Proceedings of the 4th Workshop on
Knowledge Engineering for Planning and Scheduling

KEPS 2013

Rome, Italy - June 10, 2013

Edited By:

Roman Barták, Simone Fratini, Lee McCluskey, Tiago Vaquero

Organizing Commitee

Roman Barták
Charles University, Czech Republic
Supported by the Czech Science Foundation under the contract P202-10-1188.

Simone Fratini
European Space Agency, Germany

Lee McCluskey
University of Huddersfield, UK

Tiago Vaquero
University of Toronto, Canada

Program committee

Roman Barták, Charles University, Czech Republic
Piergiorgio Bertoli, Fondazione Bruno Kessler, Italy
Mark Boddy, Adventium Labs, U.S.A.
Adi Botea, IBM, Ireland
Luis Castillo, IActive, Spain
Amedeo Cesta, ISTC-CNR, Italy
Susana Fernández, Universidad Carlos III de Madrid, Spain
Simone Fratini, ESA/ESOC, Germany
Antonio Garrido, Universidad Politecnica de Valencia, Spain
Arturo González-Ferrer, University of Haifa, Israel
Felix Ingrand, LAAS-CNRS, France
Peter A. Jarvis, PARC, USA
Ugur Kuter, SIFT, USA
John Levine, University of Strathclyde, UK
Lee McCluskey, University of Huddersfield, United Kingdom
José Reinaldo Silva, University of São Paulo, Brazil
David Smith, NASA, USA
Tiago Vaquero, University of Toronto, Canada
Gerard Verfaillie, ONERA, France
Dimitris Vrakas, Aristotle University of Thessaloniki, Greece

Foreword

Despite the progress in automated planning and scheduling systems, these systems still need to be fed by careful problem description and they need to be fine-tuned for particular domains and problems. Knowledge engineering for AI planning and scheduling deals with the acquisition, design, validation and maintenance of domain models, and the selection and optimization of appropriate machinery to work on them. These processes impact directly on the success of real planning and scheduling applications. The importance of knowledge engineering techniques is clearly demonstrated by a performance gap between domain-independent planners and planners exploiting domain dependent knowledge.

The KEPS 2013 workshop continues the tradition of ICKEPS competitions and KEPS workshops. Rather than focusing on software tools only, which is the topic of ICKEPS, the workshop covers all aspects of knowledge engineering for AI planning and scheduling.

This year the set of accepted papers covers the traditional area of knowledge engineering tools and methods, supplemented with papers on formulation, reformulation, and post planning optimization. Representational concerns are covered also, with several papers on timelines-based approaches. With coverage of plan libraries, verification and validation issues, and some key application areas, the schedule for this year's workshop is as broad as it is interesting.

Roman Barták, Simone Fratini, Lee McCluskey, Tiago Vaquero
KEPS 2013 Organizers
June 2013

Table of Contents

Requirement Analysis Method for Real World Application in Automated Planning Systems

Rosimarci Tonaco Basbaum[1] **and Tiago Stegun Vaquero**[2] and **José Reinaldo Silva**[1]
[1]Department of Mechatronic Engineering, University of São Paulo, Brazil
[2]Department of Mechanical & Industrial Engineering, University of Toronto, Canada
rosimarci@usp.br, tvaquero@mie.utoronto.ca, reinaldo@usp.br

In the intelligent design field, the requirement analysis phase has a fundamental role in automated planning-especially for "real life" systems. Using requirement analysis users has the ability to identify or redesign variables which can potentially increase the model accuracy. A great effort has been made today in the area of Artificial Intelligence for defining reliable automated planning systems that can be applied for real life applications. That leads to the need for systematic design process, in which the initial phases are not neglected and where Knowledge and Requirement Engineering tools have a fundamental role for supporting designers. This paper intent to propose a design method as well as perform a more detailed study on the adoption of UML and Petri Nets in the requirement analysis phase using the itSIMPLE framework as a KE tool.

Introduction

Planning characterizes a specific kind of design problem where the purpose is to find a set of admissible actions to solve a problem. The current approaches in the literature aim to improve the performance of the planner trying to optimize the search algorithms and the general solution (Edelkamp and Jabbar 2006). In addition, most of existing work on this direction apply synthesized and artificial problems (closed problems that have limited set of actions such as Blocks World) as a proof of concept for the proposed algorithms. Due to the extensive development in this area some authors started to apply planning techniques on real world problems (Vaquero et al. 2012) as well - like logistic problems - where the number of variables and actions are high. Those features increases the model's complexity.

That said, it is clear that the automated planning area carries an indefinition problem:

- the study made until today is historically connected to search solution methods to planning problems in an automatically way and domain independent. Thus, the solutions could be inserted in intelligent automated mechanisms, especially robots and another autonomous systems.
- the formal techniques developed domain independently led to techniques that today are very important in several research fields, like logistics, diagnostic systems, navigation, space robots, satellite systems, etc. This demand is independent of the adequacy of formal techniques to the systems based on specific knowledge to provide real solutions, while increases the discovery of new independent domain solutions.

Requirement analysis in real life systems is a topic that has been currently discussed in important Automated Planning conferences and workshops (McCluskey et al. 2003) (McCluskey 2002) (Hoffmann 2003). However, there are not still many works in this research line, especially using Petri nets in the requirement analysis and validation. Recent studies show the need of requirement analysis of real life systems in automated planning, where the problems are modeled using UML (Vaquero 2011) (Simpson 2005) (Simpson, Kitchin, and McCluskey 2007). However, the literature is still very scarce when the subject is Petri net applied to the design process, especially in real life problems.

It is important when dealing with real world problems to have a design life cycle: a defined sequence of processes that can support the designer to create a more faithful model. UML combined with Petri nets offers a disciplined design process, providing a visual tool (UML) and a validation tool (Petri nets) to analyze the models.

The design process has two major parts: the elicitation and documentation phase. Using UML the user can model all the requirements found in the elicitation phase. Throughout the Petri net the requirement analysis will be performed using the available techniques, such as invariant analysis and equation state matrix (Murata 1989). The first step detects contradictions and conflicts between the requirements, or the different view points in the diagrams. The second step identifies deep inconsistencies, that are hidden in the dynamic perspective of the plans, as well as additional information about the model that could be interpreted for the planners. Such information includes, new constraints, invariants, partial solution strategies, characteristics that could help to improve the planner performance. But, to do this analysis it necessary to translate de UML diagrams into a single network, and then make the dynamic analysis.

The tool we choose to use is the itSIMPLE framework (Vaquero and Silva 2005), which currently is not a 100%

ready to perform the UML/Petri net translation, but it will be during the development of this project. Our objective is develop a framework, that will work as part of itSIMPLE framework, in order to make requirements analysis using Petri Nets, especially the dynamic of actions.

In the section 2 we will discuss the advantages of UML in automated planning, followed by a brief description of Petri Nets, next we present a study case of a classical Logistic problem, the results, discussions and conclusions.

UML for Design of Real Life Automated Planning Problems

Accordingly to the literature (Booch, Rumbaugh, and Jacobson 2006) the increasing use of UML to model real world systems and planning problems is mainly due to its visual modeling representation (Vaquero, Tonidandel, and Silva 2005). One of the benefits of using UML to model planning problems is the viewpoints that this language can offer to the model. Dismembering the system in different viewpoints is a good approach, and can help in the design process, separating the problem into smaller parts that can be addressed using distinct diagrams. However, representing it using UML can be a challenge because some conflicting interpretations can be generated accross the different diagrams during the requirement analysis.

With that said, the following question arises: what is the minimal set of UML diagrams necessary to represent a planning application? The goal is to use diagrams that have different and complementary viewpoints. The answer is a minimal set capable to represent a planning application in an optimized way. Therefore, it is necessary to work with the independence hypothesis, considering that the knowledge are divided in two parts: work domain and planning problem, as we will explain better hereafter.

The first step is identify which requirements are represented in each part. Consider that the work domain represents the static attributes of the system, and the planning problem represents the dynamic attributes, including possible actions. Both, work domain and planning problem, can be modeled in different Class Diagrams, aiming to not overlap the viewpoints. The Class Diagram is a static representation form adequate to represent work domain. As said before, the planning problem is dynamic, but can be modeled in a Class Diagram too, in this case the methods need to be declared in the classes to represent the actions. To complement the model, the user must use the State Machine, Activity and Sequence Diagrams that contribute to the system dynamic representation. Using this design method, the work domain model remains static (if it is necessary), and the planning problem can be changed according to the necessity, offering an independent and flexible model. The Object Diagram is used to create problem instances. Such diagram can represent the initial and goal state and from these diagrams the planner can find a plan.

The Use Case Diagram is unnecessary, because it brings redundant information that is similar to a textual description of the problem. According to (Irwin and Turk 2005) in the representation via Use Case there is no consistency and accuracy. Many authors believe that the Use Case Diagram is not useful in the requirement analysis, like (Siau and Lee 2004). (Dobing and Parsons 2006) question the naturality involved in the Use Case Diagrams in the object modeling and the idea of these diagrams facilitating the communication and requirement analysis. Even with all those criticisms the Use Case Diagram can be useful to separate the problem into smaller parts, this is an interesting approach that can help the design of the Activity Diagram. However, for this first proposal the Use case Diagram will not be part of the minimal set.

The proposed minimal set consists of the following diagrams:

- Class Diagram;
- State Machine Diagram;
- Activity Diagram;
- Sequence Diagram;
- Object Diagram.

Using the minimal set of diagrams it is possible to have a concise and syntactically correct representation. However, it is impossible to validate the model since UML is a semi-formal representation language. An alternative is to use Petri nets to execute the validation, to do this it is necessary to translate the UML diagrams into a single Petri net. But when the diagrams are grouped into a single net the viewpoints are merged and with that the identification of errors becomes complex. Nevertheless, in the UML phase, a syntactic analysis can be performed to find errors in the model. This can be done by analyzing the xml file of the itSIMPLE projects.

Basics of Dynamic Analysis using Petri Nets

The use of Petri nets is quite promising in requirement analysis phase and with this technique users can perform the verification and validation of the model. In this work the place/transition Petri Nets (Murata 1989) will be used.

Petri nets is a powerful tool to validate requirements (Silva 2004). It allows a general representation of the system and it is a good formalism for real life systems. With Petri nets one can group the UML diagrams into a single net. The idea is to find interactions between the objects represented in all diagrams and translate them into a unique Petri net, that can represent the entire system process.

Lately, the itSIMPLE translates the UML model into a Petri net using only the State Machine Diagrams, (Vaquero 2007). Figure 1 shows a simple example of how this translation occurs.

The approach using only State Machine Diagrams in the translation process does not allow a correct analysis of the system. Following this approach, from each State Machine Diagram a Petri net is derived, therefore it is necessary to find the interaction points of the nets and link them into a single Petri net.

Once the Petri net was defined it is necessary to search for straightforward and trivial conflicts and the presence of deadlocks. After that, a semantic analysis is performed aiming to verify whether the network in fact represents the

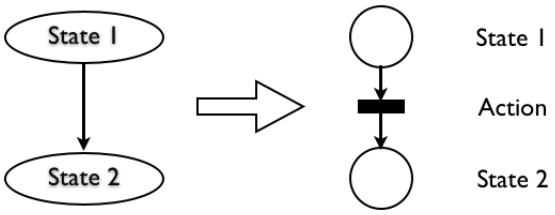


Figure 1: Translation of State Diagrams in a Petri Net.

A Petri net is a 5-tuple, *PN = (P, T, F,W, M0)* where:
$P = \{p1, p2, ...pn\}$ is a finite set of places,
$T = \{t1, t2, ..., tn\}$ is a finite set of transitions,
$F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation),
$W: F \rightarrow \{1, 2, 3, ...\}$ is a weight function,
$M0: P \rightarrow \{0, 1, 2, 3, ...\}$ is the initial marking,
$P \cap T = \oslash$ and $P \cup T \neq \oslash$.

Table 1: Formal definition of a Petri net.

planning application. The semantic analysis can be accomplished using some of the Petri nets properties, like the behavioral and structural properties (Murata 1989).

Behavioral properties of Petri nets are related to the behavior of the net and change during the simulation (Murata 1989). They are essentially dependent on the execution stage in which the network is located and are, therefore, marking-dependent (Murata 1989).

Structural properties are those that depend on the topological structures of the Petri nets. They are independent of the initial marking *M0* in the sense that these properties hold for any initial marking or are concerned with the existence of certain firing sequences from some initial marking. In this paper we only consider classical Petri nets, in this case place/transition net. A formal definition of a place/transition Petri net is given in Table 1 (Murata 1989).

A Petri net structure *N = (P, T, F, W)* without any specific initial marking is denoted by *N*. A Petri net with the given initial marking is denoted by *(N, M0)*.

A Petri net, that is a good representation for a planning problem, should have the following properties (Murata 1989):

- Reachability: The firing of an enabled transition will change the token distribution (marking) in a net according to the transition rule. A sequence of firings will result in a sequence of markings. A marking *Mn* is said to be reachable from a marking *M0* if there exists a sequence of firings that transforms *M0* to *Mn*.
- Boundedness: A Petri net *(N, M0)* is said to be bounded if the number of tokens in each place exceed a finite number k for any marking reachable from *M0*, i.e. $M(p) \leq k$ for every place p and every marking $M \in R(M0)$.
- Liveness: The concept of liveness is closely related to the complete absence of deadlocks in operating systems. A Petri net *(N, M0)* is said to be live (or equivalently *M0* is said to be a live marking for *N*) if, no matter what marking has been reached from *M0*, it is possible to ultimately fire any transition of the net by progressing through some further firing sequence. This means that a live Petri net guarantees deadlock-free operation, no matter what firing sequence is chosen.
- Reversibility: A Petri net *(N, M0)* is said to be reversible if, for each marking *M* in *R(M0)* is reachable from *M*.

Methods of analysis for Petri nets may be classified into the following groups: 1) the coverability graph method (including reachability tree); 2) the matrix-equation approach; and 3) reduction or decomposition techniques (Murata 1989). The first method involves essentially the enumeration of all reachable markings or their coverable markings. It should be able to apply to all classes of nets, but is limited to small nets due to the complexity of the state-space explosion. On the other hand, matrix equations and reduction techniques are powerful but in many cases they are applicable only to special subclasses of Petri nets or special situations (Murata 1989).

In the current version of itSIMPLE the Petri nets needs to be manually designed considering the transitions that appears in the State Machine Diagrams and pre and post conditions (these can be found at the PDDL file on the itSIMPLE project). The purpose is to migrate to high level Petri nets, because they offer a sintax representation (especially when multiplicity are present in the problem) and the complexity of the application could be represented in a clear way. The module that will generate the high level Petri net and the inclusion of Sequence and Activity Diagrams must be implemented in the next itSIMPLE releases.

Requirement and Work Domain Analysis in Automated Planning

Applications that deal with real life problems become increasingly necessary over the recent developments in the Automated Planning field. In general, the major focus of the planning community is, the pursuit of planners efficiency, and neglecting the analysis aspects. (Zimmerman and Kambhampati 2003; Upal 2005) .

To conduct the planning of an activity it is necessary to determine all features of the system in which it is embedded. Some factors must be considered, for example the sub systems evolved, internal variables, correlations with other systems, constants and constraints. Such specification is called system modeling, and from it depends the success of the result obtained from the planning process. In this aspect several points becomes important, such as the proposed model complexity, and its accuracy from the original real life system.

In the design process languages such as the traditional PDDL (McDermott 2003), or the UML (OMG 2009) are used. To help in the design and the requirement analysis phases, as said before, there are frameworks available such as itSIMPLE (Vaquero, Tonidandel, and Silva 2005) (Vaquero et al. 2007), that focus on the initial design process

phases, such as specification and modeling. After design is concluded, the output model is processed by tools called planners, that can proposes a set of actions (Ghallab, M.; Nau, D.; Traveso 2004).

To design real life systems, there are two key challenges: 1) create a design discipline for modeling real life systems, using UML as the representation language; 2) translate and synthesis of the UML diagrams in a unique high level Petri net, which will be analyzed in order to obtain information that can anticipate problems in the model helping in the design phase the generation of suitable plans.

The general purpose of this paper is to propose a design process for automated planning systems, that is composed of two layers: 1) where independent domain methods are applied; and 2) using the specific knowledge and requirements analysis applying high level Petri nets to increase the quality of planning problems solutions in Artificial Intelligence. The challenge here is to discover where to insert the specific knowledge and how to include this in the design process since we are working with three classes of problems where the specific knowledge level increases from benchmark to real life problems. The classes are: benchmarks, intermediates (like ROADEFs (Perez et al. 2006)) and real life problems (like Petrobras problem presented in the International Competition of Knowledge Engineering for Planning and Scheduling (ICKEPS) 2012) (Vaquero et al. 2012).

In the first level the purpose is to follow the original work of (Hoffmann 2003), where is proposed the problem structure concept used in this paper. From that concept to suggest a formalization of this structure based on high level Petri nets. The properties will serve to analyze the similarities, repetitive cycles, invariants and other properties between the models.

In the second level the specific knowledge can be included as dynamic relationships and actions properties, that will be inserted in UML diagrams and translated in a Petri net (therefore having a format compatible with the previous phase). The structure receives the dependent-domain knowledge, to apply the analysis techniques of high level Petri nets. These techniques are particularly sensitive when applied in real life problems, requiring a different approach from the academic applications. Real life problems must follow a very disciplined design process, grounded in Knowledge Engineering, whose initial stage is composed by elicitation and requirement analysis. Such design process is the study focus of many researchers in Automated Planning area (McCluskey et al. 2003).

Since our focus is on intermediate and real life problems, and in this kind of problems it is necessary to define more specific knowledge to generate the model. We start from the independence hypothesis, that is design the work domain separately from the planning problems, as presented before. This approach allows to develop a more flexible model.

In the design process will be used UML (OMG 2009) as the modeling language and the oriented-object Petri net (from GHENeSys, that is a class of high level Petri nets) (San and Miralles 2012) to make the requirement analysis of the dynamic of actions.

In the next section will be presented a simple case study synthesized from Logistic Problem to show our first insights with this proposal. To be clear this is not the only case study that we made to test this proposal, we modeled and analyzed two ROADEFs challenges (2005 (Nguyen 2005) and 2009 (Palpant et al. 2009)) as well. Most of our conclusiond and findings about the method are from ROADEFs problems.

Case Study - Logistic

To illustrate the design method proposed in the previous sections, we present a simple version based on Logistic domain. In which only trucks are considered in order to reduce the scope. Since itSIMPLE still does not model the Activity Sequence Diagrams, the UML model for this case study was designed in a object oriented modeling tool. Another reason to make the model ”by hand” (not using itSIMPLE) is because the design method presented here is different of the method used in itSIMPLE. As proposed earlier, the planning problem and the work domain were designed separately, to avoid overlap in the requirements definition phase. Figures 2 and 3 show respectively the Class Diagram to the planning problem and work domain.

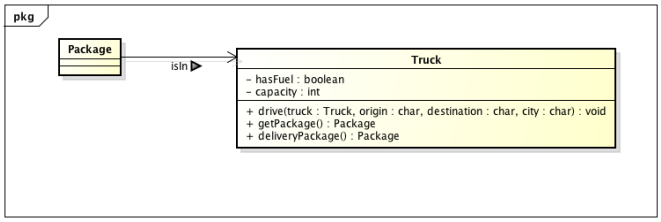


Figure 2: Class Diagram designed to represent the planning problem.

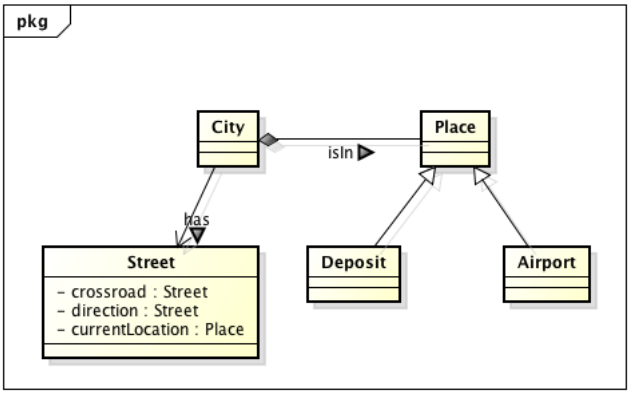


Figure 3: Class Diagram designed to represent the work domain.

Following the proposal of using a minimal set to represent planning problems, we present below the Activity, Sequence and States Diagrams. The Petri net designed for this problem does not use the itSIMPLE method to generate the nets. In order to design the Petri net we used three UML diagrams:

Activity, Sequence and State Machine, representing that way the dynamic part of the problem (the planning problem).

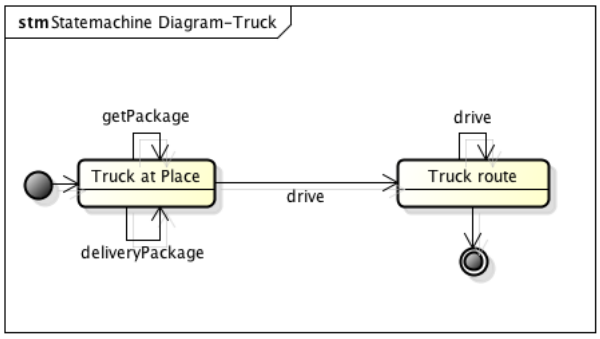


Figure 4: State Diagram designed to represent the planning problem.

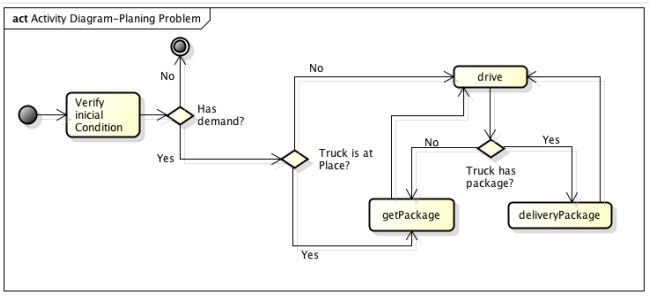


Figure 5: Activity Diagram designed to represent the planning problem.

The Activity Diagram shows the action cycles, (as an algorithm controlling the process flow) the State Machine Diagram identifies which states that each object could assume during the processing, and using the Sequence Diagram it is possible to create use cases to test the model and check which sequence of action is shot. The Petri net that represents those diagrams is presented in the Figure 7.

With this representation it is possible to verify the preconditions for the *getPackage* action which would be *Package is at place*, *Truck is at place* and *Received demand* and also its effect (in an abstract way) that is *Package in truck*. With Petri nets the analysis can be done using the techniques explained in the previous section. In this case, we use a reachability tree and a matrix-equation approach (Murata 1989).

The invariant analysis showed that the net is conservative, because the sum of markings is the same - since there is two objects modeled in the Petri net (Truck and Package), the sum of tokens must be 2. Analyzing the same results - from invariant analysis - we observed that the net is bounded as well, because the number of tokens never exceeds an integer k, where k = 2 in this example.

The reachability tree shows that the net is:

- limited because the coverability tree is finite.

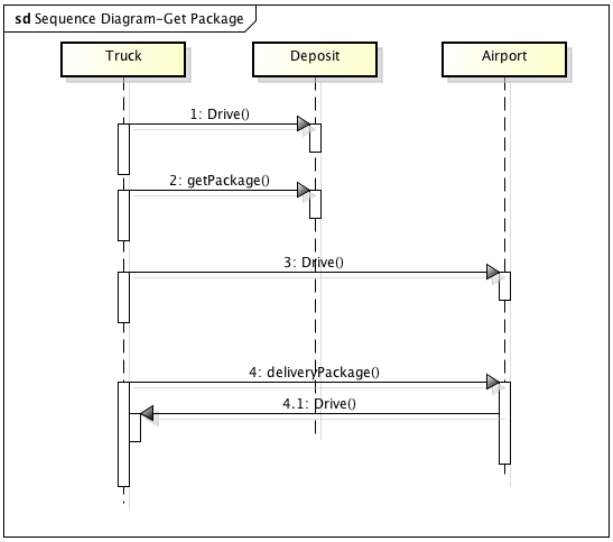


Figure 6: Sequence Diagram designed to represent the planning problem.

