28th International Conference on
Automated Planning and Scheduling

June 24–29, 2018, Delft, the Netherlands

IPC 2018 – Classical Tracks

Planner Abstracts for the

Classical Tracks

in the

International Planning Competition 2018

Edited by:

Álvaro Torralba, Florian Pommerening.

Organization

Álvaro Torralba
Saarland University, Germany

Florian Pommerening
University of Basel, Switzerland

Foreword

The International Planning Competition (IPC) is organized in the context of the International Conference on Planning and Scheduling (ICAPS). It empirically evaluates state-of-the-art planning systems on a number of benchmark problems. The goals of the IPC are to promote planning research, highlight challenges in the planning community and provide new and interesting problems as benchmarks for future research. In 2018, different tracks were organized more or less independently by different groups of organizers. This booklet describes the planners that participated in the classical tracks of the IPC 2018. This is the 9th IPC containing classical tracks making it the oldest part of IPC.

There were four classical tracks in total: in the *optimal track* planners had to find a provably optimal solution; in the *satisficing track* the goal was to find a (not necessarily optimal) plan with high quality; in the *agile track* only the time to find any plan was counted; and in the *cost-bounded track* planners had to find a plan with a cost not exceeding a given bound. A multi-core track was announced but unfortunately had to be canceled because only two teams registered for it. In parallel to the classical tracks there were also temporal and probabilistic tracks organized separately.

Across the four classical tracks, 35 planners were submitted which includes variants of some planners. Most of these planners participated in more than one track summing up to 73 entries in total. We are happy to have had a large number of 21 teams with 38 authors coming from 17 affiliations across 12 different countries.

In previous years, IPCs have driven research by using new features in the benchmarks used for evaluation and requiring participants to support them to some degree. This year, we decided to not add any new mandatory features but to push for better support of existing features. Our main focus was on conditional effects and domains that are hard to ground (e.g., because they have action schemas with many parameters). Both are features that often occur in domains that are not written by planning experts but planners only support them to a limited degree so far.

There were many submissions of domain ideas and with great help from the domain submitters we were able to include eight of them in the final set used for evaluation. We created three more domains for a total of eleven. The optimal, satisficing, and agile tracks all used ten of these, while the cost-bounded tracks used eight (leaving out domains not suited for a track).

This year planners were submitted as software containers using singularity (`singularity.lbl.gov`). We hope that this step will increase the reproducibility of results and simplify reusing the planners for future experiments. The competition was run on the sciCORE (`http://scicore.unibas.ch/`) scientific computing center at University of Basel. Each planner was run on a single core of an Intel Xeon Silver 4114 processor with no load on the other cores. The results were analyzed with Downward Lab (https://lab.readthedocs.io).

The competition results were announced at ICAPS, in June 2018, in Delft (Netherlands). Results, planners, log files, and detailed information on all tracks are available on our homepage:

`https://ipc2018-classical.bitbucket.io/`

Álvaro Torralba, Florian Pommerening
June 2018

Contents

Alien: Return of Alien Technology to Classical Planning
Masataro Asai **1**

The Freelunch Planning System Entering IPC 2018
Tomáš Balyo and Stephan Gocht **5**

IBaCoP-2018 and IBaCoP2-2018
Isabel Cenamor, Tómas de la Rosa and Fernando Fernández **9**

SaarPlan: Combining Saarland's Greatest Planning Techniques
Maximilian Fickert, Daniel Gnad, Patrick Speicher and Jörg Hoffmann **11**

OLCFF: Online-Learning $h^{C\mathbf{FF}}$
Maximilian Fickert and Jörg Hoffmann **17**

MAPlan: Reductions with Fact-Alternating Mutex Groups and h^m Heuristics
Daniel Fišer and Anonín Komenda **20**

Best-First Width Search in the IPC 2018: Complete, Simulated, and Polynomial Variants
Guillem Francès, Hector Geffner, Nir Lipovetzky and Miquel Ramírez **23**

The Complementary1 Planner in the IPC 2018
Santiago Franco, Levi H. S. Lelis, Mike Barley, Stefan Edelkamp, Moises Martinez and Ionut Moraru **28**

The Complementary2 Planner in the IPC 2018
Santiago Franco, Levi H. S. Lelis and Mike Barley **32**

The Meta-Search Planner (MSP) at IPC 2018
Raquel Fuentetaja, Michael Barley, Daniel Borrajo, Jordan Douglas, Santiago Franco and Patricia Riddle **37**

DecStar – STAR-topology DECoupled Search at its best
Daniel Gnad, Alexander Shleyfman and Jörg Hoffmann **42**

Cerberus: Red-Black Heuristic for Planning Tasks with Conditional Effects Meets Novelty Heuristic and Enchanced Mutex Detection
Michael Katz **47**

Good Old Mercury Planner
Michael Katz and Jörg Hoffmann **51**

MERWIN Planner: Mercury Enchanced With Novelty Heuristic
Michael Katz, Nir Lipovetzky, Dany Moshkovich, and Alexander Tuisov **53**

Delfi: Online Planner Selection for Cost-Optimal Planning
Michael Katz, Shirin Sohrabi, Horst Samulowitz and Silvan Sievers **57**

Planning-PDBs Planner in the IPC 2018
Ionut Moraru, Stefan Edelkamp, Moises Martinez and Santiago Franco **69**

Fast Downward Remix
Jendrik Seipp **74**

Scorpion
Jendrik Seipp **77**

Fast Downward Stone Soup 2018
Jendrik Seipp and Gabriele Röger **80**

Metis 2018
Silvan Sievers and Michael Katz **83**

Fast Downward Merge-and-Shrink
Silvan Sievers **85**

SYMPLE: Symbolic Planning based on EVMDDs
David Speck, Florian Geißer and Robert Mattmüller **91**

Alien: Return of Alien Technology to Classical Planning

Masataro Asai
guicho2.71828@gmail.com

Abstract

Recently, a modern Classical Planning framework Fast Downward (Helmert 2006) is the go-to framework in the planning community. Despite its huge contribution, the framework have several design problems: (1) code quality and extensibility, (2) preprocessing speed, (3) low-level performance. In this IPC submission, we present a new classical planning framework, Alien, with emphasis on preprocessing and low-level performance. While we do not believe this library will get as popular as Fast Downward (the only expected users of this framework are us), we believe some of the design choices might influence future planners. Tl;dr: C/C++ is too slow.

1 Introduction

In recent years (post-2000), classical planning solvers are written in languages such as C or C++, assuming that the resulting binary automatically achieves almost-optimal low-level performance thanks to the compiler improvement. This is not true, mainly due to the limitations in these languages that they cannot optimize the low-level instruction sequences for the problem instance at hand – They apply the same, fixed instruction sequence that iterates/recurses over data structures, to different data. This behavior is similar to a byte-code interpreter, which is reading and interpreting a data structure loaded on main memory instead of directly running the assembly instructions that achieve the same behavior. The approach is slower than the native compilation even if the interpreter itself is compiled by GCC or Clang.

The choice of these languages also carries a significant burden on the extensibility & composability of the resulting solver with external services such as web servers, debuggers, visualizers etc. While it is possible to connect a planner to these systems, it is typically done via a coarse-grained API such as command line options and standard I/O, and is not easily "pluggable": It does not allow users to attach knobs at every corners, unless a significant modification is performed on the source code. While such flexibility might be achieved by interpreted programming languages such as Ruby or Python, we cannot sacrifice the low-level performance for a computationally intensive task like Classical Planning. To achieve flexibility and speed, one should use a flexible compiled language.

Two examples of such lack of extensibility are the adaptation of Fast Downward for parallel processing (Jinnai and Fukunaga 2017) or external memory search (Lin and Fukunaga 2018). In the former case, they had to implement process-level parallelism via MPI because the open/closed lists in Fast Downward are assuming single-threaded execution. In the latter case, not only the state database needs to be rewritten, but also the interface to heuristic functions and other pieces should be modified because they are tightly coupled to the state database.

Finally, Fast Downward uses python-based grounding process (PDDL-SAS+ translation) which becomes slow on certain instances e.g. very large instances with repetitive structures (Asai and Fukunaga 2014) or a problem instance automatically generated from images using neural networks (Asai and Fukunaga 2018). To even start solving the problem one should improve the performance of grounding processes for actions and state variables.

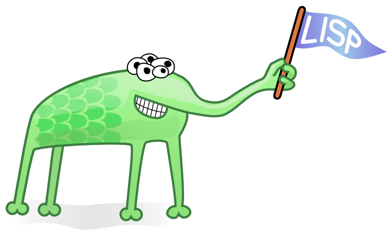


Figure 1: A **Lisp alien**. "To most programmers, Lisp seems like an entirely alien language at first- (...) this strangeness is not an arbitrary obstacle, but a necessary adjustment that imparts great power to programmers that would otherwise be unattainable. The alien Lisp mascot and quirky logo designs are designed to accentuate the awesome (and, to most people, alien) power that Lisp languages have- At the same time, they show how fun Lisp programming tends to be and that Lisp has wide appeal far beyond the stuffy academia it is sometimes wrongly associated with." (Barski 2007)

We tackle these issues by using ANSI Common Lisp programming language (Fig. 1) combined with B-Prolog (Zhou 2012), a modern high-performance Prolog solver with tabling predicates (Van Gelder, Ross, and Schlipf 1991) for preprocessing / grounding process. Common Lisp addresses the first two issues of low-level performance and extensibility. B-Prolog addresses the third issue by its heavily optimized implementation.

2 PDDL Preprocessing

2.1 Prolog the Language

Prolog is a logic programming language that describes a program with a set of horn clauses. In Prolog, a program consists of rules and facts, both described in first-order logic terms. A rule looks like `<Head> :- <Body>.`, where `Head` is a term, and `Body` consists of several *subgoal* terms `<sg>`$_1$, `<sg>`$_2$.... A fact is a rule without body, and can be written as `<Head>.`.

A term can be a number, an atom (e.g. `cat`), a variable (e.g. `X`) including wildcards (_), or a compound form `predicate(arg1, arg2...)` where each `arg`$_i$ is also a term.

To *achieve* a subgoal `predicate(arg1, arg2...)`, Prolog interpreter performs a process called *Unification*. It first looks for a rule/fact in the program whose head has the same predicate and has the same instantiated arguments (arguments that are numbers/atoms) as the subgoal. Next, it assigns a value to each unassigned variable, using existing assignments as much as possible, while it also enumerates all combinations when no existing assignment is available. For each such combination of value assignment, it tries to achieve every subgoals, hence the Prolog interpreter performs a depth-first search on the subgoals. When there is no matching rule in the program, it backtracks and tries another combination of assignments. There is a top-level clause called `initialization` and Prolog tries to achieve this on launch.

Tabling While the default depth-first search method is good for most querying purposes, it has a limitation that sometimes the program does not halt, or good performance is not achieved due to the many re-evaluation of the same subgoal. To address this issue, an alternative semantics called Well-Founded Semantics (Van Gelder, Ross, and Schlipf 1991) was proposed, in which the program is allowed to use *tabled predicates*. When the program declares a certain predicate to be *tabled*, results of achieving compound terms of the same predicate are memoized into a table and it succeeds without recursion when the same subgoal is tested next time.

A related subset of Prolog called Datalog is a much smaller subset. Instead, Prolog + Tabling is an extension of Prolog with Datalog-like efficiency.

High Performance Modern Prolog Prolog language is standardized as ISO-Prolog (Covington 1993) and many Prolog implementations (commercial / open sourced) with various focuses are available. Prolog interpreters typically process a program with a virtual machine called Warren's Abstract Machines (Warren 1985, WAM), which is heavily tuned for optimized execution of unification. While the most popular implementation is SWI-Prolog (Wielemaker et al. 2012), it's focus is the large feature set rather than the performance. In our project, we use B-Prolog (Zhou 2012) which has shown the best performance in our internal testing. B-Prolog is an originally commercial implementation which is now in public domain, and it supports tabled predicates. Typically, B-Prolog is faster than SWI by around x2, but for some corner cases by up to x120 faster (transport agl14 p01, SWI:242s, BProlog:4.4s, translate.py: 13s w/o invariant synthesis).

2.2 Preprocessing

In our planner, we reused the formal definitions described in Helmert (2009) for grounding facts/actions. We use binary formalizm instead of SAS formalism for simplicity, and thus does not perform mutex invariant synthesis, while this is future work.

The program is entirely written in Common Lisp (CL). After opening a PDDL input file, a parser written in CL parses the PDDL input, then programatically constructs a Prolog program using `cl-prolog2` (Asai 2017) library written by the author. The library makes it easy to use Prolog as a domain-specific solver by transpiling S-expressions (the same format used in PDDL: parentheses and symbols) into Prolog expressions, writes them to a file, runs a Prolog interpreter, then extracts the output.

3 Search Component

In this section, we describe the search component and the language used to implement the program, ANSI Common Lisp. In the search component, we generally follow the advice from Burns et al. (2012).

3.1 ANSI Common Lisp

ANSI Common Lisp (ANSI CL) is a *specification* of Common Lisp language, similar to C++14 or C++17. Just as C++14 has various implementations (GCC / Clang / MSVC), so does CL (SBCL / CCL / ECL / ABCL). Just as C++14 does not forbid implementing a C++ interpreter, ANSI CL does not specify if it is interpreted or compiled. **Due to historical mishaps, many misunderstand that lisp implementations are interpreters; In fact most CL implementations compile programs to native instruction sequences.** Alien uses Steel Bank Common Lisp (sbcl), the current fastest Common Lisp compiler on x86_64 environment.

Objects in Common Lisp are strongly typed and functions can be *optionally* statically typed via *declaration*. Typed functions typically compiles to a native code that is as good as code compiled by GCC. Similar to many other languages, or like `auto` keyword in C++, there is type inference mechanism and programmers should declare only a subset of variables. As typing is optional, programmers can choose to neglect it for non-performance-sensitive code, sacrificing speed for agile development.

A notable feature of ANSI CL is the inclusion of `compile` function in the standard library. That is, programmers are allowed to *compile a new code in runtime*, which allows us to generate code specifically optimized for the given problem instance / successor function / state representation / heuristic function.

Compilation of Common Lisp Programs Compilation of a Common Lisp program is quite different from those of traditional programming languages, and it allows flexibility and ultimate low-level performance.

In traditional programming languages, initially (1) there is a textual representation of the program in a file, (2) the compiler emits a compiled binary for a file, (3) the linker links the object files to produce an executable, (4) which is loaded onto main memory and the CPU runs the instructions. While it is possible to compile an additional source code while the program is running and load the object file from the running program, the process would be quite complicated. Thus, most programs are compiled off-line and is considered a fixed entity during execution.

Common Lisp has several differences from this paradigm. First, it is inherently interactive: There is a process that is always running in the background, and the compilation, linking, execution are all performed by this process.

Secondly, the compiler is separated into two controllable phases. A textual program is first parsed into a nested single-linked list structure, because the entire program is written in S-expression (Fig. 2). The compiler then compiles the linked list into an instruction sequence. Due to this separation, *programmers can systematically create a linked-list representing a certain program* and compile/execute it.

Common Lisp:

```
(defun factorial (x)
  (declare ((unsigned-byte 64) x))
  (if (= 0 x)
      1
      (* x (factorial (- x 1)))))
```

Equivalent C:

```
uint factorial(uint x){
    if (0 == x){
       return 1;
    }else{
       return x * factorial(1-x);
    }
}
```

Figure 2: Comparison of factorial implementation with Common Lisp and C.

3.2 Escaping GC for Close List

Common Lisp programs uses Garbage Collection (GC) for memory management as its inherently interactive nature allows the lifetime of certain objects to be unknown. However, GC is an expensive process: It should sweep over the entire memory, collecting and freeing the unreferenced objects. While it is acceptable to have many dead objects in performance-insensitive code such as preprocessing, object allocation inside a core inner loop is problematic, as it invokes GC and slows the entire program execution.

To completely avoid the problem of GC in inner loop, we store Close List in a large, separate memory array independently allocated by `malloc` and not managed by Lisp GC. This design choice is acceptable because in forward state-space search, close-list and other data structures are persistent, and need to be freed only when the program exits.

3.3 States and Per State Information

Memory layout of the Close List is determined after preprocessing and command line option parsing.

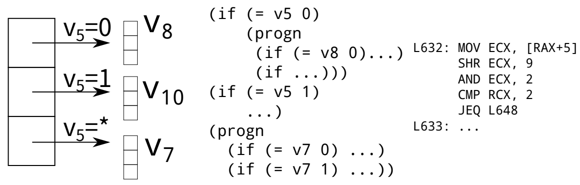
In Alien, states have binary representation. Unlike SAS formalism, the representation itself is not densely compressed. However, bit packing performed by FD is trivial in our case. Also, when a state has N propositional variables, it consumes exactly N bits in Close List, with a slight overhead of shifting some bits after reading the data from the array.

Since the layout is computed after the option parsing, per-state information such as heuristic cache or g-value can be placed right after each packed state, i.e. array-of-structures. This improves memory locality compared to Fast Downward, which has a separate array for each `PerStateInformation<T>`, i.e. structure-of-arrays (SoA) representation.

The number of bits consumed for such data is also optimized. For example, the maximum value of FF heuristics is bounded by the number of operators (maximum depth of an RPG), thus the cache takes exactly $\lceil \log |O| \rceil$ bits where $|O|$ is the number of operators.

3.4 Successor Generator as Assembly Sequence

Fast Downward uses Successor Generator (SG) to represent a successor function (Fig. 3). SG is a decision-tree whose internal node represents a precondition of an action and each node has multiple outgoing edges, one for each value in the domain of SAS variable, as well as a single don't-care edge. When generating a successor, the program recurses over this data structure, following the correct branch depending on the value of the variable in the current state, as well as following the don't care edge afterwards.



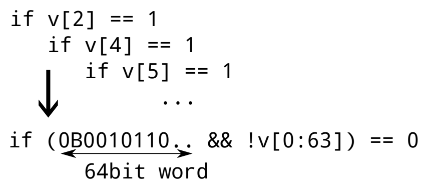
```
(if (= v5 0)
    (progn
     (if (= v8 0)...)
     (if ...)))
(if (= v5 1)
    ...)
(progn
  (if (= v7 0) ...)
  (if (= v7 1) ...))
```

```
L632: MOV ECX, [RAX+5]
      SHR ECX, 9
      AND ECX, 2
      CMP RCX, 2
      JEQ L648
L633: ...
```

Figure 3: Successor Generator and its corresponding program and the compiled binary

While this achieves a better performance compared to a naive method which checks applicability of an action one by one, Alien improves it by converting a SG into a nested if-else program that is subsequently compiled into an X86_64 instruction sequence.

This approach has two advantages over the recursion to a SG. First, it creates a function that is loaded onto L1 instruction cache rather than a L1 data cache, minimizing the data cache usage. Second, it enables every built-in CPU features including pipelining, out-of-order execution and branch prediction. Thirdly, when building a program for a SG, it could *merge* several preconditions into a single `if` statement which compiles to a single word-size test op, when their indices are within a 64bit boundary (Fig. 4).



```
if v[2] == 1
   if v[4] == 1
      if v[5] == 1
            ...

if (0B0010110.. && !v[0:63]) == 0
      64bit word
```

Figure 4: Packing nearby conditions into a single condition.

One issue with this compilation is that it takes time when the SG is large, and that the function may not fit in instruction cache. In the internal testing, compilation time increases quadratically to the number of branches. Therefore, we set a limit on the number of compiled decision nodes, and from the tip node that is not compiled, we process the remaining variables with a standard SG. The limit is heuristically determined to be 1000 nodes, which roughly keeps the function size within 20kB. For reference, Intel Haswell processor has 32kB of L1 instruction cache. Axiom evaluators and conditional effects are compiled similarly.

3.5 Heuristics and Other Search Code

After preprocessing, Alien recompiles the heuristic functions and search algorithms (e.g. eager best first search) being used. This optimizes the program by inlining the information such as the state size / number of operators.

We did not have time to implement various heuristic functions, and we have only FF (Hoffmann and Nebel 2001) heuristics based on RPG, as well as the novelty metric (Lipovetzky 2017). The planner which entered the competition is almost the same as BWFS presented in (Lipovetzky 2017).

3.6 Low-Level Performance

In our preliminary testing with blind search, Alien showed a better low-level performance compared to Fast Downward (Table 1).

In IPC2014 Agile track setting, Alien with eager FF heuristics (54 instances solved) slightly outperforms Fast Downward with FF heuristics with eager evaluation (44 instances solved).

problem	Fast Downward	Alien
sokoban p01	180095	307500
cavediving p01	255159	410255
citycar p01	178950	200229
parkprinter p01	264629	273645

Table 1: Node generation per second for Fast Downward and Alien on four easy problem instances.

4 Conclusion

We present Alien planner, which contains a new approach to write a preprocessor and the base search algorithm. While the framework is still immature, there are some notable design decisions that may also influence future planners. Future work includes the implementation of more heuristic functions, invariant synthesis, and connection to external services such as web services, online notebook or machine learning.

References

Asai, M., and Fukunaga, A. 2014. Fully Automated Cyclic Planning for Large-Scale Manufacturing Domains. In *Proc. of the International Conference on Automated Planning and Scheduling(ICAPS)*.

Asai, M., and Fukunaga, A. 2018. Classical Planning in Deep Latent Space: Bridging the Subsymbolic-Symbolic Boundary. In *Proc. of AAAI Conference on Artificial Intelligence*.

Asai, M. 2017. Cl-Prolog2 - Common Interface to the ISO Prolog implementations from Common Lisp. `github.com/guicho271828/cl-prolog2`.

Barski, C. 2007. Public domain lisp logo set. `lisperati.com/logo.html`.

Burns, E. A.; Hatem, M.; Leighton, M. J.; and Ruml, W. 2012. Implementing Fast Heuristic Search Code. In *Proc. of Annual Symposium on Combinatorial Search*.

Covington, M. A. 1993. ISO Prolog: A Summary of the Draft Proposed Standard. `fsl.cs.illinois.edu/images/9/9c/PrologStandard.pdf`.

Helmert, M. 2006. The Fast Downward Planning System. *J. Artif. Intell. Res.(JAIR)* 26:191–246.

Helmert, M. 2009. Concise finite-domain representations for pddl planning tasks. *Artificial Intelligence* 173(5-6):503–535.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation through Heuristic Search. *J. Artif. Intell. Res.(JAIR)* 14:253–302.

Jinnai, Y., and Fukunaga, A. 2017. On hash-based work distribution methods for parallel best-first search. *Journal of Artificial Intelligence Research* 60:491–548.

Lin, S., and Fukunaga, A. 2018. Revisiting Immediate Duplicate Detection in External Memory Search. In *Proc. of AAAI Conference on Artificial Intelligence*.

Lipovetzky, N. 2017. Best-First Width Search: Exploration and Exploitation in Classical Planning . In *Proc. of AAAI Conference on Artificial Intelligence*.

Van Gelder, A.; Ross, K. A.; and Schlipf, J. S. 1991. The well-founded semantics for general logic programs. *Journal of the ACM (JACM)* 38(3):619–649.

Warren, D. 1985. An abstract Prolog instruction set. *SRI Technical Note*.

Wielemaker, J.; Schrijvers, T.; Triska, M.; and Lager, T. 2012. SWI-Prolog. *Theory and Practice of Logic Programming* 12(1-2):67–96.

Zhou, N.-F. 2012. The language features and architecture of B-Prolog. *Theory and Practice of Logic Programming* 12(1-2):189–218.

The Freelunch Planning System Entering IPC 2018

Tomáš Balyo
Karlsruhe Institute of Technology
Karlsruhe, Germany
biotomas@gmail.com

Stephan Gocht
KTH Royal Institute of Technology
Stockholm, Sweden
gocht@kth.se

Abstract

The planners FREELUNCH-DOUBLY-RELAXED and FREELUNCH-MADAGASCAR consist of three components: a) an encoding of the planning problem to SAT (FREELUNCH, which also tries to solve the instance before passing it on, or MADAGASCAR) b) the driver for solving the SAT problems incrementally (INCPLAN) and c) a modern incremental SAT solver with inprocessing (LINGELING).

Introduction

The general idea of our planners is to encode the planing problem into SAT and use an of-the-shelf SAT solver to solve it. However, it turns out that lots of problems can already be solved with a simple heuristic search, which is done in FREELUNCH-DOUBLY-RELAXED before passing it to the SAT solver.

To transform the planning problems into SAT, we use MADAGASCAR (Rintanen, Heljanko, and Niemelä 2006) with the $\exists$ encoding in FREELUNCH-MADAGASCAR and in FREELUNCH-DOUBLY-RELAXED we use the *Selective* encoding (Balyo and Barták 2015) which is a heuristic selector that chooses either the *Relaxed Relaxed Exist-Step (RRES)* encoding (Balyo 2013) or the *Reinforced* encoding (Balyo, Barták, and Trunda 2015).

The *double ended incremental encoding* (Gocht and Balyo 2017) is used by the tool INCPLAN to call the SAT solver LINGELING (Biere 2013) incrementally with inprocessing.

Preliminary Definitions

Incremental SAT Solving

A *clause* is a disjunction (OR) of literals, a *literal* is a Boolean variable or its negation and a *Boolean variable* is variable with two possible values (True and False). A *conjunctive normal form* (CNF) *formula* is a conjunction (AND) of clauses. A CNF formula is satisfiable if there is an assignment of truth values to its variables that satisfies at least one literal in each clause of the formula.

The idea of incremental SAT solving is to utilize the effort already spent on a formula to solve a slightly changed but similar formula. The assumption based interface (Eén and Sörensson 2003) has two methods. One adds a clause C and the other solves the formula with additional assumptions in form of a set of literals A:

$$add(C)$$
$$solve(assumptions = A)$$

Note that we will add arbitrary formulas, but they will be transformable to CNF trivially. The method $solve$ determines the satisfiability of the conjunction of all previously added clauses under the condition that all literals in A are true. Note that it is only possible to extend the formula, not to remove parts of the formula. However, this is no restriction. If we want to add a clause C we plan to remove later we add it with an activation literal: Instead of adding C we add $(a \vee C)$. If the clause needs to be active, $\neg a$ is added to the set of assumptions for the solve step. Otherwise, no assumption is added and the solver can always satisfy the clause by assigning True to a.

SAT-Based Planning

A planning problem is to find a plan – a sequence of actions, that transforms the initial state into a goal state. The basic idea of solving planning as SAT (Kautz and Selman 1992) is to express whether a plan of length i exists as a Boolean formula F_i such that if F_i is satisfiable then there is a plan of length i. Additionally, a valid plan must be constructible from a satisfying assignment of F_i. To find a plan the plan encodings $F_0, F_1, \ldots$ are checked until the first satisfiable formula is found, which is called sequential scheduling and used in our planners. There also exist more advanced algorithms to schedule the solving of plan encodings with different makespans (Rintanen, Heljanko, and Niemelä 2006), however these approaches seem to be less beneficial, when incremental SAT solving is used.

Representation of the Plan Encoding

To apply incremental SAT solving it is necessary to break the plan encoding down to its essential parts. While an arbitrary encoding does not necessarily have this structure, all existing encodings already use this structure or are easily expressed within the presented terms.

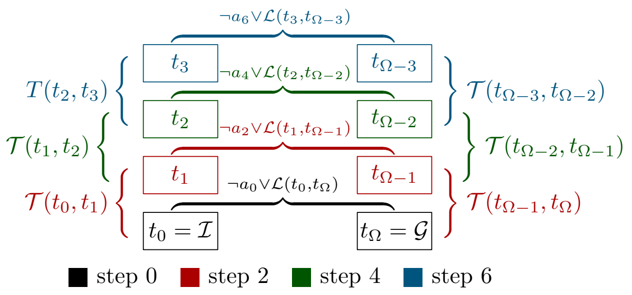

$\neg a_6 \vee \mathcal{L}(t_3, t_{\Omega-3})$
t_3
$\neg a_4 \vee \mathcal{L}(t_2, t_{\Omega-2})$
$t_{\Omega-3}$
$\mathcal{T}(t_2, t_3)$
$\mathcal{T}(t_{\Omega-3}, t_{\Omega-2})$
t_2
$\neg a_2 \vee \mathcal{L}(t_1, t_{\Omega-1})$
$t_{\Omega-2}$
$\mathcal{T}(t_1, t_2)$
$\mathcal{T}(t_{\Omega-2}, t_{\Omega-1})$
t_1
$\neg a_0 \vee \mathcal{L}(t_0, t_\Omega)$
$t_{\Omega-1}$
$\mathcal{T}(t_0, t_1)$
$\mathcal{T}(t_{\Omega-1}, t_\Omega)$
$t_0 = \mathcal{I}$
$t_\Omega = \mathcal{G}$
■ step 0 🟥 step 2 🟩 step 4 🟦 step 6

Figure 1: Visualization of the double ended incremental encoding. For clear arrangement, link clauses are only shown for every other step. The colors show which clauses are new in the particular step. All clauses from previous steps are present as well. (Gocht 2017)

The variables of the plan encoding F_i are divided into $i+1$ groups called *time points* with the same number of variables N, $v_k@j$ represents variable k at time point t_j. The clauses of F_i are divided into three groups:

- initial clauses $\mathcal{I}$: satisfied in the initial state t_0
- goal clauses $\mathcal{G}$: satisfied in the goal state t_i
- transition clauses $\mathcal{T}$: satisfied at each pair of consecutive time points $(t_0t_1, t_1t_2, \ldots, t_{i-1}t_i)$

The clauses of $\mathcal{I}, \mathcal{G}$ operate on the variables of one time point and $\mathcal{T}$ operates on the variables of two time points. $\mathcal{T}(j,k)$ indicates that the transition clauses are applied from time point j to time point k and similarly for $\mathcal{I}, \mathcal{G}$. The plan encoding F_i for makespan i can be constructed from these clause sets:

$$F_i = \mathcal{I}(0) \wedge \left(\bigwedge_{k=0}^{i-1} \mathcal{T}(k, k+1) \right) \wedge \mathcal{G}(i)$$

Double Ended Incremental Encoding

Let us also introduce the *double ended incremental encoding* (Gocht and Balyo 2017) as described in (Gocht 2017).

With non-incremental SAT solving the plan encodings are newly generated for each makespan and the SAT solver does not learn anything from previous attempts. With an incremental SAT solver it is possible to append a new time point in each step. The idea of the double ended incremental encoding is to add new states between initial state and goal state, such that clauses can be learned from both. This can be understood as having two stacks: One stack contains the time point with initial clauses at the bottom, the other contains the time point with the goal clauses at the bottom. New time points are pushed alternating to both stacks. The time points at the top of both stacks are linked together with link clauses, such that they represent the same time point, i.e. each variable has the same value in both time points: $\mathcal{L}(j,k) := \wedge_{l=1}^{N} v_l@j \Leftrightarrow v_l@k$. Activation literals ensure that only the latest link is active.

```
step (0) :
    add (I(0) ∧ G′(Ω) ∧ [¬a_0 ∨ L(0, Ω)])
    solve (assumptions = {a_0})
step (2k + 1) :                          add t_{k+1}
    add (T′(k, k + 1) ∧ [¬a_{2k+1} ∨ L(k + 1, Ω − k)])
    solve (assumptions = {a_{2k+1}})
step (2k) :                              add t_{Ω−k}
    add (T′(Ω − k, Ω − k + 1) ∧ [¬a_{2k} ∨ L(k, Ω − k)])
    solve (assumptions = {a_{2k}})
```

Note that Ω is neither a precomputed number nor a fixed upper bound but a symbol which always represents the last time point and $\Omega - k$ is the k^{th} time point before the last. In step zero there is no transition between the first time point 0 and the last time point Ω. Therefore, both time points are the same. In step one there is one transition between the first and the last time point. In step two there are two transitions in between and so on. This is visualized in Figure 1.

Implementation

To transform the planning problems into the SAT encoding, i.e. into initial clauses $\mathcal{I}$, transition clauses $\mathcal{T}$ and goal clauses $\mathcal{G}$, we use MADAGASCAR (Rintanen, Heljanko, and Niemelä 2006) with the $\exists$ encoding in FREELUNCH-MADAGASCAR.

In FREELUNCH-DOUBLY-RELAXED we use the *Selective* encoding (Balyo and Barták 2015) which is a heuristic selector that chooses either the *Relaxed Relaxed Exist-Step (RRES)* encoding (Balyo 2013) or the *Reinforced* encoding (Balyo, Barták, and Trunda 2015) based on the relative number of state variable transitions in the problem description. For more details on the selection rule see (Balyo and Barták 2015).

The tool INCPLAN takes care of the double ended incremental encoding and the interaction with the state-of-the-art incremental SAT solver Lingeling (Biere 2013), which is used to solve the formulas. An overview of this general system is shown in Figure 2.

Inprocessing

The used SAT solver Lingeling (Biere 2013) does support inprocessing. To prevent the removal of variables which are necessary to extend the formula we tell the solver to keep all variables which are contained in the link clause, which is active in the current solve step.

Pre-solving with Heuristic Forward Search

In FREELUNCH-DOUBLY-RELAXED we have an initial heuristic search phase. It is run for the first 5 minutes in the Satisfycing Track and 2 minutes in the Agile Track.

Starting with the initial state, the algorithm computes all the applicable actions in the current state that lead to a not yet visited state. For each of these actions a heuristic value is computed representing its supposed usefulness. The action

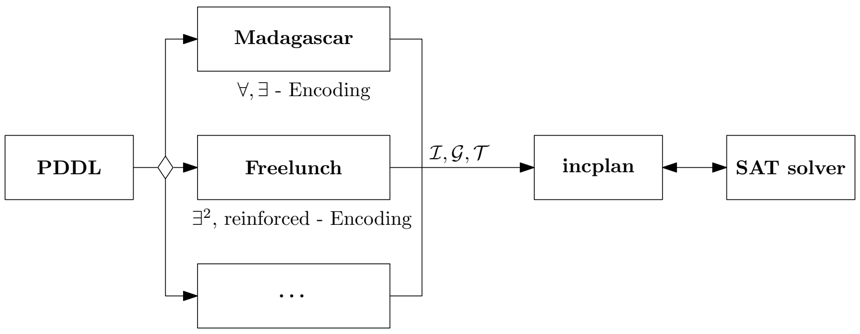

Madagascar
∀, ∃ - Encoding
PDDL
Freelunch
$\exists^2$, reinforced - Encoding
...
$\mathcal{I}, \mathcal{G}, \mathcal{T}$
incplan
SAT solver

Figure 2: System Overview

with the highest value is selected and applied on the current state. If we get to a state, that each applicable action leads to an already visited state or there is no applicable action, then we backtrack to the previous state.

The heuristic function of action usefulness is very simple and greedy. An action starts with a score of 0. If an effect of the action sets a variable to a goal value while in the current state it has a different value, then the score of the action is increased by 1. On the other hand, if an effect changes the value of a variable which already has a goal value, then the score is decreased by 1. Finally, to break the ties, the score is multiplied by 10 and a random value between 0 and 9 is added to it.

Despite its simplicity, this algorithm can solve around one half of the IPC 2011 benchmark problems very quickly. The downside of the algorithm is that it finds extremely long plans full of redundant actions. For example, for domains such as Elevators and Transport the found plans contain hundreds of thousands of actions while plans found by Fast Downward (Helmert 2006) only have a few hundred actions.

Fortunately, these extremely long plans can be easily reduced to reasonable lengths using post planning optimization techniques. Even the simplest such techniques perform very well on these plans due to their severe redundancy. The post planning optimization algorithm we used is Action Elimination (AE) (Nakhost and Müller 2010; Fink and Yang 1992). AE is a polynomial ($O(|P|^2)$) heuristic algorithm capable of removing redundant (unnecessary) actions from plans. It is not guaranteed to remove all redundant actions (which is an NP-complete problem (Fink and Yang 1992)) and it cannot add/replace actions.

In some cases the plan is very long even after the post planning optimization. The plan may be so long that the widely used plan validation tool val (Howey, Long, and Fox 2004) crashes on it.

Conclusion

In this paper we described the set of tools and the tool-chains we submitted to the IPC 2018 under the name Freelunch. We believe it represents the state-of-the-art in SAT based planning, and we hope it will perform well on the competition's benchmark problems

Post-Competition Results Commentary

Both versions of our planners performed very poorly (placed among the bottom 7 in both Tracks), which indicates that SAT based approaches are not very suitable for this years benchmarks.

The FREELUNCH-DOUBLY-RELAXED planner performed slightly better than FREELUNCH-MADAGASCAR, but this is not due to the SAT solving part. As mentioned above, FREELUNCH-DOUBLY-RELAXED has a pre-solving phase where a heuristic search with a trivial heuristic is run. This phase actually solved all the problems solved by FREELUNCH-DOUBLY-RELAXED in both Tracks and therefore none of the problems was solved in the SAT solving phase.

The heuristic search was especially successful in the Snake domain, where it performed best among all the participating solvers while solving all the benchmark instances. We believe this is thanks to the high number of goal atoms in this domain, which favors our trivial greedy heuristic.

References

Balyo, T., and Barták, R. 2015. No one satplan encoding to rule them all. In *Eighth Annual Symposium on Combinatorial Search.*

Balyo, T.; Barták, R.; and Trunda, O. 2015. Reinforced encoding for planning as sat. *Acta Polytechnica CTU Proceedings* 2(2):1–7.

Balyo, T. 2013. Relaxing the relaxed exist-step parallel planning semantics. In *ICTAI*, 865–871.

Biere, A. 2013. Lingeling and plingeling home page. `http://fmv.jku.at/lingeling/`.

Eén, N., and Sörensson, N. 2003. An extensible sat-solver. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, 502–518.

Fink, E., and Yang, Q. 1992. Formalizing plan justifications. In *In Proceedings of the Ninth Conference of the Canadian Society for Computational Studies of Intelligence*, 9–14.

Gocht, S., and Balyo, T. 2017. Accelerating SAT based planning with incremental SAT solving. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling*, 135–139.

Gocht, S. 2017. Incremental SAT solving for sat based planning.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research (JAIR)* 26:191–246.

Howey, R.; Long, D.; and Fox, M. 2004. Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In *Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on*, 294–301. IEEE.

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In *ECAI '92 Proceedings of the 10th European conference on Artificial intelligence*, volume 92, 359–363.

Nakhost, H., and Müller, M. 2010. Action elimination and plan neighborhood graph search: Two algorithms for plan improvement. In *ICAPS*, 121–128.

Rintanen, J.; Heljanko, K.; and Niemelä, I. 2006. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence* 170(12):1031–1080.

IBaCoP-2018 and IBaCoP2-2018

Isabel Cenamor, Tómas de la Rosa and Fernando Fernández
Departamento de Informatica, Universidad Carlos III de Madrid
Avda. de la Universidad, 30. Leganes (Madrid). Spain
icenamorg@gmail.com, trosa@inf.uc3m.es, ffernand@inf.uc3m.es

Abstract

This manuscript describes the IBaCoP family of planning portfolios submitted to the International Planning Competition 2018. Our portfolios are improved versions of the planners submitted to the last IPC-2014. IBaCoP-2018 is configured following a Pareto efficiency approach for selecting planners and then giving the same execution time for the selected planners. IBaCoP2-2018 decides for each problem the sub-set of planners to use. This decision is based on predictive models trained with domain/problems from previous IPCs. Both 2018 portfolios compete in the sequential satisficing and agile tracks.

Introduction

IBaCoP and IBaCoP2 are planning portfolios, which are descried in detail in (Cenamor, de la Rosa, and Fernández 2014). We build these portfolios making a pre-selection of good candidate planners from the set of known or available planners. The pre-selection technique is based on a multi-criteria approximation taking into account the time of the first solution and the quality of the best solution, both observed running planners on the training domain and problems. Then, we do planner performance modeling, for predicting the behavior of planners as a function of planning task features. From the output of the predictions, we narrow the selection of planners to finally run the portfolio in a per-instance based configuration. As in 2014, IBaCoP is the portfolio resulting from the Pareto pre-selection of planners (static configuration), and IBaCoP2 is the portfolio following the whole process described before (dynamic configuration). In IPC-2014, IBaCoP2 was the winner of the satisficing track, while IBaCoP achieved a runner-up position in the multi-core track (Cenamor, de la Rosa, and Fernández 2014).

Version for IPC-2018 have being built following the same procedure. The remarkable modifications are: version (Cenamor, de la Rosa, and Fernández 2016; Cenamor 2017).

- Models were trained with additional features regarding landmarks and relaxed plans (de la Rosa, Cenamor, and Fernández 2017).
- Data from IPC-2014 was used as part of the training data
- New base planners were included as candidates

The Components of IBaCoP

We started the construction of the portfolio with all the planners from the sequential satisficing track in IPC-2011 plus Mercury, Jasper, BFS(f) and SIW. However, there are some planners that obtained similar results, and therefore do not contribute to diversity in the portfolio. The chosen planners were selected by using the Pareto efficiency (Censor 1977) technique described before. The final components for IBaCoP-2018 are:

- jasper (Xie, Müller, and Holte 2014)
- mercury (Katz and Hoffmann 2014)
- BFS(F) (Lipovetzky et al. 2014)
- SIW (Lipovetzky et al. 2014)
- FDSS-2 (Helmert et al. 2011)
- probe (Lipovetzky and Geffner 2011)
- yashp2-mt (Vidal 2011)
- lama-2011 (Richter, Westphal, and Helmert 2011)
- lamar (Olsen and Bryce 2011)
- arvand (Nakhost, Valenzano, and Xie 2011)

We trained a predictive model for a (yes/no) classification task using Rotation Forrest (Rodriguez, Kuncheva, and Alonso 2006). The model tries to encode whether a given planner will solve the planning task or not. IBaCoP2-2018 is the result of querying this model and selecting the five planners with the best "positive" prediction confidence.

Details for Sequential Agile and Satisficing Tracks

The IBaCoP-2018 in the sequential satisficing track assigns 257 seconds to each base planner. The IBaCoP-2018 agile planner assigns the time shown in Table 1. In addition in this track, if one or more candidate planners fail, the system runs lama-2011, lamar and arvand with the remaining time. In both tracks, IBaCoP2-2018 selects five planners recommended by the predictive model, and then assigns the same time per candidate.

Planner	**Time**
jasper	80
mercury	30
BFS(F)	45
SIW	45
FDSS-2	45
probe	45
yashsp2-mt	20

Table 1: IBaCoP-2018 Agile. The list with the planners and the time in seconds per candidate.

Acknowledgements

We generated sequential portfolios of existing planners to be submitted to the International Planning Competition. Thus, we would like to acknowledge and thank the authors of the individual planners for their contribution and hard work. We would also like to thank Florian and Alvaro for their help with the compilation and bug fixing issues.

References

Cenamor, I.; de la Rosa, T.; and Fernández, F. 2014. IBaCoP and IBaCoP2 planner. *Proceedings of the 8th International Planning Competition (IPC-2014).*

Cenamor, I.; de la Rosa, T.; and Fernández, F. 2016. The IBaCoP planning system: Instance-based configured portfolios. *J. Artif. Intell. Res.* 56:657–691.

Cenamor, I. 2017. *Creating Planning Portfolios with Predictive Models*. Ph.D. Dissertation, Departamento de Informatica, Universidad Carlos III de Madrid.

Censor, Y. 1977. Pareto optimality in multiobjective problems. *Applied Mathematics and Optimization* 4(1):41–59.

de la Rosa, T.; Cenamor, I.; and Fernández, F. 2017. Performance modelling of planners from homogeneous problem sets. In Barbulescu, L.; Frank, J.; Mausam; and Smith, S. F., eds., *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017, Pittsburgh, Pennsylvania, USA, June 18-23, 2017.*, 425–433. AAAI Press.

Helmert, M.; Röger, G.; Seipp, J.; Karpas, E.; Hoffmann, J.; Keyder, E.; Nissim, R.; Richter, S.; and Westphal, M. 2011. Fast downward stone soup. *The 2011 International Planning Competition* 38.

Katz, M., and Hoffmann, J. 2014. Mercury planner: Pushing the limits of partial delete relaxation. *Proceedings of the 8th International Planning Competition (IPC-2014).*

Lipovetzky, N., and Geffner, H. 2011. Searching with probes: The classical planner probe. *The 2011 International Planning Competition* 30(29):71.

Lipovetzky, N.; Ramirez, M.; Muise, C.; and Geffner, H. 2014. Width and inference based planners: SIW, BFS (f), and PROBE. *Proceedings of the 8th International Planning Competition (IPC-2014).*

Nakhost, H.; Valenzano, R.; and Xie, F. 2011. Arvand: the art of random walks. *The 2011 International Planning Competition* 15.

Olsen, A., and Bryce, D. 2011. Randward and lamar: Randomizing the ff heuristic. *The 2011 International Planning Competition* 55.

Richter, S.; Westphal, M.; and Helmert, M. 2011. Lama 2008 and 2011. *The 2011 International Planning Competition* 50.

Rodriguez, J. J.; Kuncheva, L. I.; and Alonso, C. J. 2006. Rotation forest: A new classifier ensemble method. *IEEE transactions on pattern analysis and machine intelligence* 28(10):1619–1630.

Vidal, V. 2011. Yahsp2: Keep it simple, stupid. *The 2011 International Planning Competition* 83–90.

Xie, F.; Müller, M.; and Holte, R. 2014. Jasper: the art of exploration in greedy best first search. *The Eighth International Planning Competition (IPC-2014)* 39–42.

SaarPlan: Combining Saarland's Greatest Planning Techniques

Maximilian Fickert and **Daniel Gnad** and **Patrick Speicher** and **Jörg Hoffmann**
Saarland Informatics Campus
Saarland University
Saarbrücken, Germany
{fickert,gnad,speicher,hoffmann}@cs.uni-saarland.de

Abstract

Saarland is the smallest – yet arguably one of the most beautiful – state in Germany. And it has a lot to offer! From rich nature, over its industrial heritage, really smart people, to powerful planning techniques. SaarPlan combines the best of these planning techniques to a performant portfolio, making it the best planner in Saarland.

Introduction

Among many other things, Saarland offers a wide range of powerful planning techniques. SaarPlan combines the best of these techniques into a portfolio planner. Since in Saarland we don't care too much about optimality, but rather about getting things done, SaarPlan participates in the satisficing, agile, and bounded-cost tracks of the competition.

Some of the ingredients of SaarPlan are also used in *DecStar* (Gnad, Shleyfman, and Hoffmann 2018), and the *OLCFF* planner (Fickert and Hoffmann 2018). From DecStar, SaarPlan takes the *Star-topology decoupled search* part, trying to decompose a given planning task, if possible. In case a good problem decomposition was detected, decoupled search typically performs very well. Finding a decomposition is fast, it succeeds (or fails) quickly, so not much time is lost in the latter case. If it fails, SaarPlan tries its best with *Grey planning*, an enhancement of the red-black planning method used by the Mercury planner (Katz and Hoffmann 2014). The grey planning component of SaarPlan only considers the initial state, generates a semi-delete-relaxed plan and attempts to repair it into a real plan. Again, this process terminates quickly. If these two techniques do not manage to solve the planning task, SaarPlan uses another semi-delete relaxation method, the online-refined $h^{C\text{FF}}$ heuristic. It uses different search methods with this heuristic: an enforced hill-climbing with additional *novelty pruning*, which is followed by a greedy best-first search (GBFS) using the $h^{C\text{FF}}$ heuristic, without pruning. The GBFS part, which is very LAMA-like (Richter and Westphal 2010) – replacing the fully delete-relaxed heuristic with $h^{C\text{FF}}$ – completes SaarPlan, making it Saarland's best planner.

Big things always start in the small, ("Großes entsteht immer im Kleinen!") as we say in Saarland. Does this also hold for planning? Do great planners come from the small Saarland? The competition will answer this question.

SAARLAND
Großes entsteht immer im Kleinen.

Decoupled Search

We perform decoupled search like introduced by Gnad and Hoffmann (2018), in its integration in the Fast Downward planning system (Helmert 2006). We use the improved *fork* and *inverted-fork*, as well as the *incident-arcs* factoring methods from Gnad, Poser, and Hoffmann (2017). The outcome of the factoring process is a partitioning $\mathcal{F}$ of the variables of the planning task Π, such that $|\mathcal{F}| > 1$ and there exists $F^C \in \mathcal{F}$ such that, for every action a where $\mathcal{V}(\mathsf{eff}(a)) \cap F^C = \emptyset$, there exists $F \in \mathcal{F}$ with $\mathcal{V}(\mathsf{eff}(a)) \subseteq F$ and $\mathcal{V}(pre(a)) \subseteq F \cup F^C$. We then call $\mathcal{F}$ a *star factoring*, with *center factor* F^C and *leaf factors* $\mathcal{F}^L := \mathcal{F} \setminus \{F^C\}$.

