Knowledge Acquisition and Knowledge Engineering in the *ModPlan* Workbench

Stefan Edelkamp and Tilman Mehler
Computer Science Department
Baroper Str. 301
University Dortmund
{stefan.edelkamp,tilman.mehler}@cs.uni-dortmund.de

Abstract

In this paper we present the architecture and the abilities of the *ModPlan* Workbench; an interacive knowledge acquisition and engineering tool for AI planning. It provides automated domain analysis tools together with PDDL learning capabilities. Integrated optimal and suboptimal planning technology extends 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 enhanced plans, which, in turn, can be bootstrapped for domain description inference.

Introduction

Both knowledge acquisition and knowledge engineering for AI planning systems are crucial to improve their effectiveness and to enlarge the application focus in practice.

On the one hand – surely pushed by the series of international planning competitions (McDermott 2000; Bacchus 2001; Long and Fox 2003; Edelkamp *et al.* 2004) – the efficiency of planning technology is continously increasing. Many recent planning systems can quickly solve rather complex planning problems. The improvement of technology is to be observed especially in suboptimal, but is also noticable in optimal planning. On the other hand, as current planning technology is still underrepresented in industrial applications, there is more transfer needed.

Henceforth, 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 currently used in planning competitions (Hoffmann *et al.* 2005).

With recent extensions to PDDL (Fox and Long 2003), namely PDDL2.1, a powerful and flexible specification domain description language has been established. In *propositional planning* typed domain descriptions and ADL expressivity is available (Level 1), *metric planning* (Level 2), mixed propositional and numerical problem instances can be devised and optimized. In *temporal planning* (Level 3), a bridge between planning and scheduling has been established: action generation as in planning is merged with action arrangement as in scheduling.

Additionally, PDDL2.2 (Edelkamp and Hoffmann 2004) provides state completion based domain axioms specified in form of derived predicates, and restricted use of exogenous events in form of timed initial predicates and action time execution windows. The PDDL language definition itself is not likely to be the bottleneck for domain experts to actually integrate planning technology. All that is missing is a set of intelligent tools that take over a large amount of design automation to quickly generate planning domain models and to select and adjust different state-of-the-art solvers. As with the software development cycle, establishing a domain model is an iterative process.

All aspects that are considered in domain-independent planning have to be automated. However, for most existing planning systems, some decisions that are inferred automatically can be improved by limited user guidance. Examples range from improvements to the domain encoding (Edelkamp and Helmert 1999), the inference of domain invariances (Fox and Long 1998; Gerevini and Schubert 2000; Rintanen 2000), observed goal orderings (Koehler and Hoffmann 2000), generic types (Long and Fox 2000), via the choice of specialized exploration algorithms (Long and Fox 2001), pruning options in form of symmetry detection (Fox and Long 1999), automated reduction of operator lists (Haslum and Jonsson 2000), learning of macro operators (Botea *et al.* 2005), to pattern database selection (Edelkamp 2001), hierarchical decomposition (Nau *et al.* 1999), and control rules (Bacchus and Kabanza 2000).

As a consequence, we have designed a planning workbench that provides knowledge acquisition options to access and modify the outcome provided by static analyzers, together with visualization assistence in understanding the validity of computed plans. Moreover, it includes knowledge engineering tools to ease domain modeling. The workbench is capable to handle large fragments of current

PDDL, including ADL expressivity, derived predicates, as well as metric and durative actions.

In this work we first present components we found essential to be accessible for the domain expert. We identify the combinatorial problems that need to be solved, illustrate how intimate knowledge is inferred by the system, and how flexible it is refined by the expert. We start with the domain analysis, converting uninstantiated PDDL input files to fully instantiated and annotated PDDL, including the inference of mutivariate variables domains, as used in many recent planning modules. We then describe the learning capabilities and propose extensions to optimal and suboptimal planning. We show how visualization can be achieved and conclude with a brief summary of the contributions.

Domain Instantiation

PDDL planning task specifications usually consist of two different text files: the uninstanciated 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.