- live (there is no deadlocks), which in this case is desired and expected, since the process is cyclic.
- reachable, since all markings are due to the initial marking.

The purpose of using the reachability tree is trying to find loops that can indicate to the planner subgoals to be achieved. Thus, it is feasible to identify subgoals that could also help reaching the goal state or even if there are states that must be avoided anticipating that way possible deadlocks.

Results and Discussion

Modeling the work domain separately (Independence Hypothesis) make sense because it is common to many planning problems. With this approach we believe that is possible to analyze how the work domain constraints affects the planning problem. Besides that, one can analyze easily the structure that represents the work domain. And this representation is very closer to what (Hoffmann 2003) calls problem structure.

According to Hoffmann (2003), benchmark problems has a sort of structure which repeats in different planning applications. That structure is what the author called problem structure. Hoffmann (2003) raises a number of questions about the structure of the problem, among them two questions are relevant here: 1) What are the common patterns in the domain structures? Is it possible to find a description of these domains? The author points out that patterns could be an answer in the space-state graph topology.

The approach via Petri nets is what Hoffmann (2003) calls problem topology. And with this approach we can identify phenomena like dead-ends and goal ordering that was discussed in his lecture notes. The dead-ends is similar to deadlocks in Petri nets, and this problem is easily solved. Another

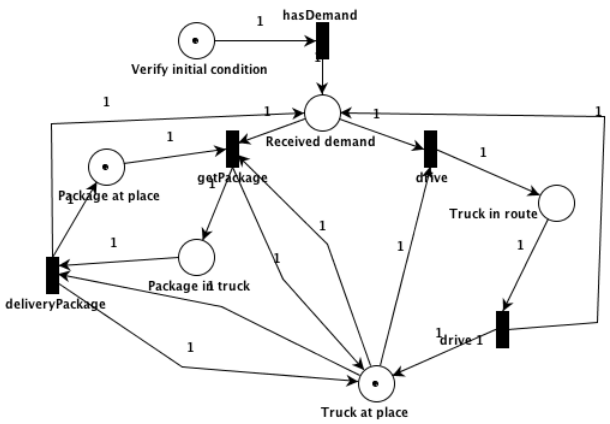


Figure 7: Petri net designed to represent the Logistc domain.

possibility is to observe if the planner can reach subgoals before reaching the goal state. With that is plausible to say that this characteristic can help to solve the ordering goals problem presented in (Hoffmann 2003). But, in this case we have positive results just for small and closed problems as will be shown below.

Let us consider the benchmark problem Blocks-World, where the problem instance is:

- Initial state: clear(b), clear(c), ontable(a), ontable(b), on(c, a), handempty.
- Goal: on(b,c) on(a,b) ontable(c).

Figure 8 shows all possible states of Blocks-World Arm, from state 1 where the blocks are on table to state 13. The initial state of our problem instance is state 11 and our goal state is state 9.

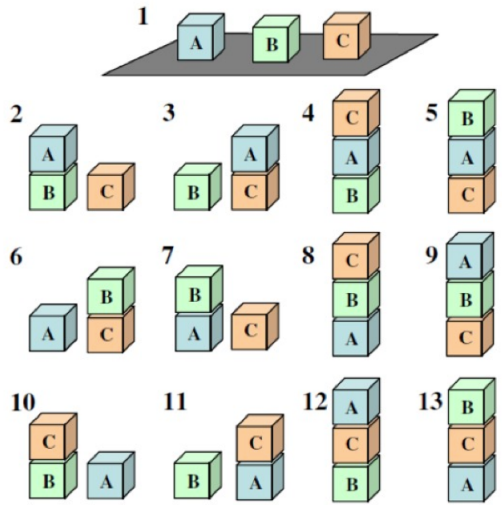


Figure 8: Possible States for Blocks-World Arm Problem.

From Figure 8 was designed to Petri net of Figure 9. In this figure each place represents a possible state of Figure 8. And there is an essential node, node 1 which divide the net in components. Each component represents the movement for each block. If we remove node 1, we will have three different Petri nets. The Sussman paradox occurs when the processing goes from essential node and comes back to the same component entering into a loop. It possible to solve that problem with just three actions: 1) it is mandatory pass through essential node; 2) since passed the essential node do not return to the same component (not repeat the initial action); 3) run the action ”move” to the respective component.

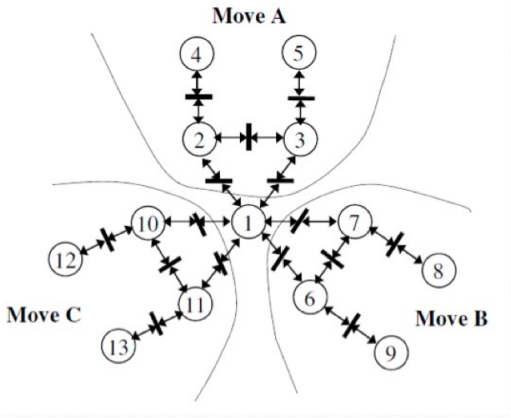


Figure 9: Blocks-World Arm Petri Net.

With this simple solution we can resolve the Sussman paradox and goal ordering phenomena presented in (Hoffmann 2003). Tests has not been done already with large and complex problems. But, the results found in small problems is a good sign that this technique can be also applied for complex problems. However, for real life problems, we believe that it will be necessary to use high level Petri nets to represent the problem.

Conclusion

After some running exercises, we found that the Petri net generated using he current version of itSIMPLE is not correct. itSIMPLE consider just the State Machine Diagrams and this is not enough to derivate the Petri net. But, currently the State Machine Diagram is the only way itSIMPLE uses to represent the dynamic part of the system, this was the main motivation to add other diagrams that can represent the dynamic aspects in a better way. Another weakness of itSIMPLE is how the planning application is modeled. There is no project discipline and this can lead the user to wrongly model the problem, confusing the work domain with the planning problem. Our proposal is to separate them in different Class Diagrams, and from the Class Diagram - that represents the planning problem - design other diagrams, since they can offer different viewpoints that can complement the information needed to generate the Petri net.

In this work we presented a different proposal for Petri nets in automated planning, that uses the Petri net in Knowledge Engineering to improve the design process. This project aims to create a better and disciplined way

to model planning applications using UML and Petri nets. Even though the method has not been implemented and validated yet, the results found in the case studies are quite promising. Using the method proposed in this work it is expected to have an improvement of the plans, with the addition of information from the Petri net analysis.

References

Booch, G.; Rumbaugh, J.; and Jacobson, I. 2006. *UML - User Guide*. Elsevier.

Dobing, B., and Parsons, J. 2006. How UML IS USED. *Communications of the ACM* 49(5):109–114.

Edelkamp, S., and Jabbar, S. 2006. Action Planning for Directed Model Checking of Petri Nets. *Electronic Notes in Theoretical Computer Science* 149(2):3–18.

Ghallab, M.; Nau, D.; Traveso, P. 2004. *Automated Planning: Theory and Practice*. CA: Morgan Kaufman.

Hoffmann, J. 2003. The Metric-{FF} Planning System: Translating Ignoring Delete Lists to Numerical State Variables. *Journal of Artificial Intelligence Research (JAIR)* 20.

Irwin, G., and Turk, D. 2005. An Ontological Analysis of Use Case Modeling. *Information Systems* 6(1):1–36.

McCluskey, T. L.; Aler, R.; Borrajo, D.; Haslum, P.; Jarvis, P.; Refanidis, I.; and SCHOLZ. 2003. Knowledge Engineering for Planning Roadmap.

McCluskey, T. 2002. Knowledge engineering: issues for the AI planning community. *The AIPS-2002 Workshop on Knowledge Engineering Tools and Techniques for AI Planning*.

McDermott, D. V. 2003. PDDL2.1 - The Art of the Possible? Commentary on Fox and Long. *Journal of Artificial Intelligence Research (JAIR)* 20:145–148.

Murata, T. 1989. Petri Nets: Properties, Analisys and Applications.

Nguyen, A. 2005. Challenge ROADEF 2005 Car Sequencing Problem. *ROADEF Challenge* 1–17.

OMG. 2009. OMG Unified Modeling Language TM (OMG UML), Superstructure.

Palpant, M.; Boudia, M.; Robelin, C.; Gabteni, S.; and Laburthe, F. 2009. ROADEF 2009 Challenge : Disruption Management for Commercial Aviation. (261):1–24.

Perez, O. J. G.; Reines, F. C. P.; Olivares, J. F.; Vidal, L. C.; and Hervas, T. G. 2006. Planning process from a user perspective. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS 2006) Workshop on Plan Analysis and Management. Cumbria, UK.*

San, A., and Miralles, P. 2012. *GHENeSys , uma rede unificada e de alto nivel*. Ph.D. Dissertation, Sao Paulo.

Siau, K., and Lee, L. 2004. Are use case and class diagrams complementary in requirements analysis? An experimental study on use case and class diagrams in UML. *Requirements Engineering* 9(4):229–237.

Silva, J. R. 2004. Applying Petri nets to requirements validation. *IFAC Symposium on Information Control Problems in Manufacturing Salvador* 1:508–517.

Simpson, R. M.; Kitchin, D. E.; and McCluskey, T. L. 2007. Planning domain definition using GIPO. *The Knowledge Engineering Review* 22(02):117.

Simpson, R. M. 2005. Gipo graphical interface for planning with objects. *Proceedings of the First International Competition on Knowledge Engineering for AI Planning*.

Upal, M. A. 2005. Learning to Improve Plan Quality. *Computational Intelligence* 21(4):440–461(22).

Vaquero, T. S., and Silva, J. R. 2005. The itSIMPLE tool for Modeling Planning Domains. *Artificial Intelligence*.

Vaquero, T. S.; Romero, V.; Tonidandel, F.; and Silva, J. R. 2007. itSIMPLE2.0: An integrated Tool for Designing Planning Environments. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS 2007). Providence, Rhode Island, USA*.

Vaquero, T. S.; Costa, G.; Tonidandel, F.; Igreja, H.; Silva, J. R.; and Beck, J. C. 2012. Planning and Scheduling Ship Operations on Petroleum Ports and Platforms. *Proceedings of the Scheduling and Planning Applications woRKshop*.

Vaquero, T. S.; Tonidandel, F.; and Silva, J. R. 2005. The itSIMPLE tool for Modelling Planning Domains. In *Proceedings of the First International Competition on Knowledge Engineering for AI Planning, Monterey, California, USA*.

Vaquero, T. S. 2007. *ITSIMPLE : Iintegrated Tools Environment For Modeling and Domain Analysis*. Dissertation, Polytechnic - University of Sao Paulo.

Vaquero, T. S. 2011. *Post-design for Automated Planning Problems: an approach combining diagnosis, virtual reality and reuse of rationales*. Tese de doutorado, Polytechnic - University of Sao Paulo.

Zimmerman, T., and Kambhampati, S. 2003. Learning-assisted automated planning: looking back, taking stock, going forward. *AI Magazine* 24(2):73–96.

Encoding Partial Plans for Heuristic Search

Pascal Bercher and **Susanne Biundo**

Institute of Artificial Intelligence,
Ulm University, D-89069 Ulm, Germany,
firstName.lastName@uni-ulm.de

Abstract

We propose a technique that allows any planning system that searches in the space of partial plans to make use of heuristics from the literature which are based on search in the space of states.

The technique uses a problem encoding that reduces the problem of finding a heuristic value for a partial plan to finding a heuristic value for a state: It encodes a partial plan into a new planning problem, s.t. solutions for the new problem correspond to solutions reachable from the partial plan. Evaluating the goal distance of the partial plan then corresponds to evaluating the goal distance of the initial state in the new planning problem.

Introduction

In most of today's classical planning approaches, problems are solved by informed (heuristic) progression search in the space of states. One reason for the big success of this approach is the availability of highly informed heuristics performing a goal-distance estimate for a given state. In plan-space-based search, search nodes correspond to partially ordered partial plans. One of the most important representatives of this technique is partial-order causal link (POCL) planning (McAllester and Rosenblitt 1991; Penberthy and Weld 1992). The least commitment principle of POCL planning seems to be advantageous compared to the more restricted state-based search techniques, as it enforces decisions such as variable bindings, only if necessary. POCL planning has greater flexibility at plan execution time (Muise, McIlraith, and Beck 2011) and eases the integration for handling resource or temporal constraints and durative actions (Vidal and Geffner 2006; Coles et al. 2010). Its knowledge-rich plans furthermore enable the generation of formally sound plan explanations (Seegebarth et al. 2012). However, due to the complex structure of partial plans, developing well-informed heuristics for POCL planning is a challenging task (Weld 2011) and heuristics are still rare. To address the lack of informed heuristics for POCL planning, we propose an idea of how to use heuristics already known from state-based search, rather than developing new *specific* heuristics.

POCL Planning

A planning domain is a tuple $\mathcal{D} = \langle \mathcal{V}, \mathcal{A} \rangle$, where $\mathcal{V}$ is a finite set of state variables and $\mathcal{A}$ is a finite set of actions, each having the form (pre, add, del), where $pre, add, del \subseteq \mathcal{V}$. $2^{\mathcal{V}}$ is the set of states and an action is applicable in a state $s \in 2^{\mathcal{V}}$ if its precondition pre holds in s, i.e., $pre \subseteq s$. Its application generates the state $(s \setminus del) \cup add$. The applicability and application of action sequences is defined as usual. A planning problem in STRIPS notation is a tuple $\pi = \langle \mathcal{D}, s_{init}, g \rangle$ with $s_{init} \in 2^{\mathcal{V}}$ being the initial state and $g \subseteq \mathcal{V}$ being the goal description. A solution to π is an applicable action sequence starting in s_{init} and generating a state $s' \supseteq g$ that satisfies the goal condition.

POCL planning is a technique that solves planning problems via search in the space of partial plans. A *partial plan* is a tuple $(PS, \prec, CL)$. PS is a set of plan steps, each being a pair $l{:}a$ with an action $a \in \mathcal{A}$ and a unique label $l \in L$ with L being an infinite set of label symbols to differentiate multiple occurrences of the same action within a partial plan. The set $\prec \subset L \times L$ represents ordering constraints and induces a partial order on the plan steps in PS. CL is a set of causal links. A causal link $(l, v, l') \in L \times \mathcal{V} \times L$ testifies that the precondition $v \in \mathcal{V}$ of the plan step with label l' is provided by the action with label l. That is, if $l{:}(pre, add, del) \in PS$, $l'{:}(pre', add', del') \in PS$, and $(l, v, l') \in CL$, then $v \in add$ and $v \in pre'$. Furthermore, we demand $l \prec l'$ if $(l, v, l') \in CL$.

Now, π can be represented as a POCL planning problem $\langle \mathcal{D}, P_{init} \rangle$, where $P_{init} := (\{l_0{:}a_0, l_\infty{:}a_\infty\}, \{(l_0, l_\infty)\}, \emptyset)$ is the *initial partial plan*. The actions a_0 and a_∞ encode the initial state and goal description: a_0 has no precondition and s_{init} as add effect and a_∞ has g as precondition and no effects. A solution to a POCL planning problem is a partial plan P with no *flaws*. There are two flaw classes: $\mathcal{F}_{OpenPrecondition}$ and $\mathcal{F}_{CausalThreat}$. An *open precondition* in $\mathcal{F}_{OpenPrecondition}$ is a tuple $(v, l) \in \mathcal{V} \times L$ and specifies that the precondition v of the plan step with label l is not yet protected by a causal link. A *causal threat* in $\mathcal{F}_{CausalThreat}$ is a tuple $(l, (l', v, l'')) \in L \times CL$ and specifies that the ordering constraints $\prec$ allow the plan step $l{:}(pre, add, del)$ with $v \in del$ to be ordered in such a way that $\prec \cup \{(l', l), (l, l'')\}$ induces a partial order. That is, the plan step with label l *threatens* the causal link (l', v, l''), since it might undo its protected condition v.

If a partial plan P has no flaws, then every linearization of its plan steps that respects the ordering constraints is a solution to the corresponding planning problem π in STRIPS notation.

POCL planning can be regarded as a refinement procedure (Kambhampati 1997), since it *refines* the initial partial plan P_{init} step-wise until a solution is generated. To that end, first a partial plan P is selected, which is based on heuristics estimating the goal-distance or quality of P. Given such a partial plan P, a flaw selection function selects one of its flaws and resolves it. For that end, all *modifications* are generated, which are all possibilities to resolve the given flaw. There are three modification classes, each specifying modifications addressing certain flaw classes. A causal threat flaw $(l, (l', v, l'')) \in \mathcal{F}_{CausalThreat}$ can only be resolved by *promotion* or *demotion*. Promotion and demotion modifications belong to the class of $\mathcal{M}_{InsOrdering}$ and are ordering constraints, which promote the plan step with label l before the one with label l' or demote it behind the one with label l''. An open precondition flaw $(v, l) \in \mathcal{F}_{OpenPrecondition}$ can only be resolved by inserting a causal link (l', v, l) which protects the open precondition v. This can be done either by using a plan step already present in the current partial plan, or by a new action from $\mathcal{A}$ – the corresponding modification classes are $\mathcal{M}_{InsCausalLink}$ and $\mathcal{M}_{InsAction}$, respectively.

The procedure of selecting a partial plan, calculating its flaws, and selecting and resolving a flaw is repeated until a partial plan P without flaws is generated. Hence, P is a solution to the POCL planning problem and returned.

Heuristics for POCL Planning

In this section we briefly review the current state of the art heuristics for selecting a partial plan in POCL planning.

Although there are many heuristics for POCL planning, most of them are based on pure syntactical criteria like the number of open precondition flaws or the ratio of certain flaws to the number of plan steps, etc. (Younes and Simmons 2003; Schattenberg 2009). However, we are only aware of *two* heuristics for POCL planning which are based on a well-informed *means-ends analysis*: the *Additive Heuristic for POCL Planning* h^r_{add} (Younes and Simmons 2003) and the *Relax Heuristic* h_{relax} (Nguyen and Kambhampati 2001). The first one is a variant of the add heuristic (Haslum and Geffner 2000), whereas the second one can be regarded as a variant of the FF heuristic (Hoffmann and Nebel 2001).

While these heuristics are the currently best-informed heuristics available for (non-temporal) POCL planning, they both ignore the negative effects of the plan steps in the current partial plan, although those could be used to strengthen their heuristic estimates. In contrast to that, our technique allows, in principle, heuristics to use all information given by the current partial plan. To pinpoint our observation, we briefly review h^r_{add}[1].

The heuristic h^r_{add} takes as input a set of open preconditions of the partial plan and estimates the effort to achieve them based on a reachability analysis assuming sub-goal independence and delete relaxation.

Let $\langle\langle \mathcal{V}, \mathcal{A}\rangle, P_{init}\rangle$ be a POCL planning problem, $V \subseteq \mathcal{V}$ a set of state variables, $v \in \mathcal{V}$ such a state variable, $A(v) := \{(pre, add, del) \in \mathcal{A} \mid v \in add\}$ the set of actions with an add effect v, and $a := (pre, add, del) \in \mathcal{A}$ an action. Then, h^r_{add} is based on the following functions:

$$h^{variables}_{add}(V) := \sum_{v' \in V} h^{aVariable}_{add}(v')$$

$$h^{aVariable}_{add}(v) := \begin{cases} 0 & \text{if } v \in s_{init} \\ \min\limits_{a \in A(v)} h^{anAction}_{add}(a) & \text{if } A(v) \neq \emptyset \\ \infty & \text{else} \end{cases}$$

$$h^{anAction}_{add}(a) := 1 + h^{variables}_{add}(pre)$$

The heuristic $h^r_{add}(P)$, which does *reuse* actions in the current partial plan $P = (PS, \prec, CL)$, is now defined by $h^{variables}_{add}(g_P)$, where g_P is a subset of all open preconditions. Let $OC \subseteq \mathcal{V} \times L$ be the set of all open preconditions of P. Then, $g_P := \{v \mid (v, l) \in OC$ and there is no l':$(pre, add, del) \in PS$, s.t. $v \in add$ and the set $\prec \cup \{(l', l)\}$ induces a partial order$\}$. Thus, g_P is the set of all open preconditions for which new plan steps must be inserted in order to resolve these flaws.

It is easy to see that the given partial plan P and its structure are only used to identify open preconditions. Positive interactions are used only to a certain extent and negative interactions are completely ignored. Of course, the very idea of this heuristic is the delete relaxation; however, to ignore the negative effects of actions in PS is an *additional* relaxation, which might lead to a strong underestimation of the heuristic. The original version of the add heuristic for state-based search takes a current *state* s as input, and the heuristic assumes that all state variables of s remain true. Since there is no such state in our setting, h^r_{add} assumes that all state variables of s_{init} remain true. However, this assumption is much more severe than in the original version of the add heuristic, since in state-based search, s reflects *all effects* of *all actions* leading to s, whereas h^r_{add} does only incorporate the *positive effects* of all actions leading to P, but not its negative ones.

We argue that the plan structure and the positive *and negative* interactions of the plan steps given in the partial plan should be used to improve heuristic estimates. Our proposed technique allows to take all these factors into account.

New Heuristics for POCL Planning

Our idea to make the heuristics from state-based planning available to POCL planning involves encoding the current partial plan by means of an altered planning problem, s.t. estimating the goal distance for that *partial plan* corresponds to estimating the goal distance for the initial *state* in the new planning problem.

Please note that a similar encoding was already proposed by Ramírez and Geffner (Ramírez and Geffner 2009). However, their transformation was used in the context of plan recognition for compiling observations away.

[1] We do not review both heuristics, because h_{relax} is basically just an improvement of the add heuristic taking into account positive interactions to a larger extent.

Transformation

Our transformation works as follows: given a planning problem in STRIPS notation $\pi = \langle\langle \mathcal{V}, \mathcal{A}\rangle, s_{init}, g\rangle$ and a partial plan $P = (PS, \prec, CL)$, let $enc_P(\pi, P) = \langle\langle \mathcal{V}', \mathcal{A}'\rangle, s'_{init}, g'\rangle$ be the *encoding* of π and P with:

$$\begin{aligned}
\mathcal{V}' &:= \mathcal{V} \cup \{l_-, l_+ \mid l{:}a \in PS, l \notin \{l_0, l_\infty\}\} \\
\mathcal{A}' &:= \mathcal{A} \cup \{enc_{PS}(l{:}a, \prec) \mid l{:}a \in PS, l \notin \{l_0, l_\infty\}\}, \\
&\text{with } enc_{PS}(l{:}(pre, add, del), \prec) := \\
&\quad (pre \cup \{l_-\} \cup \{l'_+ \mid l' \prec l, l' \neq l_0\}, \\
&\quad\ add \cup \{l_+\}, del \cup \{l_-\}), \\
s'_{init} &:= s_{init} \cup \{l_- \mid l{:}a \in PS, l \notin \{l_0, l_\infty\}\} \\
g' &:= g \cup \{l_+ \mid l{:}a \in PS, l \notin \{l_0, l_\infty\}\}
\end{aligned}$$

The transformed problem subsumes the original one and extends it in the following way: all plan steps present in P are additional actions in $\mathcal{A}'$ – we do not encode the artificial start and end actions, since their purpose is already reflected by the initial state and goal description. The new actions use the labels of their corresponding plan steps as additional state variables to encode whether they have already been executed or not. Thus, for every label we introduce two new state variables: l_- for encoding that the corresponding plan step/action has not yet been executed and l_+ to encode that it has been executed. Initially, none of these plan steps were executed and the (additional) goal is to execute all of them. Furthermore, the new actions use these labels to ensure that they can only be executed in an order consistent with the ordering present in the plan to encode. Please note that we do not encode the causal links for the sake of simplicity, although it is possible.

Before we can state the central property of the transformed problem, we need some further definitions: $ref(P) := \{\langle PS', \prec', CL'\rangle \mid PS' \supseteq PS, \prec' \supseteq \prec, CL' \supseteq CL\}$ is called the set of all *refinements* of P, i.e., the set of all partial plans which can be derived from P by adding plan elements. Let $sol(\pi)$ be the set of all *solution* plans of π. Then, $sol(\pi, P) := sol(\pi) \cap ref(P)$ is the set of all solutions of π, which are refinements of P. The cost of P is denoted by $c(P) := |PS|$.