Given a factoring $\mathcal{F}$, decoupled search is performed as follows: The search will only branch over center actions, i. e., those actions affecting (with an effect on) a variable in F^C. Along such a path of center actions π^C, for each leaf factor F^L, the search maintains a set of leaf paths, i. e., actions only affecting variables of F^L, that *comply* with π^C. Intuitively, for a leaf path π^L to comply with a center path π^C, it must be possible to embed π^L into π^C into an overall action sequence π, such that π is a valid path in the projection of the planning task Π onto $F^C \cup F^L$. A decoupled state corresponds to an end state of such a center action sequence. The main advantage over standard search originates from a decoupled state being able to represent exponentially many explicit states, avoiding their enumeration. A decoupled state can "contain" many explicit states, because by instantiating the center with a center action sequence, the leaf factors are conditionally independent. Thus, the more leaves in the factoring, the more explicit states can potentially be represented by a single decoupled state.

We will next describe a couple of extensions that have been developed for decoupled search and that we use in some of our configurations.

Symmetry Breaking in Decoupled Search

Symmetry Breaking has a long tradition in planning and many other sub-areas of computer science (Starke 1991; Emerson and Sistla 1996; Fox and Long 1999; Rintanen 2003; Pochter, Zohar, and Rosenschein 2011; Domshlak, Katz, and Shleyfman 2012). We use an extension to decoupled search, introduced by Gnad et al. (2017), which is build on *orbit search* (Domshlak, Katz, and Shleyfman 2015; Wehrle et al. 2015). An orbit is a set of states all of which are symmetric to each other. In the search, each state is mapped to a canonical representative of its orbit. In case another state from the same orbit has already been generated, a new state can safely be pruned. *Decoupled orbit search* extends this concept to decoupled states.

Decoupled Dominance Pruning

Another extension that has recently been introduced is dominance pruning (Torralba et al. 2016), where decoupled states that are dominated by other – already generated – states can be safely discarded. We only deploy a very lightweight pruning method, namely *frontier* pruning. The standard way of performing duplicate checking in decoupled search can already detect certain forms of dominance, in particular if two decoupled states have the same center state and all leaf states reachable in one state are also reachable in the other. Frontier pruning improves this by only comparing a subset of the reached leaf states, those that can possibly make so far unreached leaf states available. It has originally been developed for optimal planning, but can be easily adapted to become more efficient, when optimal solutions do not matter, by replacing the real cost of reaching a leaf state by 0, if a state has been reached at any cost.

Additionally, we also employ a leaf simulation, originally proposed by Torralba and Kissmann (2015), to remove irrelevant leaf states and leaf actions. In some domains, this can tremendously reduce the size of the leaf state spaces.

The techniques described in this sub-section are only applicable if $\mathcal{F}$ is a fork factoring.

Implementation

Decoupled Search has been implemented as an extension of Fast Downward (FD) (Helmert 2006). The implementation does not support conditional effects. By changing the low-level state representation, many of FD's built-in algorithms and functionality can be used with only minor adaptations. Of particular interest for SaarPlan are greedy best-first search (GBFS) and the h^{FF} heuristic (Hoffmann and Nebel 2001). Search algorithms and heuristics can be adapted to decoupled search using a compilation defined by Gnad and Hoffmann (2018). We will use the following notation to describe our techniques: the decoupled variant of search algorithm X is denoted **DX**. We denote fork (inverted-fork) factorings by **F** (**IF**), and factorings generated using the incident-arcs algorithm by **IA**. To combine the power of the factoring strategies, we use a portfolio approach that runs multiple strategies and picks the one with the maximum number of leaf factors. Further more, we restrict the size for the per-leaf domain-size product to ensure that the leaf state spaces are reasonably small and do not incur a prohibitive runtime overhead when generating new decoupled states. We denote this size limit by $|F^L_{max}| := \max_{F^L \in \mathcal{F}^L} \Pi_{v \in F^L} |\mathcal{D}(v)|$, where $\mathcal{D}(v)$ denotes the domain of variable v. If a fork factoring is detected, we sometimes perform frontier dominance pruning, denoted **FP** and reduce the size of the leaf state spaces removing irrelevant transitions and states (**IP**). (Decoupled) orbit search is abbreviated **(D)OSS**.

Grey Planning

In the spirit of partial delete-relaxation methods, like red-black planning, which has been used in the Mercury planner (Katz and Hoffmann 2014), SaarPlan employs an extension thereof, *grey planning* (Speicher et al. 2017). In this paper, we only give a brief summary of grey planning and refer the reader to Speicher et al. (2017) for full details.

Partial delete-relaxation methods interpolate between delete-relaxed planning and real planning (Keyder, Hoffmann, and Haslum 2012; 2014; Katz, Hoffmann, and Domshlak 2013; Domshlak, Hoffmann, and Katz 2015). Red-black planning applies the delete-relaxed semantics to a subset of state variables (the "red" ones), letting them accumulate their values, while keeping the real semantics for the others (the "black" ones). It is tractable if the dependencies between black variables are acyclic, and each black variable is *invertible*. The heuristic function based on that tractable fragment, h^{RB}, was a key part of Mercury.

Distinctions at the level of entire state variables, however, are very coarse-grained: either we remember all past values of a variable (red), or only the most recent one (black). Grey planning uses *limited-memory* state variables, instead, that allow more fine-grained relaxations through remembering a subset of their most recent values. So it partially relaxes *within* state variables, remembering only a subset of the most recent values, as needed for tractability. Limited memory can be used to substantially extend the abovementioned tractable fragment. In h^{RB}, non-invertible variables cannot be painted black, because they may not be able to go back to a previous value when required. In grey planning, the idea is to give these variables "just enough" memory to ensure this property, instead of fully delete-relaxing them. The resulting heuristic function, h^{Gray}, has proven to improve over h^{RB} in many domains.

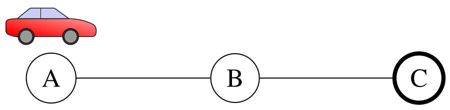
Implementation

As our other methods, h^{Gray} is implemented in Fast Downward (Helmert 2006), adopting Domshlak et al.'s *stop search* technique, which tests whether the relaxed plan is actually a real plan, and if so, stops the search. In fact, we never run an actual search with h^{Gray}, but *only* use the stop search mechanism. If it succeeds in the initial state, we are done. Else, we stop the run and proceed with the next component. We also adopted the painting strategy of Domshlak, Hoffmann, and Katz (2015), which has been used in the Mercury planner (Katz and Hoffmann 2014). The main advantage of h^{Gray} over the red-black heuristic of Mercury lies in the additional stop-search prowess, thus adding another increment to that same main advantage of h^{RB} over h^{FF}. The abbreviation of our grey-planning "no-search" approach is **GREY**. Conditional effects are not supported.

Partial Delete Relaxation with $h^{C\text{FF}}$

Like grey planning, the $h^{C\text{FF}}$ heuristic is an approach to partial delete relaxation. The partially relaxed plans must respect a given set of conjunctions C, which represent combinations of facts that must be achieved *simultaneously* (Hoffmann and Fickert 2015; Fickert, Hoffmann, and Steinmetz 2016). Whenever a conjunction is a subset of the preconditions of an action, the conjunction of these facts must be achieved instead of the facts individually.

Consider the task illustrated below. The car has to move from A to C. The car can only hold one unit of fuel, which each drive action consumes, but can be refueled at any location. Formally, there are STRIPS facts *at(x)* for the position of the car and *fuel* to indicate if the car has fuel. Initially the car is at location A and holds fuel.



A fully delete relaxed plan can ignore the fuel consumption and just apply the drive actions from A to B and B to C immediately after each other. The critical conjunction of facts that is ignored here is that the car must be at B while holding fuel before the second drive action can be executed. This conjunction can not be achieved by any of the drive actions as they consume the fuel fact. A partially relaxed plan generated by the $h^{C\text{FF}}$ heuristic respecting this conjunction would have to add the refuel action before driving from B to C, making the relaxed plan a real plan. In fact, with a sufficiently large set of conjunctions C, all plans generated by $h^{C\text{FF}}$ are real plans.

Refinement-HC with Novelty Pruning

The $h^{C\text{FF}}$ heuristic works best when the conjunctions are generated online, in particular in Refinement-HC (**RHC**) (Fickert and Hoffmann 2017a), which is an extension of enforced hill-climbing (EHC) (Hoffmann and Nebel 2001). Like standard EHC, the algorithm progresses through iterations of breadth-first search (BrFS) until a state s with lower heuristic value is found, then search continues from there. In RHC, these explorations are bounded by a fixed depth. If a state s with lower heuristic value can not be found within that bound, the heuristic is refined and the BrFS phase is restarted. Thus, RHC escapes local minima through heuristic refinement instead of brute-force search. A second extension to standard EHC are restarts from the initial state (without resetting the heuristic) whenever the search is stuck in a dead end. Due to the convergence of the partially relaxed plans generated by $h^{C\text{FF}}$ to real plans, RHC is complete.

In SaarPlan, we use an extension of Refinement-HC where the refinement criterion is based on novelty pruning instead of a simple depth bound (Fickert 2018). Instead of using BrFS with bounded depth in the local exploration phase, we perform exhaustive BrFS with incomplete novelty pruning, similar to a single iteration IW(k) of iterated width search (Lipovetzky and Geffner 2012). In our setting, a state passes the novelty test if it contains at least one novel *conjunction* $c \in C$, otherwise it is pruned. This corresponds to IW(1), but uses the conjunctions of $h^{C\text{FF}}$ instead of the individual facts. The novelty pruning is only applied in the BrFS phase, and thus only considers the states in the current BrFS exploration for pruning, not across the overall search.

GBFS and Weighted A*

Refinement-HC can not always overcome the limitations of local search. For example in Sokoban, the presence of deep dead ends has proven difficult, and global search algorithms like GBFS are much more suitable here.

Given these drawbacks, we place a time limit on Refinement-HC, as well as a growth bound on the number of conjunctions for the $h^{C\text{FF}}$ heuristic and run GBFS after Refinement-HC terminates. The growth bound on $h^{C\text{FF}}$ is motivated by the observation that in domains where Refinement-HC works well, the set of conjunctions typically does not grow excessively large.

The GBFS phase uses a dual-queue of $h^{C\text{FF}}$ and a landmarks-count heuristic (Porteous, Sebastia, and Hoffmann 2001; Richter, Helmert, and Westphal 2008). This makes it is very similar to LAMA (Richter and Westphal 2010), using $h^{C\text{FF}}$ instead of h^{FF}. The set of conjunctions for the $h^{C\text{FF}}$ heuristic is reset to a fixed size bound before starting the GBFS phase. During search, the heuristic periodically replaces old conjunctions by newly generated ones, which allows it to adapt itself to the part of the search space that is currently being explored (Fickert and Hoffmann 2017b). Again, similar to LAMA, when GBFS finds a solution, search continues with an anytime phase of weighted A* with incrementally lower weights. We cache heuristic values across the GBFS and weighted A* iterations to reduce overhead.

Implementation

Unsurprisingly, $h^{C\text{FF}}$ and the related techniques are also implemented on top of FD (Helmert 2006). Similar to the stop search technique used in our grey planning component, here we stop search whenever no conflict could be found in the refinement process, which implies that the partially relaxed plan is also a real plan.

SaarPlan Configurations

SaarPlan combines the described techniques into a sequential portfolio. In addition to the standard FD preprocessor, we perform a relevance analysis based on h^2 to simplify the planning task prior to the search (Alcázar and Torralba 2015). The mutexes found in this process are also used by the $h^{C\text{FF}}$ heuristic to reduce its computational overhead.

This section describes configuration details for the individual tracks. We use the following abbreviations:

- **PO**: dual-queue for preferred operators.
- **HA**: helpful actions pruning.
- **N**: novelty pruning.

In all tracks, we start by ignoring the action costs. Costs are ignored altogether in the agile track, and only re-introduced in the bounded-cost track if no plan below the cost bound could be found. In the satisficing track, we re-introduce the real costs upon finding the first plan.

In the following sub-sections, we detail the configurations employed in each competition track. We provide the search configurations, as well as the time each of the components is allotted (in seconds).

Satisficing Track

The portfolio configuration for the satisficing track is shown in Figure 1. By default, all configurations ignore action costs, but reintroduce them after finding the first plan.

Search	Factoring	$\|F^L_{max}\|$	Heuristic	Pruning	Runtime
DGBFS	F	$1M$	h^{FF}	FP,IP,PO	$100s$
(D)GBFS	F/IF/IA	$1/1/0.1M$	h^{FF}	(D)OSS,PO	$600s$
GREY	-	-	h^{Gray}	-	$1100s*$
RHC	-	-	$h^{C\text{FF}}$	HA,N	$1100s*$
GBFS	-	-	h^{LM},$h^{C\text{FF}}$	PO	$1100s*$
WA*	-	-	h^{LM},$h^{C\text{FF}}$	PO	$1100s*$

Figure 1: Portfolio configuration in the satisficing track. Components are launched top to bottom. Components whose timeout is marked with * share their timeout. The RHC component also has an individual timeout of $500s$.

Saarplan starts with two decoupled search configurations. The first one runs decoupled search with a fork factoring, since these typically perform better, in particular when combined with the strong leaf pruning methods (FP,IP). The second component tries all factoring strategies, and additionally enables decoupled orbit search (DOSS). If none of the factoring strategies succeeds, the component falls back to standard search using the same options (indicated by the "D" in parantheses). Both components use the h^{FF} heuristic with preferred operators in a dual-queue.

After the decoupled search components, we first attempt to find a grey plan for the initial state and check if it is a real plan. Then, we run the components of the OLCFF planner (Fickert and Hoffmann 2018), starting with Refinement-HC with novelty pruning. The final phase is a LAMA-like anytime search with GBFS and weighted A* using incrementally lower weights, where the main difference to LAMA is the use of $h^{C\text{FF}}$ instead of h^{FF}.

Agile Track

The portfolio configuration for the agile track is shown in Figure 2. All configurations ignore action costs.

In the agile track, we use a similar configuration to the satisficing track with only small differences. The second decoupled search configuration is moved to the end, and we don't need the weighted A* phase. Since the time limit is much lower in the agile track, the time limits of the individual components are reduced accordingly.

Search	Factoring	$\|F^L_{max}\|$	Heuristic	Pruning	Runtime
DGBFS	F	$1M$	h^{FF}	FP,IP,PO	$30s$
GREY	-	-	h^{Gray}	-	$170s*$
RHC	-	-	$h^{C\text{FF}}$	HA,N	$170s*$
GBFS	-	-	h^{LM},$h^{C\text{FF}}$	PO	$170s*$
(D)GBFS	F/IF/IA	$1/1/0.1M$	h^{FF}	(D)OSS,PO	$100s$

Figure 2: Portfolio configuration in the agile track. Components are launched top to bottom. Components whose timeout is marked with * share their timeout. The RHC component also has an individual timeout of $100s$.

Bounded-Cost Track

The portfolio configuration for the bounded-cost track is shown in Figure 2. All components use normal action costs, except the first two which use unit action costs and only check if the cost-bound is satisfied upon finding a solution.

Search	Factoring	$\|F^L_{max}\|$	Heuristic	Pruning	Runtime
DGBFS	F	$1M$	h^{FF}	FP,IP,PO	$100s$
(D)GBFS	F/IF/IA	$1/1/0.1M$	h^{FF}	(D)OSS,PO	$600s$
GREY	-	-	h^{Gray}	-	$800s*$
RHC	-	-	$h^{C\text{FF}}$	HA,N	$800s*$
GBFS	-	-	h^{LM},$h^{C\text{FF}}$	PO	$800s*$
(D)WA*	F/IF/IA	$10/10/1M$	h^{LM},h^{FF}	(D)OSS,PO	$300s$

Figure 3: Portfolio configuration in the cost-bounded track. Components are launched top to bottom. Components whose timeout is marked with * share their timeout. The RHC component also has an individual timeout of $400s$.

Again, we use a similar configuration to the satisficing track. We replaced the anytime phase with a (decoupled) weighted A* with $w = 3$, using h^{LM} and h^{FF} and a dual-queue for preferred operators.

Conclusion

SaarPlan combines a set of powerful planning techniques into a sequential portfolio. This portfolio is designed in a way that quick-to-terminate methods, like star-topology decoupled search or grey planning's stop-search, are applied first, to find a plan as fast as possible. More search-heavy algorithms like the online-refining $h^{C\text{FF}}$ heuristic are executed later, in case the other methods fail. We augment our techniques with recently introduced extensions like novelty pruning, and symmetry breaking in decoupled search, to further spread the range of techniques.

Acknowledgments. This work was partially supported by the German Research Foundation (DFG), under grants HO 2169/5-1 ("Critically Constrained Planning via Partial Delete Relaxation") and HO 2169/6-1 ("Star-Topology Decoupled State Space Search").

References

Alcázar, V., and Torralba, Á. 2015. A reminder about the importance of computing and exploiting invariants in planning. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*, 2–6. AAAI Press.

Domshlak, C.; Hoffmann, J.; and Katz, M. 2015. Red-black planning: A new systematic approach to partial delete relaxation. *AI* 221:73–114.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2012. Enhanced symmetry breaking in cost-optimal planning as forward search. In Bonet, B.; McCluskey, L.; Silva, J. R.; and Williams, B., eds., *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12).* AAAI Press.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2015. Symmetry breaking in deterministic planning as forward search: Orbit space search algorithm. *Technical Report IS/IE-2015-02.*

Emerson, E. A., and Sistla, A. P. 1996. Symmetry and model-checking. *Formal Methods in System Design* 9(1/2):105–131.

Fickert, M., and Hoffmann, J. 2017a. Complete local search: Boosting hill-climbing through online heuristic-function refinement. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17).* AAAI Press.

Fickert, M., and Hoffmann, J. 2017b. Ranking conjunctions for partial delete relaxation heuristics in planning. In Fukunaga, A., and Kishimoto, A., eds., *Proceedings of the 10th Annual Symposium on Combinatorial Search (SOCS'17).* AAAI Press.

Fickert, M., and Hoffmann, J. 2018. OLCFF: Online-learning $h^{C\mathrm{FF}}$. In *IPC 2018 planner abstracts.*

Fickert, M.; Hoffmann, J.; and Steinmetz, M. 2016. Combining the delete relaxation with critical-path heuristics: A direct characterization. *JAIR* 56(1):269–327.

Fickert, M. 2018. Making hill-climbing great again through online relaxation refinement and novelty pruning. In Bulitko, V., and Storandt, S., eds., *Proceedings of the 11th Annual Symposium on Combinatorial Search (SOCS'18).* AAAI Press.

Fox, M., and Long, D. 1999. The detection and exploitation of symmetry in planning problems. In Pollack, M., ed., *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, 956–961. Stockholm, Sweden: Morgan Kaufmann.

Gnad, D., and Hoffmann, J. 2018. Star-topology decoupled state space search. *AI* 257:24 – 60.

Gnad, D.; Torralba, Á.; Shleyfman, A.; and Hoffmann, J. 2017. Symmetry breaking in star-topology decoupled search. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17).* AAAI Press.

Gnad, D.; Poser, V.; and Hoffmann, J. 2017. Beyond forks: Finding and ranking star factorings for decoupled search. In Sierra, C., ed., *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17).* AAAI Press/IJCAI.

Gnad, D.; Shleyfman, A.; and Hoffmann, J. 2018. DecStar - star-topology decoupled search at its best. In *IPC 2018 planner abstracts.*

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

Hoffmann, J., and Fickert, M. 2015. Explicit conjunctions w/o compilation: Computing $h^{\mathrm{FF}}(\Pi^C)$ in polynomial time. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15).* AAAI Press.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Katz, M., and Hoffmann, J. 2014. Mercury planner: Pushing the limits of partial delete relaxation. In *IPC 2014 planner abstracts*, 43–47.

Katz, M.; Hoffmann, J.; and Domshlak, C. 2013. Who said we need to relax *all* variables? In Borrajo, D.; Fratini, S.; Kambhampati, S.; and Oddi, A., eds., *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, 126–134. Rome, Italy: AAAI Press.

Keyder, E.; Hoffmann, J.; and Haslum, P. 2012. Semi-relaxed plan heuristics. In Bonet, B.; McCluskey, L.; Silva, J. R.; and Williams, B., eds., *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, 128–136. AAAI Press.

Keyder, E.; Hoffmann, J.; and Haslum, P. 2014. Improving delete relaxation heuristics through explicitly represented conjunctions. *JAIR* 50:487–533.

Lipovetzky, N., and Geffner, H. 2012. Width and serialization of classical planning problems. In Raedt, L. D., ed., *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI'12)*, 540–545. Montpellier, France: IOS Press.

Pochter, N.; Zohar, A.; and Rosenschein, J. S. 2011. Exploiting problem symmetries in state-based planners. In Burgard, W., and Roth, D., eds., *Proceedings of the 25th National Conference of the American Association for Artificial Intelligence (AAAI'11).* San Francisco, CA, USA: AAAI Press.

Porteous, J.; Sebastia, L.; and Hoffmann, J. 2001. On the extraction, ordering, and usage of landmarks in planning. In Cesta, A., and Borrajo, D., eds., *Proceedings of the 6th European Conference on Planning (ECP'01)*, 37–48. Springer-Verlag.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR* 39:127–177.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In Fox, D., and Gomes, C., eds., *Proceedings of the 23rd National Conference of the American Association for Artificial Intelligence (AAAI'08)*, 975–982. Chicago, Illinois, USA: AAAI Press.

Rintanen, J. 2003. Symmetry reduction for SAT representations of transition systems. In Giunchiglia, E.; Muscettola, N.; and Nau, D., eds., *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS'03)*, 32–41. Trento, Italy: Morgan Kaufmann.

Speicher, P.; Steinmetz, M.; Gnad, D.; Hoffmann, J.; and Gerevini, A. 2017. Beyond red-black planning: Limited-memory state variables. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17)*, 269–273. AAAI Press.

Starke, P. 1991. Reachability analysis of petri nets using symmetries. *Journal of Mathematical Modelling and Simulation in Systems Analysis* 8(4/5):293–304.

Torralba, Á., and Kissmann, P. 2015. Focusing on what really matters: Irrelevance pruning in merge-and-shrink. In Lelis, L., and Stern, R., eds., *Proceedings of the 8th Annual Symposium on Combinatorial Search (SOCS'15)*, 122–130. AAAI Press.

Torralba, Á.; Gnad, D.; Dubbert, P.; and Hoffmann, J. 2016. On state-dominance criteria in fork-decoupled search. In Kambhampati, S., ed., *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'16)*. AAAI Press/IJCAI.

Wehrle, M.; Helmert, M.; Shleyfman, A.; and Katz, M. 2015. Integrating partial order reduction and symmetry elimination for cost-optimal classical planning. In Yang, Q., ed., *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*. AAAI Press/IJCAI.

OLCFF: Online-Learning $h^{C\textbf{FF}}$

Maximilian Fickert and **Jörg Hoffmann**
Saarland Informatics Campus
Saarland University
Saarbrücken, Germany
{fickert,hoffmann}@cs.uni-saarland.de

Abstract

OLCFF is a sequential satisficing planner based on partial delete relaxation with explicit conjunctions using the $h^{C\mathrm{FF}}$ heuristic. The heuristic can interpolate between fully delete relaxed semantics and real semantics by choosing the set of conjunctions C accordingly. Our planner is built around refining the heuristic online, which has proven to be the most effective way to use the $h^{C\mathrm{FF}}$ heuristic. The main search algorithm used by the planner is a variant of enforced hill-climbing with online refinement and novelty pruning, followed by a LAMA-like anytime phase with GBFS and weighted A* where $h^{C\mathrm{FF}}$ is used in a dual queue with a landmarks-count heuristic.

Introduction

In satisficing planning, heuristics based on the delete relaxation were part of most state-of-the-art planners for almost two decades (e.g. (Hoffmann and Nebel 2001; Richter and Westphal 2010; Katz and Hoffmann 2014)). However, the delete relaxation often ignores critical features of the planning task. These pitfalls can be diminished by "un-relaxing" part of the problem.

One approach to partial delete relaxation is red-black planning, where not all variables are relaxed, but only some of them (Domshlak, Hoffmann, and Katz 2015). The Mercury planner (Katz and Hoffmann 2014) is based on red-black planning, and was very successful at the last IPC.

We employ a different partial delete relaxation technique, where certain combinations of facts must be respected by the relaxed plans. For example in a transportation domain with fuel consumption, it can be useful to consider being in a specific location while still holding a certain amount oiif fuel. The $h^{C\mathrm{FF}}$ heuristic implements this by treating a set of conjunctions C as atomic (Fickert, Hoffmann, and Steinmetz 2016). Choosing C correctly is critical for the performance of the heuristic, since, while the accuracy increases with larger C, so does the computational complexity.

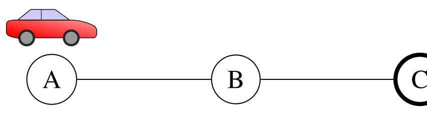
Fickert and Hoffmann (2017a) have recently shown that the $h^{C\mathrm{FF}}$ heuristic is most effective when the conjunctions are generated online. They employ a variant of enforced hill-climbing (Hoffmann and Nebel 2001), called Refinement-HC, to detect when the search is stuck in a local minimum. Whenever this happens, the heuristic is refined by adding conjunctions to C until the local minimum is removed from the search space surface.

The OLCFF planner is based on online refinement with the $h^{C\mathrm{FF}}$ heuristic. It uses an extension of Refinement-HC with novelty pruning (Lipovetzky and Geffner 2012) as its core search algorithm. The Refinement-HC phase is followed by a LAMA-like anytime search to find plans with better quality. This second phase runs $h^{C\mathrm{FF}}$ in a dual queue with a landmarks-count heuristic (Richter, Helmert, and Westphal 2008).

$h^{C\textbf{FF}}$ and Refinement-HC

Delete relaxation heuristics can be made more accurate by taking *some* delete information into account. The $h^{C\mathrm{FF}}$ heuristic generates partially relaxed plans that respect a given set of conjunctions C (Fickert, Hoffmann, and Steinmetz 2016). Achieving a conjunction $c \in C$ means achieving the individual facts represented by c *simultaneously*. Whenever a conjunction is a subset of the preconditions of an action, the conjunction of these facts must be achieved instead of the facts individually. Thus, e.g. if an action has two preconditions for which a conjunction exists, the partially relaxed plan must achieve both preconditions at the same time. A fully relaxed plan may achieve the first precondition, delete it again while achieving the second one, and can still apply the action.

Consider the task illustrated below. The car has to move from A to C. The car can only hold one unit of fuel, which each drive action consumes, but can be refueled at any location. Formally, there are STRIPS facts *at(x)* for the position of the car and *fuel* to indicate if the car has fuel. Initially the car is at location A and holds fuel.



A fully delete relaxed plan can ignore the fuel consumption and just apply the drive actions from A to B and B to C immediately after each other. The critical conjunction of facts that is ignored here is that the car must be at B while holding fuel before the second drive action can be executed. This conjunction can not be achieved by any of the drive actions as they consume the fuel fact. A partially relaxed plan generated by the $h^{C\text{FF}}$ heuristic respecting this conjunction would have to add the refuel action before driving from B to C, making the relaxed plan a real plan. In fact, with a sufficiently large set of conjunctions C, all plans generated by $h^{C\text{FF}}$ are real plans.

The $h^{C\text{FF}}$ heuristic works best when the conjunctions are generated online, in particular in Refinement-HC (Fickert and Hoffmann 2017a), which is an extension of enforced hill-climbing (EHC) (Hoffmann and Nebel 2001). Like standard EHC, the algorithm progresses through iterations of breadth-first search (BrFS) until a state s with lower heuristic value is found, then search continues from there. In Refinement-HC, these explorations are bounded by a fixed depth. If a state s with lower heuristic value can not be found within that bound, the heuristic is refined and the BrFS phase is restarted. Thus, Refinement-HC escapes local minima through heuristic refinement instead of brute-force search. A second extension to standard EHC are restarts from the initial state (without resetting the heuristic) whenever the search is stuck in a dead end. Due to the convergence of the partially relaxed plans generated by $h^{C\text{FF}}$ to real plans, Refinement-HC is complete.

Planner Components

The planner consists of two main search components. First, we run an extension of Refinement-HC with novelty pruning. The second search component runs GBFS with a dual-queue of the $h^{C\text{FF}}$ heuristic and a heuristic based on landmarks. This is followed by an anytime search with weighted A* using incrementally decreasing weights to obtain better plans (similar to LAMA).

Refinement-HC with Novelty Pruning

The core of our planner is an extension of Refinement-HC with novelty pruning (Fickert 2018). Instead of using BrFS with bounded depth in the local exploration phase, we perform exhaustive BrFS with incomplete novelty pruning, similar to a single iteration IW(k) of iterated width search (Lipovetzky and Geffner 2012). In our setting, a state passes the novelty test if it contains at least one novel *conjunction* $c \in C$, otherwise it is pruned. This corresponds to IW(1), but uses the conjunctions of $h^{C\text{FF}}$ instead of the individual facts. We denote this extension with conjunctions by IW(C). The novelty pruning is only applied in the BrFS phase, and thus only considers the states in the current BrFS exploration for pruning, not across the overall search. The simplified pseudo-code is shown in Algorithm 1.

This extension improves Refinement-HC as it takes the structure of the search space into account in the local explorations. Using novelty pruning here allows "interesting" branches (with novel states) to be explored in more depth. Sharing the set of conjunctions with $h^{C\text{FF}}$ also has a synergistic side-effect in Refinement-HC. The $h^{C\text{FF}}$ heuristic becomes more expensive to evaluate with each added conjunction, so refinement should be used carefully. On the other hand, IW(C) is less restrictive with each added conjunction. Thus, as Refinement-HC progresses, refinement will be triggered less frequently with larger C (since the novelty pruning is less aggressive), which reduces further overhead for $h^{C\text{FF}}$.

Algorithm 1: Refinement-HC with Novelty Pruning

```
$s_{best} := I$
while $h(s_{best}) \neq 0$ do
    Run IW($C$) from $s_{best}$ until a state $s$ with
      $h(s) < h(s_{best})$ is found.
    if no such state exists then
        Refine $h$ in $s_{best}$.
        continue
    $s_{best} := s$
return SOLVED
```

GBFS and Weighted A*

Refinement-HC can not always overcome the limitations of local search. For example in Sokoban, the presence of deep dead ends has proven difficult, and global search algorithms like GBFS are much more suitable here.

Given these drawbacks, we place a time limit on Refinement-HC, as well as a growth bound on the number of conjunctions for the $h^{C\text{FF}}$ heuristic and run GBFS after Refinement-HC terminates. The growth bound on $h^{C\text{FF}}$ is motivated by the observation that in domains where Refinement-HC works well, the set of conjunctions typically does not grow excessively large.

The GBFS phase uses a dual-queue of $h^{C\text{FF}}$ and a landmarks-count heuristic (Porteous, Sebastia, and Hoffmann 2001; Richter, Helmert, and Westphal 2008). This makes it is very similar to LAMA (Richter and Westphal 2010), which uses a dual queue of h^{FF} and landmarks-count, but using $h^{C\text{FF}}$ in place of h^{FF}. The set of conjunctions for the $h^{C\text{FF}}$ heuristic is reset to a fixed size bound before starting the GBFS phase. During search, the heuristic then periodically replaces old conjunctions by newly generated ones, which allows it to adapt itself to the part of the search space that is currently being explored (Fickert and Hoffmann 2017b). Again similar to LAMA, when GBFS finds a solution, search continues with an anytime phase of weighted A* with incrementally lower weights. We cache heuristic values across the GBFS and weighted A* iterations to reduce overhead.

Implementation

The planner is implemented on top of Fast Downward (Helmert 2006). It performs a relevance analysis based on h^2 in the preprocessing phase (Alcázar and Torralba 2015), which simplifies the planning task by removing superfluous actions and facts. Additionally, it finds mutex relations between facts which are used by the $h^{C\text{FF}}$ heuristic to reduce its computational overhead.

In the competition, we build our planner with profile-guided optimization[1]. We generated profiling data by run-

[1] `https://clang.llvm.org/docs/UsersManual.html#profile-guided-optimization`

ning the planner on instances from previous IPCs. This data is then used to optimize the executable, e.g. by prioritizing the layout for frequently taken branches and making better inlining decisions.

Configurations

This section describes configuration details of our planner for the individual tracks.

Satisficing Track

The planner first runs Refinement-HC with unit action costs with a timeout of 1200 seconds, and bounds the complexity growth of the $h^{C\text{FF}}$ heuristic to a factor of 8 compared to the heuristic without added conjunctions. If Refinement-HC finds a solution, it is restarted with original action costs. Otherwise, GBFS is run with unit action costs. Afterwards, we run GBFS and then weighted A^* with weights 5, 3, 2, and 1, each with original action costs. In GBFS and weighted A^*, $h^{C\text{FF}}$ is used in a dual-queue with landmarks-count, but only uses the preferred operators of the $h^{C\text{FF}}$ heuristic which improved results in preliminary experiments.

Agile and Cost-Bounded Tracks

In the agile and cost-bounded tracks, the planner also starts with Refinement-HC before running GBFS, but leaves out the anytime phase with weighted A^* since we can stop after finding the first solution. In the agile track, the time limit for Refinement-HC is set to 180 seconds.

Acknowledgments. This work was partially supported by the German Research Foundation (DFG), under grant HO 2169/5-1, "Critically Constrained Planning via Partial Delete Relaxation".

References

Alcázar, V., and Torralba, Á. 2015. A reminder about the importance of computing and exploiting invariants in planning. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*, 2–6. AAAI Press.

Clang compiler users manual, profile-guided optimization.

Domshlak, C.; Hoffmann, J.; and Katz, M. 2015. Red-black planning: A new systematic approach to partial delete relaxation. *Artificial Intelligence* 221:73–114.

Fickert, M., and Hoffmann, J. 2017a. Complete local search: Boosting hill-climbing through online heuristic-function refinement. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17)*. AAAI Press.

Fickert, M., and Hoffmann, J. 2017b. Ranking conjunctions for partial delete relaxation heuristics in planning. In Fukunaga, A., and Kishimoto, A., eds., *Proceedings of the 10th Annual Symposium on Combinatorial Search (SOCS'17)*. AAAI Press.

Fickert, M.; Hoffmann, J.; and Steinmetz, M. 2016. Combining the delete relaxation with critical-path heuristics: A direct characterization. *Journal of Artificial Intelligence Research* 56(1):269–327.

Fickert, M. 2018. Making hill-climbing great again through online relaxation refinement and novelty pruning. In Bulitko, V., and Storandt, S., eds., *Proceedings of the 11th Annual Symposium on Combinatorial Search (SOCS'18)*. AAAI Press.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Katz, M., and Hoffmann, J. 2014. Mercury planner: Pushing the limits of partial delete relaxation. In *IPC 2014 planner abstracts*, 43–47.

Lipovetzky, N., and Geffner, H. 2012. Width and serialization of classical planning problems. In Raedt, L. D., ed., *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI'12)*, 540–545. Montpellier, France: IOS Press.

Porteous, J.; Sebastia, L.; and Hoffmann, J. 2001. On the extraction, ordering, and usage of landmarks in planning. In Cesta, A., and Borrajo, D., eds., *Proceedings of the 6th European Conference on Planning (ECP'01)*, 37–48. Springer-Verlag.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39:127–177.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In Fox, D., and Gomes, C., eds., *Proceedings of the 23rd National Conference of the American Association for Artificial Intelligence (AAAI'08)*, 975–982. Chicago, Illinois, USA: AAAI Press.

MAPlan: Reductions with Fact-Alternating Mutex Groups and h^m Heuristics

Daniel Fišer and **Anonín Komenda**
Department of Computer Science, Faculty of Electrical Engineering,
Czech Technical University in Prague, Czech Republic
danfis@danfis.cz, antonin.komenda@fel.cvut.cz

Introduction

MAPlan (Fišer, Štolba, and Komenda 2015) is, originally, a multi-agent planner that we adapted for the deterministic optimal track of the International Planning Competition (IPC) 2018. The planner uses state-based heuristic search with a translator from PDDL to STRIPS (Fikes and Nilsson 1971) and then to the finite domain representation (FDR, or SAS$^+$) (Bäckström and Nebel 1995). The translator tries to reduce the STRIPS problem (i.e., to remove spurious facts and operators) using mutexes found by h^2 and h^3 heuristics (Haslum and Geffner 2000; Haslum 2009) in both progression and regression, and by removing dead-end operators (operators that cannot be part of any plan) detected using fact-alternating mutex groups (fam-groups) (Fišer and Komenda 2018). The inferred fam-groups are also useful in the disambiguation process (Alcázar et al. 2013) that is essential for the h^m heuristics, especially in regression. The fam-groups can also provide more concise encodings of the problems in FDR than the most commonly used translator from Fast Downward (Helmert 2006), although its impact on the performance is very limited (Fišer and Komenda 2018).

For IPC 2018, we prepared two configurations of the MAPlan planner. Both of the configurations use the aforementioned reductions and the A* search algorithm. They differ only in the heuristics. The first one (`maplan-1`) uses the LM-Cut heuristic (Helmert and Domshlak 2009). The second configuration (`maplan-2`) uses a simplified abstraction heuristic based on merging a certain subset of the inferred fam-groups, similarly to what is done in the merge and shrink heuristic (Helmert, Haslum, and Hoffmann 2007) and pattern databases (Culberson and Schaeffer 1998; Edelkamp 2001). This heuristic is just a preliminary work testing how big merges can fit into the memory and how many overlapping fam-groups can these merges cover.

In the following sections we briefly introduce all the methods we used.

Fact-Alternating Mutex Groups

A mutex group is a set of facts out of which maximally one can be true in any reachable state. Mutex groups are invariants that are primarily used in the translation from STRIPS to FDR for the creation of FDR variables. The inference of mutex groups is in general PSPACE-Complete. The fact-alternating mutex groups (fam-groups) were first introduced by Fišer and Komenda (2018) as a subclass of mutex groups of which inference is NP-Complete and they described an algorithm based on the integer linear programming that is complete with respect to maximal fam-groups. This algorithm is implemented in the MAPlan planner using CPLEX solver.

Fam-group of a certain form can be used for a detection of operators that can produce only dead-end states, i.e., states from which it is impossible to reach any goal state. Such operators can be safely removed from the planning task, because these operators cannot be part of any plan. This is one of the methods we use for a reduction of the input planning problem.

Another useful application of fam-groups is in the disambiguation of operators' preconditions and the goal specification. Disambiguation is a simple process that extends a set of facts with the fact that is the only possibility given a set of facts out of which exactly one is a part of every state. For example, consider an operator's precondition $\{f_1, f_2\}$ and let us assume that every state must contain one of the facts $\{f_3, f_4, f_5\}$. If we know that there are no reachable states that contain either $\{f_1, f_3\}$ or $\{f_1, f_4\}$, then we can safely extend the precondition with f_5, because it is the only possibility. A certain subset of fam-groups can be easily identified as mutex groups that has not maximally one, but exactly one fact in every reachable state. These fam-groups together with the h^m mutexes, described in the next section, are used for the disambiguation which in turn improves a pruning of operators and facts.

h^2 and h^3 Mutexes

A generalization of the h$^{\max}$ heuristic to a family of h^m heuristics (Haslum and Geffner 2000; Haslum 2009) offers a method for the generation of mutex invariants that can be used for reductions of the planning problems (Alcázar and Torralba 2015). h$^{\max}$ is a widely known and a well understood admissible heuristic for STRIPS planning. The heuristic value is computed on a relaxed reachability graph as a cost of the most costly fact from a conjunction of reachable facts. The heuristic works with single facts, but it can be generalized to consider a conjunction of at most m facts instead.

h^1 would then be equal to $h^{\max}$, h^2 would build the reachability graph with single facts and pairs of facts, etc. Unfortunately, the cost of the computation increases exponentially in m, which is why, usually, only h^1 and h^2 variants are used.

The pruning of operators and facts proposed by Alcázar and Torralba (2015) uses h^2 heuristic combined with the disambiguation process as a reachability analysis that can prove that certain operators and facts can be safely removed from the problem. The algorithm runs, in turn, in progression and in regression (on the dual planning problem (Massey 1999; Pettersson 2005; Suda 2013)) removing the unreachable facts and operators in each cycle until a fixpoint is reached. The h^2 heuristic in the algorithm can be easily switched to any heuristic from the h^m family, but with a considerably increased computational time. In the MAPlan planner, we use the pruning with h^2 and fam-groups used for the disambiguation, and we also added the variant with h^3, but we restricted the running time of a single cycle to one minute. That is, if the running time of the pruning using h^3 in any direction exceeds the limit of one minute, the pruning is prematurely terminated and it is not used anymore for that problem. This is a rather weak restriction that still could cause that the whole time limit will be consumed just by computing h^3, but our tests on the domains from IPC 2011 and 2014 showed that this simple rule should be sufficient to disable this type of pruning for large problems.

Merge without Shrink

The heuristic function used in the second configuration of the submitted MAPlan planner (`maplan-2`) is built on the inferred set of maximal fam-groups. It uses a similar approach as is used in pattern databases (PDB) (Culberson and Schaeffer 1998; Edelkamp 2001), but instead of computing the projections on the FDR variables, we use projections on the fam-groups that are usually overlapping in a sense that they contain a common subset of facts.

We start with the complete set of maximal fam-groups and we merge (i.e., compute a synchronized product from the corresponding projected transition systems) as many fam-groups as possible to fit into memory. We use a 7 GB memory block in which we are, usually, able to fit around 20 000 abstract states. From the resulting merge we save the abstract states along with the cost of the cheapest path to a goal state as a heuristic estimate in the same way as is done in PDBs. This process is repeated so that each maximal fam-group is contained in at least one big merge. The selection of the fam-groups used for merging is controlled by a greedy rule that prefers fam-groups that share the most facts. The resulting heuristic estimate is the maximum of the estimates over all merges.

We realize that we could, probably, get much better estimates if we adapted our approach to the framework of either merge and shrink heuristics or PDBs, but this work was not finished in time for IPC. However, we believe that using mutex groups directly instead of FDR variables (which can be considered as derivatives of mutex groups) can lead to better merge and shrink strategies and to improvements in PDB heuristics, which is a focus of our future research.

Acknowledgements

This research was supported by the Czech Science Foundation (grant no. 18-24965Y). Computational resources were provided by the CESNET LM2015042 and the CERIT Scientific Cloud LM2015085, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures".

References

Alcázar, V., and Torralba, Á. 2015. A reminder about the importance of computing and exploiting invariants in planning. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS)*, 2–6.

Alcázar, V.; Borrajo, D.; Fernández, S.; and Fuentetaja, R. 2013. Revisiting regression in planning. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI)*, 2254–2260.

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS+ planning. *Computational Intelligence* 11:625–656.

Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3):318–334.

Edelkamp, S. 2001. Planning with pattern databases. In *Proceedings of the 6th European Conference on Planning*, 13–24.

Fikes, R., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.* 2(3/4):189–208.

Fišer, D., and Komenda, A. 2018. Fact-alternating mutex groups for classical planning. *J. Artif. Intell. Res.* 61:475–521.

Fišer, D.; Štolba, M.; and Komenda, A. 2015. MAPlan. In *Competition of Distributed and Multi-Agent Planners (CoDMAP)*, 8–10.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS)*, 140–149.

Haslum, P. 2009. $h^m(P) = h^1(P^m)$: Alternative characterisations of the generalisation from $h^{\max}$ to h^m. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, 354–357.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS)*, 176–183.

Helmert, M. 2006. The Fast Downward planning system. *J. Artif. Intell. Res. (JAIR)* 26:191–246.

Massey, B. 1999. *Directions in planning: Understanding the flow of time in planning*. Ph.D. Dissertation, University of Oregon.

Pettersson, M. P. 2005. Reversed planning graphs for relevance heuristics in ai planning. In *Planning, Scheduling and Constraint Satisfaction: From Theory to Practice*, volume 117, 29–38. IOS Press.

Suda, M. 2013. Duality in STRIPS planning. *CoRR* abs/1304.0897.

Best-First Width Search in the IPC 2018: Complete, Simulated, and Polynomial Variants

Guillem Francès
University of Basel
Basel, Switzerland
first.last@unibas.ch

Hector Geffner
ICREA & U. Pompeu Fabra
Barcelona, Spain
first.last@upf.edu

Nir Lipovetzky
University of Melbourne
Melbourne, Australia
first.last@unimelb.edu.au

Miquel Ramírez
University of Melbourne
Melbourne, Australia
miguel.ramirez@unimelb.edu.au

Abstract

Width-based search algorithms have recently emerged as a simple yet effective approach to planning. Best-First Width Search (BFWS) is one of the most successful satisficing width-based algorithms, as it strikes a good balance between an effective exploration based on a measure of *state novelty* and the exploitation provided by traditional goal-directed heuristics. Several conceptually interesting BFWS variants have recently been shown to offer state-of-the-art performance, including a *polynomial-time* BFWS planner which is incomplete but fast and effective, and a black-box BFWS planner that can plan efficiently with *simulators*, i.e. when the transition function of the problem is represented as a black-box function. In this paper, we describe six BFWS planners involving these variations that we have entered into the 2018 International Planning Competition.

Introduction

Planning as heuristic search is one of the most successful computational approaches to classical planning developed so far (Bonet and Geffner 2001; Hoffmann and Nebel 2001), dominating several of the past editions of the International Planning Competition (IPC). The essential component of this approach is the *automatic derivation* of an heuristic function that informs the search from the declarative representation of the problem in some modeling language such as STRIPS or PDDL (Fikes and Nilsson 1971; McDermott 2000). This is usually coupled with a suitable search strategy and a number of search improvements such as helpful actions, delayed evaluation and multiple search queues (Hoffmann and Nebel 2001; Helmert 2006).

A recent and significant departure from this approach are *width-based search algorithms*. Still framed within the *planning as search* paradigm, width-based algorithms however do away with the reliance on heuristics and *means-ends* analysis (Newell and Simon 1963), and use instead a powerful exploration mechanism based on a structural, goal-agnostic notion of *state novelty*, which roughly assigns value to states based on how novel they are with respect to the states already visited by the search strategy being employed (Lipovetzky and Geffner 2012). The exploration mechanism offered by the width-based approach can be combined with traditional heuristics in greedy best-first-like search strategies to produce state-of-the-art satisficing planning strategies collectively called Best-First Width Search (BFWS) (Lipovetzky and Geffner 2017a; Katz *et al.* 2017), but it also has other interesting properties that go beyond performance. First, state novelty measures seem to be a particularly effective pruning mechanism. Lipovetzky and Geffner (2017b) have developed *incomplete but polynomial* BFWS variations with a simple modification that consists on roughly pruning from the search those nodes which are not novel enough. The resulting algorithm solves more instances from previous competitions than IPC-winning, exponential-time planners such as LAMA or FF (Richter and Westphal 2010; Hoffmann and Nebel 2001).

Second, width-based methods constitute a surprising departure from previous planning research, as they do not require a declarative definition of the action model, an essential component of virtually all previous approaches, from the first means-ends and partial-order planners (Newell and Simon 1963; Tate 1977; Nilsson 1980) to the latest SAT, OBDD, and heuristic search planners (Kautz and Selman 1996; Edelkamp and Kissmann 2009; Richter and Westphal 2010; Rintanen 2012). Francès *et al.* (2017) show how width-based methods are an effective means of dealing with the standard IPC benchmarks even when *no information on the action structure is available* to be used in the computation of e.g. heuristics or SAT models, that is, when the transition function of the problem is given as a black box. This is relevant, as there is a wide set of problems which fit the classical planning model, but whose dynamics are not easily represented in declarative languages, cf. the Atari Learning Environment (Bellemare *et al.* 2013), the games of the General Video Game competition (Perez-Liebana *et al.* 2016), Angry Birds (Renz 2015) and Minecraft (Johnson *et al.* 2016), all of which expose action models through procedural, black-box interfaces that preclude the use of most classical planners. At the same time, having effective planning algorithms that do not rely on a declarative representation of the action model greatly reduces the challenge of modeling, as arbitrary, high-level language constructs such as axioms or semantic attachments (Thiébaux *et al.* 2005; Dornhege *et al.* 2009) can be seamlessly dealt with.