Problem *Grounding* is the process of finding (supersets of) reachable actions, facts and fluents by instantiating operators, predicates and functions with the objects that come with the problem description. Most current planners perform some form of grounding to apply planning state space exploration. Exceptions are planners like TLPlan (Bacchus and Kabanza 2000) that use a distributed state representation based on persistent search trees, or SHOP (Nau *et al.* 1999) that performs unification to bind actions (and methods) to states.

Knowledge Acquisition One of the most efficient methods to infer a grounded representation is *fact space exploration* (Edelkamp and Helmert 1999) that uses a fact queue, in which the initial state is enqueued and where one fact is dequeued and processed at a time. For all operators that include this fact in the precondition list this fact is marked. If all preconditions are marked, the operator is *fired*, enqueuing all its add effects. Constant predicates are detected and removed and constant fluents are substituted to simplify numerical expressions. Quantification, conditional effects, domain axioms in form of derived predicates, numeric conditions and temporal operators have been integrated into this process.

Knowledge Engineering The in- and output format of the instantiation process itself is valid PDDL, so that the grounded representation can be fed into every existing planner by the domain expert. Even though the intermediate results might become large, it is important for the expert to access the simplified state and operator descriptors to understand the working of the plan engines.

To perform the grounding process in our workbench we integrated three different technologies: *translate*, the instantiation process that is provided with the planner *Fast-Downward* (Helmert 2004), *adl2strips*, the domain translation that comes with the planner FF (Hoffmann and Nebel 2001), and *ground*, the preprocessing result that is implicit in the planner MIPS (Edelkamp 2003).

While *translate* and *adl2strips* are capable of handling 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. We newly implemented a translator that converts *translate*'s outcome to PDDL, Level 1.

The grounded instance specific file is close to the original one with object declaration omitted.

Domain Encoding

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* (Helmert 2004), 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 SAS$^+$ 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. We plan to add the inference mechanism of VAL (Fox and Long 1998), as new versions support SAS$^+$ encodings.

Inferring *the* minimal state description is complex. The recursive approach of (Edelkamp and Helmert 1999) analyzes the effects of operators to merge partial predicates into exclusive groups, where a partial predicate is a predicate projected all but one of its parameters. Merged predicates are filled with the atoms from the result of fact space exploration. As there can be many different possible merging, one may ask the domain expert for assistance to select the ones that minimizes the encoding length.

For including knowledge acquisition and engineering options to this phase, however, we chose a different aspect.

Problem In planning with pattern databases (Edelkamp 2001), 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 (Korf and Felner 2002), in which operator projections are void in all but one abstraction. In plannning, pattern database abstraction are most effective if they consider SAS$^+$ groups in common.

Knowledge Acquisition There are different bin-packing approximation algorithms (Edelkamp 2001) 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.

Knowledge Engineering Even though we are currently working at a genetic algorithm to improve the first partition of groups, domain expert guidance in this optimization process is crucial. The expert is asked to refine the approximated, disjoint partitioning into planning pattern databases proposed in the inference module, by modifying the SAS$^+$ partition for the different pattern databases in an XML frontend.

Object Symmetry

Unless handled properly, symmetries cause an explosion in the search space of the planners. Let $(\mathcal{O}, \mathcal{I}, \mathcal{G})$ be a planning instance. A bijection ϕ within the planning state space is said to be a *symmetry* if $\phi(\mathcal{I}) = \mathcal{I}$, $\phi(G) \in \mathcal{G}$ for all $G \in \mathcal{G}$ and for any states u and v with an operator from u to v there exist an according operator from $\phi(u)$ to $\phi(v)$. Any set A of symmetries generates a subgroup $g(A)$ called a symmetry group. The subgroup $g(A)$ induces an equivalence relation $\sim_A$ on states, defined as $u \sim_A v$ if and only if $\phi(u) = v$ and $\phi \in g(A)$. Such an equivalence relation is called a *symmetry relation* on $\mathcal{P}$ induced by A. The equivalence class of u is called the *orbit* of u and denoted as $[u]_A$. Any symmetry relation on $\mathcal{P}$ is a congruence on $\mathcal{P}$. Moreover, u is reachable from $\mathcal{I}$ if and only if $[u]_A$ is reachable from $[\mathcal{I}]_A$. This reduces the search for goal $G \in \mathcal{G}$ to the reachability of state $[G]_A$.