Theorem 1. *Let π be a planning problem and P a partial plan with no causal links. Then,*

$$\min_{P' \in sol(\pi, P)} c(P') = \min_{P' \in sol(enc_P(\pi, P))} c(P')$$

This theorem states that an optimal solution for π, which also has to be a refinement of P, has the same cost as an optimal solution for the transformed problem. To prove that theorem, we provide two propositions from which it directly follows. The first proposition states that every solution of the original planning problem, which is also a refinement of the given partial plan, does also exist as a solution for the encoded problem. The second proposition states that every solution of the encoded problem can be decoded into a solution of the original one, which is a refinement of the given partial plan, too. Before we can state these propositions formally, we have to show how partial plans derived from $enc_P(\pi, P)$ can be transformed back into plans for π.

Let the *decoding* of a plan step be given by $dec_{PS}(l{:}(pre, add, del)) := l{:}(pre \cap \mathcal{V}, add \cap \mathcal{V}, del \cap \mathcal{V})$ and the decoding of a partial plan be given by $dec_P(\langle PS, \prec, CL\rangle) := \langle\{dec_{PS}(l{:}a) \mid l{:}a \in PS\}, \prec, \{(l, v, l') \in CL \mid v \in \mathcal{V}\}\rangle$.

Proposition 1. *Let π be a planning problem, P a partial plan with no causal links, and $P_{sol} \in sol(\pi, P)$. Then, there exists a plan P'_{sol} with $P'_{sol} \in sol(enc_P(\pi, P))$ and $dec_P(P'_{sol}) = P_{sol}$.*

Proposition 2. *Let π be a planning problem, P a partial plan with no causal links, and $P'_{sol} \in sol(enc_P(\pi, P))$. Then, $dec_P(P'_{sol}) \in sol(\pi, P)$.*

Proof Sketch. Follows from construction. □

Theorem 1 can be exploited by using heuristics known from state-based planning in the context of POCL planning: we want to find a heuristic function $h(\pi, P)$ that estimates the goal distance in π from the *partial plan* P. To that end, we transform π and P into the planning problem $\pi' = enc_P(\pi, P)$ and set $h(\pi, P) := \max\{h_{sb}(\pi', s'_{init}) - c(P), 0\}$, where h_{sb} is any heuristic that takes a *state* as input. We subtract the action cost of P from the estimate, since the heuristic h has to estimate the distance from P, whereas h_{sb} estimates the goal distance from the new initial state thereby including the costs of the plan steps in P. We maximize with zero, in case the heuristic h_{sb} underestimates the optimal goal distance by returning a value smaller than the action costs of the given plan.

From Theorem 1, we can also conclude that we inherit admissibility: if h_{sb} is admissible, h is admissible, too. This is an important property of our technique, as the currently best-informed heuristics for POCL planning, h^r_{add} and h_{relax}, are both not admissible. Our technique thus provides POCL planning with the first admissible heuristics by using admissible heuristics from state-based planning.

Since our transformation ignores causal links, the encoded planning problem is already relaxed if the given partial plan has causal links. Thus, given a partial plan *with* causal links, the equality in Theorem 1 is weakened in such a way that the optimal solution cost of $sol(enc_P(\pi, P))$ is just a lower bound of the optimal cost of $sol(\pi, P)$.

Evaluation

In this section, we evaluate the overhead of performing the encoding process. We begin by examining the canonical approach where the transformation is done for each partial plan independently, followed by an analysis of the costs if one performs an *incremental* transformation. To that end, let $\pi := (\mathcal{V}, \mathcal{A}, s_{init}, g)$, $P := (PS, \prec, CL)$ be a partial plan, and $enc_P(\pi, P) = (\mathcal{V}', \mathcal{A}', s'_{init}, g')$.

We can directly observe that the elements of $enc_P(\pi, P)$ are supersets of the elements in π. The number of their additional elements (state variables and actions) is bounded by a constant factor in the number of plan steps in PS. However, the preconditions of the additional actions in $\mathcal{A}' \setminus \mathcal{A}$ need some further attention. The size of the subset $\{l'_+ \mid l' \prec l, l' \neq l_0\}$ of the precondition of such an action can be linear in the number of plan steps of P. We can hence conclude

that the transformation has a time and space consumption of $\Theta(|\pi| + |PS| + |\prec|)$, which is in $O(|\pi| + |PS|^2)$.

However, we want to minimize the overhead we incur by performing the transformation; thus, we desire an *incremental* encoding of the current partial plan, where the transformed planning problem depends only on the previous one and the modification applied last.

Theorem 2. *Let π be a planning problem, P a partial plan, m a modification resolving some flaw of P thereby generating P', and $\pi' := enc_P(\pi, P)$. Then, π' can be transformed into $\pi'' := enc_P(\pi, P')$ in $O(1)$, given P, and m.*

Proof. We give a constructive proof by providing an algorithm calculating $enc_{P\text{-}inc}(\pi', P, m) := (\mathcal{V}'', \mathcal{A}'', s''_{init}, g'')$, s.t. $enc_{P\text{-}inc}(\pi', P, m) = enc_P(\pi, P')$. The modification m can only belong to one of the classes $\mathcal{M}_{InsOrdering}$, $\mathcal{M}_{InsAction}$, and $\mathcal{M}_{InsCausalLink}$. If m is the insertion of an ordering constraint or a causal link, $\mathcal{V}''$, s''_{init}, and g'' are not changed w.r.t. the elements of $\pi' = (\mathcal{V}', \mathcal{A}', s'_{init}, g')$. If m is a task insertion, these sets are extended by just one or two elements, each. Their incremental construction can thus be performed in constant time. The more interesting part is the calculation of $\mathcal{A}''$. Let $m = (l, l') \in \mathcal{M}_{InsOrdering}$ and $enc_{PS}(l'{:}a, \prec) = (pre, add, del)$ be the encoding of the plan step in PS with label l'. This action must be altered in order to represent the new ordering constraint. Thus, $\mathcal{A}'' := (\mathcal{A}' \setminus \{(pre, add, del)\}) \cup \{(pre \cup \{l_+\}, add, del)\}$. Since we only remove and add one element, we can compute this set in constant time, assuming the set operations are constant-time bounded. Since the insertion of a causal link is only reflected via an ordering constraint, we obtain the same result for $m \in \mathcal{M}_{InsCausalLink}$. For $m \in \mathcal{M}_{InsAction}$, we also have to do the same as for the previous modification classes, as m inserts a new action $a \in \mathcal{A}$ and a causal link (l, v, l') from $l{:}a$ to $l'{:}a'$. In addition, we must insert the action $enc_{PS}(l{:}a, \emptyset)$, which can also be done in constant time. Please note that we can use an empty set of ordering constraints, since this set only determines which plan steps must precede $l{:}a$ - however, since $l{:}a$ is just being inserted, there are no such plan steps, yet. Hence, no further alterations must be made. We have thus shown that $enc_{P\text{-}inc}(\pi', P, m)$ can be calculated in constant time.

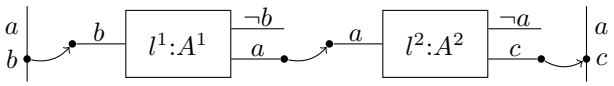
We do not show $enc_{P\text{-}inc}(\pi', P, m) = enc_P(\pi, P')$, since the proof is straight-forward. □

Given a partial plan P', its parent P and the encoding of P, $\pi' := enc_P(\pi, P)$, the theorem states that one can calculate the encoding of P', $\pi'' := enc_P(\pi, P')$, in constant time. However, note that the proof relies on an algorithm which *directly manipulates* π' in order to calculate π''. Thus, in case a partial plan has more than one successor, applying the algorithm given in the proof violates the premise that we have stored the encoding for every plan. To address that problem, it suffices to maintain a copy for every encoded planning problem, which can be done in linear time in the size of the given plan. (Since every encoding contains the original planning problem, it does not need to be copied for each individual partial plan.)

Please note that we only discussed the *runtime* of the *encoding process*. However, for our purpose of using the technique for calculating heuristic estimates, the *size* of the resulting problems is of more importance as the runtime of the heuristic calculation heavily depends on this size.

Example

Let $\pi = \langle\langle \mathcal{V}, \mathcal{A}\rangle, s_{init}, g\rangle$ be a planning problem with $\mathcal{V} := \{a, b, c\}$, $\mathcal{A} := \{(\{b\}, \{a\}, \{b\}), (\{a\}, \{c\}, \{a\})\}$, $s_{init} := \{a, b\}$, and $g := \{a, c\}$. Let P be a partial plan which was obtained by a POCL algorithm as depicted below:



The arrows indicate causal links and A^1 and A^2 are the two actions of $\mathcal{A}$. P has only one open precondition: (a, l_∞), which encodes the last remaining goal condition. Since a is already true in the initial state, both the add and the relax heuristic estimate the effort to be 0. However, the optimal goal distance is even ∞, since there is no refinement of P, which is a solution.

Due to Theorem 1, a heuristic based on the transformed problem can incorporate the negative effects of $l^1{:}A^1$ and $l^2{:}A^2$ and has thus the potential to discover the partial plan/state to be invalid and thus prune the search space. With $\mathcal{A}'$ being defined below, $enc_P(\pi, P)$ is given by $\langle\langle\{a, b, c, l^1_+, l^1_-, l^2_+, l^2_-\}, \mathcal{A}'\rangle, \{a, b, l^1_-, l^2_-\}, \{a, c, l^1_+, l^2_+\}\rangle$.

$$\begin{aligned}\mathcal{A}' := \{&(\{b\}, \{a\}, \{b\}\}),\\ &(\{b, l^1_-\}, \{a, l^1_+\}, \{b, l^1_-\})\\ &(\{a\}, \{c\}, \{a\}),\\ &(\{a, l^2_-, l^1_+\}, \{c, l^2_+\}, \{a, l^2_-\})\}\end{aligned}$$

Discussion

Relaxation *Every* (practically relevant) heuristic performs some kind of relaxation. Therefore, one must investigate which impact the relaxation of actions in $\mathcal{A}_{new} := \mathcal{A}' \setminus \mathcal{A}$ has for the resulting heuristic estimates. Since these actions encode the current partial plan, relaxing them would contradict the goal to use *all* information of the current planning progress. Thus, only relaxing the actions in $\mathcal{A}$, but none in $\mathcal{A}_{new}$ would improve the heuristic accuracy. However, one has to investigate how this can be done for each individual heuristic and how much it would influence the time to calculate its heuristic estimate. But, of course, relaxing them to a certain extent still captures *some* information obtained by the current partial plan.

Preprocessing Some heuristics, like merge and shrink abstraction (Dräger, Finkbeiner, and Podelski 2006; Helmert, Haslum, and Hoffmann 2007), perform a preprocessing step before the actual search and make up for it when retrieving each single heuristic value. Since we obtain a new planning problem for each single partial plan using the results of that preprocessing step might not be possible, directly. Thus, one would have to find a way of using this kind of heuristics in our setting, for instance by updating the result

of the preprocessing incrementally, as it can be done for the transformation itself.

Runtime Although we proved that the transformation itself can be done efficiently, we expect that the computational time of the used heuristics increases with the size of the partial plan to encode. This seems to be a strange property, since one would expect the heuristic calculation time either to remain constant (as for abstraction heuristics) or to *decrease* (as for the FF or add heuristics) as closer a partial plan comes to a solution. However, that might be a direct consequence from partial plans being complex structures, as many interesting decision problems involving them are NP hard w.r.t. their size (Nebel and Bäckström 1994).

Conclusion

We presented a technique which allows planners performing search in the space of plans to use standard classical planning heuristics known from state-based search. This technique is based on a transformation which encodes a given partial plan by means of an altered planning problem, s.t. evaluating the goal distance for the given partial plan corresponds to evaluating the goal distance for the initial state of the new planning problem. We proved that performing the transformation can be done incrementally in constant time under certain assumptions.

We conclude that our technique allows to fuse the benefits of the least-commitment principle and regression-like search of POCL planning with very strong heuristics known from state-based progression search.

An empirical evaluation showing the practical impact of using state-based heuristics in POCL planning is currently ongoing work.

Acknowledgements

This work is done within the Transregional Collaborative Research Centre SFB/TRR 62 “Companion-Technology for Cognitive Technical Systems” funded by the German Research Foundation (DFG).

References

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS 2010)*, 42–49. AAAI Press.

Dräger, K.; Finkbeiner, B.; and Podelski, A. 2006. Directed model checking with distance-preserving abstractions. In Valmari, A., ed., *SPIN*, volume 3925 of *Lecture Notes in Computer Science*, 19–34. Springer.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS 2000)*, 140–149. AAAI Press.

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 2007)*, 176–183.

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

Kambhampati, S. 1997. Refinement planning as a unifying framework for plan synthesis. *AI Magazine* 18(2):67–98.

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI 1991)*, 634–639. AAAI Press.

Muise, C.; McIlraith, S. A.; and Beck, J. C. 2011. Monitoring the execution of partial-order plans via regression. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, 1975–1982. AAAI Press.

Nebel, B., and Bäckström, C. 1994. On the computational complexity of temporal projection, planning, and plan validation. *Artificial Intelligence* 66(1):125–160.

Nguyen, X., and Kambhampati, S. 2001. Reviving partial order planning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*, 459–466. Morgan Kaufmann.

Penberthy, J. S., and Weld, D. S. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the third International Conference on Knowledge Representation and Reasoning*, 103–114. Morgan Kaufmann.

Ramírez, M., and Geffner, H. 2009. Plan recognition as planning. In Boutilier, C., ed., *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 1778–1783. AAAI Press.

Schattenberg, B. 2009. *Hybrid Planning & Scheduling*. Ph.D. Dissertation, University of Ulm, Germany.

Seegebarth, B.; Müller, F.; Schattenberg, B.; and Biundo, S. 2012. Making hybrid plans more clear to human users – a formal approach for generating sound explanations. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 225–233. AAAI Press.

Vidal, V., and Geffner, H. 2006. Branching and pruning: An optimal temporal POCL planner based on constraint programming. *Artificial Intelligence* 170(3):298–335.

Weld, D. S. 2011. Systematic nonlinear planning: A commentary. *AI Magazine* 32(1):101–103.

Younes, H. L. S., and Simmons, R. G. 2003. VHPOP: Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research (JAIR)* 20:405–430.

A Knowledge Engineering Environment for P&S with Timelines

Giulio Bernardi, Amedeo Cesta, Andrea Orlandini
CNR – National Research Council of Italy
Institute for Cognitive Science and Technology
Rome, Italy – {name.surname}@istc.cnr.it

Alberto Finzi
Università di Napoli "Federico II"
Dipartimento di Scienze Fisiche
Naples, Italy – finzi@na.infn.it

Abstract

This paper presents some of the features of a knowledge engineering environment, called KEEN, created to support a timeline based planning based on the APSI-TRF modeling assumptions. A key feature of the environment is the integration of typical tools for knowledge based modeling and refining with services for validation and verification specialized to planning with timelines.

Introduction

Planning and Scheduling (P&S) systems have been deployed in several application domains. Most of these results have been achieved by small group of specialists molding their own specialized know-how. A key objective pursued in the P&S Knowledge Engineering sub community is the synthesis of software environments that would allow the development of applications to people that, as a minimum, are not "leading edge" specialist. Example of such environments are ITSIMPLE (Vaquero et al. 2013), GIPO (Simpson, Kitchin, and McCluskey 2007b), and EUROPA (Barreiro et al. 2012).

Over the last year and a half we have been developing our own Knowledge Engineering ENvironment (KEEN) that is built around the state of the art framework for P&S with timelines called APSI-TRF[1] (Cesta et al. 2009). The particular perspective we are pursuing with KEEN is the one of integrating classical knowledge engineering features connected to support for domain definition, domain refinement, etc. with services of automated Validation and Verification (V&V) techniques as those surveyed in (Cesta et al. 2010b). That paper shows the possible role of V&V techniques in domain validation, planner validation, plan verification etc. It is worth reminding that *validation* allows to check whether models, knowledge bases, and control knowledge accurately represent the knowledge as well as the objectives of the human experts that provided them (i.e., *validation* has to do with *building the right system*), while *verification* checks whether the system (and its components) meets the specified requirements (i.e., *building the system right*). Some further motivation for our work comes from a project in robot autonomy (Ceballos et al. 2011) that pushed us to better investigate problems of robustness of plans at execution time. In particular, working on the representation of flexible temporal plans, that is the key feature of a timeline-based representation, we have obtained results related to checking the dynamic controllability property (Cesta et al. 2010a; 2011) as well as to automatically generate robust plan controllers (Orlandini et al. 2011b). In those works, the problem of verifying flexible plans has been addressed considering an abstract plan view as a set of timelines with formal tools like model checkers. Then, the flexible plan verification problem has been translated in a model checking problem with Timed Game Automata (TGA), exploiting UPPAAL-TIGA (Behrmann et al. 2007) as verification tool. The goal pursued with KEEN synthesis is to obtain an integrated environment where all these results can be situated in a rational tool design and their use facilitated by the software environment.

In a very early paper (Orlandini et al. 2011a) we described the general idea and a sketchy plan to develop KEEN as situated within the GOAC robotic project for ESA (Ceballos et al. 2011). The current paper describes aspects of the current environment. In particular we describe new features of the environment for Domain Definition and Visualization and then some of the V&V tools at work starting from the defined domain. To make the example more concrete we have used as a running example the GOAC domain where we have accumulated quite an amount of basic knowledge.

The paper is organized as follows: a section describes basic knowledge on timelines to set the context and shortly introduces the GOAC domain, then the comprehensive idea of the KEEN system is described Two following sections are dedicated to the functionalities for domain definition and to knowledge engineering services based on V&V. Related works and conclusions end the paper.

Timeline-based Planning

The main modeling assumption underlying the timeline-based approach (Muscettola 1994) is inspired by the classical Control Theory: the problem is modeled by identifying a set of *relevant features* whose temporal evolutions need to be controlled to obtain a desired behavior. In this respect, the set of domain features under control are modeled as a set of temporal functions whose values have to be decided over a time horizon. Such functions are synthesized during prob-

[1] APSI-TRF is a tool of the European Space Agency (ESA) initially designed and built by our CNR group during the Advanced Planning and Scheduling Initiative (APSI).

lem solving by posting planning decisions. The evolution of a single temporal feature over a time horizon is called the *timeline* of that feature[2].

The timeline-based planning is an approach to temporal planning which has been applied to the solution of several space planning problems – e.g., (Muscettola 1994; Jonsson et al. 2000; Smith, Frank, and Jonsson 2000; Frank and Jonsson 2003; Chien et al. 2010). This approach pursues the general idea that P&S for controlling complex physical systems consist in the synthesis of a set of desired temporal behaviors for system features that vary over time.

In this regard, we consider multi-valued *state variables* representing time varying features as defined in (Muscettola 1994; Cesta and Oddi 1996). As in classical control theory, the evolution of controlled features are described by some causal laws which determine legal temporal evolutions of timelines. For the state variables, such causal laws are encoded in a *Domain Theory* which determines the operational constraints of a given domain. Task of a planner is to find a sequence of control decisions that bring the variables into a final set of desired evolutions (i.e., the *Planning Goals*)

GOAC: a test planning domain

This work considers as running example a real world planning domain derived from a project funded by the European Space Agency (ESA). In fact, the Goal Oriented Autonomous Controller project (Ceballos et al. 2011) was an effort to create a common platform for robotic software development. In particular, the delivered GOAC architecture has integrated: (a) a timeline-based deliberative layer which integrates a planner based on the APSI Platform (Cesta et al. 2009) and an executive a la T-REX (Py, Rajan, and McGann 2010); (b) a functional layer which integrates G$^{en}{}_{o}$M and BIP (Bensalem et al. 2010).

Such robotic domain considers a planetary rover equipped with a Pan-Tilt Unit (PTU), two stereo cameras (mounted on top of the PTU) and a communication facility. The rover is able to autonomously navigate the environment, move the PTU, take pictures and communicate images to a Remote Orbiter. A safe PTU position is assumed to be (*pan*, *tilt*) = $(0, 0)$. Finally, during the mission, the Orbiter may be not visible for some periods. Thus, the robotic platform can communicate only when the Orbiter is visible. The mission goal is a list of required pictures to be taken in different locations with an associated PTU configuration. A possible mission action sequence is the following: navigate to one of the requested locations, move the PTU pointing at the requested direction, take a picture, then, communicate the image to the orbiter during the next available visibility window, put back the PTU in the safe position and, finally, move to the following requested location. Once all the locations have been visited and all the pictures have been communicated, the mission is considered successfully completed. The rover must operate following some operative rules to maintain safe and effective configurations. Namely, the following conditions must hold during the overall mission: **(C1)** While the robot is moving the PTU must be in the safe position (pan and tilt at 0); **(C2)** The robotic platform can take a picture only if the robot is motionless in one of the requested locations while the PTU is pointing at the related direction; **(C3)** Once a picture has been taken, the rover has to communicate the picture to the base station; **(C4)** While communicating, the rover has to be motionless; **(C5)** While communicating, the orbiter has to be visible.

Timeline-based specification. To obtain a timeline-based specification of our robotic domain, we consider two types of state variables: *Planned State Variables* to represent timelines whose values are decided by the planning agent, and *External State Variables* to represent timelines whose values over time can only be observed. Planned state variables are those representing time varying features like the temporal occurrence of navigation, PTU, camera and communication operations. We use four of such state variables, namely the *RobotBase*, *PTU*, *Camera* and *Communication*.

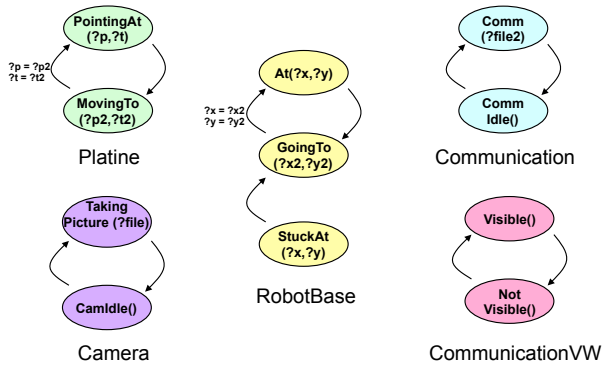


Figure 1: State variables describing the robotic platform and the orbiter visibility (durations are stated in seconds)

In Fig. 1, we detail the values that can be assumed by these state variables, their durations and the legal value transitions in accordance with the mission requirements and the robot physics. Additionally, one external state variable represents contingent events, i.e., the communication opportunities. The *Orbiter Visibility* state variable maintains the visibility of the orbiter. The allowed values for this state variable is *Visible* or *Not-Visible* and are set as an external input. The robot can be in a position (*At(x,y)*), moving towards a destination (*GoingTo(x,y)*) or Stuck (*StuckAt(x,y)*)[3]. The PTU can assume a *PointingAt(pan,tilt)* value if pointing a certain direction, while, when moving, it assumes a *MovingTo(pan,tilt)*. The camera can take a picture of a given object in a position $\langle x, y \rangle$ with the PTU in $\langle pan, tilt \rangle$ and store it as a file in the on-board memory (*TakingPicture(file-id,x,y,pan,tilt)*) or be idle (*CamIdle()*).

[2]According to Wikipedia, a *timeline* is a way of displaying a list of events in chronological order. It is worth saying that this style of planning synthesizes a timeline for each dynamic feature to be controlled. In this paper, the term "timeline-based planning" is considered because recently it is more widely used, see for instance (Chien et al. 2012). Other authors prefer "constraint-based interval planning" (Frank and Jonsson 2003) following a perspective more connected to the technical way of creating plans.

[3]Sometimes, the robot may be stuck in a certain position and the navigation module should be reset.

Similarly, the communication facility can be operative and dumping a given file (*Communicating(file-id)*) or be idle (*ComIdle()*).

Domain operational constraints are described by means of *synchronizations*. A synchronization models the existing temporal and causal constraints among the values taken by different timelines (i.e., patterns of legal occurrences of the operational states across the timelines).