Width-based methods have also recently been extended beyond classical planning to tackle finite horizon MDPs (Geffner and Geffner 2015), partially observable problems (MacNally *et al.* 2018) and problems with hybrid discrete and continuous dynamics (Ramirez *et al.* 2018). We here focus on the width-based classical planners that we have submitted to this year's International Planning Competition.

The remainder of this paper is organized as follows. We first present the essential ideas of width-based search, then briefly describe the Best-First Width Search (BFWS) framework, and highlight two interesting possibilities of BFWS-derived planners which are used in some of our submitted planners: polynomial runtime guarantees which make BFWS incomplete but still quite effective, and the ability of planning effectively on black-box representations of the transition functions. We conclude by briefly reviewing the actual planners entered into the competition. Many details have been omitted for the sake of brevity, but we provided pointers to the relevant literature where necessary.

Width-Based Search

Width-based search algorithms are forward state-space search algorithms that rely on the key notion of *novelty* of a state (Lipovetzky and Geffner 2012). Assuming that a state is a set of propositional atoms, as standard in STRIPS-based classical planning, the novelty $w(s)$ of a state s is the size of the smallest set of atoms Q such that s is *the first state encountered in the search* where $Q \subseteq s$. Thus, if s is the first state on the search that contains a certain atom p, then $w(s) = 1$. If no such atom exists, but s is the first state on the search that contains a certain pair of atoms $\{p, q\}$, then $w(s) = 2$, and so on. An important property of the novelty of a state is that it is a search-dependent but goal-independent measure whose computation requires only knowledge about the structure of the state.

The simplest width-based algorithm is the parametric IW(k), a standard breadth-first search where any newly-generated state s with novelty $w(s) > k$ is pruned (Lipovetzky and Geffner 2012). IW(k) converges to breadth-first search as the value of k approaches the number n of atoms in the problem, but its time and space complexity are exponential only in k, hence polynomial if we consider a fixed value of k. Interestingly, IW(k) has been shown to solve *any instance* of many of the standard benchmark domains with $k = 2$, i.e. in quadratic time, provided that the goal is a single atom (Lipovetzky and Geffner 2012; Lipovetzky 2014). This is because such domains have a small and bounded *width* ω that does not depend on the size of the instance and such that IW(k) with $k = \omega$ can (optimally) solve any of their instances.

When goals are however not restricted to single atoms but can be arbitrary conjunctions, strategies more sophisticated than IW(k) are necessary. Different width-based algorithms have been proposed to address that challenge, such as *Serialized IW* (SIW) (Lipovetzky and Geffner 2012), SIW$^+$ or DFS$^+$ (Lipovetzky and Geffner 2014). The most successful among these approaches, which we describe next, is the generic search schema known as Best-First Width Search.

Best-First Width Search

Lipovetzky and Geffner (2017a) have recently shown that state-of-the-art performance over the standard classical planning benchmarks can be achieved when the exploration afforded by structural measures of width is combined with the exploitation offered by traditional heuristic search methods. BFWS is a standard best-first search that uses an extended definition of *novelty* of a search node *given certain partitioning functions* as the main criterion to prioritize nodes in the open list. The novelty $w(s)$ of a state s *given functions* $h_1, \dots, h_n$ is defined as the size of the smallest set of atoms Q such that s is the first state encountered in the search where all atoms in Q are true at the same time, *considering only those previous states s' with equal h_i-values*, i.e., such that $h_i(s) = h_i(s')$ for $i = 1, \dots, n$. This novelty measure is also written as $w_{\langle h_1,\dots,h_n \rangle}(s)$.

The best-performing BFWS planner described in (Lipovetzky and Geffner 2017a) is BFWS(f_5), which uses $w = w_{\langle \#g, \#r \rangle}$, where $\#g(s)$ counts how many of the atomic goals of the problem are not true in s, and $\#r(s)$ is a path-dependent approximation of progress towards achieving a certain set $R(s)$ of atoms which are considered to be relevant to reach the problem goal from state s. A complete description of the algorithm can be found in (Lipovetzky and Geffner 2017a); for the sake of brevity, we here simply note that different alternatives in defining $R(s)$ are possible; the one that works best in (Lipovetzky and Geffner 2017a) is computed from a delete-free relaxed plan computed from s (Hoffmann and Nebel 2001).

Polynomial BFWS

Best-First Width Search provides an alternative to the IW(k) and Serialized IW(k) algorithms (Lipovetzky and Geffner 2012) which is *complete*, but this necessarily implies that the polynomial-time nature of IW(k) is lost. Lipovetzky and Geffner (2017b) present additional variations of BFWS which have guaranteed polynomial runtime, and in spite of being incomplete, can solve a surprising amount of the classical planning benchmarks from previous IPCs. The first such variant is k-BFWS, which is equal to BFWS but prunes from the search those generated states s with novelty $w(s) > k$. The second variant, k-M-BFWS, relaxes that strict pruning criterion by allowing for the expansion of at most M states with novelty higher than k, provided that they are direct descendants of some state s with novelty $w(s) < k$ (Lipovetzky and Geffner 2017b).

Black-Box BFWS

One of the properties of width-based algorithms is that the computation of novelty they rely on only requires knowledge about the structure of the state, not about the action model of the problem. This has allowed the successful use of this approach in simulated environments such as the Atari Learning Environment (Bellemare *et al.* 2013), where a declarative definition of the action model is not available or would be much harder to obtain than a procedural, black-box implementation of the transition function of the problem (Lipovetzky *et al.* 2015; Shleyfman *et al.* 2016).

On this same line, Francès *et al.* (2017) present BFWS(R), a generalization of the BFWS(f_5) algorithm described above which uses alternative strategies for computing the sets $R(s)$ *that do not require a declarative representation of the action model*. The basic idea in most of these strategies is to run a polynomial preprocessing phase where the IW(1) (and, if necessary, IW(2)) algorithm is run from the initial state of the problem to conduct an exploration of the novelty-1 (and, eventually, novelty-2) polynomial subspace of the state space that allows us to identify which of the problem atoms lie on some path that reaches at least some of the atoms in the goal conjunction. The key assumption behind this strategy is that the problem goal is expressed as a conjunction of atoms, and that most goal atoms can be *individually* reached by the polynomial IW(1) or IW(2) algorithms,[1] assumptions which many of the benchmarks from past IPCs share.

Competition Planners

We here briefly describe the characteristics of the 6 different width-based planners submitted to the competition. Table 1 summarizes the properties of each of the submitted planners in terms of completeness, complexity bounds, and their ability to deal with simulators, i.e. black-box representations of the transition function. Our two simulation-based planners, FS-blind and FS-sim, use the PDDL action model of the problem only at preprocessing, to (1) compile an efficient black-box representation of the transition function based on Fast Downward's *successor generator* (Helmert 2006), and (2) perform an ASP-based reachability analysis through the Clingo ASP solver (Gebser *et al.* 2012). The fact that these planners completely ignore the action model after this preprocessing can be seen as an unnecessary handicap in the context of the competition, but we are interested in observing the actual performance of this strategy, which has interesting applications beyond PDDL-based planning. All planners are implemented mostly in `C++`, with some parsing and preprocessing implemented in `Python`, and are built on top of the `LAPKT` planning toolkit (Ramirez *et al.* 2015).

BFWS-*preference* This is the BFWS(f_5) planner described above (Lipovetzky and Geffner 2017a), with one difference for the *satisficing* track submission: once BFWS(f_5) finds a solution, the plan cost is given as an upper bound to the weighted A* (WA*) implementation used in LAMA (Richter and Westphal 2010), which then runs to optimize solution quality until the timeout is reached. The problem given to WA* is preprocessed by h^2 to reduce the number of actions and minimize the search effort (Alcázar and Torralba 2015). This planner has been submitted to the *agile* and *satisficing* tracks.

BFWS-*polynomial* This is the polynomial k-BFWS(f_5) (Lipovetzky and Geffner 2017b), which runs BFWS(f_5) but prunes those nodes whose novelty is higher than k. The planner runs 1-BFWS first; if no solution is found, then a sequence of 2-M-BFWS calls with $M = 1, 2, 4, 8, 16, 32$ follows, where M is a parameter that stands for how many children n' of any node n with novelty $w(n) \leq 2$ have themselves novelty $w(n') > 2$ but are *not pruned*. To keep the submission polynomial, no optimization step has been used on the *satisficing* track. This planner has been submitted to the *agile* and *satisficing* tracks.

Dual-BFWS Dual-BFWS (Lipovetzky and Geffner 2017a) uses the polynomial and incomplete 1-BFWS search, pruning all nodes whose novelty is bigger than 1. If this incomplete search fails, a complete BFWS(f_6) search is run, where $f_6 = \langle w_{\langle h_L \rangle}, help, h_L, w_{\langle h_{\mathrm{FF}} \rangle}, h_{\mathrm{FF}} \rangle$ combines novelty measures with the landmark-based h_L heuristic (Richter and Westphal 2010), helpful actions and h_{FF} (Hoffmann and Nebel 2001). This type of dual architecture is present in early successful planners such as FF. This planner has been submitted to the *agile*, *satisficing* and *cost-bounded* tracks. The *satisficing* track submission includes the same WA*-based optimization as described for the BFWS-*preference* planner, whereas the *cost-bounded track* submission just uses the bound to prune solutions while searching.

DFS$^+$ This is the extension of SIW$^+$ described in (Lipovetzky and Geffner 2014), but instead of increasing the bound of IW(k) until a new goal is reached or the problem is solved, when the bound is 2, we backtrack to the last IW search and continue searching for other states that achieve one more goal. DFS$^+$ can be approximated as a BFWS(f_5) where the evaluation function is reversed as $f_5 = \langle \#g, w_{\langle \#r, \#g \rangle} \rangle$, i.e. preferring first states that achieve more goals, and then breaking ties by novelty extended with the goal and relax plan counters. DFS$^+$ is polynomial, but the number of nodes it expands depends on the number of states that decrease the count $\#g$ of unachieved goal atoms within each IW$^+$(2) call, which is hard to estimate. This planner has been submitted to the *agile* and *satisficing* tracks, with no optimization step for the latter.

FS-blind FS-blind is the BFWS(R_0) simulation-based planner as described in (Francès *et al.* 2017). The planner runs a Best-First Width Search where the set of relevant atoms $R(s)$ is always taken to be the empty set, which effectively means that the only information about the goal that the planner exploits is a simple goal-count heuristic that evaluates how many atoms in the goal conjunction are satisfied in each state. This planner has been submitted to the *agile* and *satisficing* tracks.

FS-sim FS-sim is the BFWS(R^*_G) simulation-based planner as described in (Francès *et al.* 2017), which is like FS-blind above but exploits additional information about the goal, inferred in an extra preprocessing step by running the IW(1) and IW(2) algorithms from the initial state of the problem (as described in the *Simulation-Based BFWS* section above). This planner has been submitted to the *agile* and *satisficing* tracks.

[1]Note that finding plans that reach each of the goals individually is different than finding plans that reach all goals *jointly*.

Planner	Complete	Polynomial	Black-box
BFWS-preference	x		
BFWS-polynomial		x	
Dual-BFWS	x		
DFS$^+$		x	
FS-blind	x		x
FS-sim	x		x

Table 1: Properties of width-based planners submitted to IPC 2018.

Language Expressivity

All of the planners submitted to the competition support universal quantification and conditional and universally-quantified effects, but do not have full support for axioms. The two BFWS(R)-based planners (FS-blind, FS-sim) are implemented in the FS planner (Francès and Geffner 2015; 2016), which deals with problems specified in the Functional STRIPS language (Geffner 2000), a superset of STRIPS with support for function symbols. The planner additionally supports a number of interesting extensions which inspired the development of the BFWS(R) algorithms, such as black-box specifications of the transition function, or of the procedural denotation of certain fixed predicate and function symbols, also referred to as semantic attachments (Dornhege *et al.* 2009)

The rest of the submitted planners (BFWS-*preference*, BFWS-*polynomial*, Dual-BFWS and DFS$^+$) support conditional and universally-quantified effects, as said, but their use of the Fast Downward parser, which occasionally transforms universal quantifiers into axioms, make these planners fail in these rare cases, since support for axioms is not yet implemented.

Summary

We have presented the six satisficing classical planners submitted to the 2018 International Planning Competition. These six planners explore different conceptually interesting variants of best-first width search: some are complete, some are incomplete but polynomial, and some are black-box procedures that do not require any declarative representation of actions in terms of preconditions and effects.

Acknowledgments

M. Ramírez and N. Lipovetzky have been partially funded by DST.

References

V. Alcázar and A. Torralba. A reminder about the importance of computing and exploiting invariants in planning. In *ICAPS*, pages 2–6, 2015.

M. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *JAIR*, 47, 2013.

B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1–2):5–33, 2001.

C. Dornhege, P. Eyerich, T. Keller, S. Trüg, M. Brenner, and B. Nebel. Semantic attachments for domain-independent planning systems. In *Proc. ICAPS*, 2009.

S. Edelkamp and P. Kissmann. Optimal symbolic planning with action costs and preferences. In *Proc. IJCAI*, 2009.

R. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 1:27–120, 1971.

G. Francès and H. Geffner. Modeling and computation in planning: Better heuristics for more expressive languages. In *Proc. ICAPS*, 2015.

G. Francès and H. Geffner. E-STRIPS: Existential quantification in planning and constraint satisfaction. In *Proc. 25th Int. Joint Conf. on Artificial Intelligence*, page 3082, 2016.

G. Francès, M. Ramırez, N. Lipovetzky, and H. Geffner. Purely declarative action representations are overrated: Classical planning with simulators. In *Proc. IJCAI*, 2017.

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Answer set solving in practice. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(3):1–238, 2012.

T. Geffner and H. Geffner. Width-based planning for general video-game playing. In *Proc. AIIDE*, 2015.

H. Geffner. Functional STRIPS. In J. Minker, editor, *Logic-Based Artificial Intelligence*, pages 187–205. Kluwer, 2000.

M. Helmert. The Fast Downward planning system. *JAIR*, 26:191–246, 2006.

J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *JAIR*, 14:253–302, 2001.

M. Johnson, K. Hofmann, T. Hutton, and D. Bignell. The Malmo platform for AI experimentation. In *Proc. IJCAI*, 2016.

M. Katz, N. Lipovetzky, D. Moshkovich, and A. Tuisov. Adapting novelty to classical planning as heuristic search. In *Proc. ICAPS*, 2017.

H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. AAAI*, 1996.

N. Lipovetzky and H. Geffner. Width and serialization of classical planning problems. In *Proc. ECAI*, 2012.

N. Lipovetzky and H. Geffner. Width-based algorithms for classical planning: New results. In *Proc. ECAI*, pages 88–90, 2014.

N. Lipovetzky and H. Geffner. Best-first width search: Exploration and exploitation in classical planning. In *Proc. AAAI*, 2017.

N. Lipovetzky and H. Geffner. A polynomial planning algorithm that beats lama and ff. In *Proc. ICAPS*, 2017.

N. Lipovetzky, M. Ramirez, and H. Geffner. Classical planning with simulators: Results on the atari video games. In *Proc. the Joint International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

N. Lipovetzky. *Structure and Inference in Classical Planning*. AI Access, 2014.

A. M. MacNally, N. Lipovetzky, M. Ramirez, and A. R. Pearce. Action selection for transparent planning. In *Proc. AAMAS*, 2018.

D. McDermott. The 1998 AI Planning Systems Competition. *AI Magazine*, 21(2):35–56, 2000.

A. Newell and H. Simon. GPS: a program that simulates human thought. In E. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 279–293. McGraw Hill, 1963.

N. Nilsson. *Principles of Artificial Intelligence*. Tioga Press, 1980.

D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, and S.M. Lucas. General video game AI: Competition, challenges and opportunities. In *Proc. AAAI*, 2016.

M. Ramirez, N. Lipovetzky, and C. Muise. Lightweight Automated Planning ToolKiT. `http://lapkt.org/`, 2015.

M. Ramirez, M Papasimeon, N. Lipovetzky, L. Benke, T. Miller, A. R. Pearce, E. Scala, and M. Zamani. Integrated hybrid planning and programmed control for real–time uav maneuvering. In *Proc. AAMAS*, 2018.

J. Renz. AIBIRDS: The Angry Birds artificial intelligence competition. In *Proc. AAAI*, 2015.

S. Richter and M. Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR*, 39:122–177, 2010.

J. Rintanen. Planning as satisfiability: Heuristics. *Artificial Intelligence*, 193:45–86, 2012.

A. Shleyfman, A. Tuisov, and C. Domshlak. Blind search for atari-like online planning revisited. In *Proc. IJCAI*, 2016.

A. Tate. Generating project networks. In *Proc. IJCAI*, 1977.

S. Thiébaux, J. Hoffmann, and B. Nebel. In defense of pddl axioms. *Artif. Intell.*, 168(1-2):38–69, 2005.

The Complementary1 Planner in the IPC 2018

Santiago Franco[1], **Levi H. S. Lelis**[2], **Mike Barley**[3], **Stefan Edelkamp**[4], **Moises Martinez**[4], **Ionut Moraru**[4]
[1] School of Computing and Engineering, University of Huddersfield, UK
[2] Departamento de Informática, Universidade Federal de Viçosa, Brazil
[3] Computer Science Department, Auckland University, New Zealand
[4] Computer Science Department, Kings College London, United Kingdom
s.franco@hud.ac.uk, levi.lelis@ufv.br, barley@cs.auckland.ac.nz,
stefan.edelkamp@kcl.ac.uk, moises.martinez@kcl.ac.uk, ionut.moraru@kcl.ac.uk

Abstract

This planner is an updated and significantly modified version of the heuristic (CPC) presented in (Franco *et al.* 2017).

The Complementary1 planner uses pattern databases (PDBs). A PDB is a heuristic function in the form of a lookup table that contains optimal solution costs of a simplified version of the task. In this planner we use a method that dynamically creates multiple PDBs which are later combined into a single heuristic function. At a given iteration, our method uses estimates of the search space size to create a PDB that complements the strengths of the PDBs created in previous iterations.

The biggest difference with (Franco *et al.* 2017) is that the original method always started with smaller PDBs and used *a priori* time limits to sequentially increase the PDB's size limit while the new method has no such schedule or initial bias. Complementary1 uses the UCB1 bandit algorithm to learn which PDB size bracket fits best the current problem given the previously selected PDBs. We have also added two new seeding algorithms, based on bin packing, and also added a new pattern generation algorithm based on Gamer. Finally, the code itself has been refactorized to ease the addition of evaluation methods, generation methods and other alternative configurations.

Introduction

Excerpt from (Franco *et al.* 2017)

Pattern databases (PDBs) map the state space of a classical planning task onto a smaller abstract state space by considering only a subset of the task's variables, which is called a pattern (Culberson and Schaeffer 1998; Edelkamp 2001). The optimal distance from every abstract state to an abstract goal state is precomputed and stored in a lookup table. The values in the table are used as a heuristic function to guide search algorithms such as A* (Hart *et al.* 1968) while solving planning tasks. Since a PDB heuristic is uniquely defined from a pattern, we also use the word pattern to refer to a PDB. The combination of several PDBs can result in better cost-to-go estimates than the estimates provided by each PDB alone. One might combine multiple PDBs by taking the maximum (Holte *et al.* 2006; Barley *et al.* 2014) or the sum (Felner *et al.* 2004) of the PDBs' estimates. In this paper we consider the canonical heuristic function, which takes the maximum estimate over all additive PDB subsets (Haslum *et al.* 2007). The challenge is then to find a collection of patterns from which an effective heuristic is derived.

Multiple approaches have been suggested to select good pattern collections (Haslum *et al.* 2007; Edelkamp 2006; Kissmann and Edelkamp 2011). Recent work showed that using a genetic algorithm (Edelkamp 2006) to generate a large collection of PDBs and greedily selecting a subset of them can be effective in practice (Lelis *et al.* 2016). However, while generating a PDB heuristic, Lelis *et al.*'s approach is blind to the fact that other PDBs will be considered in the selection process. Our proposed method, which we call Complementary PDBs Creation (CPC), adjusts its PDB generation process to account for the PDBs already generated as well as for other heuristics optionally provided as input.

CPC sequentially creates a set of pattern collections $\mathcal{P}_{sel}$ for a given planning task ∇. Regular CPC starts with an empty $\mathcal{P}_{sel}$ set and iteratively adds a pattern collection $\mathcal{P}$ to $\mathcal{P}_{sel}$ if it predicts that $\mathcal{P}$ will be *complementary* to $\mathcal{P}_{sel}$. We say that $\mathcal{P}$ complements $\mathcal{P}_{sel}$ if A* using a heuristic built from $\mathcal{P} \cup \mathcal{P}_{sel}$ explores a smaller search space than when using a heuristic built from $\mathcal{P}_{sel}$. CPC uses estimates of A*'s search space to guide a local search in the space of pattern collections[1]. After $\mathcal{P}_{sel}$ has been constructed, all the corresponding PDBs are combined with the canonical heuristic function (Haslum *et al.* 2007).

Comments

We evaluated our pattern selection scheme in different settings in (Franco *et al.* 2017) , including explicit and symbolic PDBs. Our results showed that combining symbolic PDB heuristics were able to outperform existing methods. Futhermore, it also showed that CPC could create complementary PDBs to other methods. Our best combination was

[1]The (Franco *et al.* 2017) CPC version explores both size and time as metrics to decide whether to select a pattern collection. For the time predictions we used SS planner we have re-factorized the code, but did not finish the time selection in time. We have instead used the simpler to code random walk method to evaluate patterns. Note that for symbolic PDBs, both time and size selection methods had very similar performance.

using our method to complement a symbolic perimeter PDB. The selected method to be complemented for this competition first generates a symbolic PDB up to a time limit of 250 seconds, a memory limit of 4GBs[2]. One advantage of seeding our algorithm with such a perimeter search is that if there is an easy solution to be found in what is basically a brute force backwards search, we are finished before we even start finding complementary PDBs. If a PDB contains all available variables, any optimal solution for such abstraction is also necessarily an optimal solution in the real search space. In such cases we stop building the perimeter and simply return the optimal plan found.

For this planner, we have added two new seeding methods besides the perimeter PDB we collectively refer to as *bin-packing*. The first one uses First Fit Increasing to try to find the smallest collection of PDB using the bin packing principle. The second method uses First Fit Decreasing to do the same. Bin packing for PDBs tries to create the smallest number of PDBs which uses all available variables. While reducing the number of PDBs used to group all possible variables does not guarantee a better PDB, the less number of collections, the less likely on average to miss interactions between variables due to being placed on different PDBs[3]. PDB selection methods tend to suffer from diminishing returns, i.e.the more time invested using a pattern generation method, the less likely it is to find a new improving one. Using different PDB generation methods or varying their parameters, e.g. PDB size limits, is how we try to ameliorate diminishing returns.

If no solution is found after the perimeter PDB is finished, our method will start generating pattern collections stochastically until either the generation time limit (900 secs) or the overall PDB memory limit (8 GBs) are reached. CPC decides whether to add a pattern collection to the list of selected patterns if it is estimated that adding such PDB will speed up search. We used the stratified selection time prediction method described in (Franco *et al.* 2017) to estimate this. Note that when a pattern collection is added, all its patterns are collected using the canonical combination method in Fast Downward (from now on referred to as FD as it was in the 2017 version we forked our code from).

(Franco *et al.* 2017) compared the pattern selection methods to the Gamer algorithm (Kissmann and Edelkamp 2011). Gamer is based on the idea of trying to discover the single best possible PDB for a problem. Its pattern selection method can be summarized as an iterative process, starting with all the goal variables in one pattern, where the casually connected variables who would increase the most the average h value of the associated PDB are added to the Gamer pattern. We have created a new "Gamer-style" pattern generation method which behaves similarly, more details on the next section. This pattern selection method is intended to be complementary to the ones in the original CPC methods.

Once all patterns have been selected, the resulting canonical PDB combination is used as an admissible heuristic to do A* search for the sequential optimal track. We also added a cost-bounded option, where we used a slightly modified version of lazy greedy search as coded in FD. The modification is that instead of pruning all generated successor nodes whose g value is above the bounded cost, we actually prune all nodes whose g+h values are above the bounded cost. This is only guaranteed to keep solution cost at or below the bounded cost if the heuristic is admissible. Since this is the case for our heuristic, there is no reason to take advantage of this. Note that this track is an experimental version for us.

We decided not to submit this planner for the Satisficing track due to the inherent incompatibility of our heuristic and the track. Generating large symbolic PDBs cost a significant amount of time. Finding which patterns make good pattern collections is even more costly. In satisficing, the critical factor is finding a solution as quickly as possible, and hence it is generally better when using heuristics to use those which do not incur in large preprocessing costs.

List of Changes and Configuration Choices

- Moved to 64 bits build, due to the increase of memory limit on the IPC to 8 GBs. It required doubling the relevant PDB and overall memory limits.
- Using only symbolic PDBs.
- After the initial Perimeter search is finished, we run two different bin-packing algorithms in order to generate optimized SAS+ variable distribution to generate the PDBs.
  - First Fit Decreasing with a time limit of 50 seconds as recommended by the Authors. This algorithm distributes the variables in different bins according to their size in bits. The variables are initially shorted by their size. Then the smaller variables are grouped in the first bins, while the bigger are grouped in the last ones and sometimes on their own.
  - First Fit Increasing with a time limit of 75 seconds. This algorithm distributes the variables in different bins according to their size as the previous one but in this case the bigger variables are grouped in the first bins while the smaller are grouped in the last ones. The authors empirical tests have showed First Fit Increasing to do better on average when compared to the Decreasing version of the algorithm.
- Dropped (Franco *et al.* 2017) Stratified Sampling (*SS*), we are still finishing porting the original SS code. Note that for symbolic PDBs performance was quite similar. This is explained in (Franco *et al.* 2017), to summarize on average when adding a symbolic PDB[4] which reduces the size of the search space it also tends to reduce the overall

[2]A maximum amount of BDD nodes in the perimeter frontier of 10 million was also used. This was used as a failsafe on the actual implementation, otherwise the code occasionally would get stuck while generating the next step for the BDD generation.

[3]The packing algorithm used here ensures that each PDB has a least one goal variable and also that all variables in a PDB are casually connected, on their own or through a chain of local variables, to at least one goal variable in the PDB.

[4]Because on average symbolic PDBs enables us to cover much larger search spaces, hence reducing the benefit of using multiple complementary smaller PDBs vs a few larger ones.

run time, hence making both evaluation methods almost equivalent performance wise.

- All PDB size limits (from a minimum of 10^8 to a maximum of 10^{20}) are equally likely to be chosen. We use the UCB1 algorithm to learn *in situ* which PDB size limits are likeliest to do better. Note that the UCB1 will change the recommended PDB size limits if diminishing returns become a significant problem for a specific PDB size bracket. On the original version, fixed time limits were given to increase the PDB size limit by an order of magnitude, potentially forcing the heuristic to keep trying a size limit not justified by the current problem data.
- New Gamer-inspired generation method. Our modified CPC algorithm decides on each iteration which pattern generation method to use. We use UCB1 to learn whether to use the CBP (Franco *et al.* 2017) generation method or the Gamer-inspired method. Note that the Gamer algorithm has a termination condition if no variable can be added to improve the average heuristic value of the selected pattern. In our case, we stop calling the Gamer generation method if we have also discovered that no variable can sufficiently increase the average heuristic value given the current time and size limits.
- Hybrid evaluation methods: Our other pattern generation methods start from scratch, however for the Gamer style pattern selection method, the choice is always whether to add variables to the previously selected pattern. For the Gamer-inspired pattern generation method, we use the average heuristic values to decide whether the next iteration is improving the pattern. If no variable can be added which sufficiently increases the average heuristic value of the Gamer style pattern, this method is dropped from the available pattern selection methods UCB1 can select from. However, in terms of comparing the Gamer style pattern with the already selected patterns by CPC we still use the *in situ* probing mechanism based on problem data, in this case whether the size of the search space is predicted to be reduced by adding the new Gamer style pattern.
- UCB1 is also used to decide how many goal variables are present in a single pattern, the original CBP method was seeded by just one goal variable per pattern. We noticed that one of the reasons Gamer does so well for some problems is that it starts with all the goal variables. For some problems, missing even one goal variable in each pattern when using CBP results in much lower accuracy. We use UCB1 to learn if this is the case in the current problem. As an added bonus, it increases the diversity of PDB generation methods we use and hence hopefully ameliorate diminishing returns.

Results

Following is an ablation-type study were we analyze which components worked best (Table 1). We used the New Zealand Nesi Cluster. Domain names have been abbreviated to either the first 3 letters or the first letter of each word for spaces saving purposes.

Complementary1, the newest implementation, solved the same number of overall problems as Complementary2, the same implementation as in (Franco *et al.* 2017) with some bug fixes, however their ablation studies tell a rather different study, A full analysis goes beyond the scope of this extended abstract, instead we are going to provide our first impressions.

Table 1 shows the overall results for our competition submission (Comp1/Reg), with and without the initial perimeter PDB (Comp1/NoPer), with and without the seeding bin packing generator and finally for each of the individual packing method on their own (CBP, Gamer).

The biggest difference with Complementary2 is that dropping the initial perimeter PDB would have increased the number of overall solved problems by 2 [5]. The initial perimeter PDB was very helpful for (Franco *et al.* 2017) and it was also helpful for the same implementation (Complementary2 planner) for the IPC2018 problems. Dropping the perimeter PDBs resulted in solving at least 10 problems less when using any combination of the pattern generators used in Complementary2. It seems that using UCB1 to learn which PDB sizes fit best the problem has reduced our dependency on the perimeter PDB to obtain best results. For example, Complementary2/CBP solved 106 problems while Complementary1/CBPNoBP solved 120. Both used the same pattern generation method(albeit with the option to choose more starting goal variables in the latest version of CBP). Neither used a perimeter PDB. For detailed Complementary2 results, see the Complementary2 extended abstract.

It was also surprising to us that the Gamer module, seeded by BinPack (Comp1/Gamer+BinPack), solved 4 more problems than if we use our selection mechanism (Comp1/NoPer+BinPack) and would have actually won the competition. This would indicate that for some problems the best option was to keep growing the Gamer-style pattern but instead we selected CBP (or run out of time before we could grow the Gamer-style PDB to the same size). Finding out which is future work.

Finally, we also included the best possible results if we knew which is the best pattern generator method *a priori* for free. 139 problems are actually solvable using the right combination of generation methods, 15 more than when using our selection mechanism. Of course this comparison is biased,i.e. when running our selection mechanism the whole available time has to be split between the preferred pattern generators, while when picking the best out of each methods, each of them had the whole 900 seconds generation time. This means the number of patterns tested is much larger when giving each method 900 seconds. What it does show is the potential of symbolic PDBs in planning.

Concluding Remarks

The competition results were quite good for optimal planning, where we were the runners up (winner was 126 problems while we solved 124). We were also the best

[5]and hence would have draw with the competition winner in terms of problems solved

Table 1: Coverage of Complementary1 Modules. Reg stands for all components active. ”NoPer” stands for perimeter PDB inactivated. ”+BinPack” stands for using PDBs generated by bin packing generator. ”CBP”, ”Cgamer” and ”BinPackOnly” rows also have Perimeter inactive.

Domain/Method	Agr	Cal	DN	Nur	OSS	PNA	Set	Sna	Spi	Ter	Total
Comp1/Reg	10	11	13	12	**12**	**19**	**9**	10	**12**	**16**	124
Comp1/NoPer+BinPack	10	**12**	**14**	**14**	**12**	6	**9**	12	11	**16**	126
Comp1/NoBinPack	6	11	13	**14**	**12**	**19**	**9**	11	11	**16**	122
Comp1/CBP+BinPack	8	**12**	**14**	13	**12**	**18**	**9**	11	11	**16**	124
Comp1/CBP-NoBinPack	6	**12**	**14**	13	**12**	18	**9**	9	11	**16**	120
Comp1/Gamer+BinPack	**13**	**12**	**14**	**14**	**12**	17	**9**	12	11	**16**	**130**
Comp1/Gamer-BinPack	8	12	12	**16**	**12**	18	**9**	**14**	11	**16**	128
Comp/BinPackOnly	7	**12**	14	12	**12**	7	**9**	11	11	12	107
Solved by any of the Comp1 methods above											
*	14	12	14	16	12	20	9	14	12	16	139
Competition result below included for Completeness											
Comp1	10	11	14	**13**	13	17	8	11	**11**	**16**	124

non-portfolio approach. However, a Gamer-style approach would have resulted in the best overall results (and easily won the competition). This is not that surprising when looking at (Franco *et al.* 2017) where even though our selector method (without gamer generators) did better overall, it was the second best method. This confirms that which Pattern Generator method is better is very much a question of which domain is it to be used for.

Interestingly, when running each pattern generation method on its own, 15 more problems are solvable. The question is if this was because there was more time allocated to generate patterns when running each method on their own, or if the selection mechanism has significant scope for improvement. This is future research.

Using the 2 bin-packing generators to seed the heuristic proved quite useful. It improved our overall results for all the methods we tested. The implementation we did of the Bin Pack generator was very last minute and did not include any stochastic method. Generally, the more diverse the pattern generators used, the more likely it is for one of them to find good patterns. The preliminary results here seem to indicate that using bin packing techniques as pattern generators is quite promising as a complement to CBP (or RBP in Complementary2) and Gamer.

Finally, it seems that Complementary1 using the UCB1 bandit algorithm to learn which PDB size bracket fits best the current problem, given the previously selected PDBs, has resulted in lowering our dependency on the perimeter PDB to obtain best results. While on the old implementation (Complementary2) dropping the perimeter would result in solving 10 fewer problems, Complementary1 actually solves more problems without the perimeter PDB, regardless of the combination of pattern generators used. A more extensive and detailed analysis is required to confirm this first impression.

Acknowledgments

Thanks to the Fast Downward (and lab testing tool) developers for sharing their code. It is easy to take it for granted, since both have been available for so long, but the constant maintenance and support work is very much appreciated. The complementary1 Planner was build on top of an early 2017 FD fork.

References

Michael W. Barley, Santiago Franco, and Patricia J. Riddle. Overcoming the utility problem in heuristic generation: Why time matters. In *Proc. ICAPS*, 2014.

Joseph C. Culberson and Jonathan Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.

Stefan Edelkamp. Planning with pattern databases. In *Proc. ECP*, pages 13–24, 2001.

Stefan Edelkamp. Automated creation of pattern database search heuristics. In *Proc. MOCHART*, pages 35–50, 2006.

Ariel Felner, Richard E. Korf, and Sarit Hanan. Additive pattern database heuristics. *Journal of Artificial Intelligence Research*, 22:279–318, 2004.

Santiago Franco, Álvaro Torralba, Levi H. S. Lelis, and Mike Barley. On creating complementary pattern databases. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 4302–4309, 2017.

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

Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonet, and Sven Koenig. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proc. AAAI*, pages 1007–1012, 2007.

R. C. Holte, A. Felner, J. Newton, R. Meshulam, and D. Furcy. Maximizing over multiple pattern databases speeds up heuristic search. *Artificial Intelligence*, 170(16–17):1123–1136, 2006.

Peter Kissmann and Stefan Edelkamp. Improving cost-optimal domain-independent symbolic planning. In *Proc. AAAI*, pages 992–997, 2011.

Levi H. S. Lelis, Santiago Franco, Marvin Abisrror, Mike Barley, Sandra Zilles, and Robert C. Holte. Heuristic subset selection in classical planning. In *Proc. IJCAI*, pages 3185–3191, 2016.

The Complementary2 Planner in the IPC 2018

Santiago Franco[1*], **Levi H. S. Lelis**[2], **Mike Barley**[3]
[1] School of Computing and Engineering, University of Huddersfield, UK
[2] Departamento de Informática, Universidade Federal de Viçosa, Brazil
[3] Computer Science Department, Auckland University, New Zealand
s.franco@hud.ac.uk, levi.lelis@ufv.br, barley@cs.auckland.ac.nz

Abstract

This planner is an implementation of the heuristic (CPC) presented in (Franco *et al.* 2017), the only updates are a few bug fixes. This paper contains a brief summary of that work with a slant on which exact configuration was used and why. **Please quote (Franco *et al.* 2017) as well when discussing this planner**.

A pattern database (PDB) for a planning task is a heuristic function in the form of a lookup table that contains optimal solution costs of a simplified version of the task. In this planner we use a method that sequentially creates multiple PDBs which are later combined into a single heuristic function. At a given iteration, our method uses estimates of the A* running time to create a PDB that complements the strengths of the PDBs created in previous iterations. We used symbolic PDBs because the current implementation supports conditional effects, a requirement in the IPC18. Additionally, in our benchmark tests, this was the best option even without conditional effects.

Introduction

This paper contains excerpts from (Franco *et al.* 2017) because this planner is an exact implementation of the CPC heuristic, specifically the CPC-S-P configuration. Other parts of the original paper have been summarized. Comments have been added to reflect the reasoning behind some of our choices. But first, we will give some context information for those not familiar with PDBs.

Excerpt from Original Paper

Pattern databases (PDBs) map the state space of a classical planning task onto a smaller abstract state space by considering only a subset of the task's variables, which is called a pattern (Culberson and Schaeffer 1998; Edelkamp 2001). The optimal distance from every abstract state to an abstract goal state is precomputed and stored in a lookup table. The values in the table are used as a heuristic function to guide search algorithms such as A* (Hart *et al.* 1968) while solving planning tasks. Since a PDB heuristic is uniquely defined from a pattern, we also use the word pattern to refer to a PDB. The combination of several PDBs can result in better cost-to-go estimates than the estimates provided by each PDB alone. One might combine multiple PDBs by taking the maximum (Holte *et al.* 2006; Barley *et al.* 2014) or the sum (Felner *et al.* 2004) of the PDBs' estimates. In this paper we consider the canonical heuristic function, which takes the maximum estimate over all additive PDB subsets (Haslum *et al.* 2007). The challenge is then to find a collection of patterns from which an effective heuristic is derived.

Multiple approaches have been suggested to select good pattern collections (Haslum *et al.* 2007; Edelkamp 2006; Kissmann and Edelkamp 2011). Recent work showed that using a genetic algorithm (Edelkamp 2006) to generate a large collection of PDBs and greedily selecting a subset of them can be effective in practice (Lelis *et al.* 2016). However, while generating a PDB heuristic, Lelis *et al.*'s approach is blind to the fact that other PDBs will be considered in the selection process. Our proposed method, which we call Complementary PDBs Creation (CPC), adjusts its PDB generation process to account for the PDBs already generated as well as for other heuristics optionally provided as input.

CPC sequentially creates a set of pattern collections $\mathcal{P}_{sel}$ for a given planning task ∇. CPC starts with an empty $\mathcal{P}_{sel}$ set and iteratively adds a pattern collection $\mathcal{P}$ to $\mathcal{P}_{sel}$ if it predicts that $\mathcal{P}$ will be *complementary* to $\mathcal{P}_{sel}$. We say that $\mathcal{P}$ complements $\mathcal{P}_{sel}$ if A* using a heuristic built from $\mathcal{P} \cup \mathcal{P}_{sel}$ solves ∇ quicker than when using a heuristic built from $\mathcal{P}_{sel}$. CPC uses estimates of A*'s running time to guide a local search in the space of pattern collections. After $\mathcal{P}_{sel}$ has been constructed, all the corresponding PDBs are combined with the canonical heuristic function (Haslum *et al.* 2007).

IPC2018 Choices

We evaluated our pattern selection scheme in different settings in (Franco *et al.* 2017) , including explicit and symbolic PDBs. Our results showed that combining symbolic PDB heuristics were able to outperform existing methods. Furthermore, it also showed that CPC could create complementary PDBs to other methods. Our best combination was using our method to complement a symbolic perimeter PDB.

[*]This work was carried out while S. Franco was a postdoctoral fellow at Universidade Federal de Viçosa, Brazil.

The selected method to be complemented for this competition first generates a symbolic PDB up to a time limit of 250 seconds, a memory limit of 4GBs[1]. One advantage of starting our algorithm with such a perimeter search is that if there is an easy solution to be found in what is basically a brute force backwards search, we are finished before we even start finding complementary PDBs. If a PDB contains all available variables, any optimal solution for such abstraction is also necessarily an optimal solution in the real search space. In such cases we stop building the perimeter and simply return the optimal plan found.

If no solution is found after the perimeter PDB is finished, our method will start generating pattern collections stochastically until either the generation time limit (900 secs) or the overall PDB memory limit (4 GBs) is reached. CPC decides whether to add a pattern collection to the list of selected patterns if it is estimated that adding such PDB will speed up search. We used the stratified selection time prediction method described in the original paper to estimate this. Note that when a pattern collection is added, all its patterns are collected using the canonical combination method in Fast Downward (from now on referred to as FD as it was in the 2017 version we forked our code from).

Once all patterns have been selected, the corresponding canonical PDB combination is used as an admissible heuristic to do A* search for the sequential optimal track. We also added a cost-bounded option, where we used a slightly modified version of lazy greedy search as coded in FD. The modification is that instead of pruning all generated successor nodes whose g (current path cost) value is above the bounded cost, we actually prune all nodes whose $g + h$ (current path cost + estimated distance to goal) values are above the bounded cost. This is only guaranteed to keep solution cost at or below the bounded cost if the heuristic is admissible. Since this is the case for our heuristic, we take advantage of the improved pruning capability. Note that this track is an experimental version for us, I personally have very little experience in cost-bounded search and make no claim this is the most efficient search method. We thought it would be nice to try the CPC heuristic in this setting as well.

We decided not to submit this planner for the Satisficing track due to the inherent incompatibility of our heuristic with respect to this track. Generating large symbolic PDBs cost a significant amount of time. Finding which patterns make good pattern collections is even more costly because most of the PDBs generated are never used for the actual search. In Satisficing, the critical factor is finding a solution as quickly as possible, and hence it is generally better when using heuristics to pick those which do not incur in large preprocessing costs.

Problem Definition

This section is identical to the original, included for completeness.

[1]A maximum amount of BDD nodes in the perimeter frontier of 10 million was also used. This was used as a failsafe on the actual implementation, otherwise the code occasionally would get stuck while generating the next step for the BDD generation.

We are interested in finding a set of pattern collections $\mathcal{P}_{sel}$ that minimizes the running time of A* using the heuristic function obtained from $\mathcal{P}_{sel}$, denoted $h_{\mathcal{P}_{sel}}$. We approximate the running time of A* guided by $h_{\mathcal{P}_{sel}}$ while solving a task ∇, denoted $\mathcal{T}(\mathcal{P}_{sel}, \nabla)$, as introduced by Lelis *et al.* (2016).

$$\mathcal{T}(\mathcal{P}_{sel}, \nabla) = J(\mathcal{P}_{sel}, \nabla) \times (t_{h_{\mathcal{P}_{sel}}} + t_{gen}) \, .$$

Here, $J(\mathcal{P}_{sel}, \nabla)$ is the number of nodes A* employing $h_{\mathcal{P}_{sel}}$ generates while solving ∇, $t_{h_{\mathcal{P}_{sel}}}$ is $h_{\mathcal{P}_{sel}}$'s average time for computing the heuristic value of a single node, and t_{gen} is the node generation time. Although the exact value of $\mathcal{T}(\mathcal{P}_{sel}, \nabla)$ is only known once A* finishes its search, one is able to compute an approximation, denoted $\hat{\mathcal{T}}(\mathcal{P}_{sel}, \nabla)$. The value of $\hat{\mathcal{T}}(\mathcal{P}_{sel}, \nabla)$ is computed by using approximations of $t_{h_{\mathcal{P}_{sel}}}$ and t_{gen}, which are obtained while computing an estimate for $J(\mathcal{P}_{sel}, \nabla)$, denoted $\hat{J}(\mathcal{P}_{sel}, \nabla)$. $\hat{J}(\mathcal{P}_{sel}, \nabla)$ is obtained by running Stratified Sampling (Chen 1992). We write $\hat{J}$ instead of $\hat{J}(\mathcal{P}_{sel}, \nabla)$ whenever $\mathcal{P}_{sel}$ and ∇ are clear from the context.

Stratified Sampling Evaluation

We used stratified sampling for our planner as described in the original paper. We briefly summarize it here, for a detailed discussion please see (Franco *et al.* 2017).

Stratified Sampling (SS) estimates numerical properties (e.g., tree size) of search trees by sampling. Lelis *et al.* (2014) showed that SS is unable to detect duplicates in the search tree in its sampling procedure. Instead, we use SS to estimate the size of the search tree $S(\mathcal{I}, b)$, for some value b, and use this estimate as an approximation $\hat{J}$ for the nodes generated by A*. SS uses a stratification of the nodes in the search tree rooted at $\mathcal{I}$ through a *type system* to guide its sampling.

The type system we use accounts for a heuristic h as follows. Two nodes n_1 and n_2 in $S(\mathcal{I}, b)$ have the same type if $f(n_1) = f(n_2)$ and if n_1 and n_2 occur at the same level of S. SS samples S and returns a set A of *representative-weight* pairs, with one such pair for every unique type seen during sampling. In the pair $\langle n, w \rangle$ in A for type $t \in T$, n is the unique node of type t that was expanded during search and w is an estimate of the number of nodes of type t in S. Since SS is non-deterministic, every run of the algorithm can generate a different set A. We call each run of SS a probe. We refer the reader to SS's original paper (Chen 1992) for details.

In our pattern selection algorithm we run multiple SS probes to generate a collection of vectors $C = \{A_1, A_2, \cdots, A_m\}$. A vector A^U is created from C by combining all representative-weight pairs in C. For each unique type t encountered in C we add to A^U a representative pair $\langle n, \bar{w} \rangle$ where n is selected at random from all nodes in C of type t, and $\bar{w}$ is the average w-value of all nodes in C of type t. Each entry in A^U represents SS's prediction for the number of nodes of a given type in the search tree.

We run SS with a time limit of 20 seconds and a space limit of 20,000 entries in the A^U structure. SS performs

1,000 probes with $b = h(\mathcal{I})$, where h is CPC's current heuristic function. If SS completes all 1,000 probes without violating the time and space limits, we increase b by 20% and run another 1,000 probes. The process is repeated until reaching either the time or the space limits. The A^U structure is built from the A vectors collected in all probes.

Since our pattern selection approach needs to test multiple heuristics, we run SS once using a type system T defined by CPC's current heuristic and store A^U in memory. Then, $\hat{J}$ is computed for a newly created heuristic h' by iterating over all representative node-weights $\langle n, \bar{w} \rangle$ in A^U and summing the $\bar{w}$-values for which $h'(n) + g(n) \leq b$, where b is the largest value used for probing with SS while building the A^U structure; this sum is our $\hat{J}$ for h'.

Adaptable Pattern Collection Generation

This section is a summary of the original papers, included for completeness.

Algorithm 1 is a high-level overview of the search CPC performs in the pattern collection space. CPC receives as input a planning task ∇, a base heuristic h_{base} (which could be the h_0 heuristic, *i.e.,* a heuristic that returns zero to all states in the state space), time and memory limits, t and m, respectively, that specify when to stop running CPC. CPC also receives another time limit, t_{stag}, for deciding when the parameters of CPC's search must be readjusted. S_{min} and S_{max} specify the minimum and maximum sizes of the PDBs constructed. We use zero-one cost partitioning on each pattern collection $\mathcal{P}$ so that its PDBs are additive. Once CPC returns a set of pattern collections $\mathcal{P}_{sel}$, we use the canonical heuristic function (Haslum *et al.* 2007) to combine all the patterns in $\mathcal{P}_{sel}$ into a heuristic function.