Problem Finding symmetries fully automatically is not easy, since it links to the computational hard problem of graph isomorphism. Note that the general problem of graph isomorphism is not completely classified. It is expected not to be NP-complete (Wegener 2003). Some complexity theoretic results are: if GI is NP-complete then $\Sigma_2 = \Pi_2$, $\overline{GI}$ has an interactive proof system with 2 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 (Rintanen 2003; Lluch-Lafuente 2003; Guéré and Alami 2001), symmetries are fully specified by the domain expert.

Some planners apply automated *object symmetries* (Fox and Long 1999). 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.

Knowledge Acquisition The important observation is that the domain file is transparent to object transpositions, so that symmetry breaking 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 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 reduce the set of possible object symmetries to the ones valid in the goal.

We are currently implementing a knowledge acquisition module that given a PDDL description file, prompts a list of detected object transpositions to the domain expert. We next aim to provide tables e.g for all facts and remaining object transpositions.

Knowledge Engineering The domain expert can delete or modify inferred object symmetries and we may expect him to add complex ones that the system connot infer.

Goal Ordering

Goal ordering is an essential part of accelerating solution finding in larger planning problems. It yields a *goal agenda* (Koehler and Hoffmann 2000) denoting, in which order goal predicates and conditions should be established. It consists of a sequence of goals atoms $\mathcal{G}_1$, ..., $\mathcal{G}_k$ 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 $\mathcal{I}_0 = \mathcal{I}$ and $\mathcal{I}_{i+1} = \textit{Planner}(\mathcal{I}_i, \mathcal{G}_i)$. The overall sequential plan is the concatenation of the individual ones. For example, the goal ordering agenda has improved heuristic search planners like FF to circumvent misguidance by the heuristic estimate value e.g. in Blockworld. It also has laid the basis of the dramatic impact of constraint partitioned problem solving in SGPlan (Chen and Wah 2004). The idea shares similarities with *macro problem solving* (Korf 1985), in which operator sequences are learned to be retrieved for sequentialized goal satisfaction. Some goal ordering can yield fairly long plans (Lin 2001).

Problem Let $(\mathcal{O}, \mathcal{I}, \mathcal{G})$ be a planning instance and $p, q \in \mathcal{G}$. Furthermore, let $s_{p,\neg q}$ be a planning state for which p has just been reached and in which q does not hold. We define the *forced ordering* $p \preceq_f q$ if for all $s_{p,\neg q}$ there is no plan $P(\mathcal{O})$ for operator set $\mathcal{O}$ with q is the result of

applying $P(\mathcal{O})$ to $s_{p,\neg q}$. Moreover, the *reasonable ordering* $p \preceq_r q$ is given if for all $s_{p,\neg q}$ there is no plan $P(\mathcal{O}_p)$ for operator set $\mathcal{O}_p$ with q is the result of applying $P(\mathcal{O}_p)$ to $s_{p,\neg q}$. Here $\mathcal{O}_p$ is the set of actions that have p not in their delete list. Unfortunately, the decision problems for $p \preceq_f q$ and $p \preceq_r q$ are both PSPACE hard.

Knowledge Acquisition We have implemented approximation $\preceq_h$ of $\preceq_r$ as an individual static analysis option. The algorithm prompts the outcome of this phase to the domain expert 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.

Knowledge Enginering 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.

Operator Dependency

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 is more complex as 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* conditions/effects, 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 ϵ time steps between any two happenings that are dependent. The default for ϵ 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.

Problem Optimizing of plans without precedence ordering is involved (Bäckström 1998), since computing the makespan for a set of operators is NP-complete.