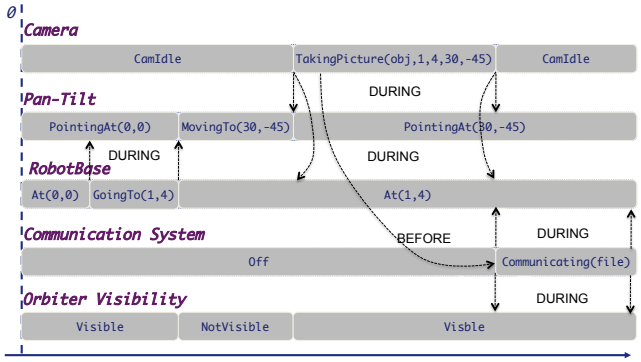


Figure 2: An example of timeline-based plan.

Fig. 2 exemplifies the use of synchronizations implementing the operative rules in our case study domain. The synchronizations depicted are: *GoingTo(x,y)* must occur during *PointingAt(0, 0)* (**C1**); *TakingPicture(pic, x, y, pan, tilt)* must occur during *At(x, y)* and *PointingAt(pan, tilt)* (**C2**); *TakingPicture(pic, x, y, pan, tilt)* must occur before *Communicating(pic)* (**C3**); *Communicating(file)* must occur during *At(x,y)* (**C4**); *Communicating(file)* must occur during *Visible* (**C5**). In addition to those synchronization constraints, the timelines must respect transition constraints among values and durations for each value specified in the domain (see again Fig. 1).

In the actual domain model, an additional state variable is considered: the *Mission Timeline*. Such state variable is used just to model the reception from the external facilities of high level mission goals, i.e., *TakePicture(pic, x, y, pan, tilt)* and *At(x,y)* to model, respectively, the goal of taking a picture with a particular position/PTU setting and just moving the rover to a certain position. These goals are set on the Camera and RobotBase timeline as actual planning goals.

The KEEN System

As explained in (Cesta et al. 2009) the APSI-TRF environment is a development environment that gives "a timeline-based support" for modeling a domain. Its sketchy representation is the core of Figure 3, where it is accessible through a Domain Description Language and a Problem Description Language (the timeline equivalent of analogous files in classical planning) and it has a software machinery (the Component-Based Modeling Engine) that essentially produces a data structure here sketched as "current plan" that indeed is a Decision Network in TRF terminology (Cesta et al. 2009) that is a richer representation for representing the domain, the current problem, the flaw to achieve a solution, and a the flexible temporal plan at the end of a problem solving session. The APSI-TRF has capabilities for plugging in different problem solvers, also more than one for the same problem. For the current purposes we are solving GOAC problems with an APSI-compliant version of OMPS (Fratini, Pecora, and Cesta 2008). In KEEN, the APSI-TRF is surrounded by a set of active services that give support during the knowledge engineering (KE) phase. Indeed in our view the knowledge engineering phase is interpreted in a very broad sense. For example we also have a Plan Execution block that contains a Dispatch Service to send actual commands to a controlled system and an Execution Feedback module that allows to receive the telemetry from an actual plan execution environment. The idea pursued is that you can connect the KEEN to an accurate simulator of the real environment, to a real physical system (e.g., a robot) and have functionalities to monitor with visual tools also the execution phase. We see in Figure 3 how KEEN is composed by "classical tools" you expect in a KE environments and by V&V services.

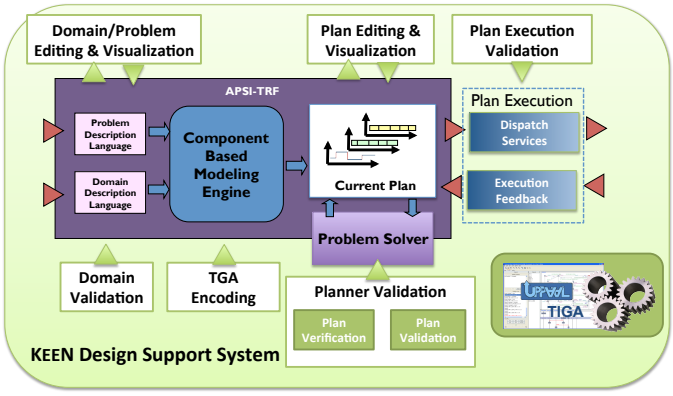


Figure 3: The Knowledge Engineering ENvironment (KEEN) Design Support System.

In particular we here describe the *Domain Editing and Visualization* module that provides initial solution for a user interaction functionality for creating planning domain models. In this respect, we have developed an *Eclipse* plugin that provides a graphical interface to model, visualize and analyze the P&S domains. Additionally, plans can be generated by means of OMPS in a continuous loop of usage. The V&V services, comes from work described in papers like (Cesta et al. 2010b; 2010a; 2011; Orlandini et al. 2011b). They are all based on the use of Timed Game Automata, exploiting UPPAAL-TIGA (Behrmann et al. 2007). As a consequence their entry point is the *TGA Encoding* module that implements a translation from P&S specification to TGA. The other services rely on that encoding. The *Domain Validation* module is to support the model building activity providing a tool to assess the quality of the P&S models with respect to system requirements. Similarly, the *Planner Validation* module is also deputed to assess the P&S solver

with respect to given requirements. But it is worth specifying that two sub-modules are needed: *Plan Verification* to verify the correctness of solution plans and *Plan Validation* to evaluate their goodness. Then, a *Plan Execution Validation and controller synthesis* module is to check whether proposed solution plans are suitable for actual execution as well as to generate robust plan controllers. To implement the modules functionalities, verification tasks are performed by means of UPPAAL-TIGA. Such a tool extends UPPAAL (Larsen, Pettersson, and Yi 1997) providing a toolbox for the specification, simulation, and verification of real-time games. As a result, UPPAAL-TIGA is an additional core engine for KEEN.

Supporting Domain Definition

The most recent work concerning KEEN has concerned the support to timeline-based domain definition. Around this problem we have a first combination of "classical" KE tools and V&V services.

Bidirectional Editing and Visualization

Our goal for KEEN is to provide an integrated environment where the user may work both visually and at the traditional code level, while having the opportunity to easily verify and validate his/her work. A knowledge engineering environment like this is a complex piece of software, and it makes no sense to reinvent the wheel: some of the features we needed are standard and already supported by state-of-the-art development tools.

For this reason, KEEN is implemented as a plugin inside Eclipse[4] platform. Eclipse is one of the most widespread *Integrated Development Environment*s (IDE) for many programming languages, Java above all. It provides a lot of features that are nowadays required for a professional IDE, like syntax highlighting, content assist, inline documentation, strong refactoring support, near real time compilation and code browsing, debugging, testing and integration with external tools, and many others. The Eclipse Platform can be extended by the means of plugins, thus providing a powerful environment for the implementors of new languages, who can leverage Eclipse's key strengths to suit their needs.

KEEN uses standard Eclipse components to provide traditional code-level functionalities:

- A syntax highlighter, which uses different colors for different parts of the code to emphasize language keywords, special types, parameters, literals and so on. The central part of Figure 4 shows the code editor performing syntax highlighting.
- A tree view (Outline view) of relevant code blocks like state variables, providing a fast visualization and navigation inside the source files. In Figure 4, in the lower left corner, the Outline view is showing the GOAC Domain state variables: *RobotBase, Platine, Camera, Communication, MissionTimeline, CommunicationVW*.
- Real time syntax checks to easily spot erroneous programming constructs.

[4]http://www.eclipse.org

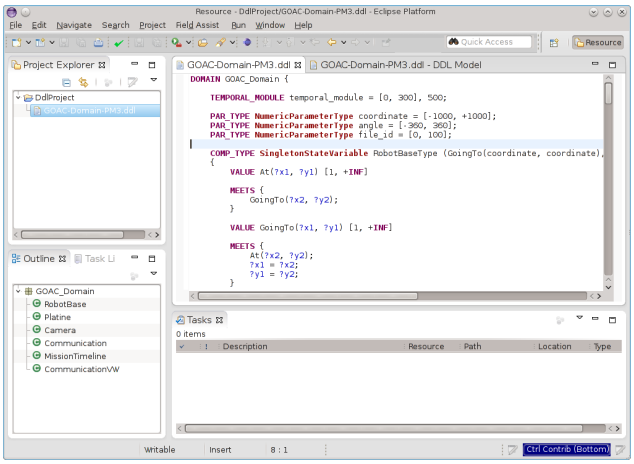


Figure 4: DDL editor based on Eclipse plugin.

As said before, the heart of KEEN is represented by the APSI framework, which, among other things, is used to maintain an updated representation of the problem being worked on. The use of APSI implies that KEEN is very loosely coupled to the particular language being used for Domain or Problem definition: while some Eclipse components (e.g. the syntax highlighter) are implemented for a specific language, their implementation is trivial and often consists in writing a grammar description file and not much more. But the more advanced features of KEEN are built on top of the APSI framework, so that concepts like state variables, timelines, or synchronizations are presented without an explicit dependency on the particular language the code is written in.

At the moment of writing, KEEN is endowed with a graphical representation of the Domain Model, as shown in Figure 5. Other graphical representations, like an execution-time timeline view, are being worked on. In this view, the state variables of the Domain Model are represented on a workbench by colored blocks, which can be moved around and expanded or collapsed to show/hide their values and constraints. Also, the desired state variables can be selected to show their synchronization relations with other state variables, represented by dotted lines between the blocks. The graphical representation has been designed to be as less tied to a particular graphical framework as possible, relying only on standard Java's AWT libraries: this allows the environment to be used inside Eclipse, but has the potentiality of being reused in ad-hoc applications too.

In Figure 5, the six state variables of the GOAC Domain are shown: some of them have been expanded (*Platine, Camera, MissionTimeline*), while the others are collapsed (*RobotBase, Communication, CommunicationVW*). The currently selected variable, *MissionTimeline*, shows its synchronizations (depicted as arrows) with other three state variables: one from *MissionTimeline.TakingPicture()* to *Camera.TakingPicture()*, one from *MissionTimeline.TakingPicture()* to an unspecified value of *Communication* (since *Communication* state variable is col-

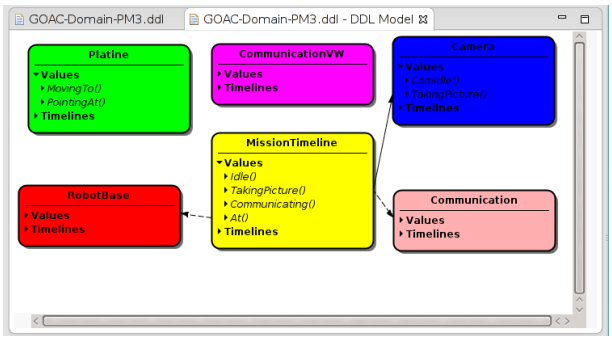


Figure 5: Detail of the graphical view of the model.

lapsed and its values are not shown), and one from *MissionTimeline.At()* to an, again, unspecified value of *RobotBase*.

The user may also use this graphical environment to define new state variables (right-clicking on an empty workbench area) and to add and edit their properties and values. The environment draws its information from the APSI framework, and immediately updates the APSI representation when the user makes a change.

At the end of the chain, when the internal representation of the model is changed, a language-specific component is used to trigger source code modifications, using Eclipse's support for code refactoring. This way the tool allows the user to perform round-trip engineering[5] by synchronizing source code and graphical views: the user can start to define a new model graphically, then switch to the traditional mode and do some hand-made editing, then switch back to the graphical mode and so on.

The integration of traditional IDE features and visual modeling functionalities should help both the experienced domain coder and the beginner or occasional writer: the former will probably use the traditional mode for the most part of its work, switching to the graphical mode to observe the results of its coding, while the latter might feel more comfortable in designing visually, switching to the code view to learn the language and experiment with it.

When the user is satisfied by the model, he/she can ask KEEN to generate a solution plan. Currently, KEEN does this by the means of the OMPS planner, but different planners will be added in the future. As for the case of the domain the plan representation is completely handled by APSI and a specific language generation component is deputed to the creation of a source file encoded with a *Problem Description Language* syntax. The user can then modify the generated solution plan at his/her will, and ask KEEN to perform plan verification using UPPAAL-TIGA (see later for further details). At the time of writing, KEEN allows the user to inspect and modify the generated plan in its textual form. In the future, a specialized plan editor similar to the one used for domain modeling will be added.

Integrating V&V Services

The deployment of formal methods techniques is to enhance the KEEN system with suitable V&V capabilities, thus, constituting one of the main advantages in its use. In this regard, the KEEN takes advantage from a set of research results based on Timed Game Automata model checking (Cesta et al. 2010a; 2011; Orlandini et al. 2011b) to provide support over all the design and development cycle of P&S application with the APSI-TRF.

Timed Game Automata (Maler, Pnueli, and Sifakis 1995) (TGA) allow to model real-time systems and controllability problems representing uncontrollable activities as *adversary moves* within a game between the controller and the environment. Following the approach presented in (Cesta et al. 2010a), flexible timeline-based plan verification can be performed by solving a Reachability Game using UPPAAL-TIGA. To this end, flexible timeline-based plans, state variables, and domain theory descriptions are compiled as a set of TGA (nTGA): (1) a flexible timeline-based plan $\mathcal{P}$ is mapped into a nTGA *Plan*. Each timeline is encoded as a sequence of locations (one for each timed interval), while transition guards and location invariants are defined according to (respectively) lower and upper bounds of flexible timed intervals; (2) the set of state variables *SV* is mapped into a nTGA *StateVar*. Basically, a one-to-one mapping is defined from state variables descriptions to TGA. In this encoding, value transitions are partitioned into controllable and uncontrollable. (3) an *Observer* automaton is introduced to check for value constraints violations and synchronizations violations. In particular, two locations are considered: an Error location, to state constraint/synchronization violations, and a Nominal (OK) location, to state that the plan behavior is correct. The *Observer* is defined as fully uncontrollable. (4) the nTGA $\mathcal{PL}$ composed by the set of automata *StateVar* $\cup$ *Plan* $\cup \{\mathcal{A}_{Obs}\}$ encapsulates flexible plan, state variables and domain theory descriptions.

Considering a Reachability Game $RG(\mathcal{PL}$, *Init*, *Safe*, *Goal*) where *Init* represents the set of the initial locations of each automaton in $\mathcal{PL}$, *Safe* is the Observer's OK location, and *Goal* is the set of goal locations, one for each automaton in *Plan*, plan verification can be performed solving the $RG(\mathcal{PL}$, *Init*, *Safe*, *Goal*) defined above. If there is no winning strategy, UPPAAL-TIGA provides a counter strategy for the opponent (i.e., the environment) to make the controller lose. That is, an execution trace showing a faulty evolution of the plan is provided. The encoding $\mathcal{PL}$ is considered as the basis for implementing the V&V functionalities discussed in the following.

Domain Validation. Similarly to (Khatib, Muscettola, and Havelund 2001), the TGA encoding $\mathcal{PL}$ can be exploited in order to validate planning domains, i.e., checking properties that are useful for ensuring correctness as well as detecting inconsistencies and flaws in the domain specification. For instance, undesired behaviors or safety properties can be checked against the planning model in order to guarantee the validity of the specification. In this regard, the KEEN Domain Validation module is to support knowledge engineers in the process of eliciting, refining and correcting the domain model w.r.t. safety- and system-critical require-

[5] http://en.wikipedia.org/wiki/Round-trip_engineering

ments[6].

To implement such a functionality, deriving from $\mathcal{PL}$ the nTGA *Dom* = *StateVar* $\cup$ $\{\mathcal{A}_{Obs}\}$, representing the allowed behaviors described by the associated planning domain, and stating a suitable Computation Tree Logic (CTL) formula ϕ, representing a given system property F to be checked, verifying ϕ in *Dom* by means of UPPAAL-TIGA corresponds to validate the planning domain with respect to the property F.

Among relevant properties, values reachability is an important aspect that can be checked. Namely, the reachability of a value stated in the planning domain is checked starting from one specific initial state (or from each possible initial state). In this regard, the KEEN environment allows to perform a full reachability test for all the values declared in the domain and, in Fig. 6, the result for the GOAC domain is depicted. In particular, all the stated values are reachable except the *StuckAt* in the navigation state variable that, obviously, cannot be planned but only detected as an abnormal system behavior. In general, finding that a certain value is unreachable may suggest either the presence of incomplete specifications or that some parts of the model are actually needed.

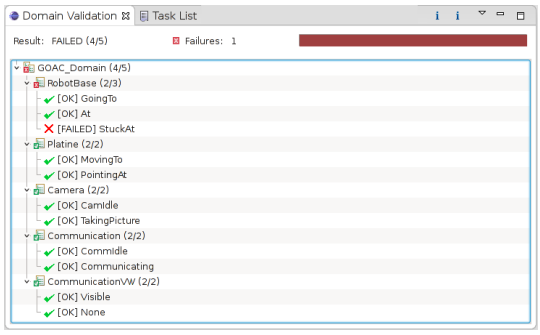


Figure 6: Detail on the Domain Validation frame reporting results of the reachability test for all the allowed domain values.

Also, the KEEN system allows to define user-defined properties to be checked (e.g., undesired or safety properties). For instance, a GOAC user may want to check that the *Communication* value is always reachable after a *TakingPicture*. Having this property satisfied would confirm that the model allows to correctly manage the downlink actions for stored science pictures. This corresponds to check the following formula: $A\Box$ *Camera.TakingPicture(file_id=x,...)* $\rightarrow$ $E\Diamond$ *Communication.Communicating(file_id=x)*.

Another relevant property the user may check through the KEEN system is the violation of mutual exclusion for timeline's allowed values. In fact, such test is useful for detecting an incomplete specification of synchronizations in the planning domain theory. For instance, in the case of a flawed GOAC domain, the property ($E\Diamond$ *RobotBase.GoingTo and Communication.Communicating)*, which reads *there exists a trace where at some point in time the rover is moving while communicating*, could be verified, then, providing an evidence that the (**C4**) domain constraints might be violated.

Planner validation. In order to validate the planner, we are interested in checking that the planning solver works properly. In this sense, the application design activity should be supported by providing effective methods to validate the solver and the generated solutions, i.e., assessing its capability of generating a correct plan and, in addition, also the quality of the generated solution plans should be checked.

For this purpose, the KEEN system has been endowed with two important submodules: Plan Verification, which systematically analyzes the solutions proposed by the planner itself, and Plan Validation, which allows to assess the plan quality. Errors or negative features possibly found in the generated plans could help knowledge engineers to revise the model (back to the domain validation step), the heuristics, or the solver. Furthermore, plan V&V is also to analyze the produced plans with respect to execution controllability issue. The KEEN Plan Verification and Plan Validation modules have been implemented exploiting the verification method presented in (Cesta et al. 2010a), i.e., solving the Reachability Game $RG(\mathcal{PL}, Init, Safe, Goal)$ defined as above.

Plan Verification and Dynamic Controllability Check. The Plan Verification module is fully relying on UPPAAL-TIGA by *winning* the Reachability Game $RG(\mathcal{PL}, Init, Safe, Goal)$. Then, the KEEN system invokes UPPAAL-TIGA for checking the CTL formula $\Phi = A$ *[Safe U Goal]* in $\mathcal{PL}$. In fact, the formula Φ states that along all its possible temporal evolutions, $\mathcal{PL}$ remains in *Safe* states until *Goal* states are reached. That is, in all the possible temporal evolutions of the timeline-based plan all the constraints and the plan is completed. Thus, if the solver verifies the above property, then the flexible temporal plan is valid. Whenever the flexible plan is not verified, UPPAAL-TIGA produces an execution strategy showing one temporal evolution that leads to a fault. Such a strategy can be analyzed in order to check for plan weaknesses or for the presence of flaws in the planning model.

In Fig. 7, a plan for the GOAC domain is verified and the system reports about its correctness taking advantage of the UPPAAL-TIGA verification process. Also, the dynamic controllability (Morris, Muscettola, and Vidal 2001) is checked and, in this case, successfully verified.

The feasibility of such method has been shown in (Cesta et al. 2010a; Orlandini et al. 2011b) where the verification methodology has been applied in two real-world planning domains.

Plan validation. Besides synchronization constraints, users may need also to take into account other constraints which cannot be naturally represented as temporal synchronizations among specific activities. Nevertheless, these constraints, that we call *relaxed constraints*, define a kind of preferences on the global behavior of the generated plan. These requirements may be not explicitly represented in the

[6]It is worth underscoring that in (Khatib, Muscettola, and Havelund 2001) only controllable events are considered while, here, also uncontrollable actions are modeled and taken into account.

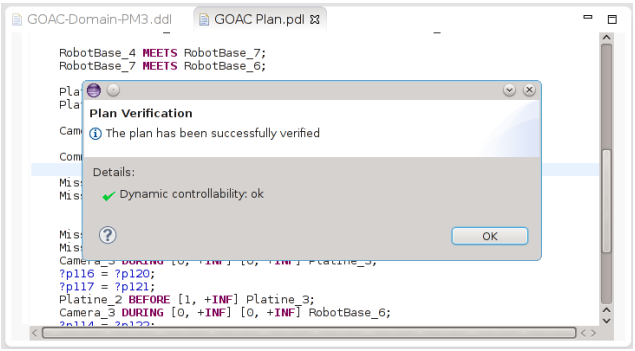


Figure 7: The KEEN system showing the textual description of a plan for the GOAC domain. The pop-up window reports the result of the UPPAAL-TIGA verification. The plan is correct and, also, it results dynamically controllable.

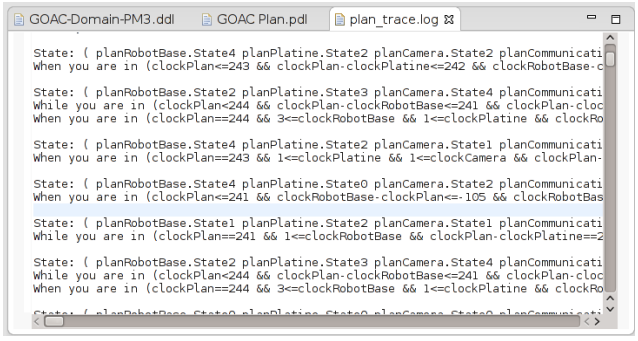


Figure 8: The execution strategy generated by UPPAAL-TIGA is currently reported in the KEEN interface.

planning model as structural constraints, but rather treated as meta-level requirements to be enforced by the planner heuristics and optimization methods. Then, to implement the Plan Validation module, it is possible to apply the same verification process as in plan verification, verifying not only plan correctness, but also other domain-dependent constraints, i.e., the relaxed constraints. In general, the additional properties to be checked carry a low additional overhead to the verification process. Thus, the verification tool performances are not affected.

Examples of such relaxed constraints in the robotic case study may be no unnecessary tasks have been planned (e.g., unnecessary robot navigation tasks). This validation task results as an important step in assessing the plan quality as well as the planner effectiveness.

Plan Controllers Synthesis. Plans synthesized by temporal P&S systems may be temporally flexible hence they identify an envelope of possible solutions aimed at facing uncertainty during actual execution. In this regard, a valid plan can be brittle at execution time due to environment conditions that cannot be modeled in advance (e.g., *disturbances*). Previous works have tackled these issues within a Constraint-based Temporal Planning (CBTP) framework deploying specialized techniques based on temporal-constraint networks. Several authors (Morris, Muscettola, and Vidal 2001; Morris and Muscettola 2005; Shah and Williams 2008) proposed a *dispatchable execution* approach where a flexible temporal plan is then used by a plan executive that schedules activities on-line while guaranteeing constraint satisfaction. Some recent works have addressed aspects of plan execution extending the approach in (Cesta et al. 2010a) by presenting the formal synthesis of a plan controller associated to a flexible temporal plan (Orlandini et al. 2011b). In particular, UPPAAL-TIGA is exploited in order to synthesize a robust execution controller of flexible temporal plans. In Figure 8, the execution strategy generated by UPPAAL-TIGA for a GOAC plan is shown by the KEEN system in a text format. Then, such strategy can be parsed and embedded in an executive system to actually implement a robust execution strategy.

Related Works and Conclusions

As said in the introduction, there are only a few general purpose KE tools for planning: GIPO (Simpson, Kitchin, and McCluskey 2007a), ITSIMPLE (Vaquero et al. 2013), and EUROPA (Barreiro et al. 2012). Several tools do exist that address specialized aspects, nevertheless these three systems are the reference ones.