CPC creates pattern collections through calls of the function BINPACKINGUCB (see line 5), which we explain in Section . Once a pattern collection $\mathcal{P}$ is created, CPC evaluates its quality with SS (see line 8), which estimates the running time of A* using a heuristic composed of the patterns already selected by CPC, $\mathcal{P}_{sel}$, added to the new $\mathcal{P}$. If SS estimates that A* solves ∇ faster with a heuristic created from the set of pattern collections $\mathcal{P}_{sel} \cup \mathcal{P}$ than with a heuristic created from $\mathcal{P}_{sel}$, CPC adds $\mathcal{P}$ to $\mathcal{P}_{sel}$ (see line 9). Whenever CPC adds a pattern collection $\mathcal{P}$ to $\mathcal{P}_{sel}$, it performs a local search by applying a mutation operator to $\mathcal{P}$ (see line 7), trying to create other similar and helpful pattern collections (the mutation operator is explained in Section). If SS estimates that $\mathcal{P}$ does not help reducing A*'s running time, then CPC creates a new $\mathcal{P}$ through another BINPACKINGUCB function call in its next iteration.

The first time EVALUATE-SS is called, CPC runs SS using h_{base} as its type system to create a vector A^U that is used to produce estimates of the A* running time. Whenever a call to EVALUATE-SS returns true, meaning that $\mathcal{P}$ helps reducing A*'s running time, CPC discards A^U and runs SS again with the heuristic constructed from $\mathcal{P}_{sel} \cup \mathcal{P}$ as its type system to generate a new A^U. The intuition behind re-running SS whenever a complementary pattern collection is found is to allow SS to explore parts of the search tree that were not explored in previous runs. Initially, the heuristic used in SS's sampling tend to be weak, and many of the states in the A^U vector SS produces will not be expanded by A* after the new $\mathcal{P}$ is added to $\mathcal{P}_{sel}$. By running SS whenever a better heuristic is constructed, one allows SS to also prune such nodes and focus its sampling on nodes that the current heuristic is not able to prune.

Algorithm 1 Complementary PDBs Creation

```
Require: Planning task $\nabla$, base heuristic $h_{base}$, time and
    memory limits $t$ and $m$ respectively, stagnation time
    $t_{stag}$, minimum/maximum PDB size $S_{min}$, $S_{max}$.
Ensure: Selected set of pattern collections $\mathcal{P}_{sel}$
 1: $\mathcal{P}_{sel} \leftarrow \emptyset$ // $\mathcal{P}_{sel}$ is a set of pattern collections
 2: $\mathcal{P} \leftarrow \emptyset$ // $\mathcal{P}$ is a pattern collection
 3: while time $t$ or memory $m$ limits are not exceeded do
 4:     if $\mathcal{P} = \emptyset$ then
 5:         $\mathcal{P} \leftarrow$ BINPACKINGUCB$(\nabla, S_{min}, S_{max})$
 6:     else
 7:         $\mathcal{P} \leftarrow$ MUTATION$(\mathcal{P})$
 8:     if EVALUATE-SS$(\mathcal{P}_{sel} \cup \mathcal{P})$ then
 9:         $\mathcal{P}_{sel} \leftarrow \mathcal{P}_{sel} \cup \mathcal{P}$
10:     else
11:         $\mathcal{P} \leftarrow \emptyset$
12:     if (time since a $\mathcal{P}$ is added to $\mathcal{P}_{sel}$) $> T_{stag}$ then
13:         adjust $S_{min}, S_{max}$
14: return $\mathcal{P}_{sel}$
```

Bin-Packing Algorithms

In this section we describe the methods we consider for generating candidate pattern collections.

Regular Bin-Packing (RBP) We adapt the genetic algorithm method introduced by Edelkamp (2006) for selecting a collection of patterns. Edelkamp's method, which we call Regular Bin-Packing (RBP), generates an initial pattern collection $\mathcal{P}$ as follows. RBP iteratively selects a unique and random variable v from $\mathcal{V}$ and adds it to a subset B of variables, called "bin", that is initially empty. Once a PDB constructed from the subset of variables in B exceeds a size limit M, RBP starts adding the randomly selected variables to another bin. This process continues until all variables from $\mathcal{V}$ have been added to a bin. Note that since RBP selects unique variables, the bins represent a collection of disjoint patterns.

Once the pattern collection $\mathcal{P}$ is generated, RBP iterates through each pattern p in $\mathcal{P}$ and removes from p any variable not causally related to other variables in p (Helmert 2004).

Causal Bin-Packing (CBP) Our CBP approach differs from RBP only in the way it selects the variables to be added to the bins. Instead of choosing them randomly as is done in RBP, CBP selects only the first variable of each bin randomly and then only adds to a bin B variables which are causally related to the variables already in B. In case there are multiple causally related variables to be added, CBP chooses one at random.

We observed empirically in (Franco *et al.* 2017) that RBP tends to generate pattern collections that result in PDBs of similar sizes, and that CBP tends to generate pattern collections that result in PDBs of various sizes. This is because RBP removes causally unrelated variables after the variable selection is done. By contrast, CBP greedily selects causally related variables as the patterns are created. As a result, usually the first pattern created by CBP will have more variables than all the other patterns created.

Combination of Bin-Packing Approaches with UCB1 UCB1 is a version of UCB whose regret grows logarithmically as a function of the number of actions take. We used this algorithm to choose *in situ* how frequently to use either of both pattern generation algorithms.

We used the UCB1 formula (Auer 2002), $\bar{x}_j + \sqrt{\frac{2 \ln n}{n_j}}$, to decide which arm (algorithm) to use next. Here, $\bar{x}_j$ is the average reward received by algorithm j, n is the total number of trials made (*i.e.,* calls to a bin-packing algorithm), and n_j is the number of times algorithm j was called. We artificially initialize $\bar{x}_j$ to 10 for all j to ensure that all algorithms are tested a few times before UCB1 can express a strong commitment to a particular option. This helps to reduce the chances of UCB1's selection being unduly influenced by the stochastic nature of the bin-packing approaches. A bin-packing algorithm receives a reward of +1 if it provides a $\mathcal{P}$ that is able to reduce the $\hat{\mathcal{T}}$-value as estimated by SS; the reward is 0 otherwise.

In (Franco *et al.* 2017) we performed a systematic experiment on the optimal STRIPS benchmark suite distributed with the FD (Helmert 2006). The coverage results for the two approaches showed using UCB1 to combine both approaches was significantly better than using either one or simply choosing them with equal probability. See the original paper for a more detailed discussion.

Mutation Operator

CPC performs mutations on a given pattern collection $\mathcal{P}$ whenever $\mathcal{P}$ is deemed as promising by SS. That is, if SS estimates that $\mathcal{P}$ will not reduce the A* running time, CPC sets $\mathcal{P}$ to $\emptyset$, and in the next iteration of CPC's while loop another $\mathcal{P}$ is created with our UCB approach. On the other hand, if SS predicts that $\mathcal{P}$ is able to reduce A*'s running time, then CPC adds $\mathcal{P}$ to $\mathcal{P}_{sel}$ and, in the next iteration of its while loop, it applies a mutation operator to $\mathcal{P}$, trying to create another pattern collection that might further reduce A*'s running time. More details in the original paper.

Dynamic Parameter Adjustment

Some of the instances benefit from a large number of small PDBs, while others require a small number of large PDBs. Thus, instead of fixing the PDB size throughout CPC's pattern selection search, we adjust the size of the PDBs, M, to be constructed during search.

To be specific, if after t_{stag} seconds we are unable to add a new complementary pattern collection to $\mathcal{P}_{sel}$, we increase the size M of the PDBs we generate. The intuition is that if our search procedure does not find complementary patterns for the current PDB size, M, then we assume that this particular planning problem might benefit from larger PDBs.

In the original paper, it was shown that a dynamic range of PDB sizes worked better for our benchmark tests compared to using any of multiple *a priori* fixed sizes.

Empirically-based Choices

1. We used CPC-S-P configuration from the original paper, because it had the overall best results.
2. We only used symbolic PDBs. (Franco *et al.* 2017) because explicit PDBs did not support conditional effects, while symbolic PDBs (as implemented) do. (Franco *et al.* 2017) did not include any domains with conditional effects. Secondly, symbolic PDBs performed significantly better overall for the paper's experiments.[2]
3. We switched to a 64 bits build. After adjusting the size and the maximum number of nodes on the frontier for symbolic PDBs, it was found that more problems were solved, when using the IPC 2018 limits. Both limits were doubled. Limiting the maximum number of nodes in the BDDs frontiers is an implementation failsafe to ensure the memory and time limits are respected.

Results

Following is an ablation-type study were we analyze which components worked best (Table 1). We used the New Zealand Nesi Cluster. Domain names have been abbreviated to either the first 3 letters or the first letter of each word for spaces saving purposes.

Table 1 shows that the combination of bin packed methods (CBP, RBP), aided by an initial perimeter PDB, and regulated by UCB1 (*Comp2/Reg*) was better than any of the individual methods on their own. This confirms our expectations and has a similar behaviour as in (Franco *et al.* 2017) which used all previous IPC domains (seq-opt). Dropping the initial perimeter PDB reduced the overall number of solved problems by 10. Interestingly there was only one domain were the perimeter PDB really helped albeit quite significantly, we would have solved approximately 11 less problems for the Petri Net Alignment (PNA) without the perimeter PDB. Otherwise, the impact of the Perimeter PDB is minimal. In general, the Perimeter PDB works well on domains where getting a good heuristic value is difficult or not that important, e.g. Openstacks. Additionally, if no perimeter is used and we only generate patterns using the RBP generator, 6 fewer problems were solved, the biggest effect is in Snake (Sna) where we solved 3 fewer problems. On the other hand, when only using CBP as a pattern generator, results in no problems lost for Snake but solving 4 fewer Agricola (Agr) and Termes (Ter) problems. However, from previous experiments do note that whether CBP, RBP or a combination of both is the best option is very much dependent on the domain. There is no obvious method to predict *a priori* which bin packing method is best for the current problem.

[2]Note that explicit PDBS can outperform symbolic PDBs on some domains. The reason is that while symbolic PDBs can deal with larger abstractions, they are also more expensive to evaluate. For those domains where it is easy to find a large amount of good quality complementary patterns, explicit can be the better option.

Table 1: Coverage of Complenmentary2 Modules. Reg stands for all components active. NoPer stands for perimeter PDB inactivated. RBP and CBP also have Perimeter inactive.

Domain	Agr	Cal	DN	Nur	OSS	PNA	Set	Sna	Spi	Ter	Total
Comp2/Reg	**6**	**12**	13	**12**	12	**19**	**9**	**14**	11	**16**	**124**
Comp2/NoPer	**6**	**12**	**14**	**12**	12	8	**9**	**14**	11	**16**	114
Comp2/RBP	5	**12**	13	**12**	12	7	**9**	11	**13**	14	108
Comp2/CBP	2	**12**	13	**12**	12	8	**9**	**14**	12	12	106
Competition result bellow included for Completeness											
Comp2	**6**	**12**	12	**12**	**13**	18	9	**14**	12	**16**	**124**

Concluding Remarks

The competition results were quite good for optimal planning, where we were the runners up (winner was 126 problems while we solved 124). We were also the best non-portfolio approach.

Future research avenues, as mentioned in (Franco *et al.* 2017), are using more (or improved) pattern generator methods, keep working on improving *in situ* selection of pattern generators, and analyzing the impact of using online PDBs for selection purposes.

Acknowledgments

S. Franco and L. Lelis were supported by Brazil's CAPES (Science Without Borders) and FAPEMIG. M. Barley was supported by the Air Force Office of Scientific Research, Asian Office of Aerospace Research and Development (AOARD) under award number FA2386-15-1-4069. Thanks to Dr. Pat Riddle for her editorial support. Thanks to the Fast Downward (and lab testing tool) developers for sharing their code. It is easy to take it for granted, since both have been available for so long, but the constant maintenance and support work is very much appreciated. The complementary1 Planner was build on top of an early 2017 FD fork.

References

P. Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3:397–422, 2002.

Michael W. Barley, Santiago Franco, and Patricia J. Riddle. Overcoming the utility problem in heuristic generation: Why time matters. In *Proc. ICAPS*, 2014.

P.-C. Chen. Heuristic sampling: A method for predicting the performance of tree searching programs. *SIAM Journal on Computing*, 21:295–315, 1992.

Joseph C. Culberson and Jonathan Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.

Stefan Edelkamp. Planning with pattern databases. In *Proc. ECP*, pages 13–24, 2001.

Stefan Edelkamp. Automated creation of pattern database search heuristics. In *Proc. MOCHART*, pages 35–50, 2006.

Ariel Felner, Richard E. Korf, and Sarit Hanan. Additive pattern database heuristics. *Journal of Artificial Intelligence Research*, 22:279–318, 2004.

Santiago Franco, Álvaro Torralba, Levi H. S. Lelis, and Mike Barley. On creating complementary pattern databases. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 4302–4309, 2017.

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

Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonet, and Sven Koenig. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proc. AAAI*, pages 1007–1012, 2007.

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

Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

R. C. Holte, A. Felner, J. Newton, R. Meshulam, and D. Furcy. Maximizing over multiple pattern databases speeds up heuristic search. *Artificial Intelligence*, 170(16–17):1123–1136, 2006.

Peter Kissmann and Stefan Edelkamp. Improving cost-optimal domain-independent symbolic planning. In *Proc. AAAI*, pages 992–997, 2011.

Levi H. S. Lelis, Roni Stern, and Nathan R. Sturtevant. Estimating search tree size with duplicate detection. In *Proc. SOCS*, pages 114–122, 2014.

Levi H. S. Lelis, Santiago Franco, Marvin Abisrror, Mike Barley, Sandra Zilles, and Robert C. Holte. Heuristic subset selection in classical planning. In *Proc. IJCAI*, pages 3185–3191, 2016.

The Meta-Search Planner (MSP) at IPC 2018

Raquel Fuentetaja[1], **Michael Barley**[2], **Daniel Borrajo**[1], **Jordan Douglas**[2],
Santiago Franco[3] and **Patricia Riddle**[2]
[1]Departamento de Informática. Universidad Carlos III de Madrid, Spain
[2]Department of Computer Science. University of Auckland, New Zealand
[3]School of Computing and Engineering, University of Huddersfield, United Kingdom

Abstract

This abstract describes the general behavior of MSP (Meta-Search Planner) for the IPC (International Planning Competition) 2018, optimal track. MSP is a meta-reasoning system that searches through the space of planners, representations and heuristics on a problem-by-problem basis. Given a planning problem, MSP perfoms two phases: meta-search phase and problem solving phase. The meta-search phase is a search process for selecting one combination suitable for optimally solving the specific problem. The solving problem phase is basically a call to the selected planner with the other elements of the selected combination.

Introduction

The Meta-Search Planner (MSP) presented at the 2018 IPC is based on an approach for optimal planning described previously in (Fuentetaja et al. 2018). Given a problem and domain, MSP searches through the space of representation changes and heuristics to find a good combination to use in solving the problem. The underlying ideas are the following: (1) it is seldom the case that a single representation or heuristic is best over all problems within a domain; (2) the choice of representations and heuristics can be modeled as a meta-level search through a space of combinations of representations and heuristics; and (3) the decision on which representation change and set of heuristics to use can be made on-line and on a problem by problem basis. The main differences between MSP and other methods such as portfolios (Cenamor, de la Rosa, and Fernández 2016; Helmert, Röger, and Karpas 2011; Gerevini, Saetti, and Vallati 2009; Núñez, Borrajo, and Linares-López 2015) are: (1) MSP does not perform a learning step; (2) porfolios do not usually focus on changing the representation of the input domain/problem; and (3) they frequently do not adapt the portfolio to the specific problem.

An important aspect of MSP is that it allows representation changes applied to the input representation. Representation changes have been much less studied in automated planning than other aspects such as heuristics and search algorithms. However, the same problem can be defined in different ways in PDDL (Planning Domain Description Language) (Ghallab et al. 1998), the standard language for compactly representing planning tasks. Each particular definition of a planning problem may have an impact on the planner performance. In fact, given the same combination of heuristics and search methods some representations will make it harder to solve the problem while others will facilitate its solution (Howe et al. 1999; Howe and Dahlman 2002; Riddle et al. 2016; Fuentetaja and de la Rosa 2016). Also, the *best* representation may not be the same for different problems within a domain.

Related also to the representation is the fact that most current planners transform the PDDL representation into more efficient internal representations such as propositional logic (Hoffmann and Nebel 2001) or SAS$^+$ (Bäckström and Nebel 1995). Thus, changes of representation can be performed either at the PDDL level, or in the procedure(s) involved in the transformation. For instance, in Fast Downward (FD) (Helmert 2006), one of the most influential planning platforms, these procedures are the translation and preprocessing steps to generate a SAS$^+$ representation. In SAT planning, the impact of different encodings has also been examined (Kautz and Selman 1992; Rintanen 2012).

In general, MSP can change the representation using various techniques as black-boxes. They range from simple ones like changing the order of actions in the PDDL domain file, à la (Vallati et al. 2015), to more complex ones like Baggy (Riddle et al. 2016), that reformulates problems into a bagged representation. It can also vary the procedure to obtain the internal representation.

MSP can use one or more underlying planners. The meta-search described in the original publication (Fuentetaja et al. 2018) uses the RIDA* planner (Barley, Franco, and Riddle 2014) for several purposes: evaluating meta-search states; selecting the set of appropiate heuristics for a meta-search state; and as the planner for solving the problem. The version presented at the IPC 2018 does not use RIDA*. Instead, it includes two planners: Optimal Fast Downward (FD) (Helmert 2006) and SYMBA* (Torralba, Linares-López, and Borrajo 2016). Thus, the evaluation of meta-search nodes and the selection of heuristics at the IPC 2018 is different from that described in the original article. Specifically, the evaluation of nodes is perfomed by a sampling process using the mentioned planners. The heuristics are included in the meta-search states, so that they are evaluated together with the other state features by an evaluation function that is also different. The next sections describe the MSP version for IPC 2018 in more detail.

Meta-search

We use the standard definition of a planning task as $\Pi = \{F, A, I, G\}$. F is a set of propositions, A is a set of actions, $I \subseteq F$ defines the initial state and $G \subseteq F$ defines the goals. A planner takes as input a planning task and returns a plan $\pi = \langle a_1, \ldots, a_n \rangle$ such that if applied to the initial state I, it will achieve the goals. That is, it will generate a state s_n after applying actions in π to I such that $G \subseteq s_n$. Actions have a cost, $c(a), \forall a \in A$. The cost of a plan is $c(\pi) = \sum_{a_i \in \pi} c(a_i)$. An optimal plan is one with minimum cost.

MSP can be described in terms of a generic search in a meta-search space followed by a call to a planner. It can be formally defined in terms of the meta-search state space ($\mathcal{MS}$), meta-search operators ($\mathcal{MO}$), and the problem solving method: the search algorithm and the heuristics.

We will denote as R_e the set of representation changes that generate a new PDDL representation, referred to as *external* representation changes. The changes that transform a PDDL representation into an internal one (such as SAS$^+$) are referred to as R_i, the *internal* representation changes. R_i is planner specific.

The input to MSP is a planning task described in terms of a domain D, and a problem P, both described in PDDL. Since MSP performs search in the space of representations, planners and heuristics, it also receives as inputs the set of external representation change operators that can be applied, R_e; the planners, Pl; the set of internal representation changes, R_i, and heuristics, $\mathcal{H}$, that every planner can use; and the time bound, T, represents the maximum time to solve the problem.

MSP is given the maximum time it can use to solve a problem. Within that time limit, it must use some of that time to select a good combination of representation changes, planners and heuristics, and use the remainder to apply that selection to solve the problem. This means that MSP must balance the benefits of spending more time in the meta-search against those of spending more time actually solving the problem. Picking the right balance is a difficult decision. Currently, MSP just splits the maximum time, T, in half. While the planner is guaranteed at least half of the total time, it will end up with more time whenever the selection process takes less than half. In other words, the planner will have a time limit of T - *consumedTime*, where *consumedTime* is the actual time consumed by the meta search.

Algorithm 1 shows a high level description of MSP that includes the call to meta-search and a call to the planner with the output of the meta-search (and the remaining time). The meta-search requires three time limits: one for the evaluation of states, another one for the meta-search and the other as an estimate of the time to be given later to the planner. The state evaluation time has been fixed to 75 seconds. For the other two, we assume $T/2$.

In the following, *end* refers to the best meta-search state found. The output of the meta-search contains: the final se-

Algorithm 1 MSP($D, P, R_e, Pl, R_i, \mathcal{H}, T_E, T$)

```
Inputs: domain $D$, problem $P$, PDDL rep. changes $R_e$, planners
    $Pl$, internal rep. changes $R_i$ and set of heuristics $\mathcal{H}$ for every
    planner, estimation time bound $T_E$, time bound $T$
Outputs: plan
1: $(D^{end}, P^{end}, pl^{end}, r_i^{end}, H^{end}, \pi) \leftarrow$
2:     META-SEARCH($D, P, R_e, Pl, R_i, \mathcal{H}, T_E, T/2, T/2$)
3: if $\pi = \emptyset$ then
4:     $T \leftarrow T-$ consumedTime
5:     plan $\leftarrow pl^{end}(D^{end}, P^{end}, r_i^{end}, H^{end}, T)$
6: else
7:     plan $\leftarrow \pi$
8: return plan
```

lected PDDL representation of the domain and problem, D^{end} and P^{end}; the selected planner pl^{end}, r_i^{end}, the internal representation, and H^{end}, the set of heuristics. The output also contains a plan π that will be empty if the problem is not solved during meta-search. In that case, the planner is called with the final domain and problem, the selected configuration and the remaining time. These components will be explained in detail in the following subsections.

Representation Changes

The general representation changes that can be included in MSP are of two types: external (R_e) and internal (R_i). Each external representation change operates at the PDDL level and generates a new PDDL domain, D, and problem, P. It can be defined as a function that operates in the space of valid PDDL descriptions, $\mathcal{P}$ (each element of $\mathcal{P}$ is a pair (D, P)): $\forall r_e \in R_e, r_e : \mathcal{P} \rightarrow \mathcal{P}$. Given that they operate over $\mathcal{P}$, the PDDL representation changes can be applied in sequence. We define the composition of two changes $\sigma_{\langle r_e^1, r_e^2 \rangle}$ over a pair (D, P) in the usual way: $\sigma_{\langle r_e^1, r_e^2 \rangle}(D, P) = (r_e^1 \circ r_e^2)(D, P) = r_e^2(r_e^1(D, P))$. The composition of these functions is not necessarily symmetric, so the order in which changes are performed is relevant.

For the internal representation, we are restricted to transformations into SAS$^+$. If $\mathcal{S}$ is the space of all SAS$^+$ descriptions, $\forall r_i \in R_i, r_i : \mathcal{P} \rightarrow \mathcal{S}$. In order to define each r_i, there are usually two processes applied consecutively to the original PDDL representation to generate the internal one in SAS$^+$. First, there is a *translation* process that generates an initial SAS$^+$ representation; and second, there is a *preprocessing* step that generates an optimized SAS$^+$. Our set of internal representation changes R_i contains pairs (t, p), a translator method t and a pre-processor method p.

States of the Meta-search

The meta-search states contain all the necessary information regarding problem representation and the set of heuristics used for solving the problem. We define meta-search states as follows:

Definition 1 (meta-search state) *A meta-search state s is a tuple $s = (\langle r_e^1, \ldots, r_e^n \rangle, D_n, P_n, pl, r_i, H)$, where $\langle r_e^1, \ldots, r_e^n \rangle$, $r_e^i \in R_e$ is a sequence of external representation changes; D_n and P_n are the resulting domain and* *problem generated by applying this sequence to the original domain and problem, $(D_n, P_n) = \sigma_{\langle r_e^1,\ldots,r_e^n\rangle}(D, P)$; $pl \in Pl$ is a planner; $r_i \in R_i$ is an internal representation change; and $H \subseteq \mathcal{H}$ is a subset of heuristics.* [1]

Regarding the PDDL representation, the states contain both the sequence of representation changes and the obtained domain and problem. As we will explain later, the selected changes can affect the applicability of operators (e.g., some representation changes can only be applied once).

With relation to the internal representation, every state of the meta-search contains one translator and pre-processor pair $r_i = (t, p)$. Therefore, meta-search states define implicitly a SAS$^+$ representation, that can be obtained by applying the internal representation procedure to the resulting domain and problem: $r_i(D_n, P_n)$. This involves executing first the translator t and then the pre-processor p on the planning task defined by D_n and P_n. While we could implement all changes done at the PDDL level (R_e) as changes at the SAS$^+$ representation, we keep both kinds of changes independent. On one hand, changes are more easily performed at the PDDL level than at the SAS$^+$ level. On the other hand, and more importantly, keeping these changes at the PDDL level could allow to use other PDDL-based planners that do not work with SAS$^+$ representations.

The subset of heuristics $H \subseteq \mathcal{H}$ represents the selected heuristic for solving the task, defined as the maximum value over all the heuristics in H. Since we are doing optimal planning all heuristics in $\mathcal{H}$ should be admissible.

The meta-search state space is composed of all the possible combinations of sequences of external representation changes, a planner, an internal representation change and a subset of heuristics. We only consider the internal representation change and heuristics which are accepted by the chosen planner.

Given that this meta-search state space can be huge and that it will be traversed at problem solving time (on-line), the challenge consists of how to perform the search efficiently.

The initial meta-search state is $(\emptyset, D, P, pl, r_i, H)$. It contains the empty sequence, the original domain and problem, and default values for pl, r_i, and H.

Meta-search Operators

Given a state $s = (\langle r_e^1, \ldots, r_e^n\rangle, D_n, P_n, pl, r_i, H)$ of the meta-search, there are four types of modifications that can be applied to generate a new state: adding an additional change to the sequence of external changes; selecting a (possibly different) planner, selecting a (possibly different) internal representation change; and selecting a (possibly different) subset of heuristics. Thus, we define meta-search operators as follows.

Definition 2 (meta-search operator) *A meta-search operator is a tuple $mo = (mo_{r_e}, mo_{pl}, mo_{r_i}, mo_H)$, where the first component defines an external representation change $r_e \in R_e$ to be added to the sequence, the second component defines a planner $pl \in Pl$, the third component defines an internal representation change $r_i \in R_i(pl)$ to be used, and the last one defines a selection of heuristics $H \subseteq \mathcal{H}(pl)$.*

[1]n refers just to the number of elements in the sequence.

The operator $mo = (mo_{r_e} = r_e^{n+1}, mo_{pl} = pl', mo_{r_i} = r_i', mo_H = H')$ applied to the state s generates a new state $s' = (\langle r_e^1, \ldots, r_e^n, r_e^{n+1}\rangle, D_{n+1}, P_{n+1}, pl', r_i', H')$, where D_{n+1} and P_{n+1} are the domain and problem generated by the new sequence: $(D_{n+1}, P_{n+1}) = \sigma_{\langle r_e^1,\ldots,r_e^n,r_e^{n+1}\rangle}(D, P)$. pl' is a planner. The internal (SAS$^+$) representation to be used by the planner pl' will be the result of: $r_i'(\sigma_{\langle r_e^1,\ldots,r_e^n,r_e\rangle}(D, P))$. All three, pl', r_i' and H', can take the same value in consecutive states.

Regarding the external changes (PDDL representation), R_e, the MSP version presented at the 2018 IPC only considers *baggy-all* and *neutral*. *baggy-all* applies bagging to all types in the domain according to the Baggy technique (Riddle et al. 2016). This technique consists of replacing by counters all objects of the same type whose name is not relevant. *neutral* makes no change to the PDDL representation.

As planners we consider FD optimal (Helmert 2006) and SYMBA* (Torralba, Linares-López, and Borrajo 2016).

Regarding the translator and pre-processor pairs $(t, p) \in R_i$ we consider the following options. For the FD planner, we defined one translator and two preprocessors. The translator is the standard FD translator (Helmert 2006). For the preprocessors the first option is the standard FD pre-processor. The second one is the h^2-based one defined in (Alcázar and Torralba 2015), that we will refer to as h^2. For the SYMBA* planner we only consider its standard translator and preprocessor.

Finally, we consider the following heuristics: for FD, potentials (Seipp, Pommerening, and Helmert 2015), operator counting (Florian Pommerening and Bonet 2014), ipdb (Haslum et al. 2007), lmcut (Helmert and Domshlak 2009) and blind; for SYMBA* only its default heuristic. For the version presented at the competition the set of heuristics of meta-search states contains just one element. Thus, only one heuristic is selected at each meta-search node.

As an example, a meta-search operator can be $mo_{r_e} =$ *baggy-all*, $mo_{pl} =$ FD, $mo_{r_i} = (t =$ FD$^+, p =$ FD$)$, and $mo_H = \{$potentials$\}$ which applies Baggy to produce a new PDDL representation, selects the planner FD, transforms that into SAS$^+$ using the FD$^+$ translator and the standard FD preprocessor and selects the heuristic *potentials*.

Meta-search Search Technique

MSP performs greedy search with a technique similar to enforced-hill-climbing (Hoffmann and Nebel 2001). Algorithm 2 shows a high-level description of the meta-search. It takes as input a domain D, a problem P, the PDDL representation changes R_e, the planners Pl, the internal representation changes R_i, the set of heuristics $\mathcal{H}$, a meta-search time bound T_m, which will be the maximum time the whole meta-search process can take and a planning time bound T_P, which is the minimum time that the meta-search process assumes the planner will have to solve the problem. The META-SEARCH returns the configuration of the best meta-search state $(D^{end}, P^{end}, pl^{end}, r_i^{end}, H^{end}, \pi)$.

META-SEARCH first generates the operators, $\mathcal{MO}$, given the possible representation changes R_e and R_i, and builds the initial state. We start the search with the original do-

Algorithm 2 META-SEARCH($D, P, R_e, Pl, R_i, \mathcal{H}, T_E, T_m, T_P$)

Inputs: domain D, problem P, PDDL rep. changes R_e, Planners Pl, internal rep. changes R_i, set of heuristics $\mathcal{H}$, estimation time bound T_E, meta-search time bound T_m, planning time bound T_P

Outputs: configuration, $((D^{end}, P^{end}, pl^{end}, r_i^{end}, H^{end}), \pi)$

```
 1: $MO \leftarrow$GENERATE-OPERATORS($R_e, Pl, R_i, \mathcal{H}$)
 2: init $\leftarrow (\emptyset, D, P, SymBA^*, (t = SymT, p = SymP), SymH)$
 3: $f, \pi \leftarrow$ EVALUATE(init,$T_E$)
 4: best $\leftarrow$ init
 5: best_$f \leftarrow f$
 6: succ $\leftarrow$ SUCCESSORS(init,$MO$)
 7: while $T_m$ not reached and $\pi = \emptyset$ and succ$\neq \emptyset$ do
 8:     $s \leftarrow$ pop(succ)
 9:     $f, \pi \leftarrow$ EVALUATE($s, T_E$)
10:     if $f >$best_$f$ then
11:         best $\leftarrow s$
12:         best_$f \leftarrow f$
13:         succ $\leftarrow$ SUCCESSORS($s, MO$)
14: return (best $= (D^{end}, P^{end}, pl^{end}, r_i^{end}, H^{end}), \pi$)
```

main and problem, and the SYMBA* planner together with its translator, preprocessor and heuristics. Then, the initial state is evaluated. The EVALUATE function returns an evaluation of the state and a plan π (that will be empty if no plan is found during evaluation). Then, it generates its successors (the SUCCESSORS function returns a list of meta-search states), and starts evaluating each successor in order. As soon as it finds a state with a better evaluation than its parent's, it stops evaluating the current set of successors, and continues the search, generating the successors of that state. MSP finishes the meta-search if: the time bound is reached; a plan is found while evaluating a meta-search state; or if the list of successors is empty (it did not find a better meta-search state than its parent at any level).

An important decision is on the maximum time EVALUATE will be allowed to evaluate a meta-search state, T_E. If EVALUATE has too little time, the quality of its answers will be poor. If it has too much time, then the meta-search will not be able to search very many meta-search states in this space.

In order to avoid visiting some uninteresting or impossible combinations, we prune any state s when any of the following conditions apply: the same meta-search operator was applied in any ancestor of s; the last r_e is Baggy, and any ancestor of s has already applied Baggy in any form; the last r_e is Baggy, and any previously evaluated state in the search tree has already tried to apply Baggy and the problem could not be bagged.

Meta-Search State Evaluation

To evaluate a meta-search state, s, the meta-search calls EVALUATE(s, T_E). Algorithm 3 shows its pseudo-code. s is the state being evaluated and T_E is the evaluation time limit. Given s and T_E, EVALUATE returns a reasonable estimate of the "goodness" of s within the time bound T_E.

The evaluation of meta-search states is a sampling process that executes the planner defined in the corresponding state until reaching the estimation timeout. Our measure of

Algorithm 3 EVALUATE(s, T_E)

Inputs: meta-search state s, evaluation time bound T_E

Outputs: evaluation f, plan π

```
1: $(\langle r_e^1, \ldots, r_e^n \rangle, D_n, P_n, pl, r_i, H) \leftarrow s$
2: $S \leftarrow r_i(D_n, P_n)$
3: max_flimit, $\pi \leftarrow pl(S, T_E, H)$
4: return $f =$ max_flimit, $\pi$
```

the goodness for meta-search states is how far the search went when sampling; that is, the maximum f-level of the expanded nodes. The basic idea is that the configuration that achieves a higher f-level is better than one that reaches a lower f-level. If a plan is found during the evaluation, that plan is also returned.

Planner

The meta-search algorithm returns the expected best combination of representation, translation, and pre-processing techniques, plus selected heuristics and possibly a plan. If the meta-search finds a solution plan while evaluating any state, MSP just returns it. Otherwise, the planner is executed with the new domain and problem definitions, as well as the selected translator, pre-processor and heuristic.

If the representation chosen was a bagged representation, then the solution will need to be translated back into the original representation as well. Luckily this is a very fast process, linear in the size of the solution path.

References

Alcázar, V., and Torralba, Á. 2015. A reminder about the importance of computing and exploiting invariants in planning. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, 2–6.

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS+ planning. *Computational Intelligence* 11(4):625–655.

Barley, M.; Franco, S.; and Riddle, P. 2014. Overcoming the utility problem in heuristic generation: Why time matters. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, 38–46.

Cenamor, I.; de la Rosa, T.; and Fernández, F. 2016. The IBaCoP planning system: Instance-based configured portfolios. *Journal of Artificial Intelligence Research* 56:657–691.

Florian Pommerening, Gabriele Roeger, M. H., and Bonet, B. 2014. Lp-based heuristics for cost-optimal planning. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, 226–234. AAAI Press.

Fuentetaja, R., and de la Rosa, T. 2016. Compiling irrelevant objects to counters. special case of creation planning. *AI Communications* 29(3):435–467.

Fuentetaja, R.; Barley, M.; Borrajo, D.; Douglas, J.; Franco, S.; and Riddle, P. 2018. Meta-search through the space of representations and heuristics on a problem by problem basis. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*.

Gerevini, A.; Saetti, A.; and Vallati, M. 2009. An automatically configurable portfolio-based planner with macro-actions: PbP. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, 350–353.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL - the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, 1007–1012.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, 162–169.

Helmert, M.; Röger, G.; and Karpas, E. 2011. Fast downward stone soup: A baseline for building planner portfolios. In *Working Notes of the ICAPS Workshop on Planning and Learning (PAL)*, 28–35.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Howe, A. E., and Dahlman, E. 2002. A critical assessment of benchmark comparison in planning. *Journal of Artificial Intelligence Research* 17:1–33.

Howe, A. E.; Dahlman, E.; Hansen, C.; Scheetz, M.; and von Mayrhauser, A. 1999. Recent advances in AI planning. In *Proceedings of the 5th European Conference on Planning (ECP)*, 62–72. Springer-Verlag.

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In *Proceedings of the 10th Europan Conference on Artificial Intelligence*, 359–363. New York, NY, USA: John Wiley & Sons, Inc.

Núñez, S.; Borrajo, D.; and Linares-López, C. 2015. Automatic construction of optimal static sequential portfolios for AI planning and beyond. *Artificial Intelligence* 226:75–101.

Riddle, P.; Douglas, J.; Barley, M.; and Franco, S. 2016. Improving performance by reformulating PDDL into a bagged representation. In *Working Notes of the ICAPS Workshop on Heuristics and Search for Domain-Independent Planning (HDIP)*.

Rintanen, J. 2012. Planning as satisfiability: Heuristics. *Artificial Intelligence* 193(Supplement C):45 – 86.

Seipp, J.; Pommerening, F.; and Helmert, M. 2015. New optimization functions for potential heuristics. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, 193–201. AAAI Press.

Torralba, Á.; Linares-López, C.; and Borrajo, D. 2016. Abstraction heuristics for symbolic bidirectional search. In *Proceedings of IJCAI'16*.

Vallati, M.; Hutter, F.; Chrpa, L.; and McCluskey, T. 2015. On the effective configuration of planning domain models. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*.

DecStar – STAR-topology DECoupled Search at its best

Daniel Gnad
Saarland University
Saarland Informatics Campus
Saarbrücken, Germany
gnad@cs.uni-saarland.de

Alexander Shleyfman
Technion - Israel Institute of Technology
Industrial Engineering & Management
Haifa, Israel
alesh@campus.technion.ac.il

Jörg Hoffmann
Saarland University
Saarland Informatics Campus
Saarbrücken, Germany
hoffmann@cs.uni-saarland.de

Abstract

DecStar extends Fast Downward by Star-Topology Decoupling (STD), a technique recently introduced in classical planning. It exploits independence between components of a planning task to reduce the size of the state-space representation. Partitioning the state variables into components, such that the interaction between these takes the form of a star topology, decoupled search only searches over action sequences affecting the center component of the topology, and enumerates reachable assignments to each leaf component, separately. This can lead to an exponential reduction in the search-space representation size. It is not always easy to find a partitioning for a given planning task, though, so we extend STD by a fallback option, that runs standard search whenever no (good) partitioning could be found.

Introduction

Star-Topology Decoupling (STD) is a recently introduced method to reduce the representation size of search spaces (Gnad and Hoffmann 2015; Gnad, Hoffmann, and Domshlak 2015; Gnad and Hoffmann 2018). By exploiting the structure of the problem within the search – as opposed to doing that within a heuristic function guiding the search – the size of the *decoupled state space* can be exponentially smaller than that of the standard state space. Decoupled search achieves that by partitioning the task into several components, called *factors*, trying to identify a *star topology*, with a single *center factor* that interacts with multiple *leaf factors*. By enforcing such a star structure, and thereby restricting the dependencies between the components, decoupled search has proven to be very efficient and competitive to state-of-the-art planners.

The performance of STD is highly influenced by the outcome of the *factoring* process, i. e., the process of finding a partitioning of the state variables. Just, how to find a good factoring, and what qualifies a factoring as being good? These questions have partially been answered by Gnad, Poser, and Hoffmann (2017), who devised two algorithms that can detect star topologies on a wide range of planning domains. Still, the proposed algorithms can *fail* to find a factoring, or succeed, but return a factoring with undesired properties, e. g. large leaf components that incur a prohibitive runtime overhead when generating new search states. In this case, we simply run standard search, instead [1].

When running STD, we enable some of the extensions that have been developed, namely partial-order reduction (POR) (Gnad, Wehrle, and Hoffmann 2016), symmetry breaking (Gnad et al. 2017), and dominance pruning (Torralba et al. 2016). POR via strong stubborn sets is a technique that is well-known in standard search and originates from the model checking community (Valmari 1989; Alkhazraji et al. 2012; Wehrle and Helmert 2012; 2014). Symmetry breaking has recently been introduced for decoupled search, too. It is a widely known approach across many areas of computer science (e. g. (Starke 1991; Emerson and Sistla 1996; Fox and Long 1999; Rintanen 2003; Pochter, Zohar, and Rosenschein 2011; Domshlak, Katz, and Shleyfman 2012)). Dominance pruning identifies states that can be safely discarded, without affecting completeness (and optimality). POR and symmetry breaking can be used in any given factoring type [2], dominance pruning, however, is only applicable if the generated factoring takes the form of a fork, i. e., the leaves have dependencies on the center, but not vice versa.

In the fallback case, i. e., when no good factoring could be detected and we run standard search, we make use of the variety of techniques that are implemented in Fast Downward (Helmert 2006). In the optimal and bounded-cost tracks, this includes a pattern database heuristic generated using a genetic algorithm (Edelkamp 2006), the LM-cut heuristic (Helmert and Domshlak 2009), a Merge-&-Shrink heuristic (Helmert, Haslum, and Hoffmann 2007; Helmert et al. 2014), and a landmark-count heuristic (Porteous, Sebastia, and Hoffmann 2001; Richter, Helmert, and Westphal 2008). In the agile and satisficing tracks, we mostly use the h^{FF} heuristic (Hoffmann and Nebel 2001) and a search configuration similar to the LAMA planning system (Richter, Westphal, and Helmert 2011).

In all tracks, we extend the standard preprocessor of Fast Downward by the h^2-based task simplification of Alcázar and Torralba (2015), which removes irrelevant and unreachable facts and actions from the task.

[1] This limitation is merely due to the factoring strategies that we use to identify suitable partitionings. In general, *every* task has a star topology and can be tackled by decoupled search.

[2] We use a so-far unpublished extension of strong stubborn sets that supports general star factorings

Preliminaries

We use a finite-domain state variable formalization (FDR) of planning (e. g. (Bäckström and Nebel 1995; Helmert 2006)), where a planning task is a quadruple $\Pi = \langle V, A, I, G \rangle$. V is a set of *state variables*, where each $v \in V$ is associated with a finite domain $\mathcal{D}(v)$. We identify (partial) variable assignments with sets of variable/value pairs. A complete assignment to V is a *state*. I is the *initial state*, and the *goal* G is a partial assignment to V. A is a finite set of *actions*. Each action $a \in A$ is a triple $\langle \mathsf{pre}(a), \mathsf{eff}(a), \mathsf{cost}(a) \rangle$ where the *precondition* $\mathsf{pre}(a)$ and *effect* $\mathsf{eff}(a)$ are partial assignments to V, and $\mathsf{cost}(a)$ is a's non-negative *cost*.

We use the usual FDR semantics. The *planning problem* is to decide if there exists a sequence of actions that transforms the *initial state* I of Π to a state that satisfies the *goal* condition G. In the optimal and bounded-cost tracks of the competition, we are looking for an action sequence with minimal, respectively bounded, summed-up cost.

Decoupled Search

We perform decoupled search like introduced by Gnad and Hoffmann (2018), in its integration in the Fast Downward planning system (Helmert 2006). We use the improved *fork* and *inverted-fork*, as well as the *incident-arcs* factoring methods from Gnad, Poser, and Hoffmann (2017). The outcome of the factoring process is a partitioning $\mathcal{F}$ of the variables of the planning task Π, such that $|\mathcal{F}| > 1$ and there exists $F^C \in \mathcal{F}$ such that, for every action a where $\mathcal{V}(\mathsf{eff}(a)) \cap F^C = \emptyset$, there exists $F \in \mathcal{F}$ with $\mathcal{V}(\mathsf{eff}(a)) \subseteq F$ and $\mathcal{V}(\mathsf{pre}(a)) \subseteq F \cup F^C$. We then call $\mathcal{F}$ a *star factoring*, with *center factor* F^C and *leaf factors* $\mathcal{F}^L := \mathcal{F} \setminus \{F^C\}$.

Given a factoring $\mathcal{F}$, decoupled search is performed as follows: The search will only branch over center actions, i. e., those actions affecting (with an effect on) a variable in F^C. Along such a path of center actions π^C, for each leaf factor F^L, the search maintains a set of leaf paths, i. e., actions only affecting variables of F^L, that *comply* with π^C. Intuitively, for a leaf path π^L to comply with a center path π^C, it must be possible to embed π^L into π^C into an overall action sequence π, such that π is a valid path in the projection of the planning task Π onto $F^C \cup F^L$. A decoupled state corresponds to an end state of such a center action sequence. The main advantage over standard search originates from a decoupled state being able to represent exponentially many explicit states, avoiding their enumeration. A decoupled state can "contain" many explicit states, because by instantiating the center with a center action sequence, the leaf factors are conditionally independent. Thus, the more leaves in the factoring, the more explicit states can potentially be represented by a single decoupled state.

We will next describe a couple of extensions that have been developed for decoupled search and that we use in some of our configurations.

Symmetry Breaking in Decoupled Search

Symmetry Breaking has a long tradition in planning and many other sub-areas of computer science (Starke 1991; Emerson and Sistla 1996; Fox and Long 1999; Rintanen 2003; Pochter, Zohar, and Rosenschein 2011; Domshlak, Katz, and Shleyfman 2012). We use an extension to decoupled search, introduced by Gnad et al. (2017), which is build on *orbit search* (Domshlak, Katz, and Shleyfman 2015; Wehrle et al. 2015). An orbit is a set of states all of which are symmetric to each other. In the search, each state is mapped to a canonical representative of its orbit. In case another state from the same orbit has already been generated (with lower g-cost), the new state can safely be pruned. *Decoupled orbit search* extends this concept to decoupled states.

Decoupled Strong Stubborn Sets

Partial-order reduction is a well-known technique that reduces the size of the search space by pruning transitions that correspond to different permutations of actions (Valmari 1989; Godefroid and Wolper 1991; Edelkamp, Leue, and Lluch-Lafuente 2004; Alkhazraji et al. 2012; Wehrle et al. 2013; Wehrle and Helmert 2014). A variant of strong stubborn sets, decoupled strong stubborn sets (DSSS), has also been introduced for decoupled search. We will employ DSSS in the optimal and bounded-cost tracks. For fork factorings, we use DSSS as defined by Gnad, Wehrle, and Hoffmann (2016). For non-fork factorings, we use a yet unpublished extension that is able to handle arbitrary factorings. To avoid the runtime overhead when DSSS are not effective, we implemented a "safety belt" mechanism, that disables DSSS if after the first 1000 expansions less than 20% of the transitions have been pruned.

Decoupled Dominance Pruning

Another extension that has recently been introduced is dominance pruning (Torralba et al. 2016), where decoupled states that are dominated by other – already generated – states can be safely discarded. We only deploy a very lightweight pruning method, namely *frontier* pruning. The standard way of performing duplicate checking in decoupled search can already detect certain forms of dominance, in particular if two decoupled states have the same center state and all leaf states reachable in one state are (at most as costly) also reachable in the other. Frontier pruning improves this by only comparing a subset of the reached leaf states, those that can possibly make so far unreached leaf states available. It has originally been developed for optimal planning, but can be easily adapted to become more efficient, when optimal solutions do not matter, by replacing the real cost of reaching a leaf state by 0, if a state has been reached at any cost.

Additionally, we also employ a leaf simulation, originally proposed by Torralba and Kissmann (2015), to remove irrelevant leaf states and leaf actions. In some domains, this can tremendously reduce the size of the leaf state spaces.

As indicated before, the techniques described in this subsection are only applicable if $\mathcal{F}$ is a fork factoring.

Implementation & Configurations

Decoupled Search has been implemented as an extension of the Fast Downward (FD) planning system (Helmert 2006). By changing the low-level state representation, many of FD's built-in algorithms and functionality can be used with only minor adaptations. Of particular interest for the DecStar planner are the A* search algorithm, and the $h^{\text{LM-cut}}$ heuristic (Helmert and Domshlak 2009) for optimal, and bounded-cost planning. In the satisficing and agile tracks, we run greedy best-first search (GBFS) using the h^{FF} heuristic (Hoffmann and Nebel 2001). The search algorithms and heuristics can be adapted to decoupled search using a compilation defined by Gnad and Hoffmann (2018). Our implementation does not support conditional effects. On top of the standard FD preprocessor, we perform a relevance analysis based on h^2, to eliminate actions and simplify the planning task prior to the search (Alcázar and Torralba 2015).

In all tracks of the competition, star-topology decoupling is the main component of our planner. However, since, as outline before, our factoring strategies are not guaranteed to find good task decompositions, we need a fallback method. Given the implementation of decoupled search in FD, we can easily make use of the many techniques that FD ships with. Thus, in the case that no good factoring could be obtained, we run standard search using some heuristics and pruning methods that are implemented in FD.