Knowledge Acquisition 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. With PERT such a plan can be computed in optimal time (Edelkamp 2003). The approach extends to timed initial literals and action execution time windows. For the sake of simplicity we assume that each proposition p can be associated to unique a time interval $[t_{\min}(p), t_{max}(p)]$, where it is valid; $t_{min}(p)$ is the time, when p is initially set and $t_{max}(p)$ is the time, where p is finally deleted. For ordinary propositions, the time intervall is $[0, \infty]$.

The execution time window $[t_{\min}, t_{\max}]$ for an operator with propositional condition list $P_{\textit{start}} \cup P_{\textit{invariant}} \cup P_{\textit{end}}$ is the intersection of the unrestricted interval $[0, \infty]$, $\cap_{p \in P_{\textit{start}}} [t_{\min}(p), t_{\max}(p) + d]$, $\cap_{p \in P_{\textit{invariant}}} [t_{\min}(p), t_{\max}(p)]$, and $\cap_{p \in P_{\textit{end}}} [t_{\min}(p) - d, t_{\max}(p)]$, where d is the duration of the operator.

Knowledge Engineering 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 visible and accessible to the domain expert with each computed plan. He may add or delete precedence constraints before scheduling is performed.

Domain Inference

The previous mechanisms separate the knowledge acquisition stage from the knowledge engeneering part, as the domain expert refines a proposed solution to a hard combinatorial task. To infer operators within a PDDL domain description, the inference mechanism needs supervision of the domain expert.

Problem *Learning* PDDL domain specifications from plan traces without any domain expert knowledge is a computationally challenging and practically infeasible task.

Knowledge Acquisition & Engineering 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.

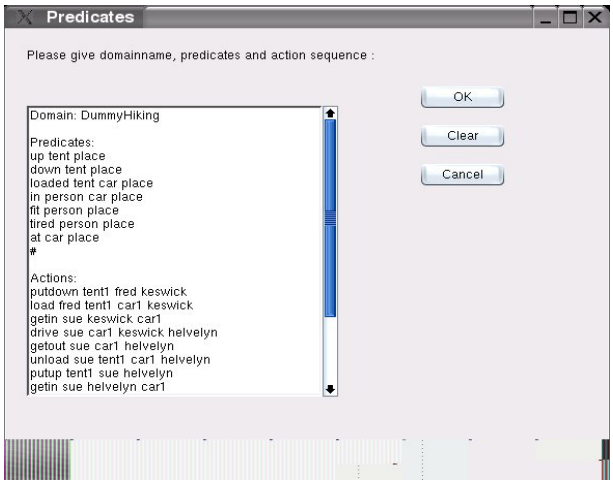

Figure 1: Input for the PDDL learning task.

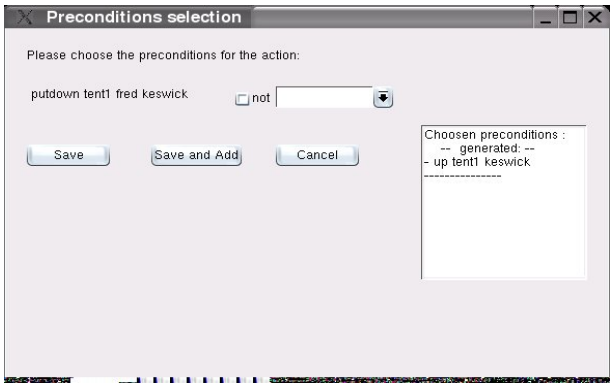

Figure 2: Supervised input of action preconditions.

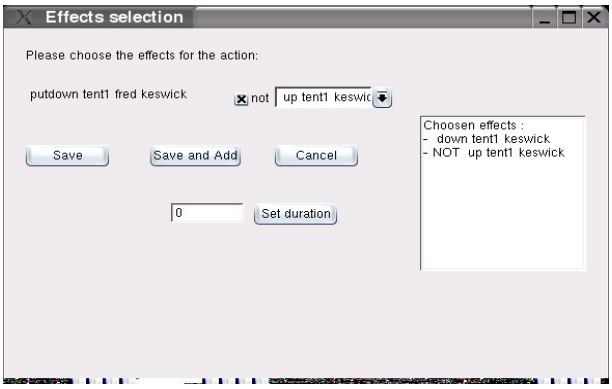

Figure 3: Supervised input of action effects and duration.

