# An Integrated Toolkit for Modern Action Planning

Miriam Bützken, Stefan Edelkamp, Abdelaziz Elalaoui, Kenneth Kahl, Rachid Karmouni, Roman Klinger, Khalid Lahiane, Andrea Matuszewski, Tilman Mehler, Mohammed Nazih, Michael Nelskamp, and Arne Wiggers

Computer Science Department
University Dortmund

**Abstract.** In this paper we introduce to the architecture and the abilities of our design and analysis workbench for modern action planning. The toolkit provides automated domain analysis tools together with PDDL learning capabilities. New optimal and suboptimal planners extend state-of-the-art technology. With the tool, domain experts assist solving hard combinatorial problems. Approximate or incremental solutions provided by the system are supervised. Intermediate results are accessible to improve domain modeling and to tune exploration in generating high quality plans, which, in turn, can be bootstrapped for domain inference.

## 1 Introduction

The research focus in AI planning shifts towards practical acceptance, with problem scenarios for transportation and routing, elevator scheduling, space applications, game playing, avionics, handheld setup, software verification, diagnosis in power networks, oil pipelining, etc., as indicated by the range of benchmarks [18] currently used in international planning competitions [30, 1, 28, 10].

Surely pushed by the success of the competitions the efficiency of planning technology is continuously raising. Many recent planning systems quickly solve rather complex planning problems. All that is missing are intelligent tools that take over a large amount of design process to quickly generate planning domain models and to maintain state-of-the-art planners.

In domain-independent planning all aspects have to be automated. For most existing planning systems, however, decisions that are inferred automatically can be improved by only a limited amount user guidance. As a consequence, we have designed a toolkit that provides options to access and modify the outcome provided by static analyzers, together with a visualization and animation of computed plans. Moreover, the workbench includes tools to ease domain modeling, that call and tune existing and new planners and different wrappers, that divide and analyze the planning process. The workbench is capable to handle large fragments of current PDDL, including ADL, timed initial literals and derived predicates.

The exposition starts with the domain analysis, converting uninstantiated PDDL input files to fully instantiated and annotated PDDL, including the inference of multi-variate variables domains, as used in many recent planning modules. We then describe the design capabilities of the system and propose different extensions to optimal and suboptimal planning. Last but not least, we indicate our approach to visualize plans.

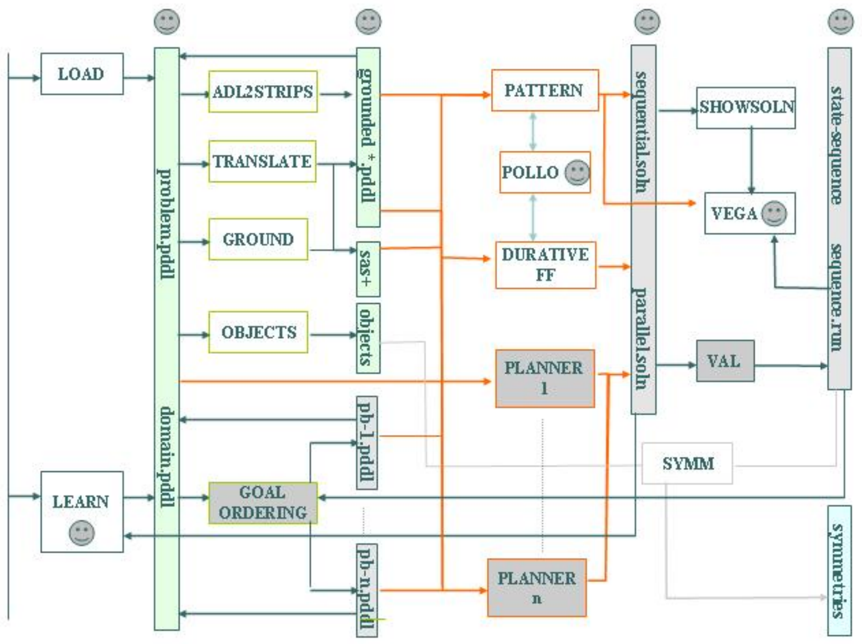


**Fig. 1.** Architecture of *ModPlan*.

## 2 Architecture of the Workbench

The *ModPlan* workbench is a fully integrated environment for the Windows operating system, with a simple adaption to Unix/Linux. It has a layered, recursive architecture. Optional user interaction is available via the top-level GUI, additional shell inputs, manipulated plan animation (Vega) as well as XML editing (Pollo) and direct file accesses. It supports an increasing fraction of PDDL expressiveness. While the learning module operates on simple timed actions and limited ADL expressions; precompilation, planning, symmetry detection all operate on full PDDL.