We will use the following notation to describe our techniques: the decoupled variant of search algorithm X is denoted $\mathbf{D}X$. We denote fork (inverted-fork) factorings by **F** (**IF**), and factorings generated using the incident-arcs algorithm by **IA**. To combine the power of the factoring strategies, we use a portfolio approach that runs multiple strategies and picks the one with the maximum number of leaf factors. Further more, we restrict the size for the per-leaf domain-size product to ensure that the leaf state spaces are reasonably small and do not incur a prohibitive runtime overhead when generating new decoupled states. We denote this size limit by $|F^L_{max}| := \max_{F^L \in \mathcal{F}^L} \Pi_{v \in F^L} |\mathcal{D}(v)|$. If a fork factoring is detected, we sometimes perform frontier dominance pruning, denoted **FP** and reduce the size of the leaf state spaces removing irrelevant transitions and states (**IP**). Decoupled strong stubborn sets will be abbreviated as **DSSS**, where we always use the safety belt with a minimum pruning ratio of 20%. In standard search, the use of strong stubborn sets pruning is denoted **SSS**. (Decoupled) orbit search is abbreviated **(D)OSS**. The use of preferred operator pruning is denoted **PO**.

In all but the optimal track, we start by ignoring the action costs. Costs are ignored altogether in the agile track, and only re-introduced in the bounded-cost track if no plan below the cost bound could be found. In the satisficing track, we re-introduce the real costs upon finding the first plan.

In the following sub-sections, we detail the configurations employed in each competition track. We provide the search configurations, as well as the time each of the components is allotted (in seconds).

Optimal Track

DecStar starts by running decoupled search with a fork factoring with a maximum leaf size of 10 million, if one exists. In this case, it employs frontier pruning, removes irrelevance in the leaves, and performs partial-order reduction (DSSS). The next component tries all factoring methods with different size constraints, and prunes states with DSSS and DOSS. This is the main component running for 15min. Both decoupled search components use the LM-cut heuristic (Helmert and Domshlak 2009), currently the strongest admissible heuristic that supports decoupled search.

Search	Factoring	$\lvert F^L_{max}\rvert$	Heuristic	Pruning	Runtime
DA*	F	$10M$	$h^{\text{LM-cut}}$	DSSS,FP,IP	$100s$
DA*	F/IF/IA	$10/10/1M$	$h^{\text{LM-cut}}$	DSSS,DOSS	$800s$
A*	-	-	$h^{\text{LM-cut}}$	SSS,OSS	$180s$
A*	-	-	$h^{\text{GA-PDB}}$	SSS	$180s$
A*	-	-	$h^{\text{M\&S}}$	-	$180s$
A*	-	-	h^{LMc}	-	$180s$
A*	-	-	blind	-	$180s$

Figure 1: Portfolio configuration in the optimal track. Components are launched top to bottom.

In case no matching factoring could be found, or when decoupled search fails, DecStar is supported by standard search with different heuristics. If the heuristic does not support conditional effects, we also enable strong stubborn sets pruning and/or orbit search, which both do not support these, either. DecStar tries the pattern database heuristic with patterns generated using a genetic algorithm ($h^{\text{GA-PDB}}$) (Edelkamp 2006), a Merge&Shrink heuristic with linear merge order and bisimulation ($h^{\text{M\&S}}$) (Helmert, Haslum, and Hoffmann 2007; Helmert et al. 2014; Sievers, Wehrle, and Helmert 2014), the landmark-count heuristic (h^{LMc}) (Porteous, Sebastia, and Hoffmann 2001; Richter, Helmert, and Westphal 2008), and finally blind search.

Satisficing Track

In the satisficing track, DecStar runs three different components. The first, similar to the optimal track, runs decoupled search with a fork factoring, since these typically perform better, in particular when combined with the strong leaf pruning methods (FP,IP). The second component tries all factoring strategies, and additionally enables decoupled orbit search. The "D" in paranthesis indicates that, if none of the factoring strategies succeeds, the component falls back to standard search using the same options. Both components use the h^{FF} heuristic and perform preferred operator pruning, using FD's dual queue mechanism.

Search	Factoring	$\lvert F^L_{max}\rvert$	Heuristic	Pruning	Runtime
$DGBFS$	F	$1M$	h^{FF}	FP,IP,PO	$100s$
$(D)GBFS$	F/IF/IA	$1/1/0.1M$	h^{FF}	(D)OSS,PO	$1000s$
$GBFS$	-	-	h^{LM},h^{FF}	PO	$700s$

Figure 2: Portfolio configuration in the satisficing track. Components are launched top to bottom.

If all is lost, DecStar gets help from his experienced friend LAMA, adopting its first iteration (Richter, Westphal, and Helmert 2011).

Bounded-Cost Track

The components that DecStar uses in the bounded-cost track are a mix of the components described above for the optimal and satisficing track. DecStar starts by running each satisficing-track component for 100s. It then uses a weighted A* search (weight 3), in case none of the previous components could find a plan within the given bound. This is followed by the components used in the optimal track.

Search	Factoring	$\lvert F^L_{max}\rvert$	Heuristic	Pruning	Runtime
$DGBFS$	F	$1M$	h^{FF}	PO,FP,IP	$100s$
$(D)GBFS$	F/IF/IA	$1/1/0.1M$	h^{FF}	(D)OSS,PO	$100s$
$GBFS$	-	-	h^{LM},h^{FF}	PO	$100s$
DWA*	F/IF/IA	$10/10/1M$	h^{FF}	DOSS	$400s$
DA*	F	$10M$	$h^{\text{LM-cut}}$	DSSS,FP,IP	$100s$
DA*	F/IF/IA	$10/10/1M$	$h^{\text{LM-cut}}$	DSSS,DOSS	$400s$
A*	-	-	$h^{\text{LM-cut}}$	SSS,OSS	$120s$
A*	-	-	$h^{\text{GA-PDB}}$	SSS	$120s$
A*	-	-	$h^{\text{M\&S}}$	-	$120s$
A*	-	-	h^{LMc}	-	$120s$
A*	-	-	blind	-	$120s$

Figure 3: Portfolio configuration in the bounded-cost track. Components are launched top to bottom.

Agile Track

Search	Factoring	$\lvert F^L_{max}\rvert$	Heuristic	Pruning	Runtime
$DGBFS$	F	$10K$	h^{FF}	FP,IP,PO	$60s$
$(D)GBFS$	F/IF/IA	$10/10/1K$	h^{FF}	(D)OSS,PO	$120s$
$GBFS$	-	-	h^{LM},h^{FF}	PO	$120s$

Figure 4: Portfolio configuration in the agile track. Components are launched top to bottom.

In the agile track, DecStar uses the same search components as in the satisficing track, but with different timeouts and leaf space size limits. The latter is due to the fact that the larger the leaves, the bigger the runtime overhead per decoupled state. Since the time limit is significantly smaller in the agile track, we try to keep the leaves as small as possible. In spite of the tight time constraint, we still run the h^2-preprocessor for 10s.

Conclusion

DecStar is the best that star-topology decoupling currently has to offer. Many extensions have been developed, allowing the use of various search algorithms, heuristic functions, and pruning techniques. Decoupled search has proved to be a method that can beat other state-of-the-art planners, also in the unsolvability IPC 2014, if the given planning task can be nicely decoupled. Even outside the planning area, namely in proving safety properties in model checking, star-topology decoupling has shown its merit (Gnad et al. 2018).

And still, there are many possible ways of further extending it. In classical planning, where it is crucial to use strong heuristics, the next steps are to do research on how to apply abstraction and LP heuristics in decoupled search. In model checking, an interesting research question is the extension of star-topology decoupling to prove liveness properties.

Acknowledgments. This work was partially supported by the German Research Foundation (DFG), under grant HO 2169/6-1, “Star-Topology Decoupled State Space Search”.

References

Alcázar, V., and Torralba, Á. 2015. A reminder about the importance of computing and exploiting invariants in planning. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*, 2–6. AAAI Press.

Alkhazraji, Y.; Wehrle, M.; Mattmüller, R.; and Helmert, M. 2012. A stubborn set algorithm for optimal planning. In Raedt, L. D., ed., *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI'12)*, 891–892. Montpellier, France: IOS Press.

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS$^+$ planning. *Computational Intelligence* 11(4):625–655.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2012. Enhanced symmetry breaking in cost-optimal planning as forward search. In Bonet, B.; McCluskey, L.; Silva, J. R.; and Williams, B., eds., *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. AAAI Press.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2015. Symmetry breaking in deterministic planning as forward search: Orbit space search algorithm. *Technical Report IS/IE-2015-02*.

Edelkamp, S.; Leue, S.; and Lluch-Lafuente, A. 2004. Partial-order reduction and trail improvement in directed model checking. *International Journal on Software Tools for Technology Transfer* 6(4):277–301.

Edelkamp, S. 2006. Automated creation of pattern database search heuristics. In *Proceedings of the 4th Workshop on Model Checking and Artificial Intelligence (MoChArt 2006)*, 35–50.

Emerson, E. A., and Sistla, A. P. 1996. Symmetry and model-checking. *Formal Methods in System Design* 9(1/2):105–131.

Fox, M., and Long, D. 1999. The detection and exploitation of symmetry in planning problems. In Pollack, M., ed., *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, 956–961. Stockholm, Sweden: Morgan Kaufmann.

Gnad, D., and Hoffmann, J. 2015. Beating LM-cut with h^{max} (sometimes): Fork-decoupled state space search. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*, 88–96. AAAI Press.

Gnad, D., and Hoffmann, J. 2018. Star-topology decoupled state space search. *Artificial Intelligence* 257:24 – 60.

Gnad, D.; Torralba, Á.; Shleyfman, A.; and Hoffmann, J. 2017. Symmetry breaking in star-topology decoupled search. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17)*. AAAI Press.

Gnad, D.; Dubbert, P.; Lluch-Lafuente, A.; and Hoffmann, J. 2018. Star-topology decoupling in spin. In del Mar Gallardo, M., and Merino, P., eds., *Proceedings of the 25th International Symposium on Model Checking of Software (SPIN'18)*, Lecture Notes in Computer Science. Springer.

Gnad, D.; Hoffmann, J.; and Domshlak, C. 2015. From fork decoupling to star-topology decoupling. In Lelis, L., and Stern, R., eds., *Proceedings of the 8th Annual Symposium on Combinatorial Search (SOCS'15)*, 53–61. AAAI Press.

Gnad, D.; Poser, V.; and Hoffmann, J. 2017. Beyond forks: Finding and ranking star factorings for decoupled search. In Sierra, C., ed., *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*. AAAI Press/IJCAI.

Gnad, D.; Wehrle, M.; and Hoffmann, J. 2016. Decoupled strong stubborn sets. In Kambhampati, S., ed., *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'16)*, 3110–3116. AAAI Press/IJCAI.

Godefroid, P., and Wolper, P. 1991. Using partial orders for the efficient verification of deadlock freedom and safety properties. In *Proceedings of the 3rd International Workshop on Computer Aided Verification (CAV'91)*, 332–342.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, 162–169. AAAI Press.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge & shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the Association for Computing Machinery* 61(3).

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In Boddy, M.; Fox, M.; and Thiebaux, S., eds., *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, 176–183. Providence, Rhode Island, USA: Morgan Kaufmann.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Pochter, N.; Zohar, A.; and Rosenschein, J. S. 2011. Exploiting problem symmetries in state-based planners. In Burgard, W., and Roth, D., eds., *Proceedings of the 25th National Conference of the American Association for Artificial Intelligence (AAAI'11)*. San Francisco, CA, USA: AAAI Press.

Porteous, J.; Sebastia, L.; and Hoffmann, J. 2001. On the extraction, ordering, and usage of landmarks in planning. In Cesta, A., and Borrajo, D., eds., *Proceedings of the 6th European Conference on Planning (ECP'01)*, 37–48. Springer-Verlag.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In Fox, D., and Gomes, C., eds., *Proceedings of the 23rd National Conference of the American Association for Artificial Intelligence (AAAI'08)*, 975–982. Chicago, Illinois, USA: AAAI Press.

Richter, S.; Westphal, M.; and Helmert, M. 2011. LAMA 2008 and 2011 (planner abstract). In *IPC 2011 planner abstracts*, 50–54.

Rintanen, J. 2003. Symmetry reduction for SAT representations of transition systems. In Giunchiglia, E.; Muscettola, N.; and Nau, D., eds., *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS'03)*, 32–41. Trento, Italy: Morgan Kaufmann.

Sievers, S.; Wehrle, M.; and Helmert, M. 2014. Generalized label reduction for merge-and-shrink heuristics. In Brodley, C. E., and Stone, P., eds., *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI'14)*, 2358–2366. Austin, Texas, USA: AAAI Press.

Starke, P. 1991. Reachability analysis of petri nets using symmetries. *Journal of Mathematical Modelling and Simulation in Systems Analysis* 8(4/5):293–304.

Torralba, Á., and Kissmann, P. 2015. Focusing on what really matters: Irrelevance pruning in merge-and-shrink. In Lelis, L., and Stern, R., eds., *Proceedings of the 8th Annual Symposium on Combinatorial Search (SOCS'15)*, 122–130. AAAI Press.

Torralba, Á.; Gnad, D.; Dubbert, P.; and Hoffmann, J. 2016. On state-dominance criteria in fork-decoupled search. In Kambhampati, S., ed., *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'16)*. AAAI Press/IJCAI.

Valmari, A. 1989. Stubborn sets for reduced state space generation. In *Proceedings of the 10th International Conference on Applications and Theory of Petri Nets*, 491–515.

Wehrle, M., and Helmert, M. 2012. About partial order reduction in planning and computer aided verification. In Bonet, B.; McCluskey, L.; Silva, J. R.; and Williams, B., eds., *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. AAAI Press.

Wehrle, M., and Helmert, M. 2014. Efficient stubborn sets: Generalized algorithms and selection strategies. In Chien, S.; Do, M.; Fern, A.; and Ruml, W., eds., *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*. AAAI Press.

Wehrle, M.; Helmert, M.; Alkhazraji, Y.; and Mattmüller, R. 2013. The relative pruning power of strong stubborn sets and expansion core. In Borrajo, D.; Fratini, S.; Kambhampati, S.; and Oddi, A., eds., *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*. Rome, Italy: AAAI Press.

Wehrle, M.; Helmert, M.; Shleyfman, A.; and Katz, M. 2015. Integrating partial order reduction and symmetry elimination for cost-optimal classical planning. In Yang, Q., ed., *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*. AAAI Press/IJCAI.

Cerberus: Red-Black Heuristic for Planning Tasks with Conditional Effects Meets Novelty Heuristic and Enchanced Mutex Detection

Michael Katz
IBM Research
Yorktown Heights, NY, USA
michael.katz1@ibm.com

Abstract

Red-black planning is the state-of-the-art approach to satisficing classical planning. A planner *Mercury*, empowered by the red-black planning heuristic, was the runner-up of the latest International Planning Competition (IPC) 2014, despite the trivial handling of conditional effects by compiling them away. Conditional effects are important for classical planning and required in many domains for efficient modeling. Another recent success in satisficing classical planning is the Novelty based heuristic guidance. When novelty of heuristic values is considered, search space is partitioned into novelty layers. Exploring these layers in the order of their novelty considerably improves the performance of the underlying heuristics. Yet another recent success relates to the translation of planning tasks from the input PDDL language to a grounded multi-valued variable based representation, such as SAS^+. Recent methods of invariants synthesis allow for deriving richer SAS^+ representations.

We herein present a satisficing classical planner which we baptize *Cerberus*, that incorporates these three recent improvements. It starts by performing enhanced mutex detection to derive a SAS^+ planning task with conditional effects. Then, it performs best first search of various greediness, exploiting red-black planning heuristic with a direct handling of conditional effects and using such red-black heuristic as a base for a novelty heuristic.

Introduction

Delete relaxation heuristics have played a key role in the success of satisficing planning systems (Bonet and Geffner 2001; Hoffmann and Nebel 2001; Richter and Westphal 2010). A well-known pitfall of delete relaxation is its inability to account for repetive achievements of facts. It has thus been an actively researched question from the outset how to take *some* deletes into account, e. g. (Fox and Long 2001; Gerevini, Saetti, and Serina 2003; Helmert 2004; Helmert and Geffner 2008; Baier and Botea 2009; Cai, Hoffmann, and Helmert 2009; Haslum 2012; Keyder, Hoffmann, and Haslum 2012). Red-black planning framework (Domshlak, Hoffmann, and Katz 2015), where a subset of *red* state variables takes on the relaxed value-accumulating semantics, while the other *black* variables retain the regular semantics, introduced a convenient way of interpolating between fully relaxed and regular planning.

Katz, Hoffmann, and Domshlak (2013b) introduced the red-black framework and conducted a theoretical investigation of tractability. Following up on this, they devised practical *red-black plan heuristics*, non-admissible heuristics generated by repairing fully delete-relaxed plans into red-black plans (Katz, Hoffmann, and Domshlak 2013a). Observing that this technique often suffers from dramatic over-estimation incurred by following arbitrary decisions taken in delete-relaxed plans, Katz and Hoffmann (2013) refined the approach to rely less on such decisions, yielding a more flexible algorithm delivering better search guidance. Subsequently, Katz and Hoffmann (2014b) presented a *red-black DAG heuristics* for a tractable fragment characterized by *DAG black causal graphs* and devise some enhancements targeted at making the resulting red-black plans executable in the real task, stopping the search if they succeed in reaching the goal. Red-black DAG heuristics are in the heart of the *Mercury* planner (Katz and Hoffmann 2014a), the runner-up of the sequential satisficing track in the latest International Planning Competition (IPC 2014). All aforementioned work on red-black planning, however, handles the SAS^+ fragment without conditional effects, despite of conditional effects being a main feature in the domains of IPC 2014. The planner *Mercury* that favorably participated in IPC 2014, handles conditional effects by simply compiling them away (Nebel 2000). Obviously, the number of actions in the resulted planning tasks grows exponentially, and thus such straight forward compiling away does not scale well. Nebel (2000) presents an alternative compilation, that does not lead to an exponential blow-up in the task size. This compilation, however does not preserve the delete relaxation. Thus, several delete relaxation based heuristics were adapted to natively support conditional effects (Haslum 2013; Röger, Pommerening, and Helmert 2014). Recently, Katz (2018) has shown that the fragment of red-black planning characterized by *DAG black causal graphs* remains tractable in the presence of conditional effects, extending the existing red-black planning heuristics to natively handling conditional effects.

Search-boosting and pruning techniques have considerably advanced the state-of-the-art in planning as heuristic search (Richter and Helmert 2009; Richter and Westphal 2010; Xie et al. 2014; Valenzano et al. 2014; Domshlak, Katz, and Shleyfman 2013; Lipovetzky and Geffner 2012).

One such technique is based on the concept of *novelty* of a state, where the search procedure prunes nodes that do not qualify as *novel*. The concept has been successfully exploited in classical planning via SIW^+ and $DFS(i)$ search algorithms and in heuristic search, in conjunction with helpful actions (Lipovetzky and Geffner 2012; 2014; 2017). and in blind state-space search for deterministic online planning in Atari-like problems (Lipovetzky, Ramirez, and Geffner 2015), where it was later generalized to account for rewards (Shleyfman, Tuisov, and Domshlak 2016; Jinnai and Fukunaga 2017). The latter work, although applied to Atari-like problems, is valid for planning with rewards in general, when rewards are defined on states. Consequently, (Katz et al. 2017) brought the concept of novelty back to heuristic search, adapting the novelty definition of Shleyfman, Tuisov, and Domshlak (2016) to a novelty of a state with respect to its heuristic estimate. The new novelty notion was no longer used solely for pruning search nodes, but rather as a heuristic function, for node ordering in a queue. However, since such heuristics are not goal-aware, Katz et al. (2017) use the base goal-aware heuristic as a secondary (tie-breaking) heuristic for node ordering.

In this work we construct a planner *Cerberus*, named after the monstrous three-headed guardian of the gates of the Underworld in Greek mythology. The planner incorporates three main recent improvements, namely enhanced mutex detection, recent novelty heuristic, and the extension of red-black planning heuristic to conditional effects. Two variants of the planner submitted to the International Planning Competition (IPC) 2018 differ in the red-black planning heuristic they use. In the reminder of this paper we describe the components in detail.

Configurations

Both *Cerberus* variants participate in three tracks, namely satisficing, agile, and bounded-cost. They are built on top of the adaptation of the *Mercury* planner (Katz and Hoffmann 2014a), runner-up of the sequential satisficing track of IPC 2014, to the recent version of the Fast Downward framework (Helmert 2006). Furhter, the implementation is extended to natively support conditional effects (Katz 2018). In contrast to *Mercury* planner, the red-black planning heuristic is enhanced by the novelty heuristic (Katz et al. 2017), replacing the queues ordered by the red-black planning heuristic h^{RB} in *Mercury* planner with queues ordered by the novelty of a state with respect to its red-black planning heuristic estimate h^{RB}, with ties broken by h^{RB}. In what follows, we describe the parts that are shared between the tracks and then detail the configuration for each track.

Enchanced Invariance Detection

As the search and the heuristic computation are performed on the finite domain representation SAS^+ (Bäckström and Nebel 1995), invariance detection plays a significant role in the quality of the translation from PDDL representation to SAS^+. To reduce the number of multi-valued state variables we exploit the h^2 mutexes detection as a preprocessing step (Alcázar and Torralba 2015). In our preliminary experiments, this step was observed to make a significant contribution to the performance of the overall planning system.

Red-Black Planning Heuristic

In order to describe the configuration of the red-black planning heuristic h^{RB}, we need to specify how a red-black task is constructed (which variables are chosen to be red and which black), also known as painting strategy, as well as how the red-black task is solved. In both cases, we followed the choices made by *Mercury* planner. Specifically, for red-black task construction followed one of the basic strategies, namely ordering the variables by causal graph level, and either (a) iteratively painting variables red until the black causal graph becomes a DAG (Domshlak, Hoffmann, and Katz 2015), or (b) iteratively painting variables black as long as the black causal graph is a DAG. There are two submitted planners, that differ in their painting strategies. While the planner that (similarly to *Mercury* planner) uses strategy (a) is called *Cerberus*, the planner that uses strategy (b) is denoted by *Cerberus-gl*. These two planners differ in red-black planning task creation only, and therefore in what follows, we describe the configurations without mentioning the actual planner. The further difference from *Mercury* planner is in the definition of invertibility in the presence of conditional effects. In our planners we follow the definition of Katz (2018).

For solving the red-black task, we use the algorithm presented in Figure 2 of Katz (2018). It is an adaptation of the algorithm of Katz and Hoffmann (2014a) to tasks with conditional effects. The algorithm receives a red-black planning task, as well as a set of red facts that is sufficient for reaching the red-black goals. Such a set is typically obtained from a relaxed solution to the task. Then, it iteratively (i) selects an action that can achieve some previously unachieved fact from that set, (ii) achieves its preconditions, and (iii) applies the action. Finally, when all the facts in the set are achieved, it achieves the goal of the task. We follow Katz and Hoffmann (2014a) in the two optimizations applied to ehnance red-black plan applicability: selecting the next action in (i) preferring actions such that achieving their black preconditions does not involve deleting facts from the set above, and selecting the sequences of actions in (ii), preferring those that are executable in the current state.

Landmarks Count Heuristic

Following the successful approaches of *Mercury* and LAMA planners, we use additional queues ordered by the landmark count heuristic (Richter and Westphal 2010).

Novelty Heuristic

The novelty heuristic used in our planners measures the novelty of a state with respect to its red-black planning heuristic estimate h^{RB}. Specifically, we use the h_{QB} heuristic, as described in Equation 3 of Katz et al. (2017). The *quantified both* novel and non-novel heuristic h_{QB} is designed not only to distinguish novel states from non-novel ones, but also to separate novel states, and even to separate non-novel ones. Consequently, we use the best performing overall configuration of Katz et al. (2017) in *Cerberus* planners.

Satisficing Track

The configuration runs a sequence of search iterations of decreasing level of greediness. The first iteration is the greedy best-first search (GBFS) with deferred heuristic evaluation, alternating between four queues. The first queue is ordered by the novelty of a state with respect to its red-black planning heuristic estimate h^{RB}, with ties broken by h^{RB}. The second queue consists of states achieved by preferred operators of the red-black planning heuristic[1] h^{RB}, ordered by h^{RB}. The third and forth queues are ordered by the landmark count heuristic, with all successors and those achieved by the preferred operators, respectively.

The next iterations perform a weighted A^* with deferred heuristic evaluation and decreasing weights $w = 5, 3, 2, 1$, continuing with $w = 1$. All these iterations alternate between the four queues as in *Mercury* planner, with the first two ordered by h^{RB}, with all successors and those achieved by the preferred operators, respectively, and the last two as in the first iteration. In case a solution is found in the previous iteration, its cost is passed as a pruning bound to the next iteration.

In case of non-unit costs, a cost transformation is performed, adding a constant 1 to all costs. Further, the first iteration is performed twice, once with unit costs and once with the increased costs.

Agile Track

The configuration in the agile track mimics the first iteration of the configuration in the satisficing track as described above.

Bounded-Cost Track

The configuration in the bounded-cost track mimics the configuration in the agile track as described above. The only difference is that the cost bound is provided as an input.

[1]These are basically the preferred operators of the full delete relaxation, the FF heuristic.

References

Alcázar, V., and Torralba, Á. 2015. A reminder about the importance of computing and exploiting invariants in planning. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 2–6. AAAI Press.

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS$^+$ planning. *Computational Intelligence* 11(4):625–655.

Baier, J. A., and Botea, A. 2009. Improving planning performance using low-conflict relaxed plans. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 10–17. AAAI Press.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1):5–33.

Cai, D.; Hoffmann, J.; and Helmert, M. 2009. Enhancing the context-enhanced additive heuristic with precedence constraints. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 50–57. AAAI Press.

Domshlak, C.; Hoffmann, J.; and Katz, M. 2015. Red-black planning: A new systematic approach to partial delete relaxation. *Artificial Intelligence* 221:73–114.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2013. Symmetry breaking: Satisficing planning and landmark heuristics. In Borrajo, D.; Kambhampati, S.; Oddi, A.; and Fratini, S., eds., *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*, 298–302. AAAI Press.

Fox, M., and Long, D. 2001. Stan4: A hybrid planning strategy based on subproblem abstraction. *AI Magazine* 22(3):81–84.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs in LPG. *Journal of Artificial Intelligence Research* 20:239–290.

Haslum, P. 2012. Incremental lower bounds for additive cost planning problems. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 74–82. AAAI Press.

Haslum, P. 2013. Optimal delete-relaxed (and semi-relaxed) planning with conditional effects. In Rossi, F., ed., *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 2291–2297. AAAI Press.

Helmert, M., and Geffner, H. 2008. Unifying the causal graph and additive heuristics. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E., eds., *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*, 140–147. AAAI Press.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In Zilberstein, S.; Koehler, J.; and Koenig, S., eds., *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*, 161–170. AAAI Press.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Jinnai, Y., and Fukunaga, A. 2017. Learning to prune dominated action sequences in online black-box planning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)*. AAAI Press.

Katz, M., and Hoffmann, J. 2013. Red-black relaxed plan heuristics reloaded. In Helmert, M., and Röger, G., eds., *Proceedings of the Sixth Annual Symposium on Combinatorial Search (SoCS 2013)*, 105–113. AAAI Press.

Katz, M., and Hoffmann, J. 2014a. Mercury planner: Pushing the limits of partial delete relaxation. In *Eighth International Planning Competition (IPC-8): planner abstracts*, 43–47.

Katz, M., and Hoffmann, J. 2014b. Pushing the limits of partial delete relaxation: Red-black DAG heuristics. In *ICAPS 2014 Workshop on Heuristics and Search for Domain-independent Planning (HSDIP)*, 40–44.

Katz, M.; Lipovetzky, N.; Moshkovich, D.; and Tuisov, A. 2017. Adapting novelty to classical planning as heuristic search. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*, 172–180. AAAI Press.

Katz, M.; Hoffmann, J.; and Domshlak, C. 2013a. Red-black relaxed plan heuristics. In desJardins, M., and Littman, M. L., eds., *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI 2013)*, 489–495. AAAI Press.

Katz, M.; Hoffmann, J.; and Domshlak, C. 2013b. Who said we need to relax *all* variables? In Borrajo, D.; Kambhampati, S.; Oddi, A.; and Fratini, S., eds., *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*, 126–134. AAAI Press.

Katz, M. 2018. Red-black heuristic for planning tasks with conditional effects. Technical report, IBM. Available at `http://ibm.biz/ceffRBTr`.

Keyder, E.; Hoffmann, J.; and Haslum, P. 2012. Semi-relaxed plan heuristics. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 128–136. AAAI Press.

Lipovetzky, N., and Geffner, H. 2012. Width and serialization of classical planning problems. In De Raedt, L.; Bessiere, C.; Dubois, D.; Doherty, P.; Frasconi, P.; Heintz, F.; and Lucas, P., eds., *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, 540–545. IOS Press.

Lipovetzky, N., and Geffner, H. 2014. Width-based algorithms for classical planning: New results. In Schaub, T.; Friedrich, G.; and O'Sullivan, B., eds., *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, 1059–1060. IOS Press.

Lipovetzky, N., and Geffner, H. 2017. Best-first width search: Exploration and exploitation in classical planning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017).* AAAI Press.

Lipovetzky, N.; Ramirez, M.; and Geffner, H. 2015. Classical planning with simulators: Results on the atari video games. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, 1610–1616. AAAI Press.

Nebel, B. 2000. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research* 12:271–315.

Richter, S., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 273–280. AAAI Press.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39:127–177.

Röger, G.; Pommerening, F.; and Helmert, M. 2014. Optimal planning in the presence of conditional effects: Extending LM-Cut with context splitting. In Schaub, T.; Friedrich, G.; and O'Sullivan, B., eds., *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, 765–770. IOS Press.

Shleyfman, A.; Tuisov, A.; and Domshlak, C. 2016. Blind search for atari-like online planning revisited. In Kambhampati, S., ed., *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, 3251–3257. AAAI Press.

Valenzano, R.; Sturtevant, N. R.; Schaeffer, J.; and Xie, F. 2014. A comparison of knowledge-based GBFS enhancements and knowledge-free exploration. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 375–379. AAAI Press.

Xie, F.; Müller, M.; Holte, R. C.; and Imai, T. 2014. Type-based exploration with multiple search queues for satisficing planning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2014).* AAAI Press.

Good Old Mercury Planner

Michael Katz
IBM Research
Yorktown Heights, NY, USA
michael.katz1@ibm.com

Jörg Hoffmann
Saarland University
Saarbrücken, Germany
hoffmann@cs.uni-saarland.de

Abstract

Mercury is a sequential satisficing planner that favorably competed in the International Planning Competition (IPC) 2014. *Mercury* planner is based mainly on the red-black planning heuristic. Red-black planning is a systematic approach to partial delete relaxation, taking into account *some* of the delete effects: Red variables take the relaxed (value-accumulating) semantics, while black variables take the regular semantics. *Mercury* planner exploits a powerful tractable fragment requiring the *black causal graph* – the projection of the causal graph onto the black variables – to be a DAG. Further, it applies techniques aimed at making red-black plans executable, short-cutting the search. As in 2014, Mercury planner is entered into sequential satisficing and agile tracks of the competition.

Planner structure

Mercury planner (Katz and Hoffmann 2014a) is a sequential satisficing planner that is implemented in the Fast Downward planning system (Helmert 2006). The planner is submitted for participation in the International Planning Competition (IPC) 2018.

Satisficing Track

The variant that competes in the *satisficing track* performs multiple iterations of heuristic search, starting with a fast and inaccurate greedy best-first search with deferred heuristic evaluation. Once a solution is found, next iterations run weighted A^* with deferred heuristic evaluation, gradually decreasing the weight parameter, similarly to the famous LAMA planning system (Richter and Westphal 2010). The cost of the best plan found so far is used in following iterations for search space pruning. Also similarly to LAMA, each search iteration alternates between four queues, two per heuristic, with all successors and successors reached by preferred operators only. The heuristics are the landmark count heuristic (Porteous, Sebastia, and Hoffmann 2001), and the red-black planning heuristic (Katz, Hoffmann, and Domshlak 2013b; 2013a; Katz and Hoffmann 2013; 2014b; Domshlak, Hoffmann, and Katz 2015). For red-black heuristic, which is based on FF (Hoffmann and Nebel 2001), the preferred operators are obtained as the preferred operators of FF heuristic.

Agile Track

The variant that competes in the *agile track* performs a single iteration of a greedy best-first search with deferred heuristic evaluation, alternating between two queues ordered by the red-black planning heuristic. These queues are filled with all successors and successors reached by preferred operators defined by the red-black planning heuristic.

Red-Black Planning Heuristic

In order to describe the configuration of the red-black planning heuristic, we need to specify how a red-black task is constructed (which variables are chosen to be red and which black), also known as painting strategy, as well as how the red-black task is solved. For red-black task construction the variables are ordered by causal graph level and iteratively painted red until the black causal graph becomes a DAG (Domshlak, Hoffmann, and Katz 2015). For solving the red-black task, the following algorithm is used: The algorithm receives a red-black planning task, as well as a set of red facts that is sufficient for reaching the red-black goals. Such a set is typically obtained from a relaxed solution to the task. Then, it iteratively (i) selects an action that can achieve some previously unachieved fact from that set, (ii) achieves its preconditions, and (iii) applies the action. Finally, when all the facts in the set are achieved, it achieves the goal of the task. There are two optimizations applied to ehnance red-black plan applicability: selecting the next action in (i) preferring actions such that achieving their black preconditions does not involve deleting facts from the set above, and selecting the sequences of actions in (ii), preferring those that are executable in the current state (Katz and Hoffmann 2014a).

Supported Features

As in the previous competition, a support for conditional effects is currently required. *Mercury* planner supports conditional effects by compiling them away. This was done by multiplying them out in the translation step. On one hand, this can lead to an exponential blow-up in the task representation size. On the other hand, it does not split up an operator application into a sequence of operator applications. Our decision was based on the speculation that the latter option could potentially decrease red-black plan applicability, one of the main advantages of the current red-black heuristics.

In order to be able to take advantage of the larger memory resource available to the participants of the current competition, the planner is built with the support for 64bit enabled.

References

Domshlak, C.; Hoffmann, J.; and Katz, M. 2015. Red-black planning: A new systematic approach to partial delete relaxation. *Artificial Intelligence* 221:73–114.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Katz, M., and Hoffmann, J. 2013. Red-black relaxed plan heuristics reloaded. In Helmert, M., and Röger, G., eds., *Proceedings of the Sixth Annual Symposium on Combinatorial Search (SoCS 2013)*, 105–113. AAAI Press.

Katz, M., and Hoffmann, J. 2014a. Mercury planner: Pushing the limits of partial delete relaxation. In *Eighth International Planning Competition (IPC-8): planner abstracts*, 43–47.

Katz, M., and Hoffmann, J. 2014b. Pushing the limits of partial delete relaxation: Red-black DAG heuristics. In *ICAPS 2014 Workshop on Heuristics and Search for Domain-independent Planning (HSDIP)*, 40–44.

Katz, M.; Hoffmann, J.; and Domshlak, C. 2013a. Red-black relaxed plan heuristics. In desJardins, M., and Littman, M. L., eds., *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI 2013)*, 489–495. AAAI Press.

Katz, M.; Hoffmann, J.; and Domshlak, C. 2013b. Who said we need to relax *all* variables? In Borrajo, D.; Kambhampati, S.; Oddi, A.; and Fratini, S., eds., *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*, 126–134. AAAI Press.

Porteous, J.; Sebastia, L.; and Hoffmann, J. 2001. On the extraction, ordering, and usage of landmarks in planning. In Cesta, A., and Borrajo, D., eds., *Proceedings of the Sixth European Conference on Planning (ECP 2001)*, 174–182. AAAI Press.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39:127–177.

MERWIN Planner: Mercury Enchanced With Novelty Heuristic

Michael Katz
IBM Research
Yorktown Heights, NY, USA
michael.katz1@ibm.com

Nir Lipovetzky
University of Melbourne
Melbourne, Australia
nir.lipovetzky@unimelb.edu.au

Dany Moshkovich
IBM Research
Haifa, Israel
mdany@il.ibm.com

Alexander Tuisov
Technion
Haifa, Israel
alexandt@campus.technion.ac.il

Abstract

Heuristic search with red-black planning heuristics is among the most effective approaches to satisficing planning and the driving power behind the state-of-the-art satisficing planner *Mercury*. Another recent success in satisficing planning is due to the introduction of novelty based heuristic guidance, in particular a guidance measuring the novelty of a heuristic estimate in a state.

A satisficing planner that we baptize *MERWIN* empowers red-black planning heuristics with novelty based guidance, measuring the novelty of red-black planning heuristic estimates in explored states. *MERWIN* planner partitions the state space into novelty layers, expanding the most novel nodes first, and breaking ties within each layer by the red-black heuristic values.

Introduction

Delete relaxation heuristics are the key component of many successful planning systems (Bonet and Geffner 2001; Hoffmann and Nebel 2001; Richter and Westphal 2010). These heuristics though have a well-known pitfall of not being able to account for multiple achievements of the same fact, which leads to a wide research on how to take at least *some* deletes into account, e. g. (Fox and Long 2001; Gerevini, Saetti, and Serina 2003; Helmert 2004; Helmert and Geffner 2008; Baier and Botea 2009; Cai, Hoffmann, and Helmert 2009; Haslum 2012; Keyder, Hoffmann, and Haslum 2012). One such approach is so-called red-black planning (Domshlak, Hoffmann, and Katz 2015), where a subset of *red* state variables takes on the relaxed value-accumulating semantics, while the other *black* variables retain the regular semantics. This allows to interpolate between fully relaxed and regular planning. The work started with the introduction of the red-black framework and a theoretical investigation of tractability (Katz, Hoffmann, and Domshlak 2013b). Following up on this, practical non-admissible *red-black plan heuristics* were introduced, extending delete-relaxed plans into red-black plans (Katz, Hoffmann, and Domshlak 2013a). The technique, however, often suffered from dramatic overestimation incurred by following arbitrary decisions taken in delete-relaxed plans, and to overcome this shortcoming, Katz and Hoffmann (2013) refined the approach to rely less on such decisions, yielding a more flexible algorithm delivering better search guidance. Subsequently, Katz and Hoffmann (2014b) presented a *red-black DAG heuristics* for a tractable fragment characterized by *DAG black causal graphs* and devised some enhancements targeted at making the resulting red-black plans executable in the real task, stopping the search if they succeed in reaching the goal. Red-black DAG heuristics are in the heart of the *Mercury* planner (Katz and Hoffmann 2014a), the runner-up of the sequential satisficing track in the latest International Planning Competition (IPC 2014). It is worth mentioning that all aforementioned work on red-black planning handles the SAS$^+$ fragment without conditional effects. Since conditional effects were a required feature to be supported by participating planners, *Mercury* handled conditional effects by simply compiling them away (Nebel 2000).

Search-boosting and pruning techniques have considerably advanced the state-of-the-art in planning as heuristic search (Richter and Helmert 2009; Richter and Westphal 2010; Xie et al. 2014; Valenzano et al. 2014; Domshlak, Katz, and Shleyfman 2013; Lipovetzky and Geffner 2012). One such technique is based on the concept of *novelty* of a state, where the search procedure prunes nodes that do not qualify as *novel*. Novelty has been successfully exploited for pruning in classical planning via SIW^+ and $DFS(i)$ search algorithms (Lipovetzky and Geffner 2012; 2014). The blind novelty pruning IW algorithm has shown great performance for classical online planning and finite horizon MDP problems over the Atari simulator (ALE) and General Video Game competition (GVG-AI) (Lipovetzky, Ramirez, and Geffner 2015; Bandres, Bonet, and Geffner 2018; Geffner and Geffner 2015), where it was later generalised to account for novelty based on rewards (Shleyfman, Tuisov, and Domshlak 2016; Jinnai and Fukunaga 2017). The latter work, although applied to Atari-like problems, is valid for planning with rewards in general, when rewards are defined on states. Consequently, Lipovetzky and Geffner (2017a) and Katz et al. (2017) brought the concept of novelty back to heuristic search, adapting the novelty definition of Shleyfman, Tuisov, and Domshlak (2016) to a novelty of a state with respect to its heuristic estimate. These two adaptations, althogh similar in nature, differ in detail (Katz et al. 2017). The new novelty notion was no longer used solely for pruning search nodes, but rather as a heuristic function, for node ordering in a queue. However, since such heuristics are not goal-aware, both Lipovetzky and Geffner

(2017a) and Katz et al. (2017) use the base heuristic as a secondary (tie-breaking) heuristic for node ordering. This general search framework is sometimes referred to as *Best First Width Search* (*BFWS*) (Lipovetzky and Geffner 2017a). Variants of *BFWS* can yield state-of-the-art polynomial planners (Lipovetzky and Geffner 2017b), and maintain good performance even when the action model is given as a Black-box simulator (Frances et al. 2017). In what follows, we exploit the notion of novelty of a state with respect to its heuristic estimate as defined by Katz et al. (2017).

In this work we construct a planner *MERWIN*, which stands for *MERcury enhanced WIth Novelty*, by exploiting both the red-black planning heuristic and the novelty of states with respect to a heuristic estimate. In particular, we modify the *Mercury* planner by replacing the queue ordered by red-black planning heuristic with a queue ordered by the novelty of a state with respect to its red-black planning heuristic estimate, breaking ties by that red-black planning heuristic estimate.

Configurations

MERWIN planner participates in three tracks, namely satisficing, agile, and bounded-cost. It is built on top of the *Mercury* planner (Katz and Hoffmann 2014a), runner-up of the sequential satisficing track of the International Planning Competition (IPC) 2014. Informally, as was mentioned above, the red-black planning heuristic in *Mercury* planner is enhanced by the novelty heuristic (Katz et al. 2017), and thus the queues ordered by the red-black planning heuristic h^{RB} in *Mercury* planner are replaced by the queues ordered by the novelty of a state with respect to its red-black planning heuristic estimate h^{RB}, with ties broken by h^{RB}. In what follows, we describe the heuristics used for all tracks and then detail the configuration for each track.

Red-Black Planning Heuristic

In order to describe the configuration of the red-black planning heuristic h^{RB}, we need to specify how a red-black task is constructed (which variables are chosen to be red and which black), also known as painting strategy, as well as how the red-black task is solved. In both cases, we followed the choices made by *Mercury* planner. Specifically, for red-black task construction followed one of the basic strategies, namely ordering the variables by causal graph level, and iteratively painting variables red until the black causal graph becomes a DAG (Domshlak, Hoffmann, and Katz 2015).

For solving the red-black task, *MERWIN* planner uses the algorithm presented in Figure 2 of Katz and Hoffmann (2014a). The algorithm receives a red-black planning task, as well as a set of red facts that is sufficient for reaching the red-black goals. Such a set is typically obtained from a relaxed solution to the task. Then, it iteratively (i) selects an action that can achieve some previously unachieved fact from that set, (ii) achieves its preconditions, and (iii) applies the action. Finally, when all the facts in the set are achieved, it achieves the goal of the task. We follow Katz and Hoffmann (2014a) in the two optimizations applied to ehnance red-black plan applicability: selecting the next action in (i) giving a preference to actions whose black preconditions can be achieved without deleting facts from the set above, and selecting the sequences of actions in (ii), preferring those that are executable in the current state.

Novelty Heuristic

The novelty heuristic used in our planners measures the novelty of a state with respect to its red-black planning heuristic estimate h^{RB}. Specifically, we use the h_{QB} heuristic, as described in Equation 3 of Katz et al. (2017). The *quantified both* novel and non-novel heuristic h_{QB} is designed not only to distinguish novel states from non-novel ones, but also to separate the degree of (non-)novelty. Consequently, we use the best performing overall configuration of Katz et al. (2017) in *MERWIN* planner.

Landmarks Count Heuristic

Following the successful approaches of *Mercury* and LAMA planners, *MERWIN* planner uses additional queues ordered by the landmark count heuristic (Richter and Westphal 2010).

Satisficing Track

The configuration runs a sequence of search iterations of decreasing level of greediness. The first iteration is the greedy best-first search (GBFS) with deferred heuristic evaluation, alternating between four queues. The first queue is ordered by the novelty of a state with respect to its red-black planning heuristic estimate h^{RB}, with ties broken by h^{RB}. The second queue consists of states achieved by preferred operators of the red-black planning heuristic[1] h^{RB}, ordered by h^{RB}. The third and forth queues are ordered by the landmark count heuristic, with all successors and those achieved by the preferred operators, respectively.

The next iterations perform a weighted A^* with deferred heuristic evaluation and decreasing weights $w = 5, 3, 2, 1$, continuing with $w = 1$. All these iterations alternate between the four queues as in *Mercury* planner, with the first two ordered by h^{RB}, with all successors and those achieved by the preferred operators, respectively, and the last two as in the first iteration. In case a solution is found in the previous iteration, its cost is passed as a pruning bound to the next iteration.

In case of non-unit costs, a cost transformation is performed, adding a constant 1 to all costs. Further, the first iteration is performed twice, once with unit costs and once with the increased costs.

Agile Track

The configuration in the agile track mimics the first iteration of the configuration in the satisficing track as described above.

[1]These are basically the preferred operators of the full delete relaxation, an FF heuristic.

Bounded-Cost Track

The configuration in the bounded-cost track mimics the configuration in the agile track as described above. The only difference is that the cost bound is provided as an input.

Supported Features

As in the last competition, planners are required to support planning tasks with conditional effects. Following the strategy of *Mercury* planner, we have chosen here as well to compile the conditional effects away. This was done in a straighforward fashion, multiplying-out the actions (Nebel 2000) in the translation step. On one hand, this can lead to an exponential blow-up in the task representation size. On the other hand, it does not split up an operator application into a sequence of operator applications. Our decision was based on the success of the approach in *Mercury* planner, as well as the speculation that the latter option could potentially decrease red-black plan applicability, one of the main advantages of the current red-black heuristics.

References

Baier, J. A., and Botea, A. 2009. Improving planning performance using low-conflict relaxed plans. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 10–17. AAAI Press.

Bandres, W.; Bonet, B.; and Geffner, H. 2018. Planning with pixels in (almost) real time. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI 2018).* AAAI Press.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1):5–33.

Cai, D.; Hoffmann, J.; and Helmert, M. 2009. Enhancing the context-enhanced additive heuristic with precedence constraints. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 50–57. AAAI Press.

Domshlak, C.; Hoffmann, J.; and Katz, M. 2015. Red-black planning: A new systematic approach to partial delete relaxation. *Artificial Intelligence* 221:73–114.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2013. Symmetry breaking: Satisficing planning and landmark heuristics. In Borrajo, D.; Kambhampati, S.; Oddi, A.; and Fratini, S., eds., *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*, 298–302. AAAI Press.

Fox, M., and Long, D. 2001. Stan4: A hybrid planning strategy based on subproblem abstraction. 22(3):81–84.

Frances, G.; Ramırez, M.; Lipovetzky, N.; and Geffner, H. 2017. Purely declarative action representations are overrated: Classical planning with simulators. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017).*

Geffner, T., and Geffner, H. 2015. Width-based planning for general video-game playing. In *Proceedings of the Eleventh Artificial Intelligence and Interactive Digital Entertainment International Conference (AIIDE 2015).* AIIDE Press.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs in LPG. *Journal of Artificial Intelligence Research* 20:239–290.

Haslum, P. 2012. Incremental lower bounds for additive cost planning problems. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 74–82. AAAI Press.

Helmert, M., and Geffner, H. 2008. Unifying the causal graph and additive heuristics. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E., eds., *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*, 140–147. AAAI Press.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In Zilberstein, S.; Koehler, J.; and Koenig, S., eds., *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*, 161–170. AAAI Press.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Jinnai, Y., and Fukunaga, A. 2017. Learning to prune dominated action sequences in online black-box planning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017).* AAAI Press.

Katz, M., and Hoffmann, J. 2013. Red-black relaxed plan heuristics reloaded. In Helmert, M., and Röger, G., eds., *Proceedings of the Sixth Annual Symposium on Combinatorial Search (SoCS 2013)*, 105–113. AAAI Press.

Katz, M., and Hoffmann, J. 2014a. Mercury planner: Pushing the limits of partial delete relaxation. In *Eighth International Planning Competition (IPC-8): planner abstracts*, 43–47.

Katz, M., and Hoffmann, J. 2014b. Pushing the limits of partial delete relaxation: Red-black DAG heuristics. In *ICAPS 2014 Workshop on Heuristics and Search for Domain-independent Planning (HSDIP)*, 40–44.

Katz, M.; Lipovetzky, N.; Moshkovich, D.; and Tuisov, A. 2017. Adapting novelty to classical planning as heuristic search. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*, 172–180. AAAI Press.