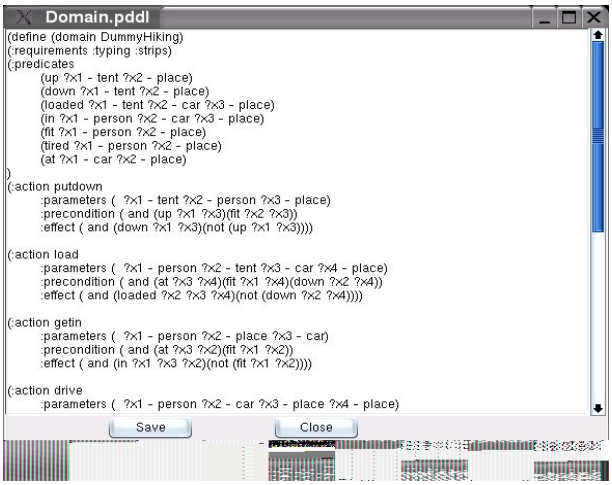

Figure 4: Inferred domain description.

Given the set of operators in a valid plan (cf. Figure 1), 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 2 and 3). The supervised PDDL learning mechanism selects 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 slightly improved implementation of the *Opmaker* algorithm (McCluskey *et al.* 2002). 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.

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 a simple user interface for both planners to selecte optional parameters.

Pattern-Plan

Current STRIPS planners are diverse in structure. While most suboptimal planners use heuristic and/or local search (Chen and Wah 2004), optimal planners range from satisfiability solving (Kautz and Selman 1996) via constraint satisfaction (Rintanen and Jungholt 1999) over planning graph approaches (Blum and Furst 1995) to integer programming (Kautz and Walser 1999) and heursitic search (Haslum and Geffner 2000; Edelkamp 2001). In our new planner *Pattern-Plan* we focus on optimal (either symbolic or explicit) pattern database planning, in particular in cooperation with external search (Edelkamp 2005).

For the domain expert, we allow interactive pattern selection in form of a partition of the SAS^+ variables.

Durative-FF

For *metric planning*, we implemented the PERT scheduling approach (Edelkamp 2003) on top of Metric-FF (Hoffmann 2003) 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 (Edelkamp 2004a), we currently include expressivity with this respect. By the choice of the underlying planner, however, generalizing the relaxed planning heuristic to *non-linear tasks* (Edelkamp 2004b) is involved.

For the domain expert, the operator dependency structure can be changed prior to the PERT scheduling.

Plan Validation and Visualization

For *plan validation* we have included VAL (Howey and Long 2003), 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 included the animation system Vega (Hipke 2000) allowing a magnification to an arbitrary part of the plan. Vega 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 using the object browser, show 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.

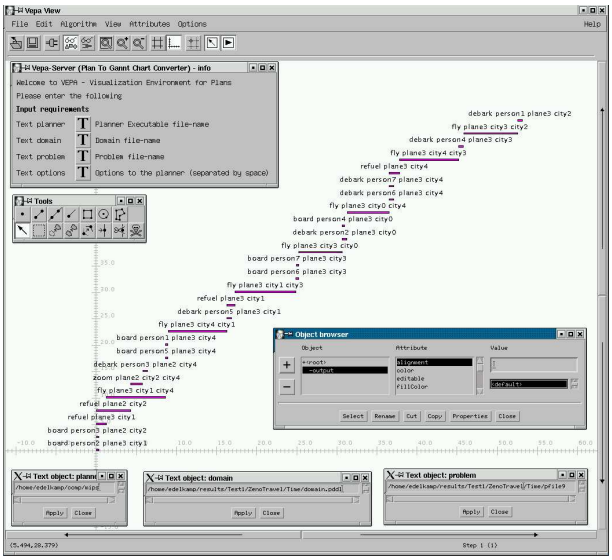

Figure 5: Visualizing a plan in Gantt chart format.