Most of the existing work has been dedicated to classical PDDL underlying language (this is the case for ITSIMPLE and to some extent for GIPO). EUROPA has been till now the only timeline-based developing environment endowed with KE features.

With respect to both EUROPA and other research we are pursuing some distinctive features. For example the round-trip engineering functionalities in KEEN are rather new. While some of the existing systems can export to PDDL, and sometimes also allow the user to edit the produced PDDL file (as in ITSIMPLE), they do not support an integrated work practice in which the users can seamlessly switch between graphical and code views while maintaining the consistency between both views.

Standalone validation tools like VAL (Howey, Long, and Fox 2004) for PDDL language do exist, and are used by integrated environments to perform validation as in ModPlan (Edelkamp and Mehler 2005). Systems like GIPO and ITSIMPLE do support static and dynamic analysis of the domains. The dynamic analysis though is performed by means of manual steppers (for GIPO) or simulation through Petri Nets (for ITSIMPLE). ITSIMPLE also supports plan analysis by simulation.

Nevertheless, it is worth underscoring that simulation is not the same as the formal validation and verification proposed in KEEN. Somehow KEEN aims at filling a hole in existing knowledge engineering tools and nicely contribute to the whole picture.

Clearly there are other aspects, for example referring to those covered in the survey (Vaquero, Silva, and Beck 2011), that are still not address in KEEN. They will deserve specific work in the future.

Acknowledgments. Authors are supported by CNR under the GECKO Project (Progetto Bandiera “La Fabbrica del Futuro”).

References

Barreiro, J.; Boyce, M.; Do, M.; Franky, J.; Iatauro, M.; Kichkaylo, T.; Morris, P.; Ong, J.; Remolina, E.; Smith, T.; and Smith, D. 2012. EUROPA: A Platform for AI Planning, Scheduling, Constraint Programming, and Optimization. In *ICK-EPS 2012: the 4th Int. Competition on Knowledge Engineering for Planning and Scheduling*.

Behrmann, G.; Cougnard, A.; David, A.; Fleury, E.; Larsen, K.; and Lime, D. 2007. UPPAAL-TIGA: Time for playing games! In *Proc. of CAV-07*, number 4590 in LNCS, 121–125. Springer.

Bensalem, S.; de Silva, L.; Gallien, M.; Ingrand, F.; and Yan, R. 2010. "Rock Solid" Software: A Verifiable and Correct-by-Construction Controller for Rover and Spacecraft Functional Levels. In *i-SAIRAS-10. Proc. of the 10^{th} Int. Symp. on Artificial Intelligence, Robotics and Automation in Space*.

Ceballos, A.; Bensalem, S.; Cesta, A.; de Silva, L.; Fratini, S.; Ingrand, F.; Ocon, J.; Orlandini, A.; Py, F.; Rajan, K.; Rasconi, R.; and van Winnendael, M. 2011. A Goal-Oriented Autonomous Controller for Space Exploration. In *ASTRA-11. 11th Symposium on Advanced Space Technologies in Robotics and Automation*.

Cesta, A., and Oddi, A. 1996. DDL.1: A Formal Description of a Constraint Representation Language for Physical Domains,. In Ghallab, M., and Milani, A., eds., *New Directions in AI Planning*. IOS Press: Amsterdam.

Cesta, A.; Cortellessa, G.; Fratini, S.; and Oddi, A. 2009. Developing an End-to-End Planning Application from a Timeline Representation Framework. In *IAAI-09. Proceedings of the 21^{st} Innovative Application of Artificial Intelligence Conference, Pasadena, CA, USA*.

Cesta, A.; Finzi, A.; Fratini, S.; Orlandini, A.; and Tronci, E. 2010a. Analyzing Flexible Timeline Plan. In *ECAI 2010. Proceedings of the 19th European Conference on Artificial Intelligence*, volume 215. IOS Press.

Cesta, A.; Finzi, A.; Fratini, S.; Orlandini, A.; and Tronci, E. 2010b. Validation and Verification Issues in a Timeline-Based Planning System. *Knowledge Engineering Review* 25(3):299–318.

Cesta, A.; Finzi, A.; Fratini, S.; Orlandini, A.; and Tronci, E. 2011. Flexible plan verification: Feasibility results. *Fundamenta Informaticae* 107:111–137.

Chien, S.; Tran, D.; Rabideau, G.; Schaffer, S.; Mandl, D.; and Frye, S. 2010. Timeline-Based Space Operations Scheduling with External Constraints. In *ICAPS-10. Proc. of the 20th International Conference on Automated Planning and Scheduling*.

Chien, S. A.; Johnston, M.; Frank, J.; Giuliano, M.; Kavelaars, A.; Lenzen, C.; and Policella, N. 2012. A Generalized Timeline Representation, Services, and Interface for Automating Space Mission Operations. In *SpaceOps*.

Edelkamp, S., and Mehler, T. 2005. Knowledge acquisition and knowledge engineering in the ModPlan workbench. In *Proceedings of the First International Competition on Knowledge Engineering for AI Planning*.

Frank, J., and Jonsson, A. 2003. Constraint Based Attribute and Interval Planning. *Journal of Constraints* 8(4):339–364.

Fratini, S.; Pecora, F.; and Cesta, A. 2008. Unifying Planning and Scheduling as Timelines in a Component-Based Perspective. *Archives of Control Sciences* 18(2):231–271.

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

Jonsson, A.; Morris, P.; Muscettola, N.; Rajan, K.; and Smith, B. 2000. Planning in Interplanetary Space: Theory and Practice. In *AIPS-00. Proceedings of the Fifth Int. Conf. on Artificial Intelligence Planning and Scheduling*, 177–186.

Khatib, L.; Muscettola, N.; and Havelund, K. 2001. Mapping Temporal Planning Constraints into Timed Automata. In *TIME-01. The Eigth Int. Symposium on Temporal Representation and Reasoning*, 21–27.

Larsen, K. G.; Pettersson, P.; and Yi, W. 1997. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer* 1(1-2):134–152.

Maler, O.; Pnueli, A.; and Sifakis, J. 1995. On the Synthesis of Discrete Controllers for Timed Systems. In *STACS*, LNCS, 229–242. Springer.

Morris, P. H., and Muscettola, N. 2005. Temporal Dynamic Controllability Revisited. In *Proc. of AAAI 2005*, 1193–1198.

Morris, P. H.; Muscettola, N.; and Vidal, T. 2001. Dynamic Control of Plans With Temporal Uncertainty. In *Proc. of IJCAI 2001*, 494–502.

Muscettola, N. 1994. HSTS: Integrating Planning and Scheduling. In Zweben, M. and Fox, M.S., ed., *Intelligent Scheduling*. Morgan Kauffmann.

Orlandini, A.; Finzi, A.; Cesta, A.; Fratini, S.; and Tronci, E. 2011a. Enriching APSI with Validation Capabilities: the KEEN environment and its use in Robotic. In *ASTRA 2011. Proc. of 11th Symposium on Advanced Space Technologies in Robotics and Automation. Noordwijk, the Netherlands*.

Orlandini, A.; Finzi, A.; Cesta, A.; and Fratini, S. 2011b. Tga-based controllers for flexible plan execution. In *KI 2011: Advances in Artificial Intelligence, 34th Annual German Conference on AI.*, volume 7006 of *Lecture Notes in Computer Science*, 233–245. Springer.

Py, F.; Rajan, K.; and McGann, C. 2010. A Systematic Agent Framework for Situated Autonomous Systems. In *AAMAS-10. Proc. of the 9^{th} Int. Conf. on Autonomous Agents and Multiagent Systems*.

Shah, J., and Williams, B. C. 2008. Fast Dynamic Scheduling of Disjunctive Temporal Constraint Networks through Incremental Compilation. In *ICAPS-08*, 322–329.

Simpson, R. M.; Kitchin, D. E.; and McCluskey, T. L. 2007a. Planning Domain Definition using GIPO. *Knowl. Eng. Rev.* 22(2):117–134.

Simpson, R. M.; Kitchin, D. E.; and McCluskey, T. L. 2007b. Planning Domain Definition Using GIPO. *Knowledge Eng. Review* 22(2):117–134.

Smith, D.; Frank, J.; and Jonsson, A. 2000. Bridging the Gap Between Planning and Scheduling. *Knowledge Engineering Review* 15(1):47–83.

Vaquero, T. S.; Silva, J. R.; Tonidandel, F.; and Beck, J. C. 2013. itsimple: Towards an integrated design system for real planning applications. *The Knowledge Engineering Review*.

Vaquero, T.; Silva, J.; and Beck, J. 2011. A brief review of tools and methods for knowledge engineering for planning & scheduling. In *Proc. of the ICAPS Workshop on Knowledge Engineering for Planning and Scheduling (KEPS 2011)*.

Towards AI Planning Efficiency: Finite-domain State Variable Reformulation

Filip Dvořák[1,2] and **Daniel Toropila**[1,3] and **Roman Barták**[1]
{filip.dvorak, daniel.toropila, roman.bartak}@mff.cuni.cz

[1]Faculty of Mathematics and Physics, Charles University
Malostranské nám. 2/25, 118 00 Prague, Czech Republic

[2]LAAS-CNRS
BP 54200, 31031 Toulouse, France

[3]Computer Science Center, Charles University
Ovocný trh 5, 116 36 Prague, Czech Republic

Abstract

AI Planning is inherently hard and hence it is desirable to derive as much information as we can from the structure of the planning problem and let this information be exploited by a planner. Many recent planners use the finite-domain state-variable representation of the problem instead of the traditional propositional representation. However, most planning problems are still specified in the propositional representation due to the widespread modeling language PDDL and it is hard to generate an efficient state-variable representation from the propositional model. In this paper we propose a novel method for automatically generating an efficient state-variable representation from the propositional representation. This method groups sets of propositions into state variables based on the mutex relations introduced in the planning graph. As we shall show experimentally, our method outperforms the current state-of-the-art method both in the smaller number of generated state variables and in the increased performance of planners.

Introduction

The task of AI Planning is to find a sequence of actions that transfers the world from some initial state to a state satisfying certain goal condition. Planning is inherently hard (Erol, Nau, and Subrahmanian 1995) and we can not expect a general solving algorithm that would be able to solve any planning problem in a reasonable time.

The efficiency of planning systems is strongly dependent on the formulation of the problem. There are many possible formulations of the same problem and planning systems behave differently on each of the formulations. Frequently, the structure of the problem is exploited to improve efficiency of planners and the relations of mutual exclusion (mutex) between the propositions and actions are among the most widely used families of structural information. Mutex between two entities says that they interfere with each other and cannot occur together in the same context.

In this paper we exploit the mutex relations introduced by the planning graph to automatically reformulate the planning problem specified in a set representation into a finite-domain state-variable representation. We use the construction of the planning graph to generate the grounded representation of the planning problem as well as to discover mutex relations between the propositions (and actions). The state variable is then defined by a set of propositions that are pairwise mutex in the fixed-point layer of the planning graph. Naturally, the larger set of propositions a state variable covers, the more compact representation we obtain. Hence we also suggest methods for finding large cliques of mutex relations and for finding a covering of propositions using these (not necessarily disjoint) cliques. The experimental results confirmed that the method generates models leading to significantly better performance for certain domains and planners.

The paper is organized as follows. We will first introduce the necessary terminology and related work. After that we will describe our method in detail, including the algorithms for finding cliques, covering the mutex graph by cliques, and generating the state variables. We conclude the paper by an experimental study showing the positive effect of the proposed method on the efficiency of planners.

Background and Related Work

The use of *state variables* for representing states in planning problems is an idea with a long history, formally first analyzed as the SAS+ representation in (Bäckström and Nebel 1995). Although the PDDL notation (McDermott et al. 1998) was originally defined using only Boolean variables, it has been shown by (Dovier, Formisano, and Pontelli 2007) and (Helmert 2009) that there is a significant number of planning approaches that can directly benefit from the state variable representation. The key benefit is a more compact encoding of the world state as a set of values of state variables instead of a set of truth-values of propositional variables. If each state variable originates from a set of pairwise mutex propositions (propositions that cannot occur at the same time point during the execution of any valid plan),

then both the state-variable and the propositional representations are equivalent. In the state-variable representation we only lose those states that could not have ever been reached.

Demonstrative Example

Let us assume an example with a ship moving between three ports. Having three facts:

Position(Ship, Port1),
Position(Ship, Port2),
Position(Ship, Port3),

where no pair of them can be true at the same time, we can encode those three facts as a single state variable:

$$\text{ShipPosition:} \rightarrow \{\text{Port1, Port2, Port3}\}$$

Although the example is very simple and the knowledge it captures could have been used by the modeller of the planning domain, other knowledge inferred from the planning problem can be non-trivial. Assume that there is a sailor, who is the only one who can sail the ship, and there is City1, where the sailor is initially located. Let Port1, where the ship is initially located, be the only accessible port from City1. Then we can deduce that if the sailor is in City1 or Port1, the ship cannot be at Port2 or Port3 (there is no one to sail the ship there). These are the additional mutex relations that we exploit when building the state variables. A new state variable can then combine the position of the sailor with the position of the ship as follows:

$$\text{Position:} \rightarrow \{\text{ShipPort2, ShipPort3, SailorPort1, SailorCity1}\}$$

Notice that we also assume the knowledge of the initial state (the problem instance in the PDDL) when finding the mutex relations. The knowledge of the initial state allows us to derive more mutex relations than if we only used the planning domain. There have been previous approaches for identifying mutex invariants in (Rintanen 2000), (Gerevini and Schubert 1998) and (Helmert 2009), while the latest is currently being used the most, for example by the planners in the International Planning Competition (IPC). It uses the monotonic invariants synthesis whose description is beyond the scope of this paper. Instead of the monotonic invariants, we use the planning graph structure not for the actual planning, but for the initial grounding of the problem and for the mutex relations it records. The use of a planning graph in a preprocessing step has already been seen in (Gazen and Knoblock 1997), however to the best of knowledge we are the first to make use of the planning graph structure to first ground the problem and then use the mutex relations captured in the planning graph for building up a finite-domain state variable representation. By itself the process of finding the same coverage of the mutex relations as the one provided by the planning graph is also known as a variant of Rintanen's invariant synthesis algorithm (Rintanen 2000).

Planning Graph

The *planning graph* was originaly introduced as a part of the GraphPlan planning system (Blum and Furst 1997) that also directly used it for planning. Though the planning graph is no more used for "complete" planning, it is still heavily exploited in planning heuristics.

For a given planning problem, the planning graph is a directed layered graph (P_0, A_1, P_1, A_2, ...), where arcs exist only from one layer to the next. Nodes in the first level P_0 represent propositions from the initial state s_0. Every further level contains two layers, an action layer A_i and a proposition layer P_i. The action layer contains a set of nodes representing actions whose preconditions are satisfied by the facts in the previous proposition layer and none of its preconditions is mutex (see below) in the proposition layer. The proposition layer contains a set of positive effects of actions from the previous layer (to resolve the frame axiom, for each proposition we add a no-op action with a single precondition and a single positive effect being the proposition). For each level i of the planning graph we maintain a set μP_i of all pairs of mutex facts and set μA_i of all pairs of mutex actions. We further define mutex relations as follows.

Two actions a and b are mutex at the level i if and only if:

$$\begin{gathered}\text{effect}^-(a) \cap (\text{precond}(b) \cup \text{effect}^+(b)) \neq \emptyset \vee \\ \text{effect}^-(b) \cap (\text{precond}(a) \cup \text{effect}^+(a)) \neq \emptyset \vee \\ \exists (p, q) \in \mu P_{i-1} : p \in \text{precond}(a),\ q \in \text{precond}(b)\end{gathered}$$

Two facts p and q are mutex at the level i if and only if:

$$\forall a, b \in A_i\text{: } p \in \text{effect}^+(a),\ q \in \text{effect}^+(b) \Longrightarrow (a, b) \in \mu A_i$$

In the essence, two actions are mutex if either one of them deletes a precondition or an effect of the other or some pair of preconditions of the actions is mutex in the previous level. Two propositions are mutex, if there does not exist any non-mutex pair of actions that would add them in the previous level.

The planning graph is constructed iteratively from the first level representing the initial state of the planning problem. The construction terminates once it eventually reaches a fixed point (no new actions have been added). Note that all the mutex relations in the fixed-point layer of the planning graph are "general". It means that those "general" mutex propositions can never be true at the same state, which is easy to verify. We will use the "general" mutex relations only so from now on, we will be talking about mutexes without the adjective "general".

An important aspect of the planning graph is that it is of polynomial size and can be computed in the time polynomial in the size of the planning problem (Blum and Furst 1997).

Building State Variables

Once we have the mutex relations we need to form the state variables. In its base form, we can perceive the state variable as a set of pair-wise mutually exclusive predicates, or in other words, a clique in a graph (V, E) where V represents all the propositional facts about the world and $(x, y) \in E$ if and only if x and y forms a general mutex. Although it is possible to relax the requirement on mutex relations, e.g. allowing a few pairs to be non-mutex as seen in (Seipp and Helmert 2011) to obtain larger groups of predicates, we shall consider only the cliques as the candidates for the state variables. We are then faced with two tasks, we first need to find the cliques (which is NP-hard in general) and after that we

need to find the most efficient covering of the original facts (set covering problem, again NP-hard in general). To denote a clique of mutexes we will also use the term *mutex set* later in the paper.

We harvest the cliques (mutex sets) from the mutex graph using a probabilistic algorithm presented in the section . For the covering we use a greedy approach as has already been seen in (Helmert 2009). We also explore a new way of covering that encodes more information than necessary by overcovering the predicates (having one predicate represented in multiple state variables).

Problem Representation

Our method works with the PDDL version 3.1. However we support only features that were used for the classical deterministic track of the International Planning Competition (Olaya, López, and Jiménez 2011), further development of the method can include extending towards axioms, conditional effects and other features improving expressivity. Since the description of the PDDL language is beyond the scope of this paper, we define only the *set representation*, which we receive after grounding the initial problem from the PDDL. We then define the *state variable representation* and describe the translation process.

Set Representation

The planning problem in the set representation is defined as a 4-tuple (V, I, G, A), where:

- V is a set of atoms; an assignment of truth values to all the atoms is called the state of the world,
- I is a set of initial atoms,
- G is a set of goal atoms,
- A is a set of actions, where an action is 4-tuple (name, PRE, EFF$^-$, EFF$^+$), where PRE is a set of atoms that need to be true before we can apply the action, EFF$^-$ (EFF$^+$) is a set of atoms that become false (true) after the execution of the action.

The usual semantic (Gelfond and Lifschitz 1998)(Ghallab, Nau, and Traverso 2004) is that the initial state I describes the world precisely (under the closed world assumption, setting all the atoms either to true, or false), while the goal set G describes the world only partially, requiring only some atoms to be true.

Finite-domain State Variable Representation

The planning problem in the state variable representation is defined as a 4-tuple (S, s_0, s^*, O), where:

- $S = \{v_1, \ldots, v_n\}$ is a set of state variables, each with an associated domain of values $D(v_i)$, the state of the world is defined as an assignment of values to all the state variables,
- O is a set of actions of the form (name, PRE, EFF), where PRE is a set of requests on the value of certain state variable (e.g., Position == SailorPort1) and EFF is a set of assignments of values into the state variables (e.g., Position $\leftarrow$ ShipPort3),
- s_0 is a set of assignments of all state variables, it represents the initial world state,
- s^* is a set of assignments of some state variables, it represents the partial description of the goal state.

Translation

The translation method can then be described in the following steps:

1. We use the planning graph structure to translate the input problem in PDDL into the set representation. We also record all the mutex relations from the fixed-point layer of the planning graph.
2. Using the mutex relations we harvest the mutex cliques (mutex sets), i.e., the candidates for the state variables.
3. We select the mutex cliques that we shall use for the state variables.
4. We translate the set representation into the state variable representation using the selected mutex cliques.

The process of translating set representation into the finite-domain state variable representation is the following:

1. For each mutex set we create a state variable whose domain contains all the elements (propositional facts) of the mutex set. We also add a value `none-of-those` that represents the situation when none of the values of the state variable is true.
2. For each action a in the set representation we create an action a' in the state variable representation. For each atom in PRE (EFF$^+$) of the action a we create a request (assignment) of the atom into the state variable whose domain contains the atom. In case a delete effect removes an atom and there is no effect that would add a value from the same variable, we add an effect that sets the value of the concerned state variable to `none-of-those`.
3. We translate the initial and goal states in the same way as the preconditions of an action.

Maximum Clique

A clique is a complete subgraph of a graph. Finding a maximum (the largest) clique in a general graph is a known NP-hard problem (Bomze et al. 1999). For the purpose of finding the candidates for the state variables we do not actually want to look for all the largest cliques since the benefit of having the largest ones is small compared to the computational price we would have to pay for finding them. Instead we shall look for a selection of large cliques that we can find in some limited time. We use a probabilistic algorithm, as described in Algorithm 1.

The main loop of the algorithm (1) records new sets of mutexes until an ending criterion is met. For each set, we first randomly permute the ordering of the facts (2), then we enter the inner loop that in each iteration chooses the first candidate (5), adds it to a new set of mutexes (6) and removes from the candidates all the facts that are not mutex with it (7). The identical algorithm is also used in (Ghallab, Nau, and Traverso 2004) for finding minimal critical sets.

Algorithm 1 Probabilistic algorithm for finding k candidates for the state variables.

```
1: repeat
2:     cands ← randomPermutationOfFacts
3:     newSet ← empty set
4:     repeat
5:         candidateFact ← popFirst(cands)
6:         newSet ← newSet ∪ {candidateFact}
7:         cands ← cands \ nonmutex(candidateFact)
8:     until cands is empty
9:     record(newSet)
10: until time is up or k sets were recorded
```

In our testing environment we have fixed the number of mutex sets we like to find to 5000, the time limitation was not needed. We have not identified any significant improvement for larger numbers of mutex sets.

Covering

Consider a universe $\mathcal{U}$ consisting of all atomic elements (facts). From the elements of the universe we can then form sets, each set corresponding to a clique in the mutex graph. Let us denote $\mathcal{S}$ a union of all such sets, i.e., $\mathcal{S}$ is a set of sets of atomic elements, or a set of cliques. Our task is then to choose such (minimal) subset of $\mathcal{S}$, the union of which will be equal to the whole universe $\mathcal{U}$. Or in other words, we want to choose a (minimal) subset of cliques that will cover all the facts.

The fact covering problem we are solving is known as the set covering problem (Vazirani 2001). As described above, the goal is to find a minimal (or the least expensive) subset of $\mathcal{S}$ such that all atomic elements are covered by the union of sets in the chosen subset. An obviously good solution would be a selection of sets covering all the facts such that there is no selection with fewer sets. However this is an NP-hard variant of set covering. Again, we shall not invest computational time into finding the solution with the fewest sets, whose benefit is discussable, but we employ a greedy algorithm, described in Algorithm 2.

Algorithm 2 Greedy set-covering algorithm.

```
1: uncovered ← allFacts
2: repeat
3:     chosen ← a mutex group with the largest cardinality
4:     remove chosen's elements from all mutex groups
5:     remove chosen's elements from uncovered
6:     record(chosen)
7: until uncovered is empty
```

The main loop of the algorithm runs until we have covered all the facts in the problem at which point all the facts are represented in a form of values of some state variables. In the loop we first choose the largest set (this is the greedy principle), we remove its content from all the mutex groups and the uncovered set. At the end we shall have a collection of sets of mutexes with pair-wise empty intersections (each fact is represented exactly once).

While having every fact represented exactly once is an expected property, we shall also experiment with covers, where each fact is represented at least once (possibly more times).

From the semantical point of view, if there is an action that sets the value of a certain fact to true, then using the state variables, the same action sets a corresponding value of the state variable instead. Having one fact represented by more state variables only extends the transformation of the action, so that a change of a fact is replaced by changes in all state variables that contain the fact. Although the size of the encoding for the over-covering is larger than for the standard greedy covering, the over-covering representation is more informative.

For the purpose of our experiments the over-covering method alters the greedy algorithm by first creating a copy of all the mutex sets and pairing them as an identity projection from the original to the copy. Running the greedy algorithm, whenever a new set should be recorded (line 6), we instead record its projected set from the copy. As a result, the intersections of the mutex sets may be non-empty.