Katz, M.; Hoffmann, J.; and Domshlak, C. 2013a. Red-black relaxed plan heuristics. In desJardins, M., and Littman, M. L., eds., *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI 2013)*, 489–495. AAAI Press.

Katz, M.; Hoffmann, J.; and Domshlak, C. 2013b. Who said we need to relax *all* variables? In Borrajo, D.; Kambhampati, S.; Oddi, A.; and Fratini, S., eds., *Proceedings of the*

Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013), 126–134. AAAI Press.

Keyder, E.; Hoffmann, J.; and Haslum, P. 2012. Semi-relaxed plan heuristics. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 128–136. AAAI Press.

Lipovetzky, N., and Geffner, H. 2012. Width and serialization of classical planning problems. In De Raedt, L.; Bessiere, C.; Dubois, D.; Doherty, P.; Frasconi, P.; Heintz, F.; and Lucas, P., eds., *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, 540–545. IOS Press.

Lipovetzky, N., and Geffner, H. 2014. Width-based algorithms for classical planning: New results. In Schaub, T.; Friedrich, G.; and O'Sullivan, B., eds., *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, 1059–1060. IOS Press.

Lipovetzky, N., and Geffner, H. 2017a. Best-first width search: Exploration and exploitation in classical planning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017).* AAAI Press.

Lipovetzky, N., and Geffner, H. 2017b. A polynomial planning algorithm that beats lama and ff. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017).* AAAI Press.

Lipovetzky, N.; Ramirez, M.; and Geffner, H. 2015. Classical planning with simulators: Results on the atari video games. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, 1610–1616.

Nebel, B. 2000. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research* 12:271–315.

Richter, S., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 273–280. AAAI Press.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39:127–177.

Shleyfman, A.; Tuisov, A.; and Domshlak, C. 2016. Blind search for atari-like online planning revisited. In Kambhampati, S., ed., *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, 3251–3257.

Valenzano, R.; Sturtevant, N. R.; Schaeffer, J.; and Xie, F. 2014. A comparison of knowledge-based GBFS enhancements and knowledge-free exploration. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 375–379. AAAI Press.

Xie, F.; Müller, M.; Holte, R. C.; and Imai, T. 2014. Type-based exploration with multiple search queues for satisficing planning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2014).* AAAI Press.

Delfi: Online Planner Selection for Cost-Optimal Planning

Michael Katz and **Shirin Sohrabi** and **Horst Samulowitz**
IBM Research
Yorktown Heights, NY, USA
michael.katz1@ibm.com, ssohrab@us.ibm.com, samulowitz@us.ibm.com

Silvan Sievers
University of Basel
Basel, Switzerland
silvan.sievers@unibas.ch

Abstract

Cost-optimal planning has not seen many successful approaches that work well across all domains. Some cost-optimal planners excel on some domains, while exhibiting less exciting performance on others. For a particular domain, however, there is often a cost-optimal planner that works extremely well. For that reason, portfolio-based techniques have recently become popular. These either decide offline on a particular resource allocation scheme for a given collection of planners or try to perform an online classification of a given planning task to select a planner to be applied to solving the task at hand.

Our planner *Delfi* is an online portfolio planner. In contrast to existing techniques, Delfi exploits deep learning techniques to learn a model that predicts which of the planners in the portfolio can solve a given planning task within the imposed time and memory bounds. Delfi uses graphical representations of a planning task which allows exploiting existing tools for image convolution. In this planner abstract, we describe the techniques used to create our portfolio planner.

Introduction

As planning is known to be computationally hard even for extremely conservative problem formalisms (Bylander 1994), no single planner should be expected to work well on all planning domains, or even on all tasks in a particular domain. As a result, research has not only focused on developing different planning techniques, such as improving search or heuristics, but also on exploiting multiple diverse approaches for solving planning tasks.

One such a approach is to aggregate multiple planners in a portfolio (Seipp et al. 2012; Vallati 2012; Cenamor, de la Rosa, and Fernández 2013; Seipp et al. 2015), which is what we do in this work. Such portfolios are often *sequential* and defined by two decisions: (i) which planner of the available to run next, and (ii) for how long to run it until the next planner is selected. Furthermore, the portfolio-based approaches can be partitioned in those that make those decisions ahead of time, called *offline* portfolios (Helmert et al. 2011; Núñez, Borrajo, and Linares López 2014; Seipp, Sievers, and Hutter 2014a; 2014b; 2014c) and those that make these decisions per given input task, called *online* portfolios (Cenamor, de la Rosa, and Fernández 2014).

Our planner, called *Delfi* for *DEap Learning of PortFoIIos*, is an online portfolio planner submitted to optimal classical track of the International Planning Competition (IPC) 2018. It consists of (a) a collection of cost-optimal planners based on Fast Downward (Helmert 2006), and (b) a module that, given a planning task, selects the planner from the collection for which the confidence that it solves the given planning task is highest. Once selected, the planner is run on the given task for the entire available time. In the remainder of this planner abstract, we describe both components in detail.

Collection of Cost-Optimal Planners

The large literature on classical planning results in an extensive pool of available planning systems that we could in principle all use. However, there are a few aspects that guided our decision to collect a rather small subset of specific planners. Firstly, the task of integrating the diverse planners within one system able to run them all in the same setting is a big (technical) challenge, and evaluating all of these planners for the training phase of learning the model would be extremely time-consuming. Secondly, portfolio planners always suffer from clearly identifying their components that are primarily responsible for the good performance of the portfolio planner.

Bearing in mind the first aspect, we restricted the pool of planners to those based on Fast Downward (Helmert 2006). This has the additional advantage that we also exploit how far a portfolio exclusively based on a single planning system fares. With respect to the second aspect, we excluded all recent (and state-of-the-art) planners that have not been evaluated in any previous competition. In particular, many of these planners are submitted independently to the IPC 2018. Furthermore, we mainly focused on planners with main components that we co-developed in order to primarily evaluate our own contributions.

These considerations result in a collection of 17 planners for our portfolio planner Delfi. With the exception of SymBA* (Torralba et al. 2014), the winner of the IPC 2014, included as-is in our collection of planners, all planners are based on a recent version of Fast Downward. These 16 planners use A* search (Hart, Nilsson, and Raphael 1968) and differ in the subsets of the following additional components they use. Please refer to the Appendix for the complete list of planner configurations of our collection, which is identical for both variants of Delfi.

- Pruning based on partial order reduction using strong stubborn sets (Wehrle and Helmert 2014). Delfi uses the implementation of strong stubborn sets available in Fast Downward, which is based on the original implementation of Alkhazraji et al. (2012) and Wehrle and Helmert (2012) that has also been used in Metis 2014 (Alkhazraji et al. 2014). However, the current implementation has been improved in terms of efficiency since its original development.[1] To support conditional effects, we extended the implementation in the same way as in Metis 2014. We also use the same mechanism that disables pruning after the first 1000 expansions if only 10% or fewer states have been pruned at this point. This component is part of all 16 planners.
- Pruning based on structural symmetries (Shleyfman et al. 2015) using DKS (Domshlak, Katz, and Shleyfman 2012) or orbit space search (OSS) (Domshlak, Katz, and Shleyfman 2015). We extended the original implementation of problem description graphs, also called symmetry graphs, which serve as basis for computing symmetries, to support conditional effects. Sievers et al. (2017) recently formally defined this extension in the context of structural symmetries of lifted representations. Out of the 16 planners, 8 use DKS search and the other 8 use OSS, without any other further difference except that merge-and-shrink configurations with OSS need to disable pruning of unreachable states to avoid incorrectly reporting pruned states as dead ends (cf. Sievers et al., 2015, for more details).
- Admissible heuristics:
  - The blind heuristic.
  - The LM-cut heuristic (Helmert and Domshlak 2009). To support conditional effects, we implemented a variant of the LM-cut heuristic that considers effect conditions in the same way as Metis 2014 (Alkhazraji et al. 2014) does. However, we refrain from choosing the regular LM-cut heuristic or the variant that supports conditional effects depending on the requirements of the input planning task, and instead always use the latter implementation that comes with a small overhead due to the need for different data structures.
  - The canonical pattern database (CPDB) heuristic with hillclimbing (HC) to compute pattern collections, also referred to as iPDB in the literature (Haslum et al. 2007). We add a time limit of 900s to the hillclimbing algorithm and denote the planner by HC-CPDB.
  - The zero-one cost partitioning pattern database (ZOPDB) heuristic with a genetic algorithm (GA) to compute pattern collections (Edelkamp 2006). We call the planner GA-ZOPDB.
  - Four variants of the merge-and-shrink heuristic (Dräger, Finkbeiner, and Podelski 2009; Helmert et al. 2014; Sievers 2017). Three of them use the state-of-the-art shrink strategy based on bisimulation (Nissim, Hoffmann, and Helmert 2011) with a size limit of 50000 states on transition systems, always allowing (perfect) shrinking, called B. The fourth variant uses a greedy variant of B, called G, not imposing any size limit on transition systems, and also always allowing shrinking. All configurations use full pruning (Sievers 2017), i.e., always prune both unreachable and irrelevant states, unless combined with OSS as discussed above, in which case pruning of unreachable states is disabled. We perform exact label reductions based on Θ-combinability (Sievers, Wehrle, and Helmert 2014) with a fixed point algorithm using a random order on factors.

    Finally, all variants use a time limit of 900s for computing the heuristic, which leads to computing so-called *partial* merge-and-shrink abstractions that do not cover all variables of the task whenever the time limit is hit. In these cases, we pick one of the remaining induced heuristics according to the following rule of thumb: we prefer the heuristic with the largest estimate for the initial state (rationale: better informed heuristic), breaking ties in favor of larger factors (rationale: more fine-grained abstraction), and choose a random heuristic among all remaining candidates of equal preference. For more details on this, we refer to the paper introducing partial abstractions (Sievers 2018b) and the separate competition entry called Fast Downward Merge-and-Shrink (Sievers 2018a) which uses the same merge-and-shrink configurations as our portfolio.

    The remaining difference between the four variants is the merge strategy, which finally results in the following merge-and-shrink configurations:
    - ∗ B-SCCdfp: the state-of-the-art merge strategy based on *strongly connected components* of the causal graph (Sievers, Wehrle, and Helmert 2016), which uses DFP (Sievers, Wehrle, and Helmert 2014) for internal merging.
    - ∗ B-MIASMdfp: the entirely precomputed merge strategy *maximum intermediate abstraction size minimizing* (Fan, Müller, and Holte 2014), which uses DFP as a fallback mechanism.
    - ∗ B-sbMIASM (previously also called DYN-MIASM): the merge strategy *score-based MIASM* (Sievers, Wehrle, and Helmert 2016), which is a simple variant of MIASM.
    - ∗ G-SCCdfp: as SCCdfp, but with the greedy variant of bisimulation-based shrinking.

  As mentioned above, each heuristic is used in two planners, once with OSS and once with DKS. For the two PDB-based heuristics that do not support conditional effects natively, we compile away conditional effects by multiplying out all operators, adding copies for each possible scenario of different subsets of satisfied effect conditions and operator preconditions.
- Postprocessing the SAS$^+$ representation obtained with the translator of Fast Downward (Helmert 2009) by using the implementation of h^2 mutex detection of Alcázar and Torralba (2015). This component is present in 14

[1]See http://issues.fast-downward.org/issue499 and http://issues.fast-downward.org/issue628.

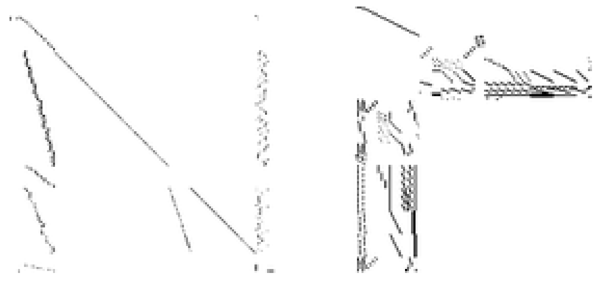

(a) Lifted representation (b) Grounded representation

Figure 1: Images constructed from lifted and grounded representations of task pfile01-001.pddl of BARMAN-OPT11.

out of 16 planners. The two planners that do not exploit h^2 mutexes use the merge-and-shrink configuration B-MIASMdfp (once with OSS, once with DKS) which heavily relies on remaining mutexes in the SAS$^+$ representation. Our preliminary experiments showed that using the postprocessing in this case significantly harmed the performance.

Online Planner Selection

The online planner selection of Delfi is based on a model that predicts for all planners of the portfolio whether they solve a given planning task within the fixed resource and time limits of the competition or not. To learn such a model, we created a collection of tasks to serve as training set, ran all planners in our collection on these tasks to find whether they solve the task or not, and used the resulting data to train a deep neural network. In what follows, we describe how we created the data as well as how we trained the model.

Data Creation

Our collection of tasks includes all benchmarks of the classical tracks of all IPCs as well as some domains from the learning tracks. We further include the domains BRIEFCASEWORLD, FERRY, and HANOI from the IPP benchmark collection (Köhler 1999), and the genome edit distance (GEDP) domain (Haslum 2011). We also use domains generated by the conformant-to-classical planning compilation (T0) (Palacios and Geffner 2009) and the finite-state controller synthesis compilation (FSC) (Bonet, Palacios, and Geffner 2009). In addition to existing tasks of these domains, we generated additional ones for some domains where generators were available. Please see the Appendix for a complete list of used domains. To filter out too hard tasks, we removed all tasks from the training set that were not solved by any of our planners.

Data Representation

To be able to take advantage of existing deep learning tools, we need to represent planning tasks in a way that can be consumed by these tools. In the context of solving other model-based problems, such as SAT and CSP, Loreggia et al. (2016) converted the textual description of input problems to a grayscale image by converting each character to a

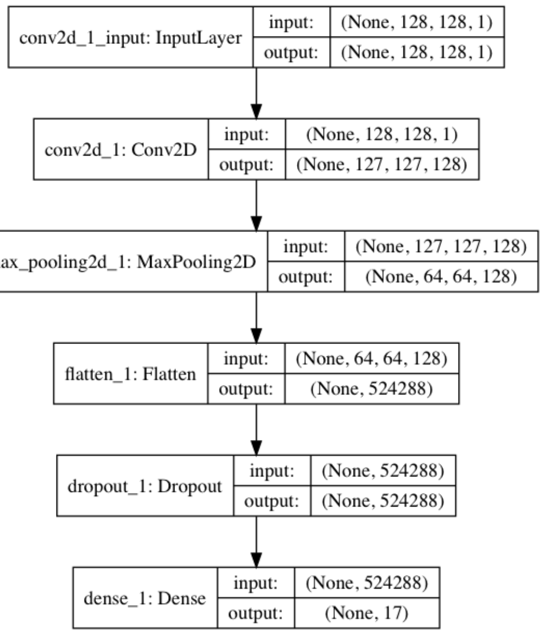

conv2d_1_input: InputLayer
input: (None, 128, 128, 1)
output: (None, 128, 128, 1)
conv2d_1: Conv2D
input: (None, 128, 128, 1)
output: (None, 127, 127, 128)
max_pooling2d_1: MaxPooling2D
input: (None, 127, 127, 128)
output: (None, 64, 64, 128)
flatten_1: Flatten
input: (None, 64, 64, 128)
output: (None, 524288)
dropout_1: Dropout
input: (None, 524288)
output: (None, 524288)
dense_1: Dense
input: (None, 524288)
output: (None, 17)

Figure 2: Visualization of the model graph structure.

pixel. Inspired by their ideas, we also chose to represent each task by a grayscale image of a constant size of $128 * 128$ pixels.

However, in contrast to Loreggia et al. (2016), we chose to abstract from the textual representation and decided to use a structural representation of planning tasks, namely the *abstract structure* (Sievers et al. 2017), which encodes the PDDL description of the task. We either directly convert this abstract structure to a graph by computing the abstract structure graph as described by Sievers et al. (2017), or we first ground the abstract structure (which corresponds to grounding the planning task) and the turn it into a graph. In the latter case, we technically do not use the abstract structure graph but the conceptually equivalent *problem description graph* (Shleyfman et al. 2015), which usually is used to compute symmetries of a ground task. Finally, we turn the graph into a grayscale image that represents the adjacency matrix of the graph and reduce the grayscale image to the desired constant size.

In some preliminary experiments, we experimented with both ways of creating an image for a planning task and decided to use both. Delfi 1 computes the image from the lifted representation of the task, and Delfi 2 from the grounded one. (This is the only difference of the two variants of our planner.) Figure 1 illustrates the different images we obtain for a task of the BARMAN domain.

Model Creation

Our tool of choice for both model creation and training is Keras (Chollet and others 2015) with Tensorflow as a backend. For our model, we employ a simple convolutional neural network (CNN) (LeCun, Bengio, and Hinton 2015) consisting of one convolutional layer, one pooling layer, one dropout layer, and one hidden layer. The main reason to choose a network with few parameters is to reduce the chances of overfitting given the comparably limited amount of data we created. Figure 2 shows the structure of the CNN.

We model planner performance by a binary feature that indicates whether the planner solves a task within the given time (1800 seconds) and memory (7744 MiB) limits or not. We also experimented with using the actual runtime, hence not predicting whether a planner solves a task or not, but rather predicting the runtime of the planner on the task. However, our preliminary tests indicated that the performance of the network when using the binary feature is comparable to when using the actual runtime. Our conjecture is that this is due to the relatively small amount of training data and due to the fact that the model learned with the binary feature bases the decision for a planner on the confidence that this planner solves the task, which means that it is likely to prefer faster planners. As a result, we decided to use the simpler representation in our model.

Consequently, we trained the CNN by optimizing for binary cross-entropy (Rubinstein 1997) so that each planner has a certain probability assigned to it that indicates how likely it is to solve a problem within the limits. Although our CNN is rather simple, it still features a range of model hyper-parameters, which we fine-tuned employing the approach by Diaz et al. (2017).[2] For both lifted and grounded representations, the hyper-parameter optimization found very similar parameters and thus we choose the same parameters in both cases, which are as follows. The convolutional layer filter size is 3, the pooling filter size is 1, and the dropout rate is 0.48. The CNN is optimized using Stochastic Gradient Decent with learning rate 0.1, decay 0.04, momentum 0.95, nesterov set to FALSE and a batch size of 52.

Post-IPC Analysis

In the following, we evaluate the performance of the two Delfi planners in the optimal classical track of the IPC 2018. While Delfi 2 finished 7 among all 16 submissions, Delfi 1 took the first place. To assess the contribution of the individual components of the portfolios, we ran them on all benchmarks under IPC conditions. We report both performance on the training set on which we learned prior to the IPC and performance on the new planning benchmarks from the competition, which we will refer to as the test set. In addition to the individual results, we include the competition results of Delfi 1 and 2[3] and of the oracle planner that takes the maximum over all planners on a per-instance base. Finally, as a baseline for portfolio performance on the test set, we also evaluate the uniform portfolio that runs each planner from the portfolio with a equal time share of the IPC limit of 30 minutes.

Table 1 shows aggregated coverage of the training set. For full domain-wise coverage on the training set, see Table 4 in the Appendix. We see that both variants of Delfi (coverage of 2282 and 2236) greatly improve over the best single planner in our portfolio, LM-cut (coverage of 1956). At the same time, the oracle portfolio solves 2350 tasks, which gives rise to the hope that the learned models of the Delfi planners are not overfitted too much on the training set.

Table 2 shows domain-wise coverage of the test set, using the same way as the IPC to compute aggregated results for the two domains where two formulations have been used (CALDERA and ORGANIC-SYNTHESIS), which is to take the task-wise maximum performance of each planner. Looking at the performance of the individual planners, we find that their coverage of different domains is diverse. In particular, Symba and HC-PDB (iPDB) are very complementary, but also LM-cut and some merge-and-shrink variants achieve best coverage in some domains.

Looking at the performance of the portfolios, we see that the uniform portfolio does not improve over the best individual planner, Symba, but even solves fewer tasks. While Delfi 1 is much stronger than the baseline uniform portfolio, the same is not true for Delfi 2, which even lacks behind the best individual planner Symba. We discuss the difference between Delfi 1 and 2 in more detail below. Finally, it is also worth pointing out that the oracle planner solves 18 tasks out of 240 more than Delfi 1, which leads us to conclude that the learned model of Delfi 1 generalized very well to the test set that the IPC 2018 benchmarks represent.

We now analyze the Delfi planners in more detail. Table 3 shows the number of times a component planner was chosen by Delfi: the first block shows the number of times a specific planner was chosen for each domain, and the second block shows the number of times each planner was chosen in total. We consider both variants of the two domains with two different formulations (CALDERA and ORGANIC-SYNTHESIS) individually rather than the combined domain, since it is not clear which planner to consider the chosen one if a different planner was chosen for both formulations.[4]

Delfi 1 often consistently chooses one or few planners in a given domain (with the exception of NURIKABE), which seems to be reasonable since we expect the same planner to be strong for different tasks across a given domain. Delfi 2, on the other hand, more frequently chooses a larger variety of planners in a domain.[5] While we cannot explain this significant difference yet, it is clearly due to the difference of the representation of planning tasks, i.e., the difference

[2]To evaluate our approach prior to the competition, we held back the IPC 2014 domains as a separate validation set. Only when learning the final model for the competition, we used the full benchmark set with the previously determined fine-tuned hyperparameters.

[3]We also re-ran both Delfi planners ourselves but decided to stick with the official competition results. The differences in coverage and planner selection per domain were marginal.

[4]However, for our set of planners, we note that we could restrict the analysis to the original formulation of CALDERA, in which OSS-LM-cut solves 13 tasks like Delfi 1, and to ORGANIC-SYNTHESIS-SPLIT, which dominates ORGANIC-SYNTHESIS for all planners.

[5]In 13 ORGANIC-SYNTHESIS tasks, the translator runs out of memory so that Delfi 2 cannot compute the problem description graph. In 3 ORGANIC-SYNTHESIS-SPLIT and 2 NURIBAKE tasks, the translator hits the time limit of 60s that we imposed for graph computation. In all of cases, Delfi 2 cannot use its model for planner selection but uses the fallback planner DSK-lmc.

	Portfolio Components																	Delfi		Orcl
	DKS								OSS									1	2	
	blind	lmc	P1	P2	M1	M2	M3	M4	blind	lmc	P1	P2	M1	M2	M3	M4	Sym			
C (3721)	1470	1956	1836	1602	1796	1645	1767	1615	1472	1948	1838	1606	1782	1643	1707	1568	1867	2282	2236	**2350**

Table 1: Coverage of the training set. Abbreviations: lmc: LM-cut; P1: HC-PDB; P2: GA-ZOPDB; M1: B-SCCdfp; M2: B-MIASMdfp; M3: B-sbMIASM; M4: G-SCCdfp; Sym: SymBA* 2014; Orcl: oracle portfolio over all component planners.

	Portfolio Components																	Unif	Delfi		Orcl
	DKS								OSS										1	2	
	blind	lmc	P1	P2	M1	M2	M3	M4	blind	lmc	P1	P2	M1	M2	M3	M4	Sym				
agricola (20)	5	0	7	5	6	0	10	5	6	0	7	6	6	0	6	6	13	7	12	11	**14**
caldera-comb (20)	12	13	**16**	13	12	0	12	12	12	13	**16**	13	12	0	12	12	12	13	13	11	**16**
data-network (20)	6	12	11	9	10	10	9	3	6	12	11	9	10	10	9	3	**13**	**13**	**13**	**13**	**13**
nurikabe (20)	10	**12**	**12**	11	11	**12**	**12**	11	10	**12**	**12**	11	11	11	11	11	11	10	**12**	11	**12**
org-syn-comb (20)	**14**	**14**	13	**14**	13	7	13	13	**14**	**14**	13	**14**	13	7	13	13	**14**	13	13	13	**14**
petri-net-al (20)	2	9	0	2	2	0	2	2	2	9	0	2	2	0	2	2	**20**	15	**20**	9	**20**
settlers (20)	8	**9**	0	0	**9**	**9**	8	**9**	8	**9**	0	0	**9**	**9**	8	**9**	**9**	6	**9**	8	**9**
snake (20)	11	7	**14**	11	11	7	11	10	11	7	**14**	11	11	6	11	10	4	10	11	7	**14**
spider (20)	11	11	**14**	11	11	0	11	3	11	11	**14**	11	11	0	11	3	7	13	11	7	**14**
termes (20)	7	6	12	10	10	10	11	6	7	6	12	10	10	10	11	6	**18**	13	12	15	**18**
Sum (200)	86	93	99	86	95	55	99	74	87	93	99	87	95	53	94	75	121	113	126	105	**144**

Table 2: Coverage of the test set (IPC 2018 benchmarks). Abbreviations: lmc: LM-cut; P1: HC-PDB; P2: GA-ZOPDB; M1: B-SCCdfp; M2: B-MIASMdfp; M3: B-sbMIASM; M4: G-SCCdfp; Sym: SymBA* 2014; Unif: uniform portfolio over all component planners; Orcl: oracle portfolio over all component planners.

	Delfi 1	Delfi 2
agricola (20)	Sym (20)	Sym (19), OSS-P1 (1)
caldera (20)	OSS-lmc (20)	Sym (10), DKS-M2 (3), DKS-M1 (2), DKS-lmc (2) DKS-P1 (1), DKS-M3 (1), OSS-lmc (1)
caldera-split (20)	Sym (7), OSS-lmc (8), DKS-lmc (5)	Sym (6), OSS-lmc (6), DKS-lmc (4), DKS-M4 (2), OSS-P1 (1), DKS-M1 (1)
data-network (20)	Sym (17), OSS-lmc (3)	Sym (16), OSS-M2 (3), OSS-P1 (1)
nurikabe (20)	Sym (6), DKS-P1 (6), DKS-blind (2), OSS-P1 (2), DKS-lmc (2), DKS-M2 (1), OSS-lmc (1)	Sym(12), DKS-lmc (3), OSS-M2 (2), DKS-M2 (1), time out (2)
organic-synthesis (20)	Sym (18), OSS-lmc (2)	OSS-P1 (3), Sym(2), DKS-P1 (1), OSS-M2 (1), memory out (13)
organic-synthesis-split (20)	Sym (20)	Sym (14), OSS-lmc (2), DKS-P1 (1), time out (3)
petri-net-alignment (20)	Sym (20)	DKS-lmc (10), OSS-lmc (9), Sym(1)
settlers (20)	OSS-lmc (20)	Sym (20)
snake (20)	OSS-M1 (9), OSS-P1 (7), DKS-P1 (2), Sym (2)	DKS-lmc (12), Sym (8)
spider (20)	DKS-M3 (6), DKS-M1 (6), OSS-M1 (5), DKS-lmc (3)	Sym (16), DKS-lmc (3), OSS-lmc (1)
termes (20)	DKS-M2 (20)	Sym (10), DKS-lmc (6), OSS-M2 (4)
Sum	240	222 (13 memory out, 5 time out)
Symba	110	134
OSS-lmc	54	19
DKS-lmc	10	40
OSS-P1	9	6
DKS-P1	8	3
OSS-M1	14	0
DKS-M1	6	3
DKS-M2	21	4
OSS-M2	0	3
DKS-M3	6	1
DKS-blind	2	0

Table 3: Top: domain-wise number of tasks a planner is selected by our portfolios. Bottom: for each planner, number of times it is selected by our portfolios in total.

between the abstract structure graph (Delfi 1) and the problem description graph (Delfi 2). One important distinguishing feature of this difference is that the problem description graph represents the grounded task (SAS$^+$), which is a specific representation that depends on the used grounding and invariant synthesis algorithms, while the abstract structure graph represents the lifted representation (PDDL) of the task.

In the benchmark set, there is an observable difference that may explain the different choices to some extent: the IPC 2018 domains exhibit much more conditional effects than the domains of our training set. When we performed initial experiments on the reduced training set (excluding the IPC 2014 domains as a validation set), we observed that using the problem description graph (Delfi 2) resulted in a stronger performance than using the abstract structure graph (Delfi 1) due to better choices of suitable planners. The same is not true anymore when looking at the full training set performance and the test set performance reported here. In future work, we plan to further investigate the differences in the representation of planning tasks and their impact on planner selection.

To assess how well Delfi selects planners for a given task, we also investigate which planners are required to achieve oracle performance. Surprisingly, we found two set covers of only size 3 that cover all tasks solved by any planner: the first consists of Symba, DKS-M3, and DKS-P1, and the second one of Symba, DKS-M3, and OSS-P1. In particular, it is enough to consider Symba, one variant of PDB-based planners, and one variant of merge-and-shrink-based planners (for NURIBAKE), and there is no need for LM-cut or any of the other heuristics at all. While both variants of Delfi frequently choose Symba and sometimes choose PDB-based and merge-and-shrink-based planners, they choose LM-cut the second most of times, even though this would not be required. A possible reason is that LM-cut performs much better on the training set than PDB-based and merge-and-shrink-based planners and hence is more likely to be chosen also on the test set.

Conclusions

In this planner abstract, we described the Delfi planners that participated in the optimal classical track of the IPC 2018. The Delfi planners are online portfolios that select a component planner deemed suitable for the given task based on a model learned with deep learning techniques. In particular, to represent planning tasks, we used two graphical representations for planning tasks, turned them into images and used a convolutional neural network to learn a model that predicts whether a planner solves a given task or not. The performance of Delfi 1, taking the first place in the competition, showed that the learned model generalized well to the new benchmarks used in the competition.

In future work, we would like to greatly extend the set of component planners of our portfolio to evaluate how far the performance of Delfi can be pushed if using many available state-of-the-art planners. We also plan to investigate alternative learning techniques that operate directly on the graph representations rather than relying on images created from these graphs.

Acknowledgments

Since any portfolio crucially depends on the quality of its component planners, we want to thank all contributors of Fast Downward, the authors of Symba 2014, and the authors of the h^2-based preprocessor.

References

Alcázar, V., and Torralba, Á. 2015. A reminder about the importance of computing and exploiting invariants in planning. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 2–6. AAAI Press.

Alkhazraji, Y.; Wehrle, M.; Mattmüller, R.; and Helmert, M. 2012. A stubborn set algorithm for optimal planning. In De Raedt, L.; Bessiere, C.; Dubois, D.; Doherty, P.; Frasconi, P.; Heintz, F.; and Lucas, P., eds., *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, 891–892. IOS Press.

Alkhazraji, Y.; Katz, M.; Mattmüller, R.; Pommerening, F.; Shleyfman, A.; and Wehrle, M. 2014. Metis: Arming Fast Downward with pruning and incremental computation. In *Eighth International Planning Competition (IPC-8): planner abstracts*, 88–92.

Bonet, B.; Palacios, H.; and Geffner, H. 2009. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 34–41. AAAI Press.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence* 69(1–2):165–204.

Cenamor, I.; de la Rosa, T.; and Fernández, F. 2013. Learning predictive models to configure planning portfolios. In *ICAPS 2013 Workshop on Planning and Learning*, 14–22.

Cenamor, I.; de la Rosa, T.; and Fernández, F. 2014. IBaCoP and IBaCoPB planner. In *Eighth International Planning Competition (IPC-8): planner abstracts*, 35–38.

Chollet, F., et al. 2015. `https://keras.io`.

Diaz, G. I.; Fokoue-Nkoutche, A.; Nannicini, G.; and Samulowitz, H. 2017. An effective algorithm for hyperparameter optimization of neural networks. *IBM Journal of Research and Development* 61(4).

Domshlak, C.; Katz, M.; and Shleyfman, A. 2012. Enhanced symmetry breaking in cost-optimal planning as forward search. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 343–347. AAAI Press.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2015. Symmetry breaking in deterministic planning as forward search:

Orbit space search algorithm. Technical Report IS/IE-2015-03, Technion.

Dräger, K.; Finkbeiner, B.; and Podelski, A. 2009. Directed model checking with distance-preserving abstractions. *International Journal on Software Tools for Technology Transfer* 11(1):27–37.

Edelkamp, S. 2006. Automated creation of pattern database search heuristics. In Edelkamp, S., and Lomuscio, A., eds., *Proceedings of the 4th Workshop on Model Checking and Artificial Intelligence (MoChArt 2006)*, 35–50.

Fan, G.; Müller, M.; and Holte, R. 2014. Non-linear merging strategies for merge-and-shrink based on variable interactions. In Edelkamp, S., and Barták, R., eds., *Proceedings of the Seventh Annual Symposium on Combinatorial Search (SoCS 2014)*, 53–61. AAAI Press.

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

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007)*, 1007–1012. AAAI Press.

Haslum, P. 2011. Computing genome edit distances using domain-independent planning. In *ICAPS 2011 Scheduling and Planning Applications woRKshop*, 45–51.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 162–169. AAAI Press.

Helmert, M.; Röger, G.; Seipp, J.; Karpas, E.; Hoffmann, J.; Keyder, E.; Nissim, R.; Richter, S.; and Westphal, M. 2011. Fast Downward Stone Soup. In *IPC 2011 planner abstracts*, 38–45.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM* 61(3):16:1–63.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence* 173:503–535.

Köhler, J. 1999. Handling of conditional effects and negative goals in IPP. Technical Report 128, University of Freiburg, Department of Computer Science.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *Nature* 521(7553):436–444.

Loreggia, A.; Malitsky, Y.; Samulowitz, H.; and Saraswat, V. A. 2016. Deep learning for algorithm portfolios. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI 2016)*, 1280–1286. AAAI Press.

Nissim, R.; Hoffmann, J.; and Helmert, M. 2011. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In Walsh, T., ed., *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, 1983–1990. AAAI Press.

Núñez, S.; Borrajo, D.; and Linares López, C. 2014. Miplan and dpmplan. In *Eighth International Planning Competition (IPC-8): planner abstracts*.

Palacios, H., and Geffner, H. 2009. Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence Research* 35:623–675.

Rubinstein, R. Y. 1997. Optimization of computer simulation models with rare events. *European Journal of Operational Research* 99(1):89–112.

Seipp, J.; Braun, M.; Garimort, J.; and Helmert, M. 2012. Learning portfolios of automatically tuned planners. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 368–372. AAAI Press.

Seipp, J.; Sievers, S.; Helmert, M.; and Hutter, F. 2015. Automatic configuration of sequential planning portfolios. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3364–3370. AAAI Press.

Seipp, J.; Sievers, S.; and Hutter, F. 2014a. Fast Downward Cedalion. In *Eighth International Planning Competition (IPC-8): planner abstracts*, 17–27.

Seipp, J.; Sievers, S.; and Hutter, F. 2014b. Fast Downward Cedalion. In *Eighth International Planning Competition (IPC-8) Planning and Learning Part: planner abstracts*.

Seipp, J.; Sievers, S.; and Hutter, F. 2014c. Fast Downward SMAC. In *Eighth International Planning Competition (IPC-8) Planning and Learning Part: planner abstracts*.

Shleyfman, A.; Katz, M.; Helmert, M.; Sievers, S.; and Wehrle, M. 2015. Heuristics and symmetries in classical planning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3371–3377. AAAI Press.

Sievers, S.; Wehrle, M.; Helmert, M.; and Katz, M. 2015. An empirical case study on symmetry handling in cost-optimal planning as heuristic search. In Hölldobler, S.; Krötzsch, M.; Peñaloza-Nyssen, R.; and Rudolph, S., eds., *Proceedings of the 38th Annual German Conference on Artificial Intelligence (KI 2015)*, volume 9324 of *Lecture Notes in Artificial Intelligence*, 151–165. Springer-Verlag.

Sievers, S.; Röger, G.; Wehrle, M.; and Katz, M. 2017. Structural symmetries of the lifted representation of classical planning tasks. In *ICAPS 2017 Workshop on Heuristics and Search for Domain-independent Planning (HSDIP)*, 67–74.

Sievers, S.; Wehrle, M.; and Helmert, M. 2014. Generalized label reduction for merge-and-shrink heuristics. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2014)*, 2358–2366. AAAI Press.

Sievers, S.; Wehrle, M.; and Helmert, M. 2016. An analysis of merge strategies for merge-and-shrink heuristics. In

Coles, A.; Coles, A.; Edelkamp, S.; Magazzeni, D.; and Sanner, S., eds., *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016)*, 294–298. AAAI Press.

Sievers, S. 2017. *Merge-and-shrink Abstractions for Classical Planning: Theory, Strategies, and Implementation*. Ph.D. Dissertation, University of Basel.

Sievers, S. 2018a. Fast Downward merge-and-shrink. In *Ninth International Planning Competition (IPC-9): planner abstracts*.

Sievers, S. 2018b. Merge-and-shrink heuristics for classical planning: Efficient implementation and partial abstractions. In *Proceedings of the 11th Annual Symposium on Combinatorial Search (SoCS 2018)*, 90–98. AAAI Press.

Torralba, Á.; Alcázar, V.; Borrajo, D.; Kissmann, P.; and Edelkamp, S. 2014. SymBA*: A symbolic bidirectional A* planner. In *Eighth International Planning Competition (IPC-8): planner abstracts*, 105–109.

Vallati, M. 2012. A guide to portfolio-based planning. In Sombattheera, C.; Loi, N. K.; Wankar, R.; and Quan, T. T., eds., *Proceedings of the 6th International Workshop on Multi-disciplinary Trends in Artificial Intelligence (MIWAI 2012)*, volume 7694, 57–68. Springer.

Wehrle, M., and Helmert, M. 2012. About partial order reduction in planning and computer aided verification. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 297–305. AAAI Press.

Wehrle, M., and Helmert, M. 2014. Efficient stubborn sets: Generalized algorithms and selection strategies. In Chien, S.; Fern, A.; Ruml, W.; and Do, M., eds., *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 323–331. AAAI Press.

Appendix

Collection of Planner Configurations

The following are the configurations for the 16 Fast Downward based planners.

1.
```
--symmetries 'sym=structural_symmetries(search_symmetries=dks)'
--search 'astar(blind,symmetries=sym,
    pruning=stubborn_sets_simple(minimum_pruning_ratio=0.01),num_por_probes=1000)'
```

2.
```
--symmetries 'sym=structural_symmetries(search_symmetries=dks)'
--search 'astar(celmcut,symmetries=sym,
    pruning=stubborn_sets_simple(minimum_pruning_ratio=0.01),num_por_probes=1000)'
```

3.
```
--symmetries 'sym=structural_symmetries(search_symmetries=dks)'
--search 'astar(
    merge_and_shrink(shrink_strategy=shrink_bisimulation(greedy=false),
        merge_strategy=merge_sccs(order_of_sccs=topological,merge_selector=score_based_filtering(scoring_functions=
        [goal_relevance,dfp,total_order(atomic_before_product=false,atomic_ts_order=reverse_level,product_ts_order=
        new_to_old)])),label_reduction=exact(before_shrinking=true,before_merging=false),max_states=50000,
        threshold_before_merge=1,max_time=900),
    symmetries=sym,pruning=stubborn_sets_simple(minimum_pruning_ratio=0.01),num_por_probes=1000)'
```

4.
```
--symmetries 'sym=structural_symmetries(search_symmetries=dks)'
--search 'astar(
    merge_and_shrink(shrink_strategy=shrink_bisimulation(greedy=false),
        merge_strategy=merge_stateless(merge_selector=score_based_filtering(scoring_functions=[sf_miasm(
        shrink_strategy=shrink_bisimulation,max_states=50000),total_order(atomic_before_product=true,
        atomic_ts_order=reverse_level,product_ts_order=old_to_new)])),label_reduction=exact(before_shrinking=true,
        before_merging=false),max_states=50000,threshold_before_merge=1,max_time=900),
    symmetries=sym,pruning=stubborn_sets_simple(minimum_pruning_ratio=0.01),num_por_probes=1000)'
```

5.
```
--symmetries 'sym=structural_symmetries(search_symmetries=dks)'
--search 'astar(
    merge_and_shrink(shrink_strategy=shrink_bisimulation(greedy=false),merge_strategy=merge_precomputed(
        merge_tree=miasm(abstraction=miasm_merge_and_shrink(),fallback_merge_selector=score_based_filtering(
        scoring_functions=[goal_relevance,dfp,total_order(atomic_ts_order=reverse_level,product_ts_order=
        new_to_old,atomic_before_product=false)]))),label_reduction=exact(before_shrinking=true,before_merging=false),
        max_states=50000,threshold_before_merge=1,max_time=900),
    symmetries=sym,pruning=stubborn_sets_simple(minimum_pruning_ratio=0.01),num_por_probes=1000)'
```

6.
```
--symmetries 'sym=structural_symmetries(search_symmetries=dks)'
--search 'astar(
    merge_and_shrink(shrink_strategy=shrink_bisimulation(greedy=true),merge_strategy=merge_sccs(order_of_sccs=
        topological, merge_selector=score_based_filtering(scoring_functions=[goal_relevance,dfp,
        total_order(atomic_before_product=false, atomic_ts_order=level,product_ts_order=random)])),
        label_reduction=exact(before_shrinking=true,before_merging=false),
        max_states=infinity,threshold_before_merge=1,max_time=900),
    symmetries=sym,pruning=stubborn_sets_simple(minimum_pruning_ratio=0.01),num_por_probes=1000)'
```

7.
```
--symmetries 'sym=structural_symmetries(search_symmetries=dks)'
--search 'astar(cpdbs(patterns=hillclimbing(max_time=900),transform=multiply_out_conditional_effects),
    symmetries=sym,pruning=stubborn_sets_simple(minimum_pruning_ratio=0.01),num_por_probes=1000)'
```

8.
```
--symmetries 'sym=structural_symmetries(search_symmetries=dks)'
--search 'astar(zopdbs(patterns=genetic(pdb_max_size=50000,num_collections=5,num_episodes=30,
    mutation_probability=0.01), transform=multiply_out_conditional_effects),symmetries=sym,
    pruning=stubborn_sets_simple(minimum_pruning_ratio=0.01), num_por_probes=1000)'
```

9.
```
--symmetries 'sym=structural_symmetries(search_symmetries=oss)'
--search 'astar(blind,symmetries=sym,pruning=stubborn_sets_simple(minimum_pruning_ratio=0.01),num_por_probes=1000)'
```

10.
```
--symmetries 'sym=structural_symmetries(search_symmetries=oss)'
--search 'astar(celmcut,symmetries=sym,pruning=stubborn_sets_simple(minimum_pruning_ratio=0.01),num_por_probes=1000)'
```

11.
```
--symmetries 'sym=structural_symmetries(search_symmetries=oss)'
--search 'astar(
    merge_and_shrink(shrink_strategy=shrink_bisimulation(greedy=false),merge_strategy=merge_sccs(order_of_sccs=
        topological, merge_selector=score_based_filtering(scoring_functions=[goal_relevance,dfp,
        total_order(atomic_before_product=false, atomic_ts_order=reverse_level,product_ts_order=new_to_old)])),
        label_reduction=exact(before_shrinking=true, before_merging=false),max_states=50000,threshold_before_merge=1,
        max_time=900,prune_unreachable_states=false),
    symmetries=sym,pruning=stubborn_sets_simple(minimum_pruning_ratio=0.01),num_por_probes=1000)'
```

12.
    ```
--symmetries 'sym=structural_symmetries(search_symmetries=oss)'
--search 'astar(
    merge_and_shrink(shrink_strategy=shrink_bisimulation(greedy=false),merge_strategy=merge_stateless(merge_selector=
        score_based_filtering(scoring_functions=[sf_miasm(shrink_strategy=shrink_bisimulation,max_states=50000),
        total_order(atomic_before_product=true,atomic_ts_order=reverse_level,product_ts_order=old_to_new)])),
        label_reduction=exact(before_shrinking=true,before_merging=false),
        max_states=50000,threshold_before_merge=1,max_time=900,prune_unreachable_states=false),
    symmetries=sym,pruning=stubborn_sets_simple(minimum_pruning_ratio=0.01),num_por_probes=1000)'
```

13.
    ```
--symmetries 'sym=structural_symmetries(search_symmetries=oss)'
--search 'astar(
    merge_and_shrink(shrink_strategy=shrink_bisimulation(greedy=false),merge_strategy=merge_precomputed(merge_tree=
        miasm(abstraction=miasm_merge_and_shrink(),fallback_merge_selector=score_based_filtering(scoring_functions=
        [goal_relevance,dfp,total_order(atomic_ts_order=reverse_level,product_ts_order=new_to_old,
        atomic_before_product=false)]))),label_reduction=exact(before_shrinking=true,before_merging=false),
        max_states=50000,threshold_before_merge=1,max_time=900,prune_unreachable_states=false),
    symmetries=sym,pruning=stubborn_sets_simple(minimum_pruning_ratio=0.01),num_por_probes=1000)'
```

14.
    ```
--symmetries 'sym=structural_symmetries(search_symmetries=oss)'
--search 'astar(
    merge_and_shrink(shrink_strategy=shrink_bisimulation(greedy=true),merge_strategy=merge_sccs(order_of_sccs=
        topological, merge_selector=score_based_filtering(scoring_functions=[goal_relevance,dfp,
        total_order(atomic_before_product=false,atomic_ts_order=level,product_ts_order=random)])),
        label_reduction=exact(before_shrinking=true, efore_merging=false),max_states=infinity,
        threshold_before_merge=1,max_time=900,prune_unreachable_states=false),
    symmetries=sym,pruning=stubborn_sets_simple(minimum_pruning_ratio=0.01),num_por_probes=1000)'
```

15.
    ```
--symmetries 'sym=structural_symmetries(search_symmetries=oss)'
--search 'astar(cpdbs(patterns=hillclimbing(max_time=900),transform=multiply_out_conditional_effects),
    symmetries=sym,pruning=stubborn_sets_simple(minimum_pruning_ratio=0.01),num_por_probes=1000)'
```

16.
    ```
--symmetries 'sym=structural_symmetries(search_symmetries=oss)'
--search 'astar(zopdbs(patterns=genetic(pdb_max_size=50000,num_collections=5,num_episodes=30,
    mutation_probability=0.01), transform=multiply_out_conditional_effects),symmetries=sym,
    pruning=stubborn_sets_simple(minimum_pruning_ratio=0.01), num_por_probes=1000)'
```

Domains of the Training Set

The following lists contain all benchmark domains we used for training, named as in the repository under `https://bitbucket.org/SilvanS/ipc2018-benchmarks`. Domains with the prefix ss are either additional domains not contained in the original repository under `https://bitbucket.org/aibasel/downward-benchmarks` or copies of already present domains containing additional tasks that we generated.

STRIPS domains:

```
['airport', 'barman-opt11-strips', 'barman-opt14-strips', 'blocks',
'childsnack-opt14-strips', 'depot', 'driverlog', 'elevators-opt08-strips',
'elevators-opt11-strips', 'floortile-opt11-strips',
'floortile-opt14-strips', 'freecell', 'ged-opt14-strips', 'grid',
'gripper', 'hiking-opt14-strips', 'logistics00', 'logistics98', 'miconic',
'movie', 'mprime', 'mystery', 'nomystery-opt11-strips',
'openstacks-opt08-strips', 'openstacks-opt11-strips',
'openstacks-opt14-strips', 'openstacks-strips', 'parcprinter-08-strips',
'parcprinter-opt11-strips', 'parking-opt11-strips', 'parking-opt14-strips',
'pathways-noneg', 'pegsol-08-strips', 'pegsol-opt11-strips',
'pipesworld-notankage', 'pipesworld-tankage', 'psr-small', 'rovers',
'satellite', 'scanalyzer-08-strips', 'scanalyzer-opt11-strips',
'sokoban-opt08-strips', 'sokoban-opt11-strips', 'storage',
'tetris-opt14-strips', 'tidybot-opt11-strips', 'tidybot-opt14-strips',
'tpp', 'transport-opt08-strips', 'transport-opt11-strips',
'transport-opt14-strips', 'trucks-strips', 'visitall-opt11-strips',
'visitall-opt14-strips', 'woodworking-opt08-strips',
'woodworking-opt11-strips', 'zenotravel', 'ss_barman', 'ss_ferry',
'ss_goldminer', 'ss_grid', 'ss_hanoi', 'ss_hiking', 'ss_npuzzle',
'ss_spanner',]
```

Domains with conditional effects:

```
['briefcaseworld', 'cavediving-14-adl', 'citycar-opt14-adl', 'fsc-blocks',
'fsc-grid-a1', 'fsc-grid-a2', 'fsc-grid-r', 'fsc-hall', 'fsc-visualmarker',
'gedp-ds2ndp', 'miconic-simpleadl', 't0-adder', 't0-coins', 't0-comm',
't0-grid-dispose', 't0-grid-push', 't0-grid-trash', 't0-sortnet',
't0-sortnet-alt', 't0-uts', 'ss_briefcaseworld', 'ss_cavediving',
'ss_citycar', 'ss_maintenance', 'ss_maintenance_large', 'ss_schedule',]
```

Domain-wise Trainingset Performance

	Portfolio Components																	Delfi		Oracle
	DKS								OSS									1	2	
	blind	lmc	P1	P2	M1	M2	M3	M4	blind	lmc	P1	P2	M1	M2	M3	M4	Sym			
airport (50)	27	29	31	28	27	2	27	27	27	29	31	28	27	2	27	27	27	27	29	**32**
barman-opt11-strips (20)	8	8	8	8	8	**12**	8	8	8	8	8	8	8	**12**	8	8	10	10	8	**12**
barman-opt14-strips (14)	3	3	3	3	3	**6**	3	3	3	3	3	3	3	**6**	3	3	**6**	**6**	3	**6**
blocks (35)	21	28	28	25	28	26	26	28	21	28	28	25	28	26	26	28	**32**	**32**	**32**	**32**
briefcaseworld (50)	8	**9**	8	8	**9**	8	8	**9**	8	**9**	8	8	8	8	8	8	8	**9**	8	**9**
cavediving-14-adl (20)	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
childsnack-opt14-strips (20)	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	4	**6**	**6**	**6**
citycar-opt14-adl (20)	**18**	**18**	**18**	**18**	**18**	10	**18**	**18**	**18**	**18**	**18**	**18**	17	10	17	**18**	**18**	**18**	**18**	**18**
depot (22)	6	9	**12**	8	9	**12**	11	10	6	9	**12**	8	9	**12**	9	10	7	**12**	9	**12**
driverlog (20)	8	14	13	13	13	14	13	14	7	14	14	13	13	14	13	14	14	**15**	14	**15**
elevators-opt08-strips (30)	17	22	22	20	19	19	19	12	17	22	22	20	19	19	19	5	**25**	**25**	**25**	**25**
elevators-opt11-strips (20)	15	18	18	17	16	16	16	10	15	18	18	17	16	16	16	3	**19**	**19**	**19**	**19**
floortile-opt11-strips (20)	8	**14**	8	8	9	10	12	8	8	**14**	8	8	9	10	10	8	**14**	12	**14**	**14**
floortile-opt14-strips (20)	8	**20**	8	8	9	11	14	8	8	**20**	8	8	9	11	11	8	**20**	17	**20**	**20**
freecell (80)	20	15	21	20	21	22	22	20	20	15	21	20	21	22	22	20	25	**27**	21	**27**
fsc-blocks (14)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
fsc-grid-a1 (16)	2	2	0	1	2	2	2	2	2	2	0	1	2	2	2	2	**3**	**3**	2	**3**
fsc-grid-a2 (2)	**1**	**1**	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
fsc-grid-r (16)	**15**	**15**	0	0	**15**	**15**	**15**	**15**	**15**	**15**	0	0	**15**	**15**	**15**	**15**	1	1	6	**15**
fsc-hall (2)	**1**	**1**	0	0	**1**	**1**	**1**	**1**	**1**	**1**	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
fsc-visualmarker (7)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
ged-opt14-strips (20)	15	15	19	19	19	19	19	5	15	15	19	18	18	15	15	5	19	19	**20**	19
gedp-ds2ndp (24)	18	18	4	4	**22**	18	**22**	**22**	18	18	4	4	18	14	18	18	18	**22**	16	**22**
grid (5)	1	2	**3**	2	2	**3**	2	2	1	2	**3**	2	2	**3**	**3**	2	2	**3**	2	**3**
gripper (20)	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
hiking-opt14-strips (20)	17	13	19	19	19	19	19	18	17	13	19	19	19	19	19	17	19	18	19	**20**
logistics00 (28)	12	20	**21**	**21**	20	20	20	19	12	20	20	20	20	20	20	19	19	**21**	**21**	**21**
logistics98 (35)	2	6	5	5	5	5	5	5	2	**7**	5	6	5	5	5	5	6	**7**	6	**7**
miconic (150)	56	142	61	61	84	78	61	57	56	142	61	61	85	78	61	56	109	143	142	**144**
miconic-simpleadl (150)	80	144	81	82	87	87	86	80	80	144	80	81	87	87	86	80	149	149	144	**150**
movie (30)	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
mprime (35)	18	23	24	23	23	21	23	23	20	22	25	23	24	21	23	24	24	25	24	**26**
mystery (30)	15	18	17	17	17	16	16	17	15	18	18	17	17	16	17	17	15	18	15	**19**
nomystery-opt11-strips (20)	9	16	**20**	**20**	**20**	**20**	**20**	14	9	16	**20**	**20**	**20**	**20**	**20**	14	16	18	16	**20**
openstacks-opt08-strips (30)	24	23	24	24	24	24	23	9	24	23	24	24	24	24	23	9	**30**	**30**	**30**	**30**
openstacks-opt11-strips (20)	18	18	18	18	18	18	18	4	18	18	18	18	18	18	18	4	**20**	**20**	**20**	**20**
openstacks-opt14-strips (20)	5	3	5	5	5	5	3	0	5	3	5	5	5	5	3	0	**20**	**20**	**20**	**20**
openstacks-strips (30)	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	**20**	19	7	**20**
parcprinter-08-strips (30)	**30**	**30**	29	**30**	27	9	26	**30**	**30**	**30**	29	**30**	22	9	24	**30**	22	28	**30**	**30**
parcprinter-opt11-strips (20)	**20**	**20**	**20**	**20**	**20**	5	19	**20**	**20**	**20**	**20**	**20**	17	5	17	**20**	17	**20**	**20**	**20**
parking-opt11-strips (20)	0	3	**8**	1	2	1	1	**8**	1	3	**8**	1	2	4	1	**8**	1	**8**	**8**	**8**
parking-opt14-strips (20)	0	4	7	0	5	4	1	7	0	3	**8**	0	4	4	3	**8**	3	**8**	**8**	**8**
pathways-noneg (30)	4	**5**	**5**	**5**	**5**	**5**	**5**	**5**	4	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
pegsol-08-strips (30)	28	29	**30**	28	**30**	29	29	28	28	29	**30**	29	**30**	29	29	20	29	**30**	**30**	**30**
pegsol-opt11-strips (20)	18	19	**20**	18	**20**	19	19	18	18	19	**20**	19	**20**	19	19	10	19	**20**	**20**	**20**
pipesworld-notankage (50)	20	21	**25**	23	21	4	20	20	20	21	**25**	24	21	4	20	20	15	**25**	22	**25**
pipesworld-tankage (50)	17	17	20	17	17	16	17	21	17	16	21	17	17	17	19	20	16	21	17	**22**
psr-small (50)	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**
rovers (40)	7	12	11	10	9	11	11	9	7	12	11	10	9	11	10	9	**14**	**14**	13	**14**
satellite (36)	7	**14**	9	9	9	9	9	9	7	**14**	9	9	9	9	9	9	10	**14**	11	**14**

scanalyzer-08-strips (30)	15	17	17	15	16	18	17	6	15	17	18	16	**19**	**19**	17	6	12	**19**	15	**19**
scanalyzer-opt11-strips (20)	11	14	13	11	12	14	13	3	11	14	14	12	**15**	**15**	13	3	9	**15**	12	**15**
sokoban-opt08-strips (30)	28	**30**	**30**	28	**30**	26	**30**	28	28	**30**	**30**	28	**30**	26	28	28	28	**30**	29	**30**
sokoban-opt11-strips (20)	**20**	**20**	**20**	**20**	**20**	16	**20**	19	**20**	**20**	**20**	**20**	**20**	16	**20**	19	**20**	**20**	**20**	**20**
ss_barman (33)	10	8	12	10	11	10	13	10	10	8	14	11	11	10	11	10	**24**	23	**24**	**24**
ss_briefcaseworld (110)	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**
ss_cavediving (100)	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	29	**30**	**30**	**30**	**30**	**30**
ss_citycar (288)	15	22	16	16	12	0	11	14	15	20	15	15	12	0	11	14	24	26	28	**31**
ss_ferry (132)	78	122	95	94	89	89	90	64	80	122	95	94	89	90	89	64	108	**123**	122	122
ss_goldminer (144)	50	79	**102**	53	98	62	80	50	50	80	**102**	53	98	62	97	50	75	**102**	**102**	**102**
ss_grid (108)	52	50	74	58	58	60	58	56	52	50	74	58	58	60	72	56	**91**	89	90	**91**
ss_hanoi (30)	**13**	10	**13**	**13**	**13**	**13**	**13**	**13**	**13**	10	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**
ss_hiking (112)	74	56	85	82	79	84	78	76	74	56	86	84	81	85	81	78	90	93	92	**96**
ss_maintenance (128)	0	45	65	0	10	26	11	11	0	44	65	0	9	24	8	2	0	64	64	**67**
ss_maintenance_large (100)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
ss_npuzzle (30)	6	6	**12**	**12**	6	6	6	**12**	6	6	**12**	**12**	6	6	6	**12**	9	**12**	**12**	**12**
ss_schedule (168)	63	148	130	95	158	144	135	156	64	147	131	98	152	145	119	158	0	158	**163**	**163**
ss_spanner (132)	86	89	95	86	95	95	**132**	95	82	85	92	82	92	92	89	92	**132**	**132**	**132**	**132**
storage (30e)	16	17	18	16	18	18	18	17	17	17	**19**	17	18	**19**	17	18	15	**19**	16	**19**
t0-adder (2)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
t0-coins (30)	10	15	0	9	10	10	10	10	10	15	0	9	10	10	10	10	15	15	**16**	**16**
t0-comm (25)	5	6	5	5	5	5	5	5	5	6	5	5	5	5	5	5	**15**	**15**	**15**	**15**
t0-grid-dispose (15)	0	**3**	0	0	0	0	0	0	0	**3**	0	0	0	0	0	0	1	1	**3**	**3**
t0-grid-push (5)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
t0-grid-trash (1)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
t0-sortnet (5)	**2**	**2**	0	0	**2**	0	**2**	**2**	**2**	**2**	0	0	**2**	0	**2**	**2**	0	1	**2**	**2**
t0-sortnet-alt (6)	**4**	**4**	1	1	**4**	**4**	**4**	**4**	**4**	**4**	1	1	**4**	**4**	**4**	**4**	2	2	**4**	**4**
t0-uts (29)	6	8	10	7	10	10	10	10	6	9	**11**	7	**11**	**11**	**11**	**11**	6	9	9	**11**
tetris-opt14-strips (17)	12	11	**13**	12	**13**	1	**13**	**13**	12	11	12	12	**13**	1	**13**	**13**	10	**13**	10	**13**
tidybot-opt11-strips (20)	10	**17**	15	13	11	1	11	10	10	**17**	15	13	11	1	10	10	14	**17**	**17**	**17**
tidybot-opt14-strips (20)	1	**13**	11	8	3	0	3	2	1	**13**	11	8	3	0	2	2	6	**13**	12	**13**
tpp (30)	7	8	7	7	9	**12**	8	7	7	8	7	8	9	11	8	7	8	11	11	**12**
transport-opt08-strips (30)	11	11	11	12	11	11	11	11	11	11	12	12	11	11	12	11	**14**	**14**	13	**14**
transport-opt11-strips (20)	6	7	7	8	6	7	6	6	6	7	8	8	7	7	8	7	**10**	9	**10**	**10**
transport-opt14-strips (20)	7	6	7	7	7	7	7	7	7	6	7	7	7	7	7	7	**9**	**9**	**9**	**9**
trucks-strips (30)	9	**12**	11	10	9	10	10	9	9	**12**	11	10	9	10	10	9	**12**	**12**	11	**12**
visitall-opt11-strips (20)	9	12	16	13	9	10	9	16	9	12	14	12	9	10	9	14	12	**17**	**17**	**17**
visitall-opt14-strips (20)	3	6	12	7	4	4	4	12	3	6	8	6	4	4	4	8	7	13	**14**	**14**
woodworking-opt08-strips (30)	22	**30**	23	22	**30**	**30**	**30**	28	22	**30**	23	22	**30**	**30**	22	28	28	**30**	**30**	**30**
woodworking-opt11-strips (20)	16	**20**	17	16	**20**	**20**	**20**	**20**	16	**20**	17	16	**20**	**20**	16	**20**	**20**	**20**	**20**	**20**
zenotravel (20)	8	**13**	12	11	12	12	11	11	8	**13**	12	11	12	**13**	11	11	11	12	12	**13**
Sum (3721)	1470	1956	1836	1602	1796	1645	1767	1615	1472	1948	1838	1606	1782	1643	1707	1568	1867	2282	2236	**2350**

Table 4: Coverage of the trainingset. Abbreviations: lmc: LM-cut; P1: HC-PDB; P2: GA-ZOPDB; M1: B-SCCdfp; M2: B-MIASMdfp; M3: B-sbMIASM; M4: G-SCCdfp; Sym: SymBA* 2014; Orcl: oracle portfolio over all component planners.

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.

Fast Downward Remix

Jendrik Seipp
University of Basel
Basel, Switzerland
jendrik.seipp@unibas.ch

Fast Downward Remix is a portfolio planner, based on the Fast Downward planning system (Helmert 2006). It uses the greedy algorithm by Streeter, Golovin, and Smith (2007) to compute a sequential static portfolio of Fast Downward configurations in an offline preprocessing phase. Fast Downward Remix participated in the sequential satisficing, bounded-cost and agile tracks of the International Planning Competition (IPC) 2018.

Definitions

Before we describe the greedy portfolio computation algorithm, we give some definitions concerning planning tasks, sequential portfolios and quality metrics.

Informally speaking, a classical planning task consists of an initial state, a goal description and a set of operators. In the setting of *satisficing planning*, solving a planning task entails finding any operator sequence that leads from the initial state to a goal state, with a preference for cheap solutions. On the other hand, in the setting of *agile planning*, the task is to find solutions as fast as possible, regardless of the solution cost. The third setting we consider in this planner abstract is *bounded-cost* planning, where plans must not be more expensive than a given bound.

We define $c(A, I, t)$ as the *cost* of the solution a planning algorithm A finds for planning task I within time t, or as ∞ if it does not find a solution in that time. Furthermore, we let $c^\star(I)$ denote the *minimum known solution cost* for task I (approximated by a set of Fast Downward configurations). Following IPC evaluation criteria, we define the *solution quality* $q_{\text{sol}}(A, I, t) = \frac{c^\star(I)}{c(A,I,t)}$ as the minimum known solution cost divided by the solution cost achieved by A in time t.

A *sequential planning portfolio* P is a sequence of pairs $\langle A, t\rangle$ where A is a planning algorithm and $t \in \mathbb{N}_{>0}$ is the time limit in seconds for A. We denote the portfolio resulting from appending a component $\langle A, t\rangle$ to a portfolio P by $P \oplus \langle A, t\rangle$.

We now define two quality scores $q(P, I)$ that evaluate the performance of a portfolio P on task I. In the satisficing and bounded-cost settings we use the solution quality $q_{\text{sol}}(P, I)$. It is the maximum solution quality any of the components in P achieves for I, i.e.,

Algorithm 1 Greedy algorithm by Streeter, Golovin, and Smith (2007) computing a sequential portfolio for a given quality function q, algorithms $\mathcal{A}$, instances $\mathcal{I}$ and total portfolio runtime T.

```
 1: function COMPUTEPORTFOLIO($q$, $\mathcal{A}$, $\mathcal{I}$, $T$)
 2:     $P \leftarrow \langle\rangle$
 3:     $t_{\text{used}} \leftarrow 0$
 4:     while $t_{\max} = T - t_{\text{used}} > 0$ do
 5:         $\langle A, t\rangle \leftarrow \arg\max_{\langle A', t'\rangle \in \mathcal{A} \times [1, t_{\max}]} q_\Delta(P, A', t', \mathcal{I})$
 6:         if $q_\Delta(P, A, t, \mathcal{I}) = 0$ then
 7:             return $P$
 8:         $P \leftarrow P \oplus \langle A, t\rangle$
 9:         $t_{\text{used}} \leftarrow t_{\text{used}} + t$
10:     return $P$
```

$$q_{\text{sol}}(P, I) = \max_{\langle A,t\rangle \in P} q_{\text{sol}}(A, I, t).$$

Following IPC 2018 evaluation criteria, for the agile planning setting we define *agile quality* as

$$q_{\text{agile}}(P, I) = \begin{cases} 0 & \text{if } t(P, I) > T \\ 1 & \text{if } t(P, I) \leq 1 \\ 1 - \frac{\log_{10} t(P,I)}{\log_{10}(T)} & \text{otherwise} \end{cases},$$

where $t(P, I)$ is the time that portfolio P needs to solve task I and T is the total portfolio runtime.

A portfolio's score on multiple tasks $\mathcal{A}$ is defined as the sum of the individual scores, i.e., $q(P, \mathcal{I}) = \sum_{I \in \mathcal{I}} q(P, I)$, and the score of the empty portfolio is always 0.

Greedy Portfolio Computation Algorithm

We now describe the greedy algorithm by Streeter, Golovin, and Smith (2007). Given a quality score q, a set of algorithms $\mathcal{A}$, a set of tasks $\mathcal{I}$ and the total portfolio runtime T, the greedy algorithm iteratively constructs a sequential portfolio.

As shown in Algorithm 1, the procedure starts with an empty portfolio P (line 2) and then iteratively selects an algorithm $A \in \mathcal{A}$ and a time limit $t \in [1, t_{\max}]$ (discretized to seconds) for A such that adding $\langle A, t\rangle$ to P improves P

the most (line 5). The quality improvement between P and $P \oplus \langle A, t\rangle$ is measured by the q_Δ function:

$$q_\Delta(P, A, t, \mathcal{I}) = \frac{\sum_{I \in \mathcal{I}} q(P \oplus \langle A, t\rangle, I) - q(P, I)}{t}$$

If appending the pair $\langle A, t\rangle$ to P does not change the portfolio quality anymore, we converged and can terminate (line 6). Otherwise, the pair is appended to P (line 8). This process iterates until the sum of the runtimes in the portfolio components exceeds the maximum porfolio runtime T (line 4).

Training Benchmark Set

Our set of training instances consists of almost all tasks from the satisficing tracks of IPC 1998–2014 plus tasks from various other sources: compilations of conformant planning tasks (Palacios and Geffner 2009), finite-state controller synthesis problems (Bonet, Palacios, and Geffner 2009), genome edit distance problems (Haslum 2011), alarm processing tasks for power networks (Haslum and Grastien 2011), and Briefcaseworld tasks from the FF/IPP domain collection.[1] In total, we use 2115 training instances.

Planning Algorithms

We collect our input planning algorithms from several sources. First, we use the component algorithms of the following portfolios that participated in the sequential satisficing track of IPC 2014:

- Fast Downward Cedalion (Seipp, Sievers, and Hutter 2014; Seipp et al. 2015): 18 algorithms[2]
- Fast Downward Stone Soup 2014 (Röger, Pommerening, and Seipp 2014): 27 algorithms[3]
- Fast Downward Uniform (Seipp, Braun, and Garimort 2014): 21 algorithms

Second, for each of the 66 algorithms A above, we add another version A' which only differs from A in that A' uses an additional type-based open list (Xie et al. 2014) with the type (g), i.e., the distance to the initial state. Both A and A' alternate between their open lists (Röger and Helmert 2010).

Third, we add 12 different variants of the configuration used in the first iteration of LAMA 2011 (Richter, Westphal, and Helmert 2011). We vary the following parameters:

- preferred_successors_first $\in$ {true, false}:
  Consider states reached via preferred operators first?
- randomize_successors $\in$ {true, false}:
  Randomize the order in which successors are generated?[4]
- additional type-based open list $\in$ {none, (g), (h^{FF}, g)}:
  Alternate between only the original open lists used by the first iteration of LAMA 2011 or include an additional type-based open list (Xie et al. 2014) with the type (g) or (h^{FF}, g)?

In total, this leaves us with $(18 + 27 + 21) \cdot 2 + 12 = 144$ planner configurations as input of the greedy portfolio computation algorithm.

Resulting Portfolios

Passing the algorithms and benchmarks described above to the greedy portfolio computation algorithm, together with the quality score q_{sol} and time limit T=1800 seconds, we obtain a portfolio for the satisficing and bounded-cost tracks that consists of 150 component algorithms, 104 of which are unique. (The greedy algorithm often adds the same planner configuration multiple times with different time limits.) The minimum and maximum time limit are 1 and 149 seconds. On the training set, the portfolio achieves an overall quality score of 2003.89, which is much better than the best component algorithm with a score of 1650.40. If we had an oracle to select the best algorithm (getting allotted the full 1800 seconds) for each instance, we could reach a total score of 2073.

When we use the q_{agile} score and a time limit of 300 seconds the resulting portfolio achieves an agile score of 1743.62 points, while the best single algorithm scores 1718.22 points. The agile portfolio consists of 47 configurations, 37 of which are unique. They are run with time limits ranging from 1 to 36 seconds.

Executing Sequential Portfolios

In the previous sections, we assumed that a portfolio simply assigns a runtime to each algorithm, leaving their sequential order unspecified. With the simplifying assumption that all planner runs use the full assigned time and do not communicate information, the order is indeed irrelevant. In reality the situation is more complex.

First, the Fast Downward planner uses a preprocessing phase that we need to run once before we start the portfolio, so we do not have the full 1800 seconds available.[5] Therefore, we treat per-algorithm time limits defined by the portfolio as relative, rather than absolute values: whenever we start an algorithm, we compute the total allotted time of this and all following algorithms and scale it to the actually remaining computation time. We then assign the respective scaled time to the run. As a result, the last algorithm is allowed to use all of the remaining time.

Second, in the satisficing setting we would like to use the cost of a plan found by one algorithm to prune the search of subsequent planner runs (in the bounded-cost and agile setting we stop after finding the first valid plan). We therefore use the best solution found so far for pruning based on

[1]`http://fai.cs.uni-saarland.de/hoffmann/ff-domains.html`

[2]The only change we make to the algorithms is disabling the YAHSP lookahead (Vidal 2004).

[3]We ignore the anytime algorithm which is run after a solution has been found.

[4]When randomizing successors and considering preferred successors first, randomization happens before preferred successors are moved to the front.

[5]The preprocessing phase consists of converting the input PDDL task (Fox and Long 2003) into a SAS$^+$ task (Bäckström and Nebel 1995) with the Fast Downward translator component and pruning irrelevant operators via computing h^2 mutexes (Alcázar and Torralba 2015)

g values: only paths in the state space that are cheaper than the best solution found so far are pursued.

Acknowledgments

For a portfolio planner, not those who *combined* the components deserve the main credit but those who *contributed* them. We therefore wish to thank all Fast Downward contributors and the people who came up with the algorithms we use in our portfolio. We are also grateful to Álvaro Torralba and Vidal Alcázar for allowing us to use their h^2 mutexes code.

References

Alcázar, V., and Torralba, Á. 2015. A reminder about the importance of computing and exploiting invariants in planning. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 2–6. AAAI Press.

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS$^+$ planning. *Computational Intelligence* 11(4):625–655.

Bonet, B.; Palacios, H.; and Geffner, H. 2009. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 34–41. AAAI Press.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124.

Haslum, P., and Grastien, A. 2011. Diagnosis as planning: Two case studies. In *ICAPS 2011 Scheduling and Planning Applications woRKshop*, 37–44.

Haslum, P. 2011. Computing genome edit distances using domain-independent planning. In *ICAPS 2011 Scheduling and Planning Applications woRKshop*, 45–51.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Palacios, H., and Geffner, H. 2009. Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence Research* 35:623–675.

Richter, S.; Westphal, M.; and Helmert, M. 2011. LAMA 2008 and 2011 (planner abstract). In *IPC 2011 planner abstracts*, 50–54.

Röger, G., and Helmert, M. 2010. The more, the merrier: Combining heuristic estimators for satisficing planning. In Brafman, R.; Geffner, H.; Hoffmann, J.; and Kautz, H., eds., *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*, 246–249. AAAI Press.

Röger, G.; Pommerening, F.; and Seipp, J. 2014. Fast Downward Stone Soup 2014. In *Eighth International Planning Competition (IPC-8): planner abstracts*, 28–31.

Seipp, J.; Sievers, S.; Helmert, M.; and Hutter, F. 2015. Automatic configuration of sequential planning portfolios. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3364–3370. AAAI Press.

Seipp, J.; Braun, M.; and Garimort, J. 2014. Fast Downward uniform portfolio. In *Eighth International Planning Competition (IPC-8): planner abstracts*, 32.

Seipp, J.; Sievers, S.; and Hutter, F. 2014. Fast Downward Cedalion. In *Eighth International Planning Competition (IPC-8): planner abstracts*, 17–27.

Streeter, M. J.; Golovin, D.; and Smith, S. F. 2007. Combining multiple heuristics online. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007)*, 1197–1203. AAAI Press.

Vidal, V. 2004. A lookahead strategy for heuristic search planning. In Zilberstein, S.; Koehler, J.; and Koenig, S., eds., *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*, 150–159. AAAI Press.

Xie, F.; Müller, M.; Holte, R. C.; and Imai, T. 2014. Type-based exploration with multiple search queues for satisficing planning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2014)*. AAAI Press.

Scorpion

Jendrik Seipp
University of Basel
Basel, Switzerland
jendrik.seipp@unibas.ch

This planner abstract describes "Scorpion", the planner we submitted to the sequential optimization track of the International Planning Competition 2018. Scorpion is implemented in the Fast Downward planning system (Helmert 2006). It uses A* (Hart, Nilsson, and Raphael 1968) with an admissible heuristic (Pearl 1984) to find optimal plans. The overall heuristic is based on component abstraction heuristics that are combined by saturated cost partitioning (Seipp and Helmert 2018).[1]

In this abstract we only list the components of Scorpion and the settings we used for them. For a detailed description of the underlying algorithms we refer to Seipp (2018).

Abstraction Heuristics

Depending on whether or not a given task contains conditional effects, we use a different set of abstraction heuristics.

Tasks Without Conditional Effects

For tasks without conditional effects we use the combination of the following heuristics:

- Cartesian abstraction heuristics (CART):
  We consider Cartesian abstractions of the landmark and goal task decompositions (Seipp and Helmert 2018). We limit the total number of non-looping transitions in all abstractions underlying the Cartesian heuristics by one million.
- pattern databases found by hill climbing (HC):
  We use the algorithm by Haslum et al. (2007) for searching via hill climbing in the space of pattern collections. We limit the time for hill climbing by 100 seconds.
- pattern databases for systematic patterns (SYS):
  We use a procedure that generates all *interesting* patterns up to size 2 (Pommerening, Röger, and Helmert 2013).

Tasks With Conditional Effects

For tasks with conditional effects we compute pattern database heuristics for systematically generated patterns of sizes 1, 2 and 3 (Pommerening, Röger, and Helmert 2013). Since generating these heuristics can take very long for some tasks, we limit the time for generating PDB heuristics by 300 seconds.

[1]We chose the name "Scorpion" since it contains the letters s(aturated) c(ost) p(artitioning) in this order.

Saturated Cost Partitioning

We combine the information contained in the component heuristics with saturated cost partitioning (Seipp and Helmert 2018). Given an ordered collection of heuristics, saturated cost partitioning iteratively assigns each heuristic h only the costs that h needs for justifying its estimates and saves the remaining costs for subsequent heuristics. Distributing the operator costs among the component heuristics in this way makes the sum of the individual heuristic values admissible.

The quality of the resulting saturated cost partitioning heuristic strongly depends on the order in which the component heuristics are considered (Seipp, Keller, and Helmert 2017). Additionally, we can obtain much stronger heuristics by maximizing over multiple saturated cost partitioning heuristics computed for different orders instead of using a single saturated cost partitioning heuristic (Seipp, Keller, and Helmert 2017). We therefore iteratively sample a state (using the sampling algorithm by Haslum et al. 2007), use a greedy algorithm for finding an initial order for the state (more concretely, we use the static greedy ordering algorithm with the $q_{\frac{h}{stolen}}$ scoring function) and afterwards optimize the order with simple hill climbing in the space of orders for at most two seconds (Seipp 2018). If the the saturated cost partitioning heuristic computed for the resulting optimized greedy order yields a higher estimate for one of a set of 1000 sample states than all previously added orders, we add the order to our set of orders. We limit the time for finding orders in this way to 200 seconds.

Operator Pruning Techniques

We employ two operator pruning techniques:

- strong stubborn sets:
  We use the variant that instantiates strong stubborn sets for classical planning in a straight-forward way (Alkhazraji et al. 2012; Wehrle and Helmert 2014). We compute the interference relation "on demand" during the search and switch off pruning completely in case the fraction of pruned successor states is less than 20% of the total successor states after 1000 expansions.

Coverage	CART	HC	SYS	HC+CART	SYS+CART	SYS+HC	SYS+HC+CART
Agricola (20)	0	**4**	1	1	0	**4**	1
Data-Network (20)	**14**	12	12	**14**	**14**	12	**14**
Organic-Synthesis (20)	**7**	**7**	**7**	**7**	**7**	**7**	**7**
Organic-Synthesis-Split (20)	**13**	**13**	12	**13**	**13**	**13**	**13**
Petri-Net-Alignment (20)	3	0	**7**	0	5	0	0
Snake (20)	11	**13**	**13**	**13**	**13**	**13**	**13**
Spider (20)	13	**15**	**15**	**15**	**15**	**15**	**15**
Termes (20)	12	13	12	**14**	12	**14**	**14**
Sum (160)	73	77	**79**	77	**79**	78	77

Table 1: Coverage scores of saturated cost partitioning over different heuristic subsets for IPC 2018 tasks that have no conditional effects after the translation phase.

- h^2 mutexes (Alcázar and Torralba 2015):
  This operator pruning method can remove irrelevant operators. We invoke it after translating a given input task to SAS$^+$ and before starting the search component of Fast Downward.

Post-IPC Evaluation

After the IPC 2018, we ran an experiment to analyze how much value each of the three sets of heuristics (CART, HC and SYS) contributes to the overall heuristic on the IPC 2018 benchmarks that have no conditional effects after the translation phase. We used a time and memory limit of 30 minutes and 7 GiB. Table 1 shows coverage results.

The seven different combinations of heuristics lead to similar total coverage scores (73–79 tasks). Using Cartesian heuristics (CART) leads to solving the lowest number of tasks, whereas using systematic PDBs by themselves (SYS) or combined with Cartesian abstractions (SYS+CART) achieves the maximal total coverage score. While coverage never decreases when adding systematic PDBs to the set of heuristics, it varies between domains whether the other types of heuristics are beneficial.

For the tasks in the Agricola domain, hill climbing PDBs (HC) are more informative (4 solved tasks) than other heuristics (0 or 1 solved task). Adding Cartesian abstractions to the hill climbing PDBs leads to worse heuristics. In principle, Scorpion should be able to produce better estimates given more heuristics, but having a larger set of heuristics can make finding a good order for saturated cost partitioning harder.

In the Data-Network domain it is beneficial to use Cartesian abstractions (14 solved tasks vs. 12 solved tasks without Cartesian abstractions), whereas all heuristic subsets solve almost the same number of tasks in both variants of Organic-Synthesis.

When using hill climbing PDB heuristics, Scorpion is unable to solve any Petri-Net-Alignment tasks within the given limits. This is the case since the hill climbing algorithm starts by computing a PDB for each goal variable and then calculates the maximal additive subsets of these PDBs. The latter step runs out of memory for all tasks from the Petri-Net-Alignment domain, because of the large number of goal facts.[2]

The Snake and Spider domains benefit from using PDB heuristics. With systematic or hill climbing PDBs Scorpion solves 13 and 15 tasks, in these two domains. Using only Cartesian abstractions leads to solving two fewer tasks in each of the two domains.

In the Termes domain hill climbing PDBs with at least one other type of heuristic solves 14 tasks, whereas the other configurations solve 12 or 13 tasks.

Acknowledgments

We would like to thank all Fast Downward contributors. Special thanks go to Álvaro Torralba and Vidal Alcázar for their h^2 mutexes code.

References

Alcázar, V., and Torralba, Á. 2015. A reminder about the importance of computing and exploiting invariants in planning. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 2–6. AAAI Press.

Alkhazraji, Y.; Wehrle, M.; Mattmüller, R.; and Helmert, M. 2012. A stubborn set algorithm for optimal planning. In De Raedt, L.; Bessiere, C.; Dubois, D.; Doherty, P.; Frasconi, P.; Heintz, F.; and Lucas, P., eds., *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, 891–892. IOS Press.

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

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007)*, 1007–1012. AAAI Press.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

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

Pommerening, F.; Röger, G.; and Helmert, M. 2013. Getting the most out of pattern databases for classical planning. In Rossi, F., ed., *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 2357–2364. AAAI Press.

[2]All tasks in the Petri-Net-Alignment domain share the same problem file, which has 268 goal facts, and differ only in the domain files.

Seipp, J., and Helmert, M. 2018. Counterexample-guided Cartesian abstraction refinement for classical planning. *Journal of Artificial Intelligence Research* 62:535–577.

Seipp, J.; Keller, T.; and Helmert, M. 2017. Narrowing the gap between saturated and optimal cost partitioning for classical planning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)*, 3651–3657. AAAI Press.

Seipp, J. 2018. *Counterexample-guided Cartesian Abstraction Refinement and Saturated Cost Partitioning for Optimal Classical Planning*. Ph.D. Dissertation, University of Basel.

Wehrle, M., and Helmert, M. 2014. Efficient stubborn sets: Generalized algorithms and selection strategies. In Chien, S.; Fern, A.; Ruml, W.; and Do, M., eds., *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 323–331. AAAI Press.

Fast Downward Stone Soup 2018

Jendrik Seipp and **Gabriele Röger**
University of Basel
Basel, Switzerland
{jendrik.seipp, gabriele.roeger}@unibas.ch

Fast Downward Stone Soup (Helmert, Röger, and Karpas 2011) is a portfolio planner, based on the Fast Downward planning system (Helmert 2006; 2009). It already participated in the International Planning Competitions (IPC) 2011 and 2014.

In this planner abstract, we present the Fast Downward Stone Soup portfolio that we submitted to the sequential satisficing and bounded-cost tracks of IPC 2018. It uses different component algorithms than the 2011 and 2014 variants but employs the same procedure for building the portfolio. Therefore, we only briefly recapitulate the procedure and refer the reader to the original Fast Downward Stone Soup paper for a more detailed discussion (Helmert, Röger, and Karpas 2011).

Building the Portfolio

The Stone Soup algorithm requires the following information as input:

- A set of *planning algorithms* $\mathcal{A}$. We use a set of 144 Fast Downward configurations, which we describe below.
- A set of *training instances* $\mathcal{I}$, for which portfolio performance is optimized. We use a set of 2115 instances, described below.
- Complete *evaluation results* that include, for each algorithm $A \in \mathcal{A}$ and training instance $I \in \mathcal{I}$,
  - the *runtime* $t(A, I)$ of the given algorithm on the given training instance on our evaluation machines, in seconds (we did not consider anytime planners), and
  - the *plan cost* $c(A, I)$ of the plan that was found.

  We use time and memory limits of 30 minutes and 3.5 GiB to generate this data. If algorithm A fails to solve instance I within these bounds, we set $t(A, I) = c(A, I) = \infty$.

The procedure computes a portfolio as a mapping $P : \mathcal{A} \rightarrow \mathbb{N}_0$ which assigns a time limit (possibly 0 if the algorithm is not used) to each component algorithm. It is a simple hill-climbing search in the space of portfolios, shown in Figure 1.

In addition to the algorithms and the evaluation results, the algorithm takes two parameters, *granularity* and *timeout*, both measured in seconds. The timeout is an upper bound on the total time for the generated portfolio, which is the sum of all component time limits. The granularity specifies the step size with which we add time slices to the current portfolio.