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, for all planners in the 2004 competition, we provide their results in Vega's text input format. For using it as a wrapper module the domain expert may choose the planner, the domain, and the problem file. Instead of calling different planners to Vega, we encounter that the simplest option to visualized plans is to convert plans in competition format into Vega scenes with an individual program. Using this option, we were able to visualize all plans generated at the 2004 planning competition in a scalable Gantt chart.

Domain-dependent visualization is more challenging. As the semantics of predicates are not known in advance, domain experts assist the animation design. From a knowledge engineering aspect, the efforts have to be as small as possible. Based on flushing sequential operator trails together with their corresponding state sequence, we have written *instance-independent* visualizations for all 2002 competition domains (see Figure 6 for an example), all with less than a hundred lines of code. The list includes *BlocksWorld*, *DriverLog*, *Satellite*, *Taxi*, *Depots*, *FreeCell*, *DesertRats*, *Jugs*, *Rover*, *Settlers*, and *ZenoTravel*. Propositional atoms can often be illustrate by showing or hiding an

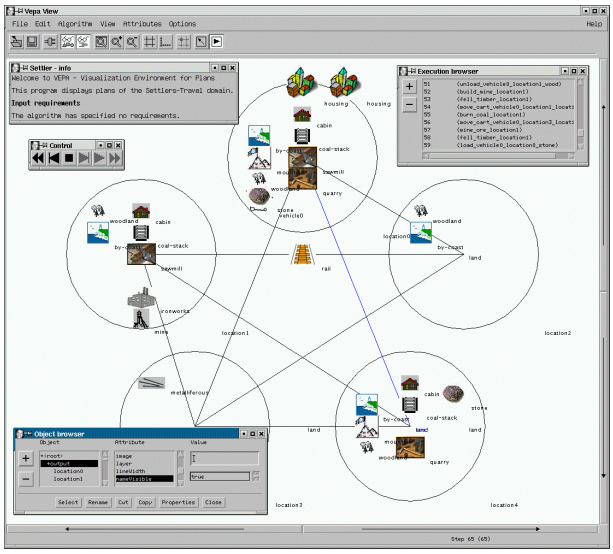

Figure 6: Animation of a plan.

image at a certain location, while numerical quantities such as fuel can best be 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 planner extension that wrote (sequential) plans together with associated state information to disk. Currently, we exploit VAL's capabilities to allow visualization 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 for further post-processing.

Conclusion

With *ModPlan* we have seen a feasible and practical approach for an integrated environment including domain modeling, static analyzes, plan finding, plan validation, and plan visualization. It features different knowledge acquisition and engineering facilities.

In our setting, *knowledge* refers to an algorithmic insight that is made accessible to and is subject to changes by the domain expert. In essence, knowledge engineering tackles hard combinatorial problems with a domain expert interacting with the system to ease solution finding.

Except for PDDL inference, we allow but do not rely on expert knowledge. We simplified domain design by offering an interactive operator learning module. We included front-ends to the knowledge that is inferred by static analysers, yieling grounded and annotated PDDL, to be exploited by different plan engines. Additionally, we have contributed two new planners: a suboptimal one based on PERT scheduling that covers a large fraction of PDDL2.2, and an optimal one based on pattern databases for propositional planning problems in SAS$^+$ encoding.

As indicated in the introduction, there are other challenges that could be made accessible to domain experts for improving 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 that perform a synchronized exploration of the state space and progressed expressions in temporal logic. Other extensions include exports to different e.g. model checking tools.

Acknowledgements *ModPlan* is being developed at the University of Dortmund within an undergraduate student project group. Thanks to Edina Kurtic, Miriam Bützken, Andrea Matuszewski, Rachid Karmouni, Michael Nelskamp, Arne Wiggers, Abdelaziz Elalaoui, Khalid Lahiane, Mohammed Nazih, Kenneth Kahl, Roman Klinger, and Abubakr Mkhdramine for all their implementation effort Moreover, we are grateful to Björn Scholz and Shahid Jabbar for improving the vizualization of plans and to Malte Helmert for providing access to *translate*. The work is supported by DFG in the projects Ed 74/2 and 74/3.

References

F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116:123–191, 2000.