Experimental evaluation

The main goal of our approach is to capture more mutex-based structural information about a planning problem and then to encode it transparently in the form of state variables, so that the existing planners using SAS$^+$ as an input could immediately leverage from this enhancement.

As described above, we created two versions of our encoding, both of them being based on the greedy mutex groups selection, but with the difference of the second encoding not ensuring the empty intersections of facts between the selected mutex groups. The first encoding, which we call *pg-greedy*, leads to a more compact representation of the problem and also action definitions. The second encoding, named *pg-greedy-nofilter*, captures more mutex relations between the facts but inherently creates larger representations of both actions and state variables, reaching thus a certain level of over-covering the facts by the selected state variables (mutex groups).

In order to evaluate our approach we have generated the new encodings for seven PDDL domains used in the sequential-optimization track of the latest International Planning Competition IPC-2011 (Olaya, López, and Jiménez 2011), namely barman, elevators, floortile, openstacks, parcprinter, pegsol, and transport. We have chosen domains, where we expected the new encoding to make difference. From each domain we selected the first 10 problem instances, which was enough to reach at minimum one unsolved problem within each domain. For the comparison, we also generated the encodings of the same domains using the translator component (called *fd-greedy* in further text) of the Fast Downward Planning System (Helmert 2009), obtaining thus three sets of the encodings. The computational time required to generate the encodings was not an object of our study, however based on our observations the time was comparable in all three cases and negligible with regard to the runtime of the planner. Note that the focus on the smaller instances and a subset of domains was caused mainly by the time requirements of such experiments (in fact, we would be

completely re-running the deterministic optimal part of IPC, if we tested all).

For the evaluation we chose seven planning systems, in alphabetic order: BJOLP (Domshlak et al. 2011a), Fast Downward Autotune (optimizing version) (Fawcett et al. 2011), Fast Downward Stone Soup 1 (optimizing version) (Helmert et al. 2011), LM-Cut (Helmert and Domshlak 2011), Merge and Shrink (Nissim, Hoffmann, and Helmert 2011), SASE (Huang, Chen, and Zhang 2010), and Selective Max (Domshlak et al. 2011b). Except for SASE, all of the planners are cost-optimal and participated in the IPC 2011. SASE optimizes the makespan of the plan with the unified action durations and ignores the action costs, which corresponds to the step-parallel semantics (Balyo, Barták, and Toropila 2011).

We then conducted the experiments using all combinations of the generated encodings and chosen planners. The computations ran with the time limit of 30 minutes. For each planner and each type of encoding we observed the time required to solve a given problem instance, and also the number of problems solved in each domain. The experiments were run on the Ubuntu Linux machine equipped with Intel Coreł i7-2600 CPU @ 3.40 GHz and 8GB of memory.

In optimal planning, one of the most interesting performance metrics is the number of solved instances (within given limit). In Table 1 we show the improvement (or degradation) of using the *pg-greedy* encoding compared to the *fd-greedy* encoding used in Fast Downward. The columns represent planners, the rows correspond to the domains, and the value corresponding to a combination of planner and domain denotes how many more problems were solved using the *pg-greedy* encoding compared to using the classical *fd-greedy* encoding. Similarly, Table 2 shows the comparison of the *pg-greedy-nofilter* encoding compared to the *fd-greedy* encoding.

Table 1: Comparison of the *pg-greedy* and *fd-greedy* encodings from the perspective of the number of solved problems. Values denote the improvement of the *pg-greedy* encoding compared to the *fd-greedy* encoding.

	bjolp	fdAutotune	fdss1	lmCut	mergeAndShrink	sase	selmax	**sum**
barman	0	0	0	0	0	**4**	0	**4**
elevators	0	-1	0	0	0	0	0	-1
floortile	0	0	0	0	-1	0	**1**	0
openstacks	0	0	0	0	0	0	0	0
parcprinter	-1	0	0	0	**1**	0	0	0
pegsol	0	0	0	0	0	0	0	0
transport	0	0	0	0	**3**	0	0	**3**
sum	-1	-1	0	0	**3**	**4**	**1**	**6**

As can be seen, the difference between the three encodings is small, though the results show that in average case the novel encodings outperform the *fd-greedy* encoding, with the *pg-greedy* encoding being the winner. The use of this encoding caused degradation only at two problem instances for two planners (BJOLP and Fast Downward Autotune), while for other three planners it helped to solve together 8 instances more (Merge and Shrink, SASE, Selective Max).

Table 2: Comparison of the *pg-greedy-nofilter* and *fd-greedy* encodings from the perspective of the number of solved problems. Values denote the improvement of the *pg-greedy-nofilter* encoding compared to the *fd-greedy* encoding.

	bjolp	fdAutotune	fdss1	lmCut	mergeAndShrink	sase	selmax	**sum**
barman	0	0	0	0	0	**4**	0	**4**
elevators	0	0	0	0	0	0	0	0
floortile	0	0	0	0	-1	**2**	0	**1**
openstacks	-4	0	0	0	0	0	0	-4
parcprinter	0	0	-2	0	**1**	0	0	-1
pegsol	0	0	0	0	0	0	0	0
transport	0	0	0	0	**3**	0	0	**3**
sum	-4	0	-2	0	**3**	**6**	0	**3**

The differences between the encodings can be, however, better seen when inspecting the runtimes required to solve the testing instances. Table 3 shows the comparison of the solving time between the *pg-greedy* and *fd-greedy* encodings. Values represent the average increment of time (in percent) required to solve problems within a domain to the slower of the two compared encodings. Positive values denote that the use of *pg-greedy* encoding reached solution faster, negative values mean the use of *fd-greedy* helped to find an optimal solution faster. For example, value 7 means that the computation using the *pg-greedy* performed faster and the same computation using the *fd-greedy* took 7 per cent longer (i.e., value 100 denotes that the computation using *pg-greedy* was twice as fast, value 200 three times, etc.). Value -8 denotes that the computation using the *fd-greedy* was faster and the use of *pg-greedy* took 8 per cent longer time. Table 4 then shows the same comparison between *pg-greedy-nofilter* and *fd-greedy* encodings.

Table 3: Comparison of the *pg-greedy* and *fd-greedy* encodings from the perspective of the time required to solve the problem instances. Values denote the improvement of the *pg-greedy* encoding compared to the *fd-greedy* encoding (computational overhead in per cent).

	bjolp	fdAutotune	fdss1	lmCut	mergeAndShrink	sase	selmax	**average**
barman	**13**	**10**	**41**	**9**	**39**	**205**	**10**	**47**
elevators	-3	-37	**3**	-53	**6**	-15	-10	-16
floortile	-1	-2	-135	-2	-5	-10	-5	-23
openstacks	-3	-2	**1**	**1**	-2	**0**	**12**	**1**
parcprinter	-3	-1	**30**	**35**	**32**	**11**	**30**	**19**
pegsol	**7**	**23**	**153**	**4**	**2062**	**33**	**3**	**326**
transport	-8	**8**	-3	**7**	**1**	**11**	**1**	**2**
average	**0**	0	**13**	**0**	**305**	**34**	**6**	**51**

We can see that the use of *pg-greedy* helped to reach the optimal solution faster in many cases, while the degradation

Table 4: Comparison of the *pg-greedy-nofilter* and *fd-greedy* encodings from the perspective of the time required to solve the problem instances. Values denote the improvement of the *pg-greedy-nofilter* encoding compared to the *fd-greedy* encoding (computational overhead in per cent).

	bjolp	fdAutotune	fdss1	lmCut	mergeAndShrink	sase	selmax	**average**
barman	-60	-41	-2	-41	-3	-5	-36	-27
elevators	-2	**112**	**29**	**10**	-1	**3**	**35**	**27**
floortile	-6	-9	**224**	-9	**238**	**930**	-8	**194**
openstacks	-1291	**2**	-4	**1**	**1**	**0**	-1	-185
parcprinter	-68	-83	**93**	-76	**104**	-21	-65	-17
pegsol	-3	-20	-16	-14	-16	-1	-1	-10
transport	-2	-2	**3**	**1**	-1	-14	-9	-3
average	-205	-6	**47**	-18	**46**	**127**	-12	**-3**

occurred only rarely and only with small difference. Especially in the logically complicated domains, such as barman and pegsol we can see that the use of *pg-greedy* leads in consistently better performance for all of the planners. The reason for this is the fact that employing the problem definition and the knowledge of the initial state into the reconstruction of the state variables leads to the structurally richer encodings, which helps to improve the performance significantly. We can also observe that the use of *pg-greedy-nofilter* was beneficial only for a single domain floortile, while for other domains the larger encodings of the problems caused complications to the planners.

With regard to the number of state variables and actions, the most significant benefit of our encoding was the parcprinter domain, reducing the number of state variables by 41 per cent and the barman domain reducing state variables by 65 per cent. Also, fewer actions were generated in the pegsol (7%), barman(16%) and parcprinter (7%) domains. The transport, elevators and openstacks domains reached the same number of actions and states variables in all encodings. There is not a single case when the *fd-greedy* would provide fewer state variables or actions than *pg-greedy*.

Looking at the planners, some of them exploit the state variable encodings more than the others. It can be seen that LM-Cut does not really benefit from the encoding, since its heuristic is independent on the encoding. On the other hand, SASE is a planner that encodes the state variables directly into SAT and benefits strongly from the more compact encodings. The Merge and Shrink seems to benefit significantly from more compact encodings (parcprinter and barman), while if we look deeper into the pegsol domain, we find that the *pg-greedy* actually finds new large cliques not discovered by the *fd-greedy*.

Conclusion

We have proposed a new method for constructing the finite-domain state-variable representation of planning problems. This method is fully automated, it uses the planning graph structure for both grounding and discovering the mutual exclusion between facts. Our input can be both a PDDL formulation of the problem that we can ground and translate to the state variable representation, or the state variable representation itself, which we can reconstruct by translating it into the set representation and then back.

We have shown that compared to the current techniques, there are planning domains and planning systems that can significantly benefit from our method, allowing them both to solve more problem instances, and solve them faster.

Although the construction of the planning graph runs in polynomial time, it is still computationaly expensive and it cannot scale with the size of the problem as well as the invariant synthesis used in *fd-greedy*. However, to plan efficiently in practical purposes we want to balance the time we have between the preprocessing and actual planning. Optimal planning is a field where the scale of the problems does not get out of reach for the construction of the planning graph (since it would get out of reach for the planning itself as well) and this is the spot, where we believe our contribution fits the most; in good cases saving a significant amount of computational time, while not having significant bad cases.

Future Work

We have shown that there is still space for exploring new methods for covering the mutex sets and that the results can often be rewarding. The planning graph provides strong mutex relations, however we can still go further and try to enrich the number of mutex relations with other techniques. And probably the most important direction can be the relaxation of cliques allowing some limited amount of edges to be non-mutex, in favor of getting less and larger state variables.

Acknowledgements

Research is supported by the Czech Science Foundation under the contract P103/10/1287 and under the Grant Agency of Charles University as the project no. 306011.

References

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

Balyo, T.; Barták, R.; and Toropila, D. 2011. Two Semantics for Step-Parallel Planning: Which One to Choose? In *Proceedings of the 29th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2011)*, 9–15.

Blum, A., and Furst, M. L. 1997. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence* 90(1-2):281–300.

Bomze, I. M.; Budinich, M.; Pardalos, P. M.; and Pelillo, M. 1999. The Maximum Clique Problem. *Handbook of Combinatorial Optimization* 4:1–74.

Domshlak, C.; Helmert, M.; Karpas, E.; Keyder, E.; Richter, S.; Seipp, J.; and Westphal, M. 2011a. BJOLP: The Big Joint Optimal Landmarks Planner. In *Seventh International Planning Competition (IPC 2011), Deterministic Part*, 91–95.

Domshlak, C.; Helmert, M.; Karpas, E.; and Markovitch, S. 2011b. The SelMax Planner: Online Learning for Speeding up Optimal Planning. In *Seventh International Planning Competition (IPC 2011), Deterministic Part*, 108–112.

Dovier, A.; Formisano, A.; and Pontelli, E. 2007. Multivalued Action Languages with Constraints in CLP(FD). In *Proceedings of the 23rd International Conference on Logic Programming*, ICLP'07, 255–270. Berlin, Heidelberg: Springer-Verlag.

Erol, K.; Nau, D. S.; and Subrahmanian, V. S. 1995. Complexity, Decidability and Undecidability Results for Domain-Independent Planning. *Artificial Intelligence* 76(1-2):75–88.

Fawcett, C.; Helmert, M.; Hoos, H.; Karpas, E.; Röger, G.; and Seipp, J. 2011. FD-Autotune: Automated Configuration of Fast Downward. In *Seventh International Planning Competition (IPC 2011), Deterministic Part*, 31–37.

Gazen, B. C., and Knoblock, C. A. 1997. Combining the Expressivity of UCPOP with the Efficiency of Graphplan. In *Fourth European Conference on Planning (ECP'97)*, 221–233.

Gelfond, M., and Lifschitz, V. 1998. Action Languages. *Electronic Transactions on AI* 3.

Gerevini, A., and Schubert, L. K. 1998. Inferring State Constraints for Domain-Independent Planning. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98*, 905–912. AAAI Press / The MIT Press.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers.

Helmert, M., and Domshlak, C. 2011. LM-Cut: Optimal Planning with the Landmark-Cut Heuristic. In *Seventh International Planning Competition (IPC 2011), Deterministic Part*, 103–105.

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 *Seventh International Planning Competition (IPC 2011), Deterministic Part*, 38–45.

Helmert, M. 2009. Concise Finite-Domain Representations for PDDL Planning Tasks. *Artificial Intelligence* 173(5-6):503–535.

Huang, R.; Chen, Y.; and Zhang, W. 2010. A Novel Transition Based Encoding Scheme for Planning as Satisfiability. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010*. AAAI Press.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL - The Planning Domain Definition Language. Technical report, Yale Center for Computational Vision and Control.

Nissim, R.; Hoffmann, J.; and Helmert, M. 2011. The Merge-and-Shrink Planner: Bisimulation-based Abstraction for Optimal Planning. In *Seventh International Planning Competition (IPC 2011), Deterministic Part*, 106–107.

Olaya, A.; López, C.; and Jiménez, S. 2011. International Planning Competition, Retrieved from http://ipc.icaps-conference.org/.

Rintanen, J. 2000. An Iterative Algorithm for Synthesizing Invariants. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, AAAI 2000*, 806–811.

Seipp, J., and Helmert, M. 2011. Fluent Merging for Classical Planning Problems. In *Proceedings of the ICAPS-2011 Workshop on Knowledge Engineering for Planning and Scheduling*, 47–53.

Vazirani, V. 2001. *Approximation Algorithms*. Springer-Verlag.

What is a Timeline?

Jeremy Frank
NASA Ames Research Center
Mail Stop N269-3
Moffett Field, California 94035-1000

Abstract

Most planning applications require reasoning about actions that take time, consume or produce resources, depend on numbers to characterize, and that contain complex constraints over these elements. In the past several years, various attempts have been made to characterize the notion of *Timelines* as a key part of fielded planning and scheduling applications. These attempts have not been satisfactory. The thesis of this paper is that the Timeline provides a set of services allowing the planner to interact with the computed history of the state variables defined in the planning problem. The Timeline must be computed from partial plans that incompletely specify the history; that is, there may be parts of the planning horizon where the value of the variable is constrained, but not yet known. The paper examines how different assumptions on the commitments made by planners influence the computational complexity of determining these histories, and responding to queries.

Why don't we really know what a Timeline is?

Timeline (n): a graphical representation of a period of time, on which important events are marked. - Oxford English Dictionary (Online)

The word *Timeline* has been used to mean many different things by many different people. Recently, (Chien et al. 2012) describe a set of features shared by the technologies used to build many different planning and scheduling systems, mostly for space applications. These features include:

1. Ability to represent actions that overlap in time
2. Ability to represent finite capacity renewable resources (e.g. resource with integer capacity that can be used and returned)
3. Ability to represent multi-valued discrete state variables and constraints on state transitions.
4. Ability to represent infinite valued state variables.
5. Ability to represent hierarchical activity decomposition.
6. Ability to represent metric temporal constraints.
7. Ability to represent complex functional constraints between activity properties or parameters.

Chien et al. also describe the services that these planners offer:

- Detect constraint violations
- Insert activities
- Check whether an activity insertion violates a constraint (e.g. lookahead)
- Return a list of valid times to place an activity
- Invoke custom code to calculate complex quantities
- Remove activities

Of this list of features, hierarchical decomposition and complex functional constraints are key representational features of many powerful modeling languages, but have nothing to do with Timelines per-se. With the exception of multi-valued discrete state variables, the remainder are all features of PDDL, starting with 2.1 (Fox and Long 2003), the defacto standard academic modeling language. Multi-valued state variables are a feature of the SAS modeling language (Jonsson and Backstrom 1998), and it is possible to automatically translate PDDL models into SAS (Edelkamp and Helmert 1999), so that some predicates in the original PDDL model are combined into the values of a state variable. This suggests that Timelines are not just a modeling formalism, but something more.

Of the list of services, many PDDL planners make use of few of these services, or employ them in a restricted form. Progression and regression planners, for example, can add activities to the suffix of a plan (progression) or prefix (regression) but nowhere else. All planners check the validity of the plan relative to the modeled constraints, but many planners are heuristics-driven chronological backtracking planners that retract decisions once constraint violations are detected. Other services also are not specific to Timelines, e.g. the ability to invoke custom code. Only some of these services, e.g. the ability to return a list of all valid times when an activity can be placed, are directly relevant to the notion of a Timeline. For this reason, a more deliberate attempt to distill the essence of Timelines is in order.

The Contentious Proposition

This paper offers one approach to determining what, exactly, a Timeline is, what role it plays in automated planners, and

how Timelines should be designed. The position of this paper is that *a Timeline is a complete history of the changes to the value of a variable, as computable by a partially specified plan, over the specified planning horizon*. The Timeline must also either be, or interact closely with, a repository of all of the commitments made by a planner that are necessary to determine what the history is. The Timeline also *provides a set of services allowing the planner to interact with the variables' histories*. The histories must be computed from commitments that incompletely specify the history; that is, there may be parts of the planning horizon where the value of the variable is constrained, but not yet known. The paper also examines how different assumptions on the commitments made by planners influence the computational complexity of determining these histories, and responding to queries.

Assumptions and Definitions of Terms

- We assume that the planning and scheduling system (hereafter the planner) is model-based.
- We assume the model contains durative actions and temporally qualified conditions and effects.
- We assume the modeling language allows numerical fluents.
- We assume that the planner can produce plans with flaws; flaws may include actions whose preconditions are not satisfied, constraint violations including resource limit violations, and so on.

We make no specific assumptions regarding the planning algorithms used to find plans, even those with flaws. Part of the motivation of this paper (and others before it) is to provide a basic reasoning component that can be used as part of many planning algorithms and applications. A single component must be useful in any of the above contexts.

Before proceeding, we provide a few definitions of terms. These definitions are an amalgam of those in (Ghallab, Nau, and Traverso 2004) (Chapters 14 and 15), (Frank and Jónsson 2003) and (Ghallab and Laurelle 1994) with a few modifications to suit the purposes of this paper[1]. Timelines have also been formally defined in a similar manner in (Fratini, Pecora, and Cesta 2008), and using a 'constraints' based formalism in (Verfaillie and Pralet 2008).

Definition 1 *Let $\mathcal{V}$ be a set of (possibly infinite) domain variables v; denote the domain of v by $d(v)$. Let t be a variable denoting time; w.l.o.g. $d(t) = \mathcal{R}^+$. A* State Variable *x is a function $x : t \rightarrow v^i \times ...v^j$. The domain of x is therefore $\mathcal{R}^+ \times d(v^i) \times ... \times d(v^j)$.*

Note that the above definition differs from prior definitions of State Variables, but only in a cosmetic way. If $|d(v^i)| > 2$ then state variables are multi-valued. It is always possible (if somewhat ungainly) to construct a single multi-valued domain out of many state variables with smaller domains. Let $V = \cup_i v^i$; without loss of generality, state variables partition V [2].

Definition 2 *A* Temporal Assertion *is a relation over variables $\{v^1...v^i, s, e\}$ partially specifying the function for state variable x; specifically, that values of the variables $v^1...v^i$ are constrained over the half-open interval $[s, e)$. The variables s, e are called events. We denote the lower and upper bound on events by s_{lb}, s_{ub} respectively.*

Definition 3 *A* Transaction *is a relation over variables $\{v, s\}$ partially specifying the function for state variable x; specifically, the* relative *change in the value of variable v occurs at time s.*

Temporal assertions and transactions are the primitives that will define the behavior of our state variables. Again, these definitions differ from those used previously. Foreshadowing a bit, the value $x(t)$ is either defined by all temporal assertions that must hold at t, that is, by all temporal assertions such that $s \leq t \leq e$, or by all transactions that must occur at or before t, that is, given a set of transactions $\{x_i, v_i, s_i\}, x(t) = \sum_{s_i \leq t} v_i$. We also take an "interval-centric" view rather than an "event-centric" view of temporal assertions. Finally, we note persistence can be modeled as flexibility in the duration of a temporal assertion.

Constraints are an important foundation of automated reasoning (Dechter 2003), and in the last decade automated planners have incorporated many types of constraints. We distinguish between constraint types below.

Definition 4 *A* Constraint *is a tuple $w^1...w^i, s^1...s^j, e^1...e^k, R$ where R is a relation defining the allowed combinations of assignments to the variables. A* Temporal Constraint *is a tuple s, e, t_1, t_2 with semantics $t_1 \leq |e - s| \leq t_2$. Distinguished events z, h with $z = 0$ and $h \in \mathcal{R}+$ define the earliest and latest times any temporal assertion may hold. A* Parameter Constraint *is a constraint whose scope only includes parameter variables $w^1...w^i$. A* Hybrid Constraint *is a constraint whose scope includes both parameters and events.*

The scope of the constraint types defined is not limited to the variables of a single temporal assertion, transaction, or state variable. We now introduce two other constraint types, whose scope is limited to the variables from temporal assertions on the same state variable.

Definition 5 *A* Mutual Exclusion Constraint *states that a state variable x takes on a single value at any time t, that is, if two temporal assertions of the same state variable x necessarily overlap, then the values of their variables must be identical. Formally if the two assertions are $\{x, v^{i,t}...v^{j,t}, s^t, e^t\}$ and $\{x, v^{i,u}...v^{j,u}, s^u, e^u\}$ and $\exists r | s^t \leq r \leq e^t \wedge s^u \leq r \leq e^u$ for all valid assignments to s^t, e^t, s^u, e^u, then $\forall j v^{j,t} = v^{j,u}$.*

Definition 6 *A* Resource Constraint *is a pair of functions U, L s.t. $U, V : v \times t \rightarrow \mathcal{R}$ and $\forall t\, U(v, t) \geq L(v, t)$. Formally, if $x(t) = d$, then $U(v, t) \geq d \geq L(v, t)$.*

[1] A more complete history of formalisms for specifying temporally qualified intervals and planning deserves its own book!

[2] Equality constraints can be used to impose relationships between state variables' parameters.

We use the term 'resource' constraint here to make clear what the intended semantics of the constraint is, even though this is a fairly generic constraint that could apply to any single variable element of a state variable.

Definition 7 *A* Timeline *is a set of temporal assertions or transactions and constraints on a state variable* x.

It is worth noting that (Ghallab, Nau, and Traverso 2004) define a *chronicle* as the set of all state variables, temporal assertions, events and constraints; they define Timelines as the set of constraints whose scope is the variables and events for a single state variable. For them, the focus was on how to build models and plans with chronicles; here, we are focused on the Timeline.

These definitions leave considerable room for defining Timelines. For instance, a Timeline can have mixes of finite and infinite domains; a Timeline need not enforce either mutual exclusion or numerical constraints, or could enforce both; a Timeline could permit hybrid constraints on variables and events. This definition also fails to cover the set of services on Timelines. Finally, the definition does not distinguish between temporal assertions or constraints arising from 'facts' versus 'goals'. (Neither, we note, do the comparable definitions of (Ghallab, Nau, and Traverso 2004).) The remainder of the paper explores these questions and the consequences of allowing (or disallowing) Timelines to have these features.