Figure 1 gives a rough overview of the architecture for the system. The *smilies* highlight, where the user may interact with the system. Domains are either inferred in a learning module or read from disk. The user can choose among different grounding options and start planning either from the original or from the instantiated files. Visualization for timed-step plans and animations of plan happenings have been included at the back-end. As additional options a goal ordering wrapper and an object symmetry analysis module for generated plans is provided. The two new planners that are featured are: *DurativeFF*, a satisficing planner with full PDDL expressiveness based on heuristic search, and *PatternPlan*, an optimal propositional planner based on pattern databases.

## 3 Domain Instantiation and Encoding

PDDL [12] is a an expressive domain description language hierarchy. In Level 1, typed domain descriptions and ADL expressivity are founded. In Level 2, mixed propositional and numerical problems can be devised and optimized. In Level 3, temporal planning is addressed: action generation as in planning is combined with action arrangement as in scheduling. Recent extensions to PDDL [9] include domain axioms in form of derived predicates, and restricted use of exogenous events in form of timed initial literals.

PDDL planning tasks usually consist of two different ASCII files: the problem independent *domain file* and the instance specific *problem file*. In the domain file, parametric predicates and functions are declared, as well as operators with precondition and effect lists. In the problem file, we find the definition of objects and their types, the initial state, the definition of the goal predicate, and the plan object function to be optimized.

*Grounding* is the process of finding (supersets of) reachable actions, facts and fluents by instantiating operators, predicates and functions with the objects that are specified in the problem description. Most current planners perform some form of grounding to apply planning state space exploration. We integrated three technologies: *translate*, the instantiation process that is implemented in the planner *Fast-Downward* [15], *adl2strips*, the domain translation that comes with the planner FF [19], and *ground*, the preprocessing result that is implicit in the planner MIPS [6]. While *translate* and *adl2strips* are capable of handling rather complex ADL expression in propositional planning, *ground* can deal with metric and durative domain formulations. As *adl2strips* provides grounded PDDL Level 1 syntax, with *ground* we contribute a tool to generate grounded PDDL, Level 1-3 syntax.

A pure propositional encoding can have efficiency drawbacks during the exploration. A multivariate representation for the atom set is often preferable. In the *SAS*$^+$ *encoding* [15], groups of mutually exclusive atoms are generated. This encoding serves as an optional input for existing planners that can exploit this facility. For the workbench we choose the multivariate encoding of *translate* and *ground*. The output file format for this domain analysis step is a Lisp-like representation of the set of reachable atoms and their partitioning into SAS$^+$ variable domains.

## 4 Symmetry Detection

Unless handled properly, uncaught symmetries cause an explosion in the search space of the planners. Detecting symmetries fully automatically is not easy, since it links to the computational hard problem of finding graph isomorphisms. Note that the general problem is not completely classified. It is expected not to be NP-complete [32]. Some complexity theoretic insights are: if GI is NP-complete then $\Sigma_2 = \Pi_2$, $\overline{GI}$ has an interactive proof system with two communication rounds, and GI has an interactive proof system with perfect zero knowledge property.

To explore a state space with respect to a state symmetry, the exploration engine additionally has to determine a representative element for each equivalence class. In most existing approaches [31, 27, 13], symmetries are fully specified by the user. Some planners apply automated *object symmetries* [11]. Two objects are symmetrical, if they can be changed in the current state without affecting solvability and optimality in the remaining planning problem. Such symmetries appear frequently in benchmark problems. For object symmetries, there is the additional problem that symmetries may vanish or reappear during the exploration.