F. Bacchus. The AIPS'00 planning competition. 22(3):47–56, 2001.

C. Bäckström. Computational aspects of reordering plans. *Journal of Artificial Intelligence Research*, 9:99–137, 1998.

A. Blum and M. L. Furst. Fast planning through planning graph analysis. In *IJCAI*, pages 1636–1642, 1995.

A. Botea, M. Müller, and J. Schaeffer. Learning partial-order macros from solutions. In *ICAPS*, 2005. To apppear.

Y. Chen and B. W. Wah. Subgoal partitioning and resolution in planning. In *Proceedings of the International Planning Competition*, 2004.

S. Edelkamp and M. Helmert. Exhibiting knowledge in planning problems to minimize state encoding length. In *ECP*, pages 135–147, 1999.

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.

S. Edelkamp, J. Hoffmann, M. Littman, and H. Younes, editors. *Proceedings of the International Planning Competition*. JPL, 2004.

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

S. Edelkamp. Taming numbers and durations in the model checking integrated planning system. *Journal of Artificial Research*, 20:195–238, 2003.

S. Edelkamp. Extended critical paths in temporal planning. In *Proceedings ICAPS-Workshop on Integrating Planning Into Scheduling*, 2004.

S. Edelkamp. Generalizing the relaxed planning heuristic to non-linear tasks. In *KI*, 2004. 198–212.

S. Edelkamp. External symbolic heuristic search. In *ICAPS*, 2005. To appear.

M. Fox and D. Long. The automatic inference of state invariants in TIM. *Artificial Intelligence Research*, 9:367–421, 1998.

M. Fox and D. Long. The detection and exploration of symmetry in planning problems. In *IJCAI*, pages 956–961, 1999.

M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 2003.

A. Gerevini and L. Schubert. Discovering state constraints in DISCOPLAN: Some new results. In *AAAI*, pages 761–767, 2000.

E. Guéré and R. Alami. One action is enough to plan. In *IJCAI*, 2001.

P. Haslum and H. Geffner. Admissible heuristics for optimal planning. In *AIPS*, pages 140–149, 2000.

P. Haslum and P. Jonsson. Planning with reduced operator sets. In *AIPS*, pages 150–158, 2000.

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

C. A. Hipke. *Distributed Visualization of Geometric Algorithms*. PhD thesis, University of Freiburg, 2000.

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

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.

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.

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.

H. Kautz and B. Selman. Pushing the envelope: Planning propositional logic, and stochastic search. In *AAAI*, pages 1194–1201, 1996.

H. Kautz and J. Walser. State-space planning by integer optimization. In *AAAI*, 1999.

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.

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. Macro-operators: A weak method for learning. *Artificial Intelligence*, 26:35–77, 1985.

F. Lin. System R. *AI-Magazine*, pages 73–76, 2001.

A. Lluch-Lafuente. Symmetry reduction and heuristic search for error detection in model checking. In *MoChArt*, 2003.

D. Long and M. Fox. Automatic synthesis and use of generic types in planning. In *AIPS*, pages 196–205, 2000.

D. Long and M. Fox. Hybrid stan: Identifying and managing combinatorial optimisation sub-problems in planning. In *IJCAI*, pages 445–452, 2001.

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.

T. L. McCluskey, N. E. Richardson, and R. M. Simpson. An interactive method for inducing operator descriptions. In *AIPS*, 2002.

D. McDermott. The 1998 AI Planning Competition. *AI Magazine*, 21(2), 2000.

D. Nau, Y. Cao, A. Lotem, and H. Munoz-Avila. SHOP: simple hierarchical ordered planner. In *IJCAI*, pages 968–973, 1999.

J. Rintanen and H. Jungholt. Numeric state variables in constraint-based planning. In *ECP*, pages 109–121, 1999.

J. Rintanen. An iterative algorithm for synthesizing invariants. In *AAAI*, pages 806–811, 2000.

J. Rintanen. Symmetry reduction for SAT representations of transition systems. In *ICAPS*, 2003.

I. Wegener. *Komplexitätstheorie*. Springer, 2003. (in German).