```
build-portfolio(algorithms, results, granularity, timeout):
    portfolio := $\{A \mapsto 0 \mid A \in \mathit{algorithms}\}$
    repeat $\lfloor \mathit{timeout}/\mathit{granularity} \rfloor$ times:
        candidates := successors(portfolio, granularity)
        portfolio := $\arg\max_{C \in \mathit{candidates}}$ score(C, results)
    portfolio := reduce(portfolio, results)
    return portfolio
```

Figure 1: Stone Soup algorithm for building a portfolio.

The search starts from a portfolio that assigns a time limit of 0 seconds to all algorithms. In each hill-climbing step, it generates all possible *successors* of the current portfolio. There is one successor per algorithm A, where the only difference between the current portfolio and the successor is that the time limit of A is increased by the given granularity.

We evaluate the quality of a portfolio P by computing its *portfolio score* $s(P)$. The portfolio score is the sum of *instance scores* $s(P, I)$ over all instances $I \in \mathcal{I}$. The function $s(P, I)$ is similar to the scoring function used for the International Planning Competitions since 2008. The only difference is that we use the best solution quality among our algorithms as reference quality (instead of taking solutions from other planners into account): if no algorithm in a portfolio P solves an instance I within its allotted runtime, we set $s(P, I) = 0$. Otherwise, $s(P, I) = c_I^*/c_I^P$, where c_I^* is the lowest solution cost for I of any input algorithm $A \in \mathcal{A}$ and c_I^P denotes the best solution cost among all algorithms $A \in \mathcal{A}$ that solve the instance within their allotted runtime $P(A)$.

In each hill-climbing step the search chooses the successor with the highest portfolio score. Ties are broken in favor of successors that increase the timeout of the component algorithm that occurs earliest in some arbitrary total order.

The hill-climbing phase ends when all successors would exceed the given time bound. A post-processing step reduces the time assigned to each algorithm by the portfolio. It considers the algorithms in the same arbitrary order used for breaking ties in the hill-climbing phase and sets their time limit to the lowest value that would still lead to the same portfolio score.

Training Benchmark Set

Our set of training instances consists of almost all tasks from the satisficing tracks of IPC 1998–2014 plus tasks from various other sources: compilations of conformant planning tasks (Palacios and Geffner 2009), finite-state controller synthesis problems (Bonet, Palacios, and Geffner 2009), genome edit distance problems (Haslum 2011), alarm processing tasks for power networks (Haslum and Grastien 2011), and Briefcaseworld tasks from the FF/IPP domain collection.[1] In total, we use 2115 training instances.

Planning Algorithms

We collect our input planning algorithms from several sources. First, we use the component algorithms of the following portfolios that participated in the sequential satisficing track of IPC 2014:

- Fast Downward Cedalion (Seipp, Sievers, and Hutter 2014; Seipp et al. 2015): 18 algorithms[2]
- Fast Downward Stone Soup 2014 (Röger, Pommerening, and Seipp 2014): 27 algorithms[3]
- Fast Downward Uniform (Seipp, Braun, and Garimort 2014): 21 algorithms

Second, for each of the 66 algorithms A above, we add another version A' which only differs from A in that A' uses an additional type-based open list (Xie et al. 2014) with the type (g), i.e., the distance to the initial state. Both A and A' alternate between their open lists (Röger and Helmert 2010).

Third, we add 12 different variants of the configuration used in the first iteration of LAMA 2011 (Richter, Westphal, and Helmert 2011). We vary the following parameters:

- preferred_successors_first $\in$ {true, false}:
  Consider states reached via preferred operators first?
- randomize_successors $\in$ {true, false}:
  Randomize the order in which successors are generated?[4]
- additional type-based open list $\in$ {none, (g), (h^{FF}, g)}:
  Alternate between only the original open lists used by the first iteration of LAMA 2011 or include an additional type-based open list (Xie et al. 2014) with the type (g) or (h^{FF}, g)?

In total, this leaves us with $(18 + 27 + 21) \cdot 2 + 12 = 144$ planner configurations as input of the hill-climbing procedure. For the timeout parameter we use 1800 seconds, the time limit used for IPC 2018. We tried different values for the granularity parameter and achieved the best results (computed for the training set) with a granularity of 30 seconds.

[1]`http://fai.cs.uni-saarland.de/hoffmann/ff-domains.html`

[2]The only change we make to the algorithms is disabling the YAHSP lookahead (Vidal 2004).

[3]We ignore the anytime algorithm which is run after a solution has been found.

[4]When randomizing successors and considering preferred successors first, randomization happens before preferred successors are moved to the front.

Resulting Portfolio

The resulting portfolio uses 41 of the 144 possible algorithms, running them between 8 and 135 seconds. On the training set, the portfolio achieves an overall score of 1999.93, which is much better than the best component algorithm with a score of 1650.40. If we had an oracle to select the best algorithm (getting allotted the full 1800 seconds) for each instance, we could reach a total score of 2073.

Executing The Sequential Portfolio

In the previous sections, we assumed that a portfolio simply assigns a runtime to each algorithm, leaving their sequential order unspecified. With the simplifying assumption that all planner runs use the full assigned time and do not communicate information, the order is indeed irrelevant. In reality the situation is more complex.

First, the Fast Downward planner uses a preprocessing phase that we need to run once before we start the portfolio, so we do not have the full 1800 seconds available.[5] Therefore, we treat per-algorithm time limits defined by the portfolio as relative, rather than absolute values: whenever we start an algorithm, we compute the total allotted time of this and all following algorithms and scale it to the actually remaining computation time. We then assign the respective scaled time to the run. As a result, the last algorithm is allowed to use all of the remaining time.

Second, in the satisficing setting we would like to use the cost of a plan found by one algorithm to prune the search of subsequent planner runs (in the bounded-cost setting we stop after finding the first plan that is at most as expensive as the given bound). We therefore use the best solution found so far for pruning based on g values: only paths in the state space that are cheaper than the best solution found so far are pursued.

Third, planner runs often terminate early, e.g., because they run out of memory or find a plan. Since we would like to use the remaining time to continue the search for a plan or improve the solution quality, we sort the algorithms by their coverage scores in decreasing order, hence beginning with algorithms likely to succeed quickly.

Acknowledgments

For a portfolio planner, not those who *combined* the components deserve the main credit but those who *contributed* them. We therefore wish to thank all Fast Downward contributors and the people who came up with the algorithms we use in our portfolio. We are also grateful to Álvaro Torralba and Vidal Alcázar for allowing us to use their h^2 mutexes code.

References

Alcázar, V., and Torralba, Á. 2015. A reminder about the importance of computing and exploiting invariants in plan-

[5]The preprocessing phase consists of converting the input PDDL task (Fox and Long 2003) into a SAS$^+$ task (Bäckström and Nebel 1995) with the Fast Downward translator component and pruning irrelevant operators via computing h^2 mutexes (Alcázar and Torralba 2015)

ning. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 2–6. AAAI Press.

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS$^+$ planning. *Computational Intelligence* 11(4):625–655.

Bonet, B.; Palacios, H.; and Geffner, H. 2009. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 34–41. AAAI Press.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124.

Haslum, P., and Grastien, A. 2011. Diagnosis as planning: Two case studies. In *ICAPS 2011 Scheduling and Planning Applications woRKshop*, 37–44.

Haslum, P. 2011. Computing genome edit distances using domain-independent planning. In *ICAPS 2011 Scheduling and Planning Applications woRKshop*, 45–51.

Helmert, M.; Röger, G.; and Karpas, E. 2011. Fast Downward Stone Soup: A baseline for building planner portfolios. In *ICAPS 2011 Workshop on Planning and Learning*, 28–35.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence* 173:503–535.

Palacios, H., and Geffner, H. 2009. Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence Research* 35:623–675.

Richter, S.; Westphal, M.; and Helmert, M. 2011. LAMA 2008 and 2011 (planner abstract). In *IPC 2011 planner abstracts*, 50–54.

Röger, G., and Helmert, M. 2010. The more, the merrier: Combining heuristic estimators for satisficing planning. In Brafman, R.; Geffner, H.; Hoffmann, J.; and Kautz, H., eds., *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*, 246–249. AAAI Press.

Röger, G.; Pommerening, F.; and Seipp, J. 2014. Fast Downward Stone Soup 2014. In *Eighth International Planning Competition (IPC-8): planner abstracts*, 28–31.

Seipp, J.; Sievers, S.; Helmert, M.; and Hutter, F. 2015. Automatic configuration of sequential planning portfolios. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3364–3370. AAAI Press.

Seipp, J.; Braun, M.; and Garimort, J. 2014. Fast Downward uniform portfolio. In *Eighth International Planning Competition (IPC-8): planner abstracts*, 32.

Seipp, J.; Sievers, S.; and Hutter, F. 2014. Fast Downward Cedalion. In *Eighth International Planning Competition (IPC-8): planner abstracts*, 17–27.

Vidal, V. 2004. A lookahead strategy for heuristic search planning. In Zilberstein, S.; Koehler, J.; and Koenig, S., eds., *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*, 150–159. AAAI Press.

Xie, F.; Müller, M.; Holte, R. C.; and Imai, T. 2014. Type-based exploration with multiple search queues for satisficing planning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2014)*. AAAI Press.

Metis 2018

Silvan Sievers
University of Basel
Basel, Switzerland
silvan.sievers@unibas.ch

Michael Katz
IBM Research
Yorktown Heights, NY, USA
michael.katz1@ibm.com

Abstract

Metis 2018 is a Fast Downward based planner that uses the pruning techniques partial order reduction and structural symmetries. The two variants that participate in the competition use the LM-cut and and the landmark heuristic. The former essentially is a remake of Metis that participated in the IPC 2014, and the latter only differs in the used heuristic and the symmetry-based pruning algorithm.

Metis 2018

Metis is a planner that participated in the IPC 2014 (Alkhazraji et al. 2014), called Metis 2014 henceforth. Our planner, Metis 2018, is a remake of Metis 2014, and comes with similar three core components:

- an admissible heuristic: LM-cut (Helmert and Domshlak 2009) or the max heuristic over LM-cut and the landmark heuristic with the landmark generation method of LAMA (Richter, Helmert, and Westphal 2008) and h^mlandmarks with $m = 2$ (Keyder, Richter, and Helmert 2010),
- pruning based on structural symmetries (Shleyfman et al. 2015) using DKS (Domshlak, Katz, and Shleyfman 2012) or orbit space search (OSS) (Domshlak, Katz, and Shleyfman 2015), and
- pruning based on partial order reduction using strong stubborn sets (Wehrle and Helmert 2014).

Notable differences to Metis 2014 are that we do not use the incremental computation of the LM-cut heuristic (Pommerening and Helmert 2013) and that we include the landmark heuristic for one of our planner variants. Furthermore, we do not only use OSS, but also the DKS algorithm for symmetry-based pruning in one of the variants. The partial order reduction component is the same as in Metis 2014.

In addition to the above differences in ingredients of the planner, Metis 2018 is implemented on top of a recent version of Fast Downward (Helmert 2006). To support conditional effects, we implemented a variant of the LM-cut heuristic that considers effect conditions in the same way as Metis 2014 does. However, we refrain from choosing the regular LM-cut heuristic or variant that supports conditional effects depending on the requirements of the input planning task, and instead always use the latter implementation that comes with a small overhead due to the need for different data structures.

The implementation of symmetry-based pruning is the same in both versions, including the extension of the symmetry graph to support conditional effects, which was recently also defined formally by Sievers et al. (2017) in the context of structural symmetries of lifted representations.

Metis 2018 uses the implementation of strong stubborn sets available in Fast Downward, which is based on the original implementation of Alkhazraji et al. (2012) and Wehrle and Helmert (2012) that has also been used in Metis 2014. However, the current implementation has been improved in terms of efficiency since its original development.[1] To support conditional effects, we extended the implementation in the same way as in Metis 2014. We also use the same mechanism that disables pruning after the first 1000 expansions if only 1% or fewer states have been pruned at this point.

To conclude this abstract, we describe the variants of our planner submissions to the IPC 2018. Both use a post-processing step to transform the SAS$^+$ representation obtained through the translator of Fast Downward (Helmert 2009) by using the implementation of h^2 mutexes by Alcázar and Torralba (2015). Furthermore, both use A* search (Hart, Nilsson, and Raphael 1968) with an admissible heuristic and with the same configuration of strong stubborn sets described above for pruning. Regarding the other components, the two variants have the following differences:

- Metis 2018 version 1 essentially is a remake of Metis 2014 and uses OSS for symmetry-based pruning and the LM-cut heuristic.
- Metis 2018 version 2 uses DKS for symmetry-based pruning and the maximum heuristic over the LM-cut heuristic and the landmark heuristic, with the two landmark generation methods described above.

References

Alcázar, V., and Torralba, Á. 2015. A reminder about the importance of computing and exploiting invariants in planning. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International*

[1]See `http://issues.fast-downward.org/issue499` and `http://issues.fast-downward.org/issue629`.

Conference on Automated Planning and Scheduling (ICAPS 2015), 2–6. AAAI Press.

Alkhazraji, Y.; Wehrle, M.; Mattmüller, R.; and Helmert, M. 2012. A stubborn set algorithm for optimal planning. In De Raedt, L.; Bessiere, C.; Dubois, D.; Doherty, P.; Frasconi, P.; Heintz, F.; and Lucas, P., eds., *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, 891–892. IOS Press.

Alkhazraji, Y.; Katz, M.; Mattmüller, R.; Pommerening, F.; Shleyfman, A.; and Wehrle, M. 2014. Metis: Arming Fast Downward with pruning and incremental computation. In *Eighth International Planning Competition (IPC-8): planner abstracts*, 88–92.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2012. Enhanced symmetry breaking in cost-optimal planning as forward search. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 343–347. AAAI Press.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2015. Symmetry breaking in deterministic planning as forward search: Orbit space search algorithm. Technical Report IS/IE-2015-03, Technion.

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

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 162–169. AAAI Press.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence* 173:503–535.

Keyder, E.; Richter, S.; and Helmert, M. 2010. Sound and complete landmarks for and/or graphs. In Coelho, H.; Studer, R.; and Wooldridge, M., eds., *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, 335–340. IOS Press.

Pommerening, F., and Helmert, M. 2013. Incremental LM-cut. In Borrajo, D.; Kambhampati, S.; Oddi, A.; and Fratini, S., eds., *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*, 162–170. AAAI Press.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, 975–982. AAAI Press.

Shleyfman, A.; Katz, M.; Helmert, M.; Sievers, S.; and Wehrle, M. 2015. Heuristics and symmetries in classical planning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3371–3377. AAAI Press.

Sievers, S.; Röger, G.; Wehrle, M.; and Katz, M. 2017. Structural symmetries of the lifted representation of classical planning tasks. In *ICAPS 2017 Workshop on Heuristics and Search for Domain-independent Planning (HSDIP)*, 67–74.

Wehrle, M., and Helmert, M. 2012. About partial order reduction in planning and computer aided verification. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 297–305. AAAI Press.

Wehrle, M., and Helmert, M. 2014. Efficient stubborn sets: Generalized algorithms and selection strategies. In Chien, S.; Fern, A.; Ruml, W.; and Do, M., eds., *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 323–331. AAAI Press.

Fast Downward Merge-and-Shrink

Silvan Sievers
University of Basel
Basel, Switzerland
silvan.sievers@unibas.ch

Abstract

Fast Downward Merge-and-Shrink uses the optimized, efficient implementation of the merge-and-shrink framework available in the Fast Downward planning system. We describe the techniques used in this implementation. To further push the performance of single-heuristic merge-and-shrink planners, we additionally discuss and evaluate partial merge-and-shrink abstractions, which we obtain through imposing a simple time limit to the merge-and-shrink computation.

Classical Planning

In this planner abstract, we discuss most of the concepts informally. We consider planning tasks in the SAS$^+$ representation (Bäckström and Nebel 1995), which are defined over a finite set of finite-domain variables. States are assignments over these variables. The planning task comes with a set of operators that have preconditions, effects, and a cost, and which allow to transform a state which satisfies the precondition into another state that satisfies the effect and remains unchanged otherwise. The task also specifies an initial state and a goal condition. The semantics of a planning task can naturally be described in terms of the *labeled transition system* it induces.

A labeled transition system, or transition system for short, has a set of states, a set of labels with associated costs, a transition relation that specifies the transitions which are triples of predecessor state, label, and successor state, an initial state from the set of states, and a set of goal states which is a subset of the set of states. Paths are sequences of labels that lead from a given state to some goal state. Their cost is the sum of the label costs of the sequence.

The transition system induced by a planning task consists of the states of the planning task and has transitions induced by the operators of the task, respecting the applicability of operators. Planning is the task of finding a path from the initial state to some goal state, called a plan. Optimal planning, which we are concerned with, deals with finding plans of minimal cost or proving that no plan exists.

Merge-and-Shrink

Merge-and-shrink (Dräger, Finkbeiner, and Podelski 2009; Helmert et al. 2014) is an algorithm framework to compute abstractions of transition systems. While it has very successfully been used to compute heuristics for planning tasks (e.g., Sievers, Wehrle, and Helmert 2014; Fan, Müller, and Holte 2014; Sievers et al. 2015; Sievers, Wehrle, and Helmert 2016; Fan, Müller, and Holte 2017; Fan, Holte, and Müller 2018), it can in principle be used for any problem that can be represented as a state space which exhibits a *factored representation*. Using such compact factored representations of both transition systems and abstraction mappings is a key aspect of merge-and-shrink that allows computing arbitrary abstractions of transition systems of interest which are generally too large to be explicitly represented.

Factored transition systems are tuples of labeled transition systems, also called factors, with the same label set that serve as a compact representation of their *synchronized product*. The synchronized product is the transition system consisting of the Cartesian product of states, where labels are used to synchronize the factors of the factored transition system via the labeled transitions: there is a transition between two states in the product system iff all factors have a transition between the corresponding component states labeled with the same label. A state is an initial/goal state in the product if all its components are initial/goal states in the respective factors.

To represent state mappings, merge-and-shrink uses *factored mappings* (Sievers 2017), which have previously also been called cascading tables (Helmert et al. 2014; Torralba 2015) and merge-and-shrink representations (Helmert, Röger, and Sievers 2015). Factored mappings are tree-like data structures where each leaf node is associated with a variable and a table that maps values of the variable to some values, and each inner node has two children factored mappings and a table that maps pairs of values computed by the children to some values. Factored mappings represent a function defined on assignments over the associated variables of all leaf nodes to some value set. To represent state mappings between factored transition systems, merge-and-shrink uses a tuple of factored mappings, called F2F mapping, that each correspond to one factor of the target factored transition system, i.e., each factored mapping computes the state mapping from states of the source factored transition system to the corresponding factor of the target factored transition system.

With the addition of generalized label reduction (Sievers, Wehrle, and Helmert 2014), the merge-and-shrink algorithm can be understood as a framework that repeatedly applies *transformations* of a factored transition system, which essentially need to specify the transformed factored transition system and the F2F mapping that maps from the given factored transition system to the transformed one. In the context of planning, the algorithm first computes the induced factored transition system of the given task that consists of *atomic factors* which each represent a single variable of the task. It further initializes the F2F mapping to the identity mapping of the factored transition system.

In the main loop, the algorithm then repeatedly selects a transformation of the current factored transition system, choosing from the four available types of merge-and-shrink transformations: *merge* transformations replace two factors by their synchronized product, *shrink* transformations apply an abstraction to a single factor, *prune* transformations discard unreachable or irrelevant states, i.e., states from which no goal state can be reached, of a single factor, and *label reductions* map the common label set of the factored transition system to a smaller one. Applying the selected transformation means to replace the previous factored transition system by the transformed one, and to compose the previous F2F mapping with the one of the transformation. The main loop terminates if the maintained factored transition system only contains a single factor. Together with the factored mapping, this factor induces the merge-and-shrink heuristic.

Concrete instantiations of the algorithm framework need to decide on a general strategy that decides on which type of transformation to apply in each iteration of the main loop, and it needs to provide *transformation strategies* that specify how to compute the individual transformations. For example, *shrink strategies* compute a state equivalence relation for a given transition system, reducing the size of the transition system below a given limit, and *merge strategies* decide which two factors to replace by their synchronized product.

Since our efficient implementation of the merge-and-shrink relies on label equivalence relations, we briefly discuss this concept in the context of label reductions. Sievers, Wehrle, and Helmert (2014) showed that label reductions are exact, i.e., preserve the perfect heuristic, if they only combine labels of the same cost that are Θ-*combinable* for some factor Θ of a given factored transition system F. Labels are Θ-combinable if they are *locally equivalent* in all factors $\Theta' \neq \Theta$ of F, i.e., if they label exactly the same transitions in all other factors than Θ.

For more details and a formal presentation of the transformation framework and the merge-and-shrink transformations, we refer to the work by Sievers (2017).

Implementation

In this section, we briefly mention some of the techniques used in the efficient implementation of the merge-and-shrink framework in Fast Downward (Helmert 2006). More details can be found in the work by Sievers (2017).

To represent transition systems, we do not store transitions as an adjacency list as it is commonly done to represent graphs, but rather store all transitions grouped by labels. This allows an efficient application of all merge-and-shrink transformations, as we will see below. Furthermore, we store *label groups* of locally equivalent labels for each factor, disregarding their cost (the cost of a label group is the minimum cost of any participating label). This allows storing the transitions of locally equivalent labels once rather than separately for each label.

Depending on the chosen transformation strategies, we need to compute g- and h-values of individual factors already during the merge-and-shrink computation. (Of course, we need to compute h-values in the end to compute the heuristic.) These are computed using Dijkstra's algorithm (Dijkstra 1959). This is the only place where we need an explicit adjacency list representation of transition systems.

We now turn our attention to the different merge-and-shrink transformations. When applying a shrink transformation, the shrink strategy computes a state equivalence relation for the given factor. We first compute the explicit state mapping from this equivalence relation, assigning a consecutive number to each equivalence class to allow a compact representation. Then we use this state mapping for an in-place modification of the factor by going over all transitions and updating their source and target states (compared to an adjacency list, this avoids the need to move transitions of different states), and for an in-place modification of the corresponding factored mapping by applying the state mapping to its table. From the equivalence relation on states, we get the set of new goal states.

When applying a merge transformation to the factored transition system, merging two transition systems Θ_1 and Θ_2, we do not compute the full product of states and their transitions because this would require to compute the local equivalence relation on labels from scratch after computing the product. Instead, we use a more efficient, bucket-based approach to directly compute the refinement of the local equivalence relations on labels of Θ_1 and Θ_2, collecting their transitions accordingly. Computing the factored mapping that maps states to the product factor is straightforward and merely a composition of the two component factored mappings.

When applying a prune transformation, we first determine the set of to-be-pruned states using g- and/or h-values. We prune them by entirely removing them and their transitions from the factor. The table of the corresponding factored mapping is updated to map removed states to a special symbol which is evaluated to ∞ by the heuristic.

For an efficient computation of exact label reductions based on Θ-combinability, we need to be able to efficiently refine the local equivalence relations of all (but one) factors of a factored transition system. This is possible using linked lists, which we therefore use to store label equivalence classes, i.e., label groups, for each factor. Applying the label reduction, i.e., the label mapping, is simple for all factors $\Theta' \neq \Theta$ for which we know that the reduced labels are locally equivalent: all we need to do is to relabel the set of transitions of the reduced labels, remove the labels from their group and add the new label to it. For the factor Θ, we need to collect all transitions of all reduced labels and combine them to form the transitions of the new label. We update the local equivalence on labels by removing reduced labels from their (different) groups and the groups themselves if they become empty, and by adding a new singleton group for the new label.

Partial Merge-and-Shrink Abstractions

To the best of our knowledge, the literature on merge-and-shrink so far always considered computing merge-and-shrink abstractions over all variables of a given planning task. That is, the main loop of the algorithm is stopped only if the factored transition system contains a single factor. However, there is no conceptual or technical reason to not stop the algorithm early, ending up with several factors and factored mappings that represent so-called *partial abstractions* because they do not cover all variables of the given planning task. The set of partial abstractions in turn induces a set of *factor heuristics* in the same way as usually the single factor and factored mapping does.

Additionally, we observed that state-of-the-art merge-and-shrink planners fail to finish computing the abstraction in the given time and memory limits in a non-negligible number of cases (152–272 out of 1667 tasks for state-of-the-art-configurations[1]). As a simple stop-gap measure for this phenomenon, we suggest adding a time limit to the merge-and-shrink algorithm, allowing to terminate the computation even before having computed all atomic factors. As a consequence, we obtain a set of partial merge-and-shrink heuristics as described above whenever the time limit stops the merge-and-shrink computation early.

Whenever this happens, we face the decision of computing a heuristic from the set of factor heuristics induced by the remaining factors and factored mappings. A straightforward way is to compute the *max-factor heuristic* (h^{mf}) that maximizes over all factor heuristics. The second, presumably less expensive alternative is to choose a *single* factor heuristic (h^{sg}) and use it as the merge-and-shrink heuristic. We use the following simple rule of thumb in the latter case: we prefer the factor heuristic with the largest estimate for the initial state (rationale: better informed heuristic), breaking ties in favor of larger factors (rationale: more fine-grained abstraction), and choose a random heuristic among all remaining candidates of equal preference.

A recent paper that was published after the IPC describes and evaluates this technique in more detail (Sievers 2018).

Competition Planner

In the following, we describe the two variants of the planner submitted to the IPC 2018. To decide how to compute partial merge-and-shrink abstractions, we also evaluate different choices experimentally. To do so, we ran our planner on the (optimal) benchmarks of all IPCs up to 2014, a set comprised of 1667 planning tasks distributed across 57 domains,[2] using A* search in conjunction with different merge-and-shrink heuristics. We limit time to 30 minutes and memory to 3.5 GiB per task, using Downward-Lab

[1]See rows “# constr” of column “base” of Table 1.

[2]From the collection at `https://bitbucket.org/aibasel/downward-benchmarks`, we use the “optimal strips” benchmark suite.

			h^{sg}			h^{mf}		
	base	450s	900s	1350s	450s	900s	1350s	
Coverage	802	835	**836**	**836**	**836**	**836**	835	FDMS2
# constr	1395	**1637**	1629	1615	1636	1629	1614	FDMS2
Constr time	241.90	135.99	197.57	230.70	**135.73**	196.58	229.45	FDMS2
Constr oom	**21**	**21**	**21**	**21**	**21**	**21**	**21**	FDMS2
Constr oot	251	**9**	17	31	10	17	32	FDMS2
E 75th perc	**1342k**	1368k	**1342k**	**1342k**	1368k	**1342k**	**1342k**	FDMS2
Coverage	814	**844**	**844**	842	**844**	**844**	841	FDMS1
# constr	1505	**1622**	1620	1611	**1622**	1621	1611	FDMS1
Constr time	97.93	61.62	80.59	91.17	**61.29**	79.82	89.84	FDMS1
Constr oom	**21**	**21**	**21**	**21**	**21**	**21**	**21**	FDMS1
Constr oot	141	**24**	26	35	**24**	25	35	FDMS1
E 75th perc	**1860k**	**1860k**	**1860k**	**1860k**	**1860k**	**1860k**	**1860k**	FDMS1

Table 1: Comparison of the baseline against two versions of partial merge-and-shrink, using different time limits.

(Seipp et al. 2017) for conducting the experiments on a cluster of machines with Intel Xeon Silver 4114 CPUs running at 2.2 GHz.

Both variants of our planner, FDMS1 and FDMS2, use the state-of-the-art shrink strategy based on bisimulation (Nissim, Hoffmann, and Helmert 2011) with a size limit of 50000, always allowing (perfect) shrinking. We use full pruning, i.e., we always prune both unreachable and irrelevant states, and we perform exact label reductions based on Θ-combinability with a fixed point algorithm using a random order on factors. FDMS1 uses the state-of-the-art merge strategy based on *strongly connected components* of the causal graph (Sievers, Wehrle, and Helmert 2016), which uses DFP (Sievers, Wehrle, and Helmert 2014) for internal merging (SCCdfp). FDMS2 uses the merge strategy *score-based MIASM* (sbMIASM, previously also called DYN-MIASM), which is a simple variant of the entirely precomputed merge strategy *maximum intermediate abstraction size minimizing* (Fan, Müller, and Holte 2014).

Table 1 shows the number of solved tasks (coverage), the number of tasks for which the heuristic construction completed (# constr), the runtime of the heuristic construction (constr time), the number of failures of the heuristic construction due to running out of time (constr oot) or memory (constr oom), and the number of expansions until the last f-layer. The table compares the baseline (base) with the two variants of computing a single merge-and-shrink heuristic (h^{sg} and h^{mf}) using time limits of 450s, 900s, and 1350s.

As expected, adding a time limit is a very effective measure for greatly increasing the number of successful heuristic constructions, which also directly transfers to a significant increase in coverage of all configurations, with 900s being a sweet spot for both planners. Stopping the computation early does *not* affect the heuristic quality as one might have expected. The likely reason is that with limiting the time, we catch precisely those tasks for which the construction otherwise does not terminate or terminate too late for a successful search. Tasks which we can already solve without imposing a time limit (base) usually require a rather short construction

	Compl.		FDMS		PP	Scor
	1	2	1	2		
Sum previous IPCs (1667)	1026	1056	939	936	1065	**1150**
Sum IPC 2018 (200)	**125**	**125**	101	104	123	108
Sum (1867)	1151	1181	1040	1040	1188	**1258**

Table 2: Overall coverage of the top IPC 2018 planners on all IPC domains, split in the set prior to IPC 2018 and the set used in IPC 2018. Compl: Complementary, PP: Planning-PDBs, Scor: Scorpion.

	Compl.		FDMS		PP	Scor
	1	2	1	2		
Previous IPCs (57)	22	27	17	17	28	**41**
IPC 2018 (10)	2	4	2	3	4	**6**
Sum ()						

Table 3: Overall coverage of the top IPC 2018 planners on all IPC domains, split in the set prior to IPC 2018 and the set used in IPC 2018. Compl: Complementary, PP: Planning-PDBs, Scor: Scorpion.

time, and therefore limiting the time to 900s or more does not stop the heuristic computation early and hence does not reduce heuristic quality in these cases.

We also observe that there is no significant difference between h^{mf} and h^{sg}. While h^{mf} theoretically dominates any single factor heuristic by definition, evaluating the former can be slightly more expensive. Furthermore, in scenarios where at the end, there is one (large) factor that covers many variables and many smaller factors that cover few variables (e.g., atomic factors), the large one likely dominates the others, and thus h^{sg} is equally informed as h^{mf}.

For the competition, we decided to use a time limit of 900s and to compute h^{mf} in both planner variants FDMS1 and FDMS2. In addition to pure A* search with the described merge-and-shrink heuristics, they use pruning based on partial order reduction by using strong stubborn sets (Alkhazraji et al. 2012; Wehrle and Helmert 2014). We extended the implementation in Fast Downward with support for conditional effects and with a mechanism that disables pruning if, after the first 1000 expansions, only 1% or fewer states have been pruned. Both planners further use pruning based on structural symmetries (Shleyfman et al. 2015) by using the DKS algorithm (Domshlak, Katz, and Shleyfman 2012). Finally, after translating PDDL with the translator of Fast Downward (Helmert 2009), we also post-process the resulting SAS$^+$ representation using the implementation of h^2 mutexes by Alcázar and Torralba (2015).

Post-IPC Discussion

Our merge-and-shrink planners finished seventh and eighth out of 16 entries. The winning planner, Delfi 1, and the sixth placed Delfi 2, both are portfolios that contain both merge-and-shrink planners of this submission. Furthermore, the post-IPC analysis of Delfi shows that FDMS2 is necessary to be included to achieve oracle performance over all component planners (Katz et al. 2018). Leaving these portfolios aside, the other entries above FDMS are three PDB-based planners (Complementary1 by Franco et al., 2018, Complementary2 by Franco, Lelis, and Barley, 2018, and Planning-PDBs by Martinez et al., 2018) and a planner based on Cartesian abstractions and PDBs (Scorpion by Seipp, 2018). For the following analysis, we ran these planners under IPC conditions.

Table 2 shows coverage of all mentioned planners, aggregating domains of all previous IPCs and domains of IPC 2018. Tables 4 and 5 show the full domain-wise results. It is clear that Scorpion was the state of the art planner prior to the competition, leaving both the three PDB-based and the two merge-and-shrink-based planners behind by a large margin. The more notable are the results of the IPC 2018, where the PDB-based planners are clearly ahead of Scorpion, which is very closely followed by our two merge-and-shrink planners. The main reason is that Scorpion struggles on AGRICOLA and PETRI-NET-ALIGNMENT, where the former seems well-suited for merge-and-shrink planners, and the latter for planners using symbolic heuristics or search (the baseline planner using symbolic search finishes above Scorpion).

Furthermore, while absolute coverage is certainly a useful performance indicator, it is problematic for the domains of older IPCs because they contain a largely varying number of tasks. In more recent editions of IPCs, domains have the same number of tasks and hence only comparing total coverage makes more sense in these cases. For completeness and as an alternative performance indicator, we therefore also count the number of domains where each planner achieves the highest coverage. Table 3 shows that Scorpion is the winner in that category for both the previous and the new domains, however the distance to the competitors is less pointed out for the IPC 2018 domains.

References

Alcázar, V., and Torralba, Á. 2015. A reminder about the importance of computing and exploiting invariants in planning. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 2–6. AAAI Press.

Alkhazraji, Y.; Wehrle, M.; Mattmüller, R.; and Helmert, M. 2012. A stubborn set algorithm for optimal planning. In De Raedt, L.; Bessiere, C.; Dubois, D.; Doherty, P.; Frasconi, P.; Heintz, F.; and Lucas, P., eds., *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, 891–892. IOS Press.

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS$^+$ planning. *Computational Intelligence* 11(4):625–655.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1:269–271.

	Compl.		FDMS		PP	Scor
	1	2	1	2		
airport (50)	27	28	27	27	27	**37**
barman-opt11-strips (20)	**8**	**8**	**8**	**8**	**8**	**8**
barman-opt14-strips (14)	**3**	**3**	**3**	**3**	**3**	**3**
blocks (35)	**30**	**30**	28	28	**30**	**30**
childsnack-opt14-strips (20)	0	0	**6**	**6**	4	0
depot (22)	7	8	9	12	8	**14**
driverlog (20)	14	**15**	13	13	**15**	**15**
elevators-opt08-strips (30)	23	**25**	19	17	**25**	**25**
elevators-opt11-strips (20)	18	**19**	16	15	**19**	**19**
floortile-opt11-strips (20)	**14**	**14**	9	10	**14**	8
floortile-opt14-strips (20)	17	**20**	9	11	**20**	8
freecell (80)	35	33	21	22	38	**70**
ged-opt14-strips (20)	**20**	**20**	19	19	**20**	**20**
grid (5)	**3**	**3**	2	**3**	**3**	**3**
gripper (20)	**20**	**20**	**20**	**20**	**20**	8
hiking-opt14-strips (20)	17	**20**	19	19	**20**	16
logistics00 (28)	21	22	20	20	22	**25**
logistics98 (35)	6	6	5	5	6	**11**
miconic (150)	108	105	84	76	106	**143**
movie (30)	**30**	**30**	**30**	**30**	**30**	**30**
mprime (35)	24	25	23	23	24	**31**
mystery (30)	15	15	17	16	16	**19**
nomystery-opt11-strips (20)	**20**	**20**	**20**	**20**	**20**	**20**
openstacks-opt08-strips (30)	**30**	**30**	24	23	**30**	23
openstacks-opt11-strips (20)	**20**	**20**	18	18	**20**	18
openstacks-opt14-strips (20)	**14**	**14**	5	3	**14**	3
openstacks-strips (30)	10	**11**	7	7	**11**	9
parcprinter-08-strips (30)	25	24	27	26	21	**30**
parcprinter-opt11-strips (20)	17	16	**20**	19	19	**20**
parking-opt11-strips (20)	1	1	2	1	4	**8**
parking-opt14-strips (20)	3	4	5	4	4	**8**
pathways-noneg (30)	**5**	**5**	**5**	**5**	**5**	**5**
pegsol-08-strips (30)	29	29	**30**	29	29	**30**
pegsol-opt11-strips (20)	19	19	**20**	19	19	**20**
pipesworld-notankage (50)	16	24	21	20	20	**26**
pipesworld-tankage (50)	16	**18**	**18**	**18**	17	**18**
psr-small (50)	**50**	**50**	**50**	**50**	**50**	**50**
rovers (40)	**14**	13	9	10	13	11
satellite (36)	**11**	10	9	**11**	**11**	9
scanalyzer-08-strips (30)	12	12	17	**18**	14	**18**
scanalyzer-opt11-strips (20)	9	9	13	14	11	**15**
sokoban-opt08-strips (30)	**30**	28	**30**	**30**	**30**	**30**
sokoban-opt11-strips (20)	**20**	**20**	**20**	**20**	**20**	**20**
storage (30)	16	15	**18**	**18**	15	16
tetris-opt14-strips (17)	11	13	13	12	**14**	13
tidybot-opt11-strips (20)	**18**	17	11	13	17	**18**
tidybot-opt14-strips (20)	14	13	3	5	13	**15**
tpp (30)	12	**15**	9	8	8	8
transport-opt08-strips (30)	**14**	**14**	11	12	**14**	**14**
transport-opt11-strips (20)	10	10	6	8	11	**13**
transport-opt14-strips (20)	9	9	7	7	9	**10**
trucks-strips (30)	14	11	9	10	12	**16**
visitall-opt11-strips (20)	12	**18**	9	9	**18**	17
visitall-opt14-strips (20)	6	**15**	4	4	**15**	13
woodworking-opt08-strips (30)	26	28	**30**	**30**	27	**30**
woodworking-opt11-strips (20)	**20**	19	**20**	**20**	19	**20**
zenotravel (20)	**13**	**13**	12	12	**13**	**13**
Sum previous IPCs (1667)	1026	1056	939	936	1065	**1150**

Table 4: Domain-wise coverage on previous IPC domains.

	Compl.		FDMS		PP	Scor
	1	2	1	2		
agricola-opt18 (20)	10	6	9	**14**	6	2
caldera-opt18-combined (20)	11	**12**	**12**	**12**	**12**	**12**
data-network-opt18 (20)	13	13	10	9	13	**14**
nurikabe-opt18 (20)	12	12	12	12	12	**13**
organic-synthesis-opt18-combined (20)	**13**	**13**	**13**	**13**	**13**	**13**
petri-net-alignment-opt18 (20)	18	18	2	2	**19**	0
settlers-opt18 (20)	9	9	9	8	9	**10**
snake-opt18 (20)	11	**14**	11	11	12	13
spider-opt18 (20)	12	12	11	11	11	**17**
termes-opt18 (20)	**16**	**16**	12	12	**16**	14
Sum IPC 2018 (200)	**125**	**125**	101	104	123	108

Table 5: Domain-wise coverage on the IPC 2018 domains.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2012. Enhanced symmetry breaking in cost-optimal planning as forward search. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 343–347. AAAI Press.

Dräger, K.; Finkbeiner, B.; and Podelski, A. 2009. Directed model checking with distance-preserving abstractions. *International Journal on Software Tools for Technology Transfer* 11(1):27–37.

Fan, G.; Holte, R.; and Müller, M. 2018. Ms-lite: A lightweight, complementary merge-and-shrink method. In de Weerdt, M.; Koenig, S.; Röger, G.; and Spaan, M., eds., *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS 2018)*. AAAI Press.

Fan, G.; Müller, M.; and Holte, R. 2014. Non-linear merging strategies for merge-and-shrink based on variable interactions. In Edelkamp, S., and Barták, R., eds., *Proceedings of the Seventh Annual Symposium on Combinatorial Search (SoCS 2014)*, 53–61. AAAI Press.

Fan, G.; Müller, M.; and Holte, R. 2017. Additive merge-and-shrink heuristics for diverse action costs. In Sierra, C., ed., *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017)*, 4287–4293. AAAI Press.

Franco, S.; Lelis, L. H. S.; Barley, M.; Edelkamp, S.; Martines, M.; and Moraru, I. 2018. The Complementary1 planner in the IPC 2018. In *Ninth International Planning Competition (IPC-9): planner abstracts*, 27–29.

Franco, S.; Lelis, L. H. S.; and Barley, M. 2018. The Complementary2 planner in the IPC 2018. In *Ninth International Planning Competition (IPC-9): planner abstracts*, 30–34.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM* 61(3):16:1–63.

Helmert, M.; Röger, G.; and Sievers, S. 2015. On the expressive power of non-linear merge-and-shrink representations. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 106–114. AAAI Press.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence* 173:503–535.

Katz, M.; Sohrabi, S.; Samulowitz, H.; and Sievers, S. 2018. Delfi: Online planner selection for cost-optimal planning. In *Ninth International Planning Competition (IPC-9): planner abstracts*, 55–62.

Martinez, M.; Moraru, I.; Edelkamp, S.; and Franco, S. 2018. Planning-PDBs planner in the ipc 2018. In *Ninth International Planning Competition (IPC-9): planner abstracts*, 63–66.

Nissim, R.; Hoffmann, J.; and Helmert, M. 2011. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In Walsh, T., ed., *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, 1983–1990. AAAI Press.

Seipp, J.; Pommerening, F.; Sievers, S.; and Helmert, M. 2017. Downward Lab. `https://doi.org/10.5281/zenodo.790461`.

Seipp, J. 2018. Fast Downward Scorpion. In *Ninth International Planning Competition (IPC-9): planner abstracts*, 70–71.

Shleyfman, A.; Katz, M.; Helmert, M.; Sievers, S.; and Wehrle, M. 2015. Heuristics and symmetries in classical planning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3371–3377. AAAI Press.

Sievers, S.; Wehrle, M.; Helmert, M.; Shleyfman, A.; and Katz, M. 2015. Factored symmetries for merge-and-shrink abstractions. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3378–3385. AAAI Press.

Sievers, S.; Wehrle, M.; and Helmert, M. 2014. Generalized label reduction for merge-and-shrink heuristics. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2014)*, 2358–2366. AAAI Press.

Sievers, S.; Wehrle, M.; and Helmert, M. 2016. An analysis of merge strategies for merge-and-shrink heuristics. In Coles, A.; Coles, A.; Edelkamp, S.; Magazzeni, D.; and Sanner, S., eds., *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016)*, 294–298. AAAI Press.

Sievers, S. 2017. *Merge-and-shrink Abstractions for Classical Planning: Theory, Strategies, and Implementation*. Ph.D. Dissertation, University of Basel.

Sievers, S. 2018. Merge-and-shrink heuristics for classical planning: Efficient implementation and partial abstractions. In *Proceedings of the 11th Annual Symposium on Combinatorial Search (SoCS 2018)*, 90–98. AAAI Press.

Torralba, Á. 2015. *Symbolic Search and Abstraction Heuristics for Cost-Optimal Planning*. Ph.D. Dissertation, Universidad Carlos III de Madrid.

Wehrle, M., and Helmert, M. 2014. Efficient stubborn sets: Generalized algorithms and selection strategies. In Chien, S.; Fern, A.; Ruml, W.; and Do, M., eds., *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 323–331. AAAI Press.

SYMPLE: Symbolic Planning based on EVMDDs

David Speck and **Florian Geißer** and **Robert Mattmüller**
University of Freiburg, Germany
{speckd, geisserf, mattmuel}@informatik.uni-freiburg.de

Abstract

SYMPLE is a classical planner which performs bidirectional symbolic search. Symbolic search has proven to be a useful approach to optimal classical planning and is usually based on Binary Decision Diagrams (BDDs). Our approach is based on an alternative data structure called Edge-valued Multi-valued Decision Diagrams (EVMDDs), which have some structural advantages over BDDs.

Introduction

The motivation for SYMPLE originates from two related sources. First, the observation that symbolic planning is a useful and powerful approach to optimal planning. Symbolic planners are similar to their explicit counterparts, but operate on entire sets of states instead of single states. Usually, decision diagrams are used as underlying symbolic data structure. The most popular decision diagrams are Binary Decision Diagrams (BDDs) (Bryant 1986) in symbolic search (Kissmann, Edelkamp, and Hoffmann 2014; Torralba et al. 2014). Second, that an alternative symbolic data structure, the so-called Edge-Multi-Valued Decision Diagrams (EVMDDs) (Lai, Pedram, and Vrudhula 1996; Ciardo and Siminiceanu 2002), were successfully used in planning with state-dependent action costs (Geißer, Keller, and Mattmüller 2015; 2016; Mattmüller et al. 2018). In BDD-based symbolic planning, each BDD represents a set of states. Multiple BDDs are required to encode at what cost the states are reachable. In contrast, EVMDDs can be used as underlying data structure to encode the costs of states in the same diagram which also encodes the reachability of states. Regarding planning tasks with diverse action costs, BDD-based approaches have to bucket over different costs (e.g. g-values), while our EVMDDs-based approach maintains a single decision diagram and can therefore be more compact.

SYMPLE and the underlying concept was original presented in Speck, Geißer, and Mattmüller (2018). The focus of our previous work was on the theory of EVMDD-based symbolic search for (optimal) planning. Representations of state sets, transition relations and new EVMDD operations required for EVMDD-A* were presented. While an empirical evaluation showed that BDD-A* is superior in many tasks with unit-costs, EVMDD-A* outperforms other approaches in domains with state-dependent costs. Unfortunately, the IPC 2018 has no track with state-dependent action costs, yet. Nevertheless, the compactness of SYMPLE can be an advantage over other symbolic approaches for planning tasks with diverse action costs.

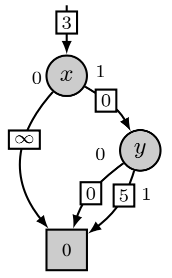
In this paper we will focus on the capabilities and implementation of SYMPLE, a bidirectional symbolic search planner based on EVMDDs. We give a short description of how SYMPLE represents states, costs and actions. EVMDD-A* is described, which is used for the actual search. In addition, the implementation of our planner is presented in detail. Finally, the differences between SYMPLE-1 and SYMPLE-2 are presented, which includes a new method of automated reformulating of planning tasks to simplify grounding.

Planning with EVMDDs

In this chapter we briefly describe how symbolic planning with EVMDDs can be realized.

EVMDD. A possible representations for functions of the form $f : \mathcal{S} \to \mathbb{Q} \cup \{\infty\}$ are *Edge-Valued Multi-Valued Decision Diagrams* (EVMDDs), where $\mathcal{S}$ denotes the set of factored states of a given planning task. An EVMDD is a rooted directed acyclic graph with a dangling incoming edge to the root node. Internal nodes correspond to variables v, and each node has $|\mathcal{D}_v|$ successors with an assigned weight to the edge, where $\mathcal{D}_v$ is the finite domain of variable v. A function can be evaluated by traversing the graph according to the variable assignment and simultaneously adding up the edge weights. The resulting sum is finally the function value for the corresponding variable assignment. An example is shown in Figure 1 where edge labels are written next to the edges and edge weights are written in boxes on the edges.

Symbolic Structures. Symbolic search operates on sets of states by performing operations. Here, states are represented as functions that map each state to the associated cost with which the state can be reached. Note, that a state s which is mapped to ∞ has infinity cost and thus is not reachable. For example, consider Figure 1. The EVMDD $\mathcal{E}$ represents the set of states $S = \{s|s(x) = 1\}$, since all other states are mapped to ∞. At the same time, the EVMDD encodes the cost of these states: $s_1 = \{x \doteq 1, y \doteq 0\}$ has a cost



$\mathbf{x}$	0	0	1	1
$\mathbf{y}$	0	1	0	1
$\mathbf{cost}(\mathbf{x},\mathbf{y})$	∞	∞	3	8

Figure 1: Left: An EVMDD $\mathcal{E}$ which represents a set of states and their costs. Right: The function represented by $\mathcal{E}$.

of 3 while $s_2 = \{x \doteq 1, y \doteq 1\}$ has a cost of 8. Similarly, actions can be represented as so called *transition relations*. Such transition relations are used to generate successor states and their costs which corresponds to applying actions in explicit planning. For more details we refer to (Speck, Geißer, and Mattmüller 2018).

EVMDD-A*. Once states and actions, both associated with costs, are represented as EVMDDs it is possible to start the actual search. Similar to explicit search, the main idea is to represent all promising states which might lead to the goal in an open list. The open list is a single EVMDD encoding the g-values (reachability costs) of each state. In each iteration, states with the smallest g-value are expanded. More specifically, all applicable actions represented as transition relations are applied to these states, resulting in new successor states. These states are again mapped to their g-value and added to the open list. As SYMPLE performs bidirectional search, separate open and closed lists for forward and backward search are maintained. A search step consists either of a backward or a forward search step (and modifies the respective open and closed lists). If a state of the current search is expanded and was already contained in the closed list of the search in the opposite direction, a goal path is found. Its cost is determined by adding the respective EVMDDs. If an optimal plan is desired, search has to continue, until it is proven that there is no cheaper goal path. Finally, plan reconstruction is executed for both directions and the returned plans are combined.

Implementation of SYMPLE

This chapter describes the technical aspects of the SYMPLE planner in detail. Furthermore, we describe the different configuration for each classical track of the IPC 2018. SYMPLE is build on top of the Fast Downward Planning System (Helmert 2006).

Preprocessing. SYMPLE's preprocessing is taken from GAMER (Kissmann, Edelkamp, and Hoffmann 2014) and SYMBA (Torralba et al. 2014), two former winners of IPCs (2008 and 2014). This includes the following procedures:

- GAMER's SAS$^+$ encoding (Kissmann, Edelkamp, and Hoffmann 2014)
- h^2 invariant computation and pruning of spurious actions (Alcázar and Torralba 2015)
- GAMER's and SYMBA's variable ordering (Kissmann and Edelkamp 2011)

Furthermore, in SYMPLE, we combine as many actions as possible into a transition relation, until the representation exceeds 100k nodes. Similarly, invariants are merged together and represented as EVMDDs in order to prune unreachable states.

Search. SYMPLE performs a bidirectional variant of EVMDD-A* (Speck, Geißer, and Mattmüller 2018) using the blind heuristic. At each iteration either a forward or backward search step is performed. In order to decide which direction appears to be more promising, the runtime of the last forward step is compared to the runtime of the last backward step. Conditional effects are encoded by extending the transition relations (Kissmann, Edelkamp, and Hoffmann 2014). To the authors' best knowledge, MEDDLY is currently the only decision diagram library supporting EVMDDs. Thus, the underlying library for EVMDD operations is an extended version of MEDDLY-0.14 (Babar and Miner 2010). The extension consists of operations necessary to realize symbolic planning (Speck, Geißer, and Mattmüller 2018) and the encoding of infinite costs. In order to save memory, we uses the "pessimistic" node deletion policy of MEDDLY, i.e. nodes are removed as soon as they become disconnected.

Track-Configurations. As SYMPLE was developed for optimal planning with state-dependent action costs using an A* variant, the main focus is on optimal planning. Nevertheless, optimal planners can usually be easily modified to participate at other tracks. SYMPLE participates in four different classical planning tracks at the IPC 2018: the optimal, the bounded-cost, the satisficing and the agile track. Generally it would be desirable to use different search techniques and heuristics tailored to the requirements of each track, but we have not yet studied these techniques for EVMDDs. In the following we describe the small changes made to SYMPLE to fit the requirements of the individual tracks.

- **All Tracks.** Bidirectional EVMDD with blind heuristic.
- **Optimal planning.** Once a plan is found, the search is continued until a cheaper plan is found or it is proven that no cheaper plan can exist.
- **Bounded-cost planning.** A plan found is only returned if it costs less than the specified bound.
- **Satisficing planning.** All plans found are returned. The search continues until an optimal plan has been found or the time has elapsed.
- **Agile planning.** As soon as a plan is found, it is returned.

SYMPLE-1 vs. SYMPLE-2

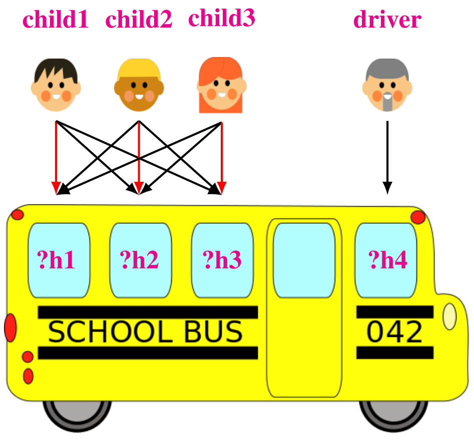
SYMPLE-2 differs from SYMPLE-1 only in the translation step. Both versions are based on the Fast Downward Planning System (Helmert 2006), and thus use the same translation unit to ground the lifted PDDL representation. By using expert knowledge, PDDL tasks are often reformulated beforehand, so that it is easy for planners to ground the planning task. In principle, such reformulations can be performed by the planner, and the IPC 2018 organizers announced that they plan to introduce tasks which are difficult to ground for current planners. SYMPLE-2 tackles the problem of the generation of duplicate redundant actions. Due to symmetries, grounded planning tasks may contain several identical actions that only differ in name, as the order of the action arguments does not affect precondition and effects. Detecting these symmetries and fixing the order of the arguments is the core addition of SYMPLE-2, which results in fewer redundant actions.

To illustrate this, consider a simple planning domain where the goal is to drive children to their school (Figure 2). The corresponding planning instance consists of four humans: one bus driver and three children. Without symmetry detection, grounding action `drive-to-school` results in six different actions: for parameter $?h4$, the only possible substitution is the bus driver, as he is the only one with a `license`. For parameters $?h1$, $?h2$ and $?h3$ however, all combinations of the three children are possible, which leads to $3! = 6$ grounded actions. It is easy to see that these actions are redundant, as they result in the exact same precondition, effect and cost. Although grounded actions can easily be checked for equivalence, this still implies that for n action parameters, an action similar to `drive-to-school` results in $n!$ generated actions and $\mathcal{O}(n!)$ equivalence checks. Our approach now reformulates the lifted PDDL action by introducing an ordering on the objects of symmetric action arguments. This ordering is only used during action generation and automatically discarded afterwards. Therefore, apart from omitted redundant actions, the resulting task is equivalent to a task where no symmetry detection is performed.

To identify such symmetries, we compute graph automorphisms of the induced planning graph of the lifted PDDL representation similar in spirit to Sievers et al. (2017), and Pochter, Zohar, and Rosenschein (2011). After symmetries in the action arguments are detected, we fix the order of these action arguments by introducing a predicate **succ**, as shown in Figure 2. The predicate **succ** is reflexive, since there may be actions where several parameters can be substituted by the same object constant (here: human). Note that the planning graph is not affected by the reformulation, as only redundant actions are discarded.

Conclusion

SYMPLE is a new planner based on symbolic search, which is focused on optimal planning with state-dependent action costs. The main objective was to determine the strengths and weaknesses of the system in comparison to other state-of-the-art planning systems. The benchmark set of the optimal



```
;;;;;;;;;;;;;;  Domain  ;;;;;;;;;;;;;;;
(define (domain bus)
 (:types human location)
 (:constants bus school - location)
 (:predicates
  (at ?l - location ?h - human)
  (license ?h - human)
  (succ ?h1 ?h2 - human))

 (:action drive-to-school
  :parameters (?h1 ?h2 ?h3 ?h4 - human)
  :precondition (and (not (= ?h1 ?h2))
   (not (= ?h1 ?h3)) (not (= ?h1 ?h4))
   (not (= ?h2 ?h3)) (not (= ?h2 ?h4))
   (not (= ?h3 ?h4))
   (at bus ?h1) (at bus ?h2)
   (at bus ?h3) (at bus ?h4)
   (license ?h4)
   (succ ?h1 ?h2) (succ ?h2 ?h3))
   :effect (and (at school ?h1)
   (at school ?h2) (at school ?h3)
   (not (at bus ?h1)) (not (at bus ?h2))
   (not (at bus ?h3)))))

;;;;;;;;;;;;;;  Problem  ;;;;;;;;;;;;;;;
(define (problem bus-3-1)
 (:domain bus)
 (:objects
  driver child1 child2 child3 - human)
 (:init (at bus driver)
  (at bus child1)
  (at bus child2)
  (at bus child3)
  (license driver)
  (succ driver driver) (succ driver child1)
  (succ driver child2) (succ driver child3)
  (succ child1 child1) (succ child1 child2)
  (succ child1 child3) (succ child2 child2)
  (succ child2 child3) (succ child3 child3))
 (:goal (and
  (at school child1)
  (at school child2)
  (at school child3))))
```

Figure 2: A domain and instance description where six redundant action are generated by grounding the task. The **succ** predicate is automatically added. The reformulation of the task contains only one action.

track of the IPC 2018 consists of five tasks with unit costs and five task with constant costs.[1] In addition, a strong focus was put on conditional effects, which are supported by SYMPLE, but not highly optimized. The plan reformulation used in SYMPLE-2 detected redundant actions in the "Organic Synthesis" domain (Matloob and Soutchanski 2016). However, the IPC organizer provided an alternative version of this domain, which was rewritten by hand and easier to solve. In summary, SYMPLE performed as expected and was competitive in some domains of the optimal track. The high focus on conditional effects and few domains with diverse action costs did not favor our system.

Acknowledgments

David Speck was supported by the German National Science Foundation (DFG) research unit FOR 1513 on Hybrid Reasoning for Intelligent Systems (`http://www.hybrid-reasoning.org`).

References

Alcázar, V., and Torralba, Á. 2015. A Reminder about the Importance of Computing and Exploiting Invariants in Planning. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 2–6. AAAI Press. Palo Alto, CA, USA.

Babar, J., and Miner, A. 2010. MEDDLY: Multi-terminal and edge-valued decision diagram library. In *Proceedings of the Seventh International Conference on the Quantitative Evaluation of Systems (QEST 2010)*, 195–196. IEEE Computer Society. Los Alamitos, CA, USA.

Bryant, R. E. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers* 35(8):677–691.

Ciardo, G., and Siminiceanu, R. 2002. Using edge-valued decision diagrams for symbolic generation of shortest paths. In Aagaard, M. D., and O'Leary, J. W., eds., *Proceedings of the Fourth International Conference on Formal Methods in Computer-Aided Design (FMCAD 2002)*, 256–273. Berlin, Heidelberg, Germany: Springer.

Geißer, F.; Keller, T.; and Mattmüller, R. 2015. Delete relaxations for planning with state-dependent action costs. In Yang, Q.; Kong, H.; and Wooldridge, M., eds., *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)*, 1573–1579. AAAI Press. Palo Alto, CA, USA.

Geißer, F.; Keller, T.; and Mattmüller, R. 2016. Abstractions for planning with state-dependent action costs. In Coles, A.; Coles, A.; Edelkamp, S.; Magazzeni, D.; and Sanner, S., eds., *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016)*, 140–148. AAAI Press. Palo Alto, CA, USA.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Kissmann, P., and Edelkamp, S. 2011. Improving Cost-Optimal Domain-Independent Symbolic Planning. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2018).* AAAI Press. Menlo Park, CA, USA.

Kissmann, P.; Edelkamp, S.; and Hoffmann, J. 2014. Gamer and Dynamic-Gamer: Symbolic search at IPC 2014. In *Eighth International Planning Competition (IPC 2014)*, 77–84.

Lai, Y.; Pedram, M.; and Vrudhula, S. B. K. 1996. Formal verification using edge-valued binary decision diagrams. *IEEE Transactions on Computers* 45(2):247–255.

Matloob, R., and Soutchanski, M. 2016. Exploring Organic Synthesis with State-of-the-Art Planning Techniques. In *Proceedings of the ICAPS Workshop on Scheduling and Planning Applications Workshop (SPARK)*, 52–61.

Mattmüller, R.; Geißer, F.; Wright, B.; and Nebel, B. 2018. On the Relationship Between State-Dependent Action Costs and Conditional Effects in Planning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI 2018).* AAAI Press. Menlo Park, CA, USA.

Pochter, N.; Zohar, A.; and Rosenschein, J. S. 2011. Exploiting problem symmetries in state-based planners. In Burgard, W., and Roth, D., eds., *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2011)*, 1004–1009. AAAI Press.

Sievers, S.; Röger, G.; Wehrle, M.; and Katz, M. 2017. Structural symmetries of the lifted representation of classical planning tasks. In *ICAPS 2017 Workshop on Heuristics and Search for Domain-independent Planning*, 67–74.

Speck, D.; Geißer, F.; and Mattmüller, R. 2018. Symbolic Planning with Edge-Valued Multi-Valued Decision Diagrams. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 250–258.

Torralba, Á.; Alcázar, V.; Borrajo, D.; Kissmann, P.; and Edelkamp, S. 2014. SymBA*: A symbolic bidirectional A* planner. In *Eighth International Planning Competition (IPC 2014)*, 105–109.

[1]There are two domains that have unit costs in their original form and constant costs in a provided alternative form.