The important observation is that the domain file is transparent to object transpositions, so that symmetry detection is possible only with respect to the current state and the goal specification. For $n$ objects we have $n!$ possible object permutations. Taking into account all type information reduces the number of all possible permutation to $n!/(t_1! \cdot \ldots \cdot t_k!)$. where $t_i$ is the number of objects with type $i$, $i \in \{1, \ldots, k\}$. To reduce the number of potential symmetries to a tractable size, we might further restrict symmetries to object transpositions, for which we have at most $n(n-1)/2$ candidates. Including type information this number further reduces to $\sum_{i=1}^{k} t_i(t_i - 1)/2$. For forward chaining planners we can reduce this set of possible object symmetries to the ones that are valid in the goal. Symmetry detection in our workbench is based on plan happenings. The systems induces a sequence of object symmetries, while the user may add complex symmetries that the system connot infer.

## 5 Goal Ordering and Planner Wrapping

*Goal ordering* is an essential part of accelerating solution finding in larger planning problems. It yields a *goal agenda* [23] denoting, in which order goal predicates and conditions should be established. It consists of a sequence of goals atoms $\mathcal{G}_1, \ldots, \mathcal{G}_k$ with $\mathcal{G}_i \subset \mathcal{G}_{i+1}$, $i \in \{1, \ldots, k-1\}$. Using a goal ordering, the planning process for any *Planner* can be reduced as follows. Set $\mathcal{I}_1 = \mathcal{I}$ and $\mathcal{I}_{i+1} = \textit{Planner}(\mathcal{I}_i, \mathcal{G}_i)$, $i \in \{0, \ldots, k-1\}$. It has laid the basis of the dramatic impact of constraint partitioned problem solving in SGPlan [4]. The idea shares similarities with *macro problem solving* [24], in which operator sequences are learned to be retrieved for sequentialized goal satisfaction. Note that some goal orderings can yield fairly long plans [26].

We have implemented an approximation $\preceq_h$ of $\preceq_r$ as an individual static analysis option. The algorithm prompts the outcome of this phase to the user so that he can refine the induced goal agenda. If the agenda is fixed, a sequence of PDDL files is generated wrapping any selected planning module. As finding the *best* goal ordering is hard, we leave it to the domain expert to adjust the approximated one. The inference $\mathcal{I}_{i+1}$ given $\mathcal{I}_i$ is transparent to the expert, as it simulates plan execution within the validator VAL (cf. section on plan validation and visualization). Using our extension to flush state sequences according to plan happenings, we apply VAL to $\mathcal{I}_i$ and write the result $\mathcal{I}_{i+1}$ together with goal $\mathcal{G}_{i+1}$ back to disk.

## 6 Operator Dependency and Parallel Plans

There are two main optimization metrics in planning, the *plan length* (number of actions) and its *makespan* (minimum parallel execution time). For propositional planning, each operator is asserted to a duration of 1, so that the latter corresponds to the minimal parallel plan length. To express parallel plans, a *mutex relation* of operators has to be provided, which in the case of metric planning, naturally extends to the standard mutex relation for STRIPS. It additionally includes conflicts between numerical variables. Two grounded operators are *mutex*, if one of the following three conditions holds:

1. The precondition list of one operator has a non-empty intersection with the add or delete lists of the other one.
2. The head of a numerical modifier of one operator is contained in some condition of the other one.
3. The head of the numerical modifier of one operator is contained in the body of the modifier of the other one.

For temporal planning with *start*, *invariant* and *end* modalities, we have eight different mutexes, i.e. *start/start*, *start/invariant*, *start/end*, *invariant/start*, *invariant/end*, *end/start*, *end/invariant* and *end/end*. If there is more than one conflict for one operator pair, we have to compute the maximum value derived for the individual conflicts.

The semantics of operator duration in PDDL2.1 demands a slack of $\epsilon$ time steps between any two happenings that are dependent. The default value for $\epsilon$ is 0.01. The idea is that if a proposition or a numerical quantity is accessed by different actions, some time for resolving has to pass.