Defining Timelines by their Services

Metric time, resources, numbers, and constraints are part of many planning problems. Time dependent facts (modeled as timed initial literals in PDDL) are common, as are temporally qualified goals. Complex resource profiles, extensive use of numerical Timelines (e.g. for spacecraft attitude or robot pose), and complex constraints over such quantities are common. The simple progression or regression algorithms used by many planners, even recent ones applied to problems with complex constraints, do not need a sophisticated set of services supported by a Timeline representation. However, over time, increasingly sophisticated planning algorithms have introduced more sophisticated services that require more and more behind-the-scenes support. To explain this, consider a brief (and woefully incomplete) history of planning algorithms:

- State-space planning (whether progression or regression). These planners only maintain an internal state consisting of the list of true propositions after the set of steps in the prefix (progression) or suffix (regression). No complete history of states are maintained. A useful metaphor for progression planning is that the planner maintains a representation of the 'gap' between commitments made by the planner (the current state) and the goals. This metaphor is 'inverted' for regression planning; the gap now is between the justifications constructed to reach the goals and the facts.
- Plan-space planning. The internal state now consists of the partial order of actions and (possibly) the part of the state needed to support actions (i.e. the causal links). Again, no complete history of states over time is maintained. The metaphor used above must be extended; there is a 'ragged horizon' on one side of the gap, representing the set of unordered actions at the end of the plan and their effects, and on the other side of the gap are the goals.

  In both of these cases, the planning algorithms are limited to adding actions to the end of the plan (state space progression) beginning of the plan (state space regression) or in limited intermediate cases between actions (POCL).
- Graphplan (Blum and Furst 1995). Graphplan maintains a data structure closer to a Timeline, but the original Graphplan is limited to total ordering of sets of noninterfering actions, from which sets of actions that are not mutually exclusive must be extracted.
- Temporal Graphplan (TGP) (Smith and Weld 1999). This is closer yet to a Timeline, in that orderings are constrained only by explicit temporal constraints among actions, but still does not represent a complete history of values of the state variables.

  The internal structures maintained by Graphplan and TGP provide more flexibility than the state representation maintained by state- or plan-space planning. They explicitly represent potential violations of constraints (mutexes) that must be avoided in the final plan, and thus both can support algorithms that make commitments to arbitrary actions in the plan, and possibly also constraint violations [3]. However, the constraints are represented directly on the actions, and no explicit representation of the values of the state variables are maintained.

Many planning problems may be posed in such a way that they have no feasible solution. While it is tempting to conclude that problems with no solution should be transformed into optimization problems and appropriate algorithms used, the customers of planning often prefer not to do so. Either it is too difficult to settle on a suitable optimization criteria, or they prefer to waive violations that cannot be resolved. This is because many of the constraints may be best guesses, include considerable slack, and are difficult to refine prior to planning (Clement et al. 2012). Many applications employ a mixed-initiative planning approach, in which much of the responsibility for action insertion or plan manipulation is performed by a human operator. Actions taken by the operator include invoking reasoning components or services that modify the plan, compute the implications of a change of the plan, flag problems, and make suggestions for resolving the problem, adding or removing constraints, and waiving violations (e.g. (Aghevli et al. 2006)).

Some planners (e.g. EUROPA (Frank and Jónsson 2003), ASPEN (Fukunaga et al. 1997), IxTeT (Ghallab and Laurelle 1994), APSI (Fratini, Pecora, and Cesta 2008) and SPIFe (Aghevli et al. 2006)) employ the most general version of these services. This suggests that the notion of Timelines may have more to do with how to offer a specific set of services to a planner that implements a specific algorithm or uses a specific set of heuristics.

[3] To our knowledge no Graphplan based planner has been built that does this.

Timeline Design Features

It is important to recognize that the Timeline is the foundation of a set of services that is provided to a planner. This means it must be distinguished from the other parts of the planner that provide other services. In order to do that, we provide a simple 'reference design' of a planner, similar to EUROPA [4].

- The Rules Engine is responsible for determining the consequences of adding an action to a plan (states, resources, constraints).
- The Constraint Engine is responsible for collecting heterogeneous constraints and propagating them.
- The Temporal Engine is responsible for collecting many temporal constraints and propagating them. Temporal constraints are distinguished due to their relative ubiquity and the powerful algorithms available for propagating temporal constraints.
- The Search Engine is responsible for identifying options, such as commitment to actions or assigning variables, and committing to one of them.
- The Heuristics engine is responsible for evaluating options and ranking them or choosing one for the Search Engine.
- The Timeline is responsible for representing the chronology of values of a state variable, along with the temporal constraints that arise from the current chronology.

Mixed initiative planners arguably drive much of the motivation for Timeline-based planners. However, we omit the UI from this list of planner components. While this is largely in the interests of brevity, the Timeline provides no direct services to the UI of mixed-initiative planners, i.e. it is logically isolated from the UI by other components.

In the rest of this section we describe the features of the Timeline that underly the design of the Timeline services. For the purposes of this paper there are two distinct types of Timelines, corresponding to two different types of State Variables:

- Mutex: Timelines enforce mutual exclusion of temporal assertions (or states) on a state variable (Definition 5); that is, a consistent Timeline can be in one, and only one, state at any time.
- Resource: Timelines track the value of a *single* number, e.g. the amount of available resource may take on a single value at any instant in time. Thus, resource state variables are functions $x : t \rightarrow v$. The resource bounds are associated profiles of minimum and maximum values. A consistent Timeline has the property that the value is within bounds at all times. The available resource is defined by the transactions on the resource (Definition 6).

[4]This is no coincidence, since EUROPA is the planner we are most familiar with. It is not the intention of this paper to stipulate the specific architecture of automated planners; it is no accident, though, that the Timeline constrains the architecture of planners in a specific way, and we expect many planners that use Timelines to employ a similar architecture.

The number of ground states of even a Mutex Timeline can be infinite. Resource upper and lower bounds may be arbitrary functions. Practically speaking, the most common cases are either piecewise constant (Muscettola 2002) or piecewise linear bounds (Frank and Morris 2007), or are not declarative and computed by external code, as was done for power by MAPGEN (Bresina et al. 2005). We will consider only piecewise constant resources for the remainder of the paper.

Constraining Timelines (1): Value Constraints

In this section we describe how the value of a Timeline can be constrained by the planner (and by extension in the initial state of the planning problem).

- Assert: A planner may add a temporal assertion to a Timeline but not constrain its parameters.
- Value: A planner may constrain a temporal assertion on a Timeline to a single value.
  - All non-temporal variables in the temporal assertion may be assigned a value, resulting in temporal assertion $\{x, v^i = d^i ... v^j = d^j, s, e\}$ where $d^i \in d(v^i)$.
  - Resource transactions may be assigned a single numerical value, resulting in transaction $\{x, v = d, s\}$ where $d \in d(v)$.
- Simple Constraint: A planner may constrain a temporal assertion. A simple constraint is a hybrid constraint on a single temporal assertion that identifies a subset $R \subset d(v^i) \times ... d(v^j) \times d(s) \times d(e)$ of valid assignments to the variables. For resources this devolves to $R \subset d(v)$.
- Complex Constraint: A planner may exclude a specified subset of values on a Timeline over a specified interval, but not constrain the Timeline to take on a single value over this interval. Complex constraints include:
  - A Hybrid Constraint on the variables of multiple temporal assertions or transactions.
  - A subset of $S \subset d(v^i) \times ... d(v^j)$ that constrains *every* temporal assertion that necessarily overlaps the half-open interval $[s, e)$.
  - A tighter Resource constraint L, U constraints the available resource over the interval, but the resource may still vary over this interval.

Simple constraints can degenerate to temporal or parameter constraints; complex constraints cannot. The relation of simple constraints must enforce the assignment of the same values of the variables $v^1 ... v^i$ to all times in the half-open interval $[s, e)$, while the relation of the complex constraints need not.

Examples of these values are shown in Figure 1.

Constraining Timelines (2): Temporal Constraints

In this section we describe the temporal constraints that can be imposed on temporal assertions on the Timeline. For Mutex Timelines the options are:

- Fixed: all start and end times of temporal assertions are fixed, and all intervals are totally ordered.

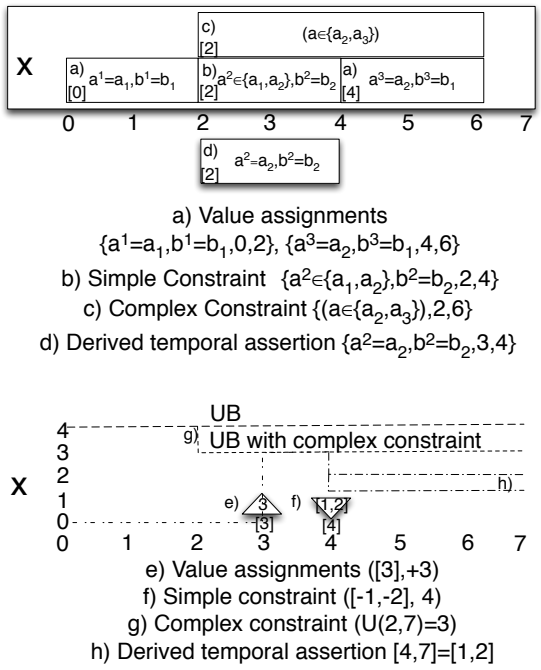


Figure 1: Types of values and value constraints on Timelines. The top of the figure shows a Mutex Timeline, with a) two value assignments, b) a simple constraint, c) a complex constraint, and d) one of the derived temporal assertions. The bottom shows a Resource, with e) one value assignments, f) one simple constraint, g) a complex constraint on the upper bound, and h) the derived temporal assertion (resource usage). (The resource transaction notation in this figure originated with (Laborie 2001).

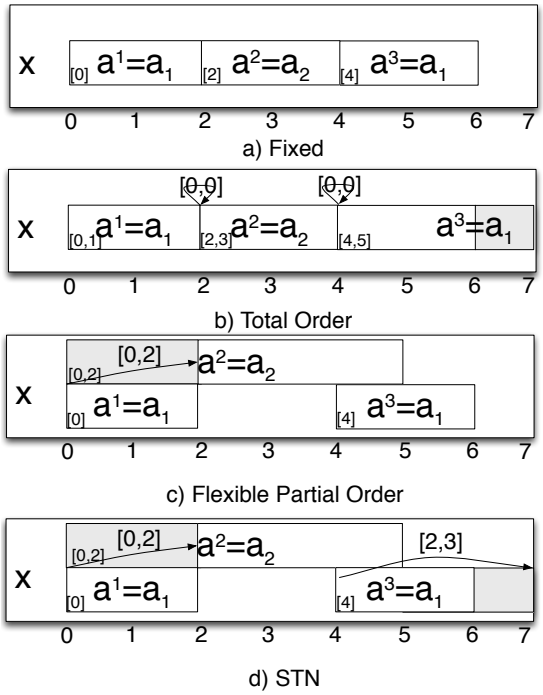


Figure 2: Types of temporal commitments allowed on Mutex Timelines. a) Fixed temporal commitments: every temporal assertion's start and end time are fixed. b) Total order: All temporal assertions are totally ordered but start and end times may be flexible. c) Flexible partial order: temporal assertion durations are fixed but may overlap. d) STN: Arbitrary simple temporal constraints are allowed between events on the temporal assertions.

- Total Order: flexible start and end times of temporal assertions are allowed (with the implication that interval durations are flexible), but all intervals on the Timeline are totally ordered.
- Partial Order: all start and end times of temporal assertions are fixed, but intervals may overlap (be partially ordered) on the Timeline.
- Flexible Partial Order: durations of temporal assertions are fixed, but arbitrary temporal constraints between events are otherwise supported.
- STN: arbitrary temporal constraints may be imposed on the events.
- DTN: disjunctive temporal networks.

For Resources:

- Fixed: all event times are fixed (but events may be simultaneous).
- Total Order: all event times are totally ordered but event times may be flexible.
- STN: arbitrary temporal constraints may be imposed on the events.
- DTN: disjunctive temporal networks.

Note that these restrictions do not apply to constraints that are maintained in the constraint network. Examples of these constraints are shown in Figure 2.

States of Time

In this section we describe what values and other properties may hold at an instant of time for a Timeline. This description will help further define the Timeline services. As distinguished from the previous section on the variable semantics, this section describes the 'state' of consistency and completeness that a Timeline can be in.

At an instant in time t, a Timeline's state or value can:

- Be unknown (i.e. no temporal assertion holds at t).
- Be possibly unknown, i.e. there are one or more temporal assertions or transactions whose temporal extent is flexible. For example, in the case of a single temporal assertion $\{x, v^i = d^i ... v^j = d^j, s, e\}$ the value of x is known for $s_{lb} \leq t \leq e_{ub}$ and unknown for $t \leq s_{ub}$ or $t_{lb} \leq t$.

- Be constrained to one of a subset of the values of the state variable or range of values of the resource.
- Take on a single value (values of all variables or value of resource is known).
- Be inconsistent due to incompatible assignments to single values and / or constraints.
  - A Mutex Timeline inconsistency consists of a set of constraints on v^i excluding all elements of $d(v^i)$.
  - A Resource Timeline inconsistency can be due to exceeding upper resource bound or lower resource bound, or to a set of constraints excluding all possible values of a transaction, either explicitly or implicitly.
  - A set of temporal constraints on events can exhibit an inconsistency.
- Be potentially inconsistent due to a set of constraints or assignments.
  - A Mutex Timeline flaw of this sort consists of e.g. a pair of value assertions in the same interval that have not been totally ordered.
  - A Resource Timeline flaw of this sort consists of a pair of inconsistent value assertions on the same transaction that have not been resolved or a set of resource consumption and production actions that are unordered and could lead to violation of the resource lower or upper bound.

Inconsistencies and potential inconsistencies merit some more discussion. Constraint propagation or restrictions on Timeline services (i.e. disallowing insertion of constraints causing inconsistency) would often eliminate these cases. However, since it is possible that planners will allow inconsistencies to be either waived or permitted, Timelines may be forced to maintain these possible inconsistent cases since no propagation engine may be present to eliminate them. Also note parameter and hybrid constraints between Timelines leading to constraint violations may not be considered, depending on the specific implementation of constraint propagation with the broader Constraint Network; this point is discussed more below. Finally, there is the question of what derived temporal assertions should be in the presence of inconsistencies. When resource bounds are exceeded this is less of a problem, but what about mutually inconsistent temporal assertions on a Mutex Timeline? One solution is to generate a temporal assertion with a symbol indicating no value exists for the relevant variables, but other solutions are possible.

We choose this time to discuss facts vs goals for Timelines. Facts may be further divided into facts asserted in the initial state, and commitments made during planning. (It is unfortunate that goals are also specified in the initial state.) In some sense, an inconsistency arising due to factual conflicts is a more profound problem than one arising because of facts and goals. For example, violating a goal of 50% battery state of charge is less problematic than violating the 100% battery state of charge limit (which may lead to explosions). Similarly, a plan in which the camera is commanded on and off at the same time is a more problematic one than a plan in which there are no constraint violations, but the goal of sending a camera image to Earth has not been achieved. In order to determine what manner of inconsistency arises, we must know whether a temporal assertion, transaction or constraint arises because of a fact, commitment, or goal. There is no specific formalism associated with the source of the primitives in our exposition. It is also an interesting design question whether this information should be stored in Timelines, or whether it could be stored elsewhere, e.g. in the Rules Engine.

Description of Services

The basic services for Timelines are as follows:

- Add or Remove temporal assertions or transactions
- Add or Remove {simple, complex, temporal} constraint
- Propagate constraints
- Get inconsistencies
- Get (derived) temporal assertions

These basic services can be extended by composing the above primitive services into more sophisticated services. Some examples:

- Is time consistent (add temporal assertion at time t; get inconsistency; if there are none return true; else return false)
- Get consistent start times (for all times t if is time consistent is true add t to list; return list)
- Move temporal assertion (remove assertion, add assertion at new time t)
- Add at legal start time (get consistent start times, select one, add temporal assertion)
- Fix violations

Extension of Services to Multiple Timelines

Some application domains include multiple similar or identical subsystems; a common example is multiple (almost) identical cameras on a single spacecraft. Satisfying a goal of taking an image may be done by choosing one camera. If each camera is modeled with a different Timeline, this can be accomplished by using two state variables $\{x, v^1...v^i\}$ $\{y, w^i...w^j\}$ with the same variable types $(d(v^i) = d(w^i))$, and a 'generic' temporal assertion $\{w, v^1...v^i, s, e\}$ which could be inserted on either state variables; if w is treated as yet another variable, then $d(w) = x, y$.

In order to naturally represent choosing which Timeline a temporal assertion or transaction is assigned to, we may rewrite Definition 1 and redefine state variables as follows:

Definition 8 *Let V be the set of variables. Let $U \subset V$ be a set of finite domain variables. Let t be a variable representing time. A state variable is a function $u_1 \times ...u_j : t \rightarrow v^i \times ...v^j$.*

This makes explicit the choice a planner can make for satisfying the goal, and in fact is more consistent with the definition of state variables in (Ghallab, Nau, and Traverso 2004). The finite domain variables U represent objects such

as rovers, cameras, locations, and other primitives that define the state variables. This perspective is very useful when considering the integration of planning with classical machine scheduling, where tasks could be assigned to multiple machines. Disjunctive temporal constraints (Tsamardinos and Pollack 2003) over these variables can now be added and propagated to let planners reason about the options. It also provides a natural way to structure the set of state variables at modeling time. If this feature is not required for the domain variables can either be bound to single values in the initial problem description, or no explicit variables used at all.

The services described above also extend naturally to services that provide a 'global' view of the plan across multiple Timelines. The generic notion of the chronicle introduced in (Ghallab, Nau, and Traverso 2004) can be made more fine-grained by identifying subsets of Timelines (perhaps based on the generic variables defining Timelines used in the above definition) for different purposes.

TImeline Design Criteria

Should there be a single Timeline implementation? On the face of it, differentiating between Mutex and Resource Timelines appears to be a good idea, especially since we have limited Resource Timelines to a single parameter to be tracked. Whether refinement should proceed further based on the value constraints coupled with the temporal constraints that may be inserted onto the Timeline is a more complex question.

How many Timeline specializations are there? There are two types of Timeline; Mutex and Resource. For each of these, we have

- 3 choices for value constraints (values only, simple, complex.)
- 6 choices for temporal constraints for Mutex Timelines, 4 for Resource Timelines.

According to this view, there are eighteen possible specializations for Mutex Timelines and twelve for Resources.

Consider first the case of Mutex Timelines. If we consider only the simplest Mutex Timeline implementation, that is, that only fully grounded temporal assertions may be inserted, then it is obvious that a Timeline implementation can be very efficiently implemented, with most services requiring at most $O(n)$ space or time. Essentially, all variables and events must be bound, and if arbitrary constraints on these variables are to be posted and / or propagated at all, then this must be handled by the Constraint Network. However, for many applications, such a limited Timeline may be sufficient. By contrast, assume that arbitrary temporal constraints are maintained in the Timeline but that all other variables must be fully grounded. Now propagation of constraints costs as much as $O(n^2)$ space and time (Dechter, Meiri, and Pearl 1991). If we now consider complex value constraints but grounded event times, we see that this can lead to $O(n^2)$ time to determine the set of derived temporal assertions and that n complex value constraints can lead to $O(n^2)$ derived temporal assertions. (This type of constraint was implemented for EUROPA (Frank and Jónsson 2003) but later discarded due to its complexity and lack of common use cases; it is provided here as an example of what is possible.) The combination of these could be quite cumbersome, both from a performance point of view as well as a complexity of implementation point of view.

Similarly, consider Resource Timelines. Again, the extremes demonstrate the potential value of specialized implementations. Fully grounded assertions permit efficient calculation ($O(n)$ time and space) for all services. Relaxation to arbitrary temporal constraints means that propagation for piecewise constant resources now requires $O(n^2)$ time (Laborie 2001) or $O(n^3)$ (Muscettola 2002). Interestingly, relaxation to simple value constraints may not increase the complexity much, since an upper bound and lower bound on resource consumption can be produced by assuming each resource impact is maximal to generate one and minimal to generate the other. Similarly, changing the resource bounds may only incur $O(n)$ penalty.

There is a complimentary way of thinking about restricting the types of constraints that may be added to a Timeline; restricting the type of constraint propagation permitted over the constraints on the Timeline. In some cases, restricting the class of constraints that can be added directly corresponds to a limited form of constraint propagation. In other cases, notably for resource propagation, the situation is more nuanced. Consistency enforcement can also be performed only on those constraints on a Timeline, ignoring parameter constraints, constraints on other Timelines, and constraints that span Timelines. Certainly the many types of consistency that can be enforced greatly increase the design space of algorithms.

Finally, Timelines could be specialized by providing a subset of the services described above. However, since part of the benefit of Timelines is to provide services that many planners can use, this may be a poor decision to make. Rather, all of the basic Timeline services should all be provided so that they can be composed as required for a planner.

Conclusions and Future Work

A Timeline is a component that maintains the history of values of a well-defined subset of a particular planning problem (a single state variable). Formally, the state variable is defined by a set of variables. The timeline computes derived temporal assertions, which provides a concise description of the interval of time over which the state of the Timeline is known, and the degree to which it is known. The elements that provide this knowledge are temporal assertions or transactions and constraints. Specialized Timelines are defined by limitations on the form of the temporal assertion and the types of constraints that can be added to the Timeline. These limitations may lead to efficient implementation of basic services that Timelines provide to other planner components.

There are some intricacies involved in fully understanding the semantics of the temporal assertions (both simple and complex) and resource constraints. A thorough formulation of the underlying dynamic constraint network is provided in (Frank and Jónsson 2003), but this treatment omits resources, and could be revisited in light of this formalization of Timelines.

Specializations for decreasing computational complexity must be described in more detail to both flesh out the options, as well as provide more insight into whether specialized Timeline implementations are valuable or necessary. In particular, this paper gives short shrift to planning heuristics as users of Timeline services.

Resources with linear production and consumption rates (Frank and Morris 2007) are omitted from this paper for simplicity. Formalization of linear resources includes intervals over which a resource is consumed or produced at a linear rate; the complexity of reasoning about linear resource state with temporal constraints is probably worse than that of piecewise constant resources with temporal constraints. A more complete treatment of them requires distinguishing between different resource Timeline types and a further investigation into the computational complexity questions surrounding Timeline services. In addition, it is often convenient to employ a special transaction to *set* the value of a resource (this is a feature of ASPEN (Fukunaga et al. 1997).) Formalizing this properly requires some extra machinery.

This view of Timelines may not extend to planning under uncertainty. Simple extensions like uncontrollable timepoints and uncontrollable resource usage may not require significant modifications to the set of definitions and services used, but supporting explicit conditional branching on the temporal assertion probably will.

The key design perspective on Timeline types taken in this paper is to map syntactic restrictions on constraints to the computational complexity of the services. However, a different perspective is to focus on the *types of disjunction* supported by different Timeline types. An ontology of Timelines of this sort would offer a different, and complementary, view of the differences between Timeline types.

References

Aghevli, A.; Bachmann, A.; Bresina, J.; Greene, J.; Kanefski, B.; Kurien, J.; McCurdy, M.; Morris, P. H.; Pyrzak, G.; Ratterman, C.; Vera, A.; and Wragg, S. 2006. Planning Applications for Three Mars Missions with Ensemble. In *Proceedings of the International Workshop on Planning and Scheduling in Space*.

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In *Proceedings of the 14^{th} International Joint Conference on Artificial Intelligence*, 1636–1642.

Bresina, J.; Jonsson, A.; Morris, P.; and Rajan, K. 2005. Activity planning for the Mars Exploration Rovers. In *Proceedings of the 15^{th} International Conference on Automated Planning and Scheduling*, 40 – 49.

Chien, S.; Johnston, M.; Frank, J.; Giuliano, M.; Kavelaars, A.; Lenzen, C.; and Policella, N. 2012. A Generalized Timeline Representation, Services, and Interface for Automating Space Mission Operations. In *Proceedings of Confernce on Space Operations*.