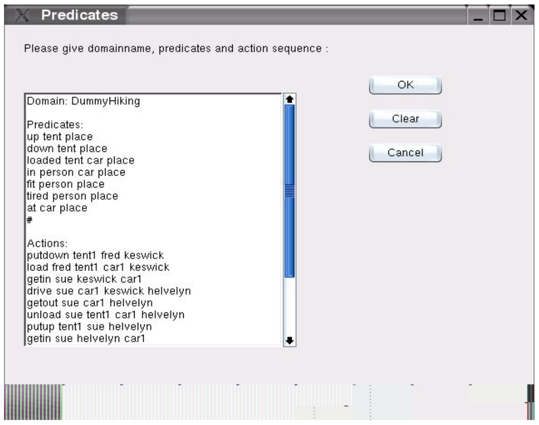
Optimizing of plans without precedence ordering is involved [2], since computing the makespan for a set of operators is NP-complete. However, given a sequence of operators in a plan, a precedence ordering among them, an optimal parallel plan that respects the given timing constraints, is polynomially solvable. Moreover, with critical path scheduling (PERT) such a plan can be computed in optimal time [6]. The approach extends to timed initial literals and action execution time windows. Operator dependency induces a partial ordering in a sequence of actions. In order to derive posterior schedules of sequential plans in temporal planning, pairwise dependencies are precomputed and made accessible to the user with each computed plan. He may add or delete precedence constraints before scheduling is performed.

## 7 Domain Inference

*Learning* PDDL domain specifications from plan traces without any domain expert knowledge is a computationally challenging and practically almost infeasible task. To infer operators within a PDDL domain description, the PDDL inference mechanism needs supervision of the user. We newly implemented a supervised learning algorithm to interactively infer the PDDL domain description.

We assume that a domain expert tries to infer a valid domain description from a set of operators that form a valid plan. This plan can be generated in a previous run of a planner. If we start from scratch, an initial sequence of operators has to be provided manually. The additional inputs of the algorithm are the prefixes of domain and problem file, namely the declaration of objects, and the set of predicates. If not already present, object type information may interactively be attached.

Given the set of operators in a valid plan (cf. Figure 2), the designer is confronted with choice boxes on how the set of preconditions and effects of an operator to be inferred are composed (cf. Figures 3). The supervised PDDL learning mechanism selects


Predicates

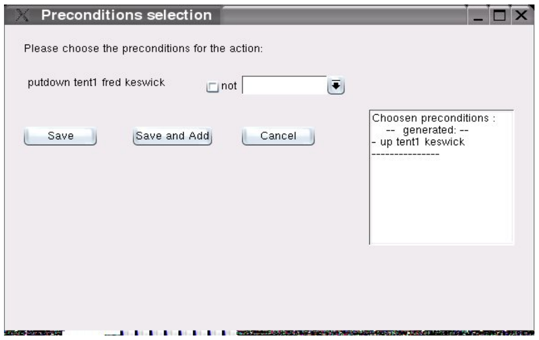
Please give domainname, predicates and action sequence :

```
Domain: DummyHiking

Predicates:
up tent place
down tent place
loaded tent car place
in person car place
fit person place
tired person place
at car place
#

Actions:
putdown tent1 fred keswick
load fred tent1 car1 keswick
getin sue keswick car1
drive sue car1 keswick helvelyn
getout sue car1 helvelyn
unload sue tent1 car1 helvelyn
putup tent1 sue helvelyn
getin sue helvelyn car1
```

**Fig. 2.** Input for the PDDL learning task.


Preconditions selection

Please choose the preconditions for the action:

putdown tent1 fred keswick

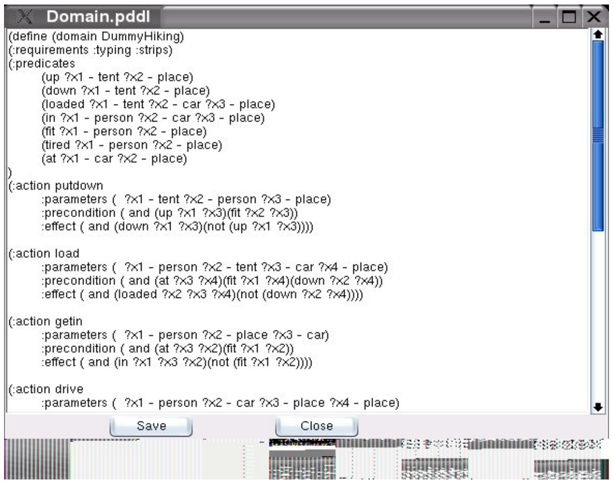
```
Choosen preconditions :
   --  generated: --
- up tent1 keswick
---------------
```

**Fig. 3.** Supervised input of action preconditions.

the operator to infer next and steadily reduces the set of options until a domain model has been established. For long plans, the inference task is almost fully automatic. The learning algorithm underneath provides an improved implementation of the *Opmaker* algorithm [29]. One of the distinctive features is the option to attach durations to actions and to allow *incremental learning*, as the output of a planner can be used as input for another inference step.

## 8 Suboptimal and Optimal Planning

As the intermediate results produced by the grounding procedures are valid PDDL files, any planner can be used as a back-end solver. To push the development of planning technology and to highlight the applicability of our knowledge engineering tool, we enlarged the set of existing planners by two different contributions: one optimal propositional planner and one suboptimal metric and temporal solving module. In our workbench, we have included an interface to both planners.



```
(define (domain DummyHiking)
(:requirements :typing :strips)
(:predicates
        (up ?x1 - tent ?x2 - place)
        (down ?x1 - tent ?x2 - place)
        (loaded ?x1 - tent ?x2 - car ?x3 - place)
        (in ?x1 - person ?x2 - car ?x3 - place)
        (fit ?x1 - person ?x2 - place)
        (tired ?x1 - person ?x2 - place)
        (at ?x1 - car ?x2 - place)
)
(:action putdown
        :parameters (  ?x1 - tent ?x2 - person ?x3 - place)
        :precondition ( and (up ?x1 ?x3)(fit ?x2 ?x3))
        :effect ( and (down ?x1 ?x3)(not (up ?x1 ?x3))))

(:action load
        :parameters (  ?x1 - person ?x2 - tent ?x3 - car ?x4 - place)
        :precondition ( and (at ?x3 ?x4)(fit ?x1 ?x4)(down ?x2 ?x4))
        :effect ( and (loaded ?x2 ?x3 ?x4)(not (down ?x2 ?x4))))

(:action getin
        :parameters (  ?x1 - person ?x2 - place ?x3 - car)
        :precondition ( and (at ?x3 ?x2)(fit ?x1 ?x2))
        :effect ( and (in ?x1 ?x3 ?x2)(not (fit ?x1 ?x2))))

(:action drive
        :parameters (  ?x1 - person ?x2 - car ?x3 - place ?x4 - place)
```

**Fig. 4.** Inferred domain description.

Current STRIPS planners are diverse in structure. While most suboptimal planners use heuristic and/or local search [4], optimal planners range from satisfiability solving [21], planning graph approaches [3] to integer programming [22] and heursitic search [14]. With *Pattern-Plan* we contribute an optimal (either symbolic or explicit) pattern database planner.

In planning with pattern databases [5], the automated selection of possible abstraction functions to yield informative pattern databases is a hard combinatorial task. This is especially true for the creation of disjoint databases [25], in which operator projections are void in all but one abstraction. In planning, pattern database abstraction are most effective if they consider SAS$^+$ groups in common. There are different bin-packing approximation algorithms [5] that infer a partitioning of variable groups before constructing the databases. The maximum size of a pattern database is bounded by the multiplication of the cardinalities of the selected variable domains. We are currently working at a genetic algorithm to improve the first partition of groups. Moreover, we allow user guidance to refine the approximated, disjoint partitioning into planning pattern databases as proposed in the inference module is helpful. We allow these interactions in an XML front end.

For *metric planning*, we implemented the PERT scheduling approach [6] on top of Metric-FF [17] to generate parallel plans. We use posterior scheduling for complete plans as well as for partial and relaxed ones. As *temporal planning* leads to plan schedules rather than plan sequences, we have successfully lifted Metric-FF to *Durative-FF*, a new planner that is capable of parsing PDDL2.1 Level 3 and to apply PERT scheduling on top of the set of generated plans. First results on the set of competition planning benchmarks are promising. As the approach generalized to *timed initial literals* in form of action execution time windows [7], we currently include expressiveness with this respect. By the choice of the underlying planner, however, generalizing the relaxed planning heuristic to *non-linear tasks* [8] is involved.

## 9 Plan Validation and Visualization

For *plan validation* we have included VAL [20], as provided by the Strathclyde planning group. The main capabilty of VAL is the simulation, i.e. the execution of almost any plan in PDDL syntax. It has recently been extended to capture *continous effects*, *exogenous events* and *processes*.

For *plan visualization* we integrated the animation system Vega [16] to the workbench, allowing a magnified view to an arbitrary part of the plan. Vega itself is implemented as a Client-Server architecture that runs an annotated algorithm on the server side to be visualized on the client side in a Java frontend. Annotation are visualization requests that (minorly) extend the existing source code by (simple) commands like *send point*$(x, y)$.

In the system, visualization objects can be associated with hierarchical structured identifiers. The client is used both as the user front-end and the visualization engine. Thus, it allows server and algorithm selection, input of data, running and stopping algorithms, and customization of the visualization. It can be used to manipulate scenes with hierarchically named objects, view algorithm lists at the server and display algorithm information, control the algorithm execution using a VCR-like panel or the execution browser, adjust view attributes directly or by using the object browser. It can display several algorithms simultaneously in multiple scenes and open different views for a single scene, load and save single scenes, complete runs, and attribute lists and export scenes in *xfig* or *gif* format.

Vega allows both *on-line* and *off-line* presentations. The main purpose of the server is to make algorithms accessible through TCP/IP. The server is able to receive commands from multiple clients at the same time. It allows the client to choose from the list of available algorithms, to retrieve information about the algorithm, to specify input data, to start it and to receive output data. The server maintains a list of installed algorithms. This list may be changed without stopping and restarting the server.

*Gantt charts* are plots for temporal plans, in which a horizontal open oblong is drawn against each activity indicating estimated duration. To access precise e.g. temporal information on the operators representatives can selected with the mouse. Our visualization module depicts the Gantt chart of plans in *competition format*. An example is provided in Figure 5. As the essence of the task is translating temporal operators into rectangles and associated text labels we compile planner results into Vega scenes.

Domain-specific visualization is more challenging. Based on flushing sequential operator trails together with their corresponding state sequence, we have written *instance-independent* visualizations for many competition domains. Propositional atoms are illustrated by showing or hiding an image at a certain location, while numerical quantities such as fuel are expressed by using scalable graphical items. The according figures for displaying domain objects are collected with an image web search engine. Previously, the visualizer worked in cooperation with a specialized planner extension that wrote (sequential) plans together with associated state information to disk. In the workbench, we additionally exploit VAL's capabilities to enable the animation of parallel and temporal plan execution. For this purpose, we extended the validator to flush a sequence of states at each *happening* together with its time stamp in form a Vega run to disk.

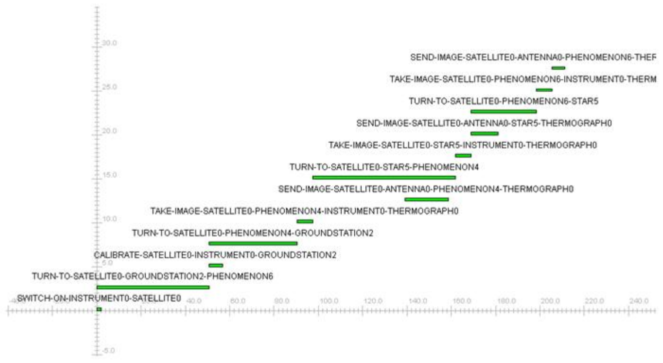


**Fig. 5.** Visualisation of a plan in Gantt chart format.

## 10 Conclusion

With *ModPlan* we have implemented an integrated environment including domain modeling, static analyzes, plan finding, plan validation, and plan visualization. Except for PDDL inference, we allow but do not rely on expert knowledge. We simplified the domain design by offering an interactive operator learning module. Static analyzers yield grounded and annotated PDDL, to be exploited by different plan engines. Additionally, we have contributed two planners: a sub-optimal planner that covers a large fraction of PDDL2.2, and an optimal planner for propositional problems with SAS$^+$ annotations.

There are many other challenges that can be made accessible for improved approximation results in domain-independent planning. For example, we have not realized the usage of *control rules* to prune exploration especially in forward-chaining planners.

**Acknowledgments** The work is supported by DFG in projects Ed 74/2 and 74/3.

## References

1. F. Bacchus. The AIPS'00 planning competition. 22(3):47–56, 2001.
2. C. Bäckström. Computational aspects of reordering plans. *Journal of Artificial Intelligence Research*, 9:99–137, 1998.
3. A. Blum and M. L. Furst. Fast planning through planning graph analysis. In *IJCAI*, pages 1636–1642, 1995.
4. Y. Chen and B. W. Wah. Subgoal partitioning and resolution in planning. In *Proceedings of the International Planning Competition*, 2004.
5. S. Edelkamp. Planning with pattern databases. In *ECP*, pages 13–24, 2001.
6. S. Edelkamp. Taming numbers and durations in the model checking integrated planning system. *Journal of Artificial Research*, 20:195–238, 2003.

7. S. Edelkamp. Extended critical paths in temporal planning. In *Proceedings ICAPS-Workshop on Integrating Planning Into Scheduling*, 2004.
8. S. Edelkamp. Generalizing the relaxed planning heuristic to non-linear tasks. In *KI*, 2004. 198–212.
9. S. Edelkamp and J. Hoffmann. PDDL2.2: The language for the classical part of the 4th international planning competition. Technical Report 195, University of Freiburg, 2004.
10. S. Edelkamp, J. Hoffmann, M. Littman, and H. Younes, editors. *Proceedings of the International Planning Competition*. JPL, 2004.
11. M. Fox and D. Long. The detection and exploration of symmetry in planning problems. In *IJCAI*, pages 956–961, 1999.
12. M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 2003.
13. E. Guéré and R. Alami. One action is enough to plan. In *IJCAI*, 2001.
14. P. Haslum and H. Geffner. Admissible heuristics for optimal planning. In *AIPS*, pages 140–149, 2000.
15. M. Helmert. A planning heuristic based on causal graph analysis. In *ICAPS*, pages 161–170, 2004.
16. C. A. Hipke. *Distributed Visualization of Geometric Algorithms*. PhD thesis, University of Freiburg, 2000.
17. J. Hoffmann. The Metric FF planning system: Translating "Ignoring the delete list" to numerical state variables. *Journal of Artificial Intelligence Research*, 20:291–341, 2003.
18. J. Hoffmann, R. Englert, F. Liporace, S. Thiebaux, and S. Trüg. Towards realistic benchmarks for planning: the domains used in the classical part of IPC-4. *Journal of Artificial Intelligence Research*, 2005. Submitted.
19. J. Hoffmann and B. Nebel. Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.
20. R. Howey and D. Long. VAL's progress: The automatic validation tool for PDDL2.1 used in the international planning competition. In *ICAPS-workshop on the Competition*, 2003.
21. H. Kautz and B. Selman. Pushing the envelope: Planning propositional logic, and stochastic search. In *AAAI*, pages 1194–1201, 1996.
22. H. Kautz and J. Walser. State-space planning by integer optimization. In *AAAI*, 1999.
23. J. Koehler and J. Hoffmann. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *Journal of Artificial Intelligence Research*, 12:338–386, 2000.
24. R. E. Korf. Macro-operators: A weak method for learning. *Artificial Intelligence*, 26:35–77, 1985.
25. R. E. Korf and A. Felner. *Chips Challenging Champions: Games, Computers and Artificial Intelligence*, chapter Disjoint Pattern Database Heuristics, pages 13–26. Elsevier, 2002.
26. F. Lin. System R. *AI-Magazine*, pages 73–76, 2001.
27. A. Lluch-Lafuente. Symmetry reduction and heuristic search for error detection in model checking. In *MoChArt*, 2003.
28. D. Long and M. Fox. The 3rd international planning competition: Overview and results. *Journal of Artificial Intelligence Research*, 20, 2003. Special issue on the 3rd International Planning Competition.
29. T. L. McCluskey, N. E. Richardson, and R. M. Simpson. An interactive method for inducing operator descriptions. In *AIPS*, 2002.
30. D. McDermott. The 1998 AI Planning Competition. *AI Magazine*, 21(2), 2000.
31. J. Rintanen. Symmetry reduction for SAT representations of transition systems. In *ICAPS*, 2003.
32. I. Wegener. *Komplexitätstheorie*. Springer, 2003. (in German).