Clement, B.; Frank, J.; Chachere, J.; Smith, T.; and Swanson, K. 2012. The Challenge of Configuring Model Based Space Mission Planners. In *Proceedings of the Workshop on Knowledge Engineering for Planning and Scheduling*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–94.

Dechter, R. 2003. *Constraint Processing*. Morgan Kauffmann.

Edelkamp, S., and Helmert, M. 1999. Exhibiting knowledge in planning problems to minimize state encoding length. In *Proceedings of the 5^{th} European Conference on Planning*, 135–147.

Fox, M., and Long, D. 2003. PDDL 2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61 – 124.

Frank, J., and Jónsson, A. 2003. Constraint-Based Attribute and Interval planning. *Journal of Constraints Special Issue on Constraints and Planning*.

Frank, J., and Morris, P. 2007. Bounding the resource availability of activities with linear resource impact. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

Fratini, S.; Pecora, F.; and Cesta, A. 2008. Unifying planning and scheduling as timelines in a component-based perspective. *Archives of Control Sciences* 18(2):231–271.

Fukunaga, A.; Rabideau, G.; Chien, S.; and Yan, D. 1997. Toward an application framework for automated planning and scheduling. In *Proceedings of the 15^{th} International Joint Conference on Artificial Intelligence*.

Ghallab, M., and Laurelle, H. 1994. Representation and Control in IxTeT, a Temporal Planner. In *Proceedings of the 4th International Conference on AI Planning and Scheduling*, 61–677.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kauffman.

Jonsson, P., and Backstrom, C. 1998. State variable planning under structural assumptions: Algorithms and complexity. *Artificial Intelligence* 100(1-2):125–176.

Laborie, P. 2001. Algorithms for propagating resource constraints in ai planning a nd scheduling: Existing approaches and new results. In *Proceedings of the 6th European Conference on Planning*.

Muscettola, N. 2002. Computing the envelope for stepwise constant resource allocations. In *Proceedings of the 8^{th} International Conference on the Principles and Practices of Constraint Programming*.

Smith, D., and Weld, D. 1999. Temporal planning with mutual exclusion reasoning. In *Proceedings of the 16^{th} International Joint Conference on Artificial Intelligence*, 326–337.

Tsamardinos, I., and Pollack, M. 2003. Efficient so- lution techniques for disjunctive temporal reasoning problems. *Artificial Intelligence* 151(1-2):43–90.

Verfaillie, G., and Pralet, C. 2008. How to model planning and scheduling problems using timelines. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

A Service Oriented approach for the Interoperability of Space Mission Planning Systems

Simone Fratini, Nicola Policella, Alessandro Donati
European Space Agency, ESA/ESOC
Darmstadt, Germany
name.lastname@esa.int

Abstract

The ambitious long term goal of evolving planning systems in use at the European Space Agency for Mission Operations is generating a debate on how to define a set of standardized interfaces to allow rapid and efficient construction of co-operating space systems. The current scenario sees a set of software systems that perform, at various levels, planning and scheduling tasks, with little or no standardization of interfaces and services offered, with the result that to use these systems you have to know in detail the system and how to use it.

The introduction of AI technologies can simplify the problem on one side, because the use of the model-based approach moves the complexity from the software to the model, but on the other side is turning a purely software engineering problem (the interoperability of the systems) into a Knowledge Engineering problem: how to define and standardize the objects manipulated and the services offered by these model-based systems. This paper presents a proposal for a definition of classes and levels of services to be used as basis for the implementation of a Service Oriented Architecture to support interoperability between Planning and Scheduling systems in space domains.

Introduction

The Spacecraft Monitoring & Control (SM&C) Working Group of the Consultative Committee for Space Data Systems (CCSDS), which sees the active participation of 10 space agencies, has been working since 2003 on the definition of a service oriented architecture for space mission operations. The goal of the Working Group is to define a set of standardized, interoperable mission operation services, which allow rapid and efficient construction of co-operating space systems. It has been the common shared view of the community that planning problems in space domain can be very diverse. It is therefore not an easy task to define generic, reusable planning services. The size of the problem, the location of the planning logic (on the ground or on board the spacecraft or even a hybrid solution), the decision-making authority (AI decision making vs. human based conflict resolution or a so-called mixed initiative hybrid solution) and the distribution of the planning knowledge (centralized vs. decentralized planning) are factors which contribute to the diversification of space mission operation planning problems. As a result the community has recommended that standardization work shall initially focus on the boundaries of a "black-box" planning system for space missions.

Since this black box should provide services very general but at the same time detailed enough for being practically usable, the main issue to tackle is to find the right balance between 1) the level of abstraction applied to the black box and the assumptions which can be made on it and 2) the level of standardization required for specifying generic boundary planning services.

The less is known about the implementation of the black box, the more difficult to specify a planning request and to interpret the resulting feedback (and the generated plan). This is due to the fact that the information model adopted by each planning system is strongly dependent on the adopted planning technique. For instance, the information model of a constraint-based scheduler compared to a PDDL (Fox and Long 2003) or HTN (Hierarchical Task Network) planner can be very different, requiring different sets of information. Also the overall business process[1] plays an important role in the content of the exchanged information at the boundaries of the planning system, hence in the specification of the boundary planning services. Unless rigid assumptions are made, which must then be met by all compliant systems, it is difficult to agree on an abstract information model for planning requests and resulting feedback from the planning system and the eventual plan.

The difficult task is to find the right balance between a purely syntactical standard (which would require an agreement among the different software system designers to entail the interoperability), and a more generic interface based on a semantic description of the data and processes (which conversely would entail very powerful, automatic interoperability among the systems but would require a wider agreement that seems impossible to reach at the current stage).

By focusing only on standardizing the syntax, all the relevant semantics for the planning problem must be already defined within the black box and has to be considered part of the system. The boundary services would in that case con-

[1] That is, the workflow of which the planning process is a part, e.g. which are the involved planning cycles and how is the conflict resolution process, etc.

tain only services to specify minimal syntactic information, by referring to models and processes already pre-defined within the black box. This approach relies however on agreeing on a set of common assumptions about availability and unambiguity of the relevant planning information for each possible request that can be posted to the planning system. In practice these assumptions prove often to be too optimistic so that a minimum set of semantics in the form of constraints and context parameters must be provided along with the reference to the pre-defined activity.

Conversely, by focusing on standardizing the semantics, the interfaces (or API) of the black box shall allow the specification of all the information required by the planning system for carrying out the task of planning (i.e. the model). This would however either require exact knowledge of the adopted planning technique or require agreeing on an standardized abstract planning meta-model. In order to specify generic boundary planning services, obviously the latter should be the case.

The pros and cons of the two extremes are at hand, while the boundary planning services of the first approach would be much easier to specify from an standardization point of view (much less to agree on, and only at syntactical level since the information model in question would be very small), it relies on a large number of assumptions which can impact significantly the implementation of the black boxes, hence again a source of debate at standardization level. Also the usability of the resulting generic services in real world complex planning solutions can be questionable and should be demonstrated. The second approach would require much more effort at standardization level as many agreement must be reached to come up with a generic abstract planning information model which is independent from the actually adopted planning technique. A number of initiatives and languages in AI planning (such the PDDL language for instance) are already attempting on specifying such a generic and abstract planning data model, but the experiences of initiatives in this direction are not very promising, as specialized dialects and extensions have proved to be necessary to address specific needs of certain planning techniques (e.g. expression of temporal and generic constraints).

A good middle point can be an agreement on the syntax and semantics of a *limited set of services* provided by the system to manipulate the information. From our viewpoint this can be obtained looking at some previous works in the area of advanced AI planning solutions developed both at ESA and at other space agencies. Following sections will elaborate on how that experience can contribute to identify and define such a set of services.

Services or Languages?

Main objectives of the standardization of the Planning Services are to allow rapid prototyping design and to re-use planning modules between different missions (to shorten software development time and cost, as well as the training of mission operation engineers).

In this regard, a concept that has demonstrated to be very useful is the AI model-based approach. This allows reusing of software modules across different domains (missions in our case) because of the great flexibility of the symbolic representation of goals, constraints, logic, parameters to be optimized, and so on. As the system is not designed for achieving goals in a single domain but for manipulating symbolic entities, the software deployment and test is substantially independent from the specific mission. However, it is worth remarking how a great effort might be necessary to both understand the domains and the problems, capturing all the requirements, and to create a model for these domains when proper symbolic constructs are not available for modeling.

The standardization of the model based approach would require an agreement on (at least) a common language to specify models, problems, constraints and so on. And this is quite far from the reality right now, both for practical and political reasons. From a practical point of view, there is not yet an agreement in the community on the terminology (activities, tasks, processes and procedures for instance are often used for similar concepts, as well as constraints, rules; even what planning and scheduling is, is often source of debate). Moreover current languages do not allow to define and manage the workflow of which the planning process is part. In fact in practical usage, it is not sufficient to solve the planning problem, a series of processes need to be performed and constraints to be satisfied upstream and downstream of the main solving process. Also from a political viewpoint the problem is not trivial. The commitment on a specific language, for instance, would constitute a strong bias towards a specific planning technology, something that should be avoided to have a widely accepted standard. Last, there is the issue at the infrastructure software design level of introducing concepts like Model Driven Engineering (Schmidt 2006) in very conservative environments, where most of the software has been carefully written and verified and even Object Oriented languages are often seen as unnecessary complications and a potential source of ambiguity!

Hence, to have a real chance of getting to a general agreement on how planning information is specified and manipulated (pre-requisite to discuss a standard), a good starting point would be to focus on the greatest common denominator among all the different types of information in use in mission planning systems, i.e., *time tagged data*, and design a set of services aimed at managing this information to achieve some objectives.

Among the current proposals from the AI planning community, the closest one to the problem of managing time tagged data can be probably considered the timeline-based paradigm (Frank and Jonsson 2003; Fratini, Pecora, and Cesta 2008), where the planning problem is conceived as a problem of assigning values to sequences of ordered time intervals (the timelines). This approach to planning has proved to be particularly suitable for space applications, mainly because it is very close to the way problems and constraints are naturally represented in space applications.

There are already timeline based software and platforms in use at NASA, ESA and other space agencies (Chien et al. 2012). These platforms unfortunately do not use standardized languages or interfaces, while follow a conceptually similar approach. Hence the similarities at the level of the services they provide can be analyzed to draw a possible

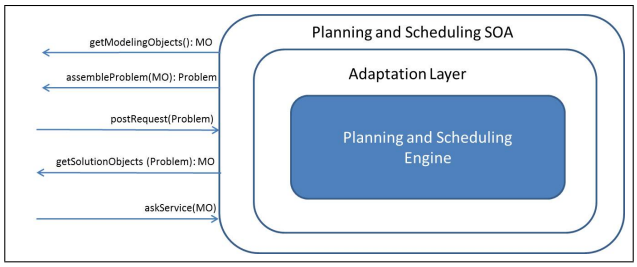


Figure 1: SOA for Mission Planning

starting point for a discussion on a standardization based on services provided instead of languages in use.

Classes of Planning Services

To design the Service Oriented Architecture (SOA) we define: (1) Classes of services, (2) Levels of service for each class and (3) Dependencies among service levels. The goal is to entail interoperability based on levels of service provided and dependencies among requested levels of service. A possible break down to classify services could be: (1) services to represent the basic entities that constitute a planning problem and its solution; (2) services to interact with the system, to post problem, control the solving process and provide feedback to the system; (3) services to use and manage problems and solutions. Figure 1 sketches the proposed architecture, constituted by a planning and scheduling engine, an adaptation layer and an API to access the services.

The API is very simple: the user asks for objects to describe the problem, posts a request to the system, retrieves objects that constitute the solution to the posted problem and asks for services for the solution objects.

In this architecture, the inner P&S engine definitely relies on its own languages for defining constraints, rules and problems, data formats for representing plans and schedules, and specific APIs to access the platform services. The classes of services identified above are mapped into internal services by means of the adaptation layer, reducing the complexity of the standardization process that is focused on external services and leave to the expert of a specific P&S technology the burden of mapping the standard services (or a sub set of the services, because in general a specific system is foreseen to be able to provide only a sub set of services) into technology specific languages and APIs. Hence the point of the standardization is not the specific syntax of a given language or a specific API for low level services, but what we need is an agreement on general and simple classes of high level services (see the conceptual diagram in Figure 2). In the following, we try to identify a possible set of such services, but the problem is still quite far away to be exhaustively analyzed

Modeling Services

The services to manage the modeling entities that constitute a planning problem and its solution should entail the capability of representing and use: basic modeling entities, problems, domain theories and solutions. Basic modeling entities constitutes the object used to assemble problems and to represent solutions. Among the basic modeling entities we classify data, punctual *events* (data defined in time instants), interval *activities* (data defined over time intervals), timelines, temporal and data constraints and objective functions. In terms of timeline representation, the classification is based on 5 parameters: type of the timeline (Symbolic or Numeric), temporal model (Fix, Bounded or Flexible[2]), data type (Ground, Parameterized, Multi-Valued, Generic Functions[3]), type of transition constraint (markovian or global) and type of specification (complete or partial). The services for defining domain theories allow the definition of classical concepts in mission planning: state variable and resource constraints, classical timeline-based synchronization or compatibility constructs among planning objects (see (Muscettola 1994; Fratini, Pecora, and Cesta 2008) for examples of formal definitions) and rules/procedures on planning objects (events, activities and timelines). Procedures are HTN decompositions of planning objects, while rules are imperative specification of if-then-else constructs and conditional activation of basic constraints.

Problems are represented as collections of basic modeling objects, domain theory objects and, possibly, objective function objects. Solutions are represented as collections of modeling entities. An interesting feature of this representation is that both problems and solutions are modeled as collection of the same entities. This allow the use of solutions of a planning process as problems for another solving process.

Problem Solving and Management Services

The services to interact with the system should generically allow the ability of managing solving and optimization processes. In practice this can be reduced to the capability of stating and propagating constraints, querying the status of the timelines, detecting and reporting conflicts in the above constraints, scheduling, generating and optimizing timelines. Regarding the definition of service levels, we classify problem solving services into: timeline extraction and querying (i.e. no domain theory is specified), scheduling, planning and scheduling (involve activity ordering and generation) and optimization. Regarding the services to manage the type of problem solving performed, we distinguish among: solving for single solution (time fix, ground timelines), solving for computing the kernel (the solution reported is a sub set of all the solutions of the problem) and solving for the envelope (compute boundaries of the solution set).

A further set of services allows to control the boundaries and authorities of the solving process when different systems are interacting with each others. In mission planning, different users have different levels oh authority to interact with

[2]For bounded timelines only distance constraints between consecutive transition points are allowed, for flexible timelines distance constraints among any pair of transition points are allowed.

[3]Examples of values that can be taken by the different types of timelines between two transition points are: k (constant), $f(?x)$ (functions of parameters), $f(?x) \vee g(?y)$ (disjunctions), $f(t)$ (value not constant in time).

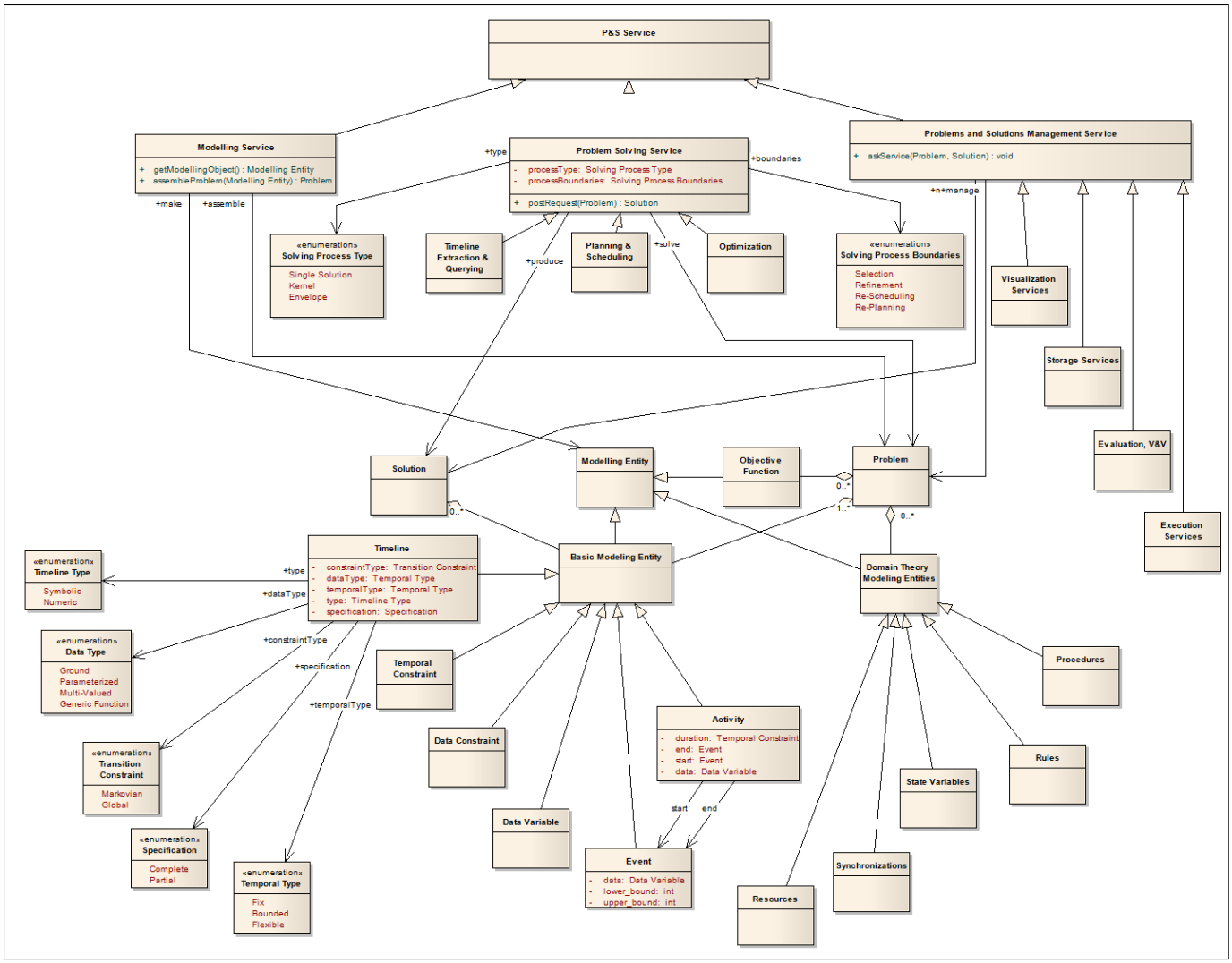


Figure 2: Conceptual Model of Services for Mission Planning

the systems and the solutions of the problems. Some users for instance can only select some solutions in the envelope or kernel (the ones that optimize some features for instance) some others can have the authority for relaxing constraints, disrupt and recalculate the solution. For this reason we classify the solving process boundaries into: selection (can interact with the solution only in within the pre-calculated envelope), refinement (can add constraints and restrict the solution set), re-scheduling (can change the activity schedule but cannot insert or delete activities) and re-planning.

Finally, problems and solutions management services includes validation and verification of properties on problems, solutions and domain theories; evaluation of solutions (quality, robustness, flexibility and so on); execution of timelines (monitoring the process); storing and visualization of problems and solutions (to entail a mixed-initiative approach).

Use Cases

The different classes and levels of service sketched aim at entailing system interoperability by means of a definition of the requirements for modeling and problem solving that are independent from the actual engine that will process the input and provide the solution. Let us think to three practical problems very common in space activities: (1) scheduling a set of tasks against a given resource to check the feasibility of a pre-calculated schedule against a decreased level of available resource, (2) planning to define a spacecraft dump plan and (3) managing P&S on-board to entail autonomy for remote operations. The three problems induce increasing levels of solving capabilities, as well as different needs both at modeling level and in terms of the solution expected.

An RCPSP-MAX Scheduling problem would for instance require: at least activities and quantitative temporal constraints as modeling objects (simple precedence constraints would not be enough); a set of time tagged data as solu-

tion objects would be enough to check the feasibility of the schedule; would require scheduling capabilities (while planning is not needed). Hence the user, facing the service oriented architecture, would look for a service provider able to manage these entities, with no need of knowing which engine will actually solve the problem.

Planning for dumping with ground station visibilities would require (at minimum): events, activities and quantitative temporal constraints as modeling objects, timelines as solution objects, planning capabilities with state variable and renewable resources (to manage memory allocations). Planning for on board autonomy would require, besides planning and scheduling, at least re-planning and rescheduling capabilities as well as dynamic insertion of activities into the timelines (which in turn requires time-flexible timelines) to entail timeline execution services.

Conclusions

This paper presents preliminary results of an ongoing activity aimed at defining a set of standardized services to allow rapid and efficient construction of co-operating planning and scheduling systems at the European Space Agency. The introduction of AI technologies can simplify the problem on one side, because the use of the model-based approach, but on the other side is turning a purely software engineering problem (the interoperability of the systems) into a Knowledge Engineering problem, i.e., how to identify a set of standard objects manipulated and services offered by these model-based planning systems.

The classes of services sketched in this paper, although far from being exhaustive of all the current and future needs of mission planning systems, aim at covering the whole life-cycle of the planning and scheduling information: from modeling to post-solving processes. The assumption behind this attempt is that the know how in knowledge engineering in AI P&S can be of a great help in the process of abstracting services for fielded mission planning systems.

This attempt is certainly meant as a challenge for future mission operation frameworks design. Nevertheless the challenge is not impossible, since most of the concepts and services presented here have already been implemented, tested, and successfully used, and the state of the art of AI planning demonstrates that is feasible to take up this challenge.

References

Chien, S.; Johnston, M., a. F. J.; Giuliano, M.; Kavelaars, A.; Lenzen, C.; and Policella, N. 2012. A Generalized Timeline Representation, Services, and Interface for Automating Space Mission Operations. In *Proceedings of SpaceOps 2012*.

Fox, M., and Long, D. 2003. PDDL 2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124.

Frank, J., and Jonsson, A. 2003. Constraint based attribute and interval planning. *Journal of Constraints* 8(4):339–364.

Fratini, S.; Pecora, F.; and Cesta, A. 2008. Unifying Planning and Scheduling as Timelines in a Component-Based Perspective. *Archives of Control Sciences* 18(2):231–271.

Muscettola, N. 1994. HSTS: Integrating Planning and Scheduling. In Zweben, M. and Fox, M.S., ed., *Intelligent Scheduling*. Morgan Kauffmann.

Schmidt, D. C. 2006. Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-* 39(2):25.

Policies for Maintaining the Plan Library in a Case-based Planner

Alfonso E. Gerevini[†] and **Anna Roubíčková**[‡] and **Alessandro Saetti**[†] and **Ivan Serina**[†]

[†]Dept. of Information Engineering, University of Brescia, Brescia, Italy
[‡]Faculty of Computer Science, Free University of Bozen-Bolzano, Bolzano, Italy
{gerevini, saetti, serina}@ing.unibs.it, anna.roubickova@stud-inf.unibz.it

Introduction

It is well known that AI planning is a computationally very hard problem (Bylander 1991; Ghallab, Nau, and Traverso 2004). In order to address it, over the last two decades several syntactical and structural restrictions that guarantee better computational properties have been identified (Bäckström et al. 2012; Bäckström and Nebel 1996), and various algorithms and heuristics have been developed (e.g. (Richter and Westphal 2010)). Another complementary approach, that usually gives better computational performance, attempts to build planning systems that can exploit additional knowledge not provided in the classical planning domain model. This knowledge is encoded as, e.g., domain-dependent heuristics, hierarchical task networks and temporal logic formulae controlling the search, or it can be automatically derived from the experiences of the planning system in different forms.

Case-based planning (Gerevini, Saetti, and Serina 2012; Liberatore 2005; Spalazzi 2001) follows this second approach and concerns techniques that improve the overall performance of the planning system by reusing its previous experiences (or "cases"), provided that the system frequently encounters problems similar to those already solved and that similar problems have similar solutions. If these assumptions are fulfilled, a well-designed case-based planner gradually creates a plan library that allows more problems to be solved (or higher quality solutions to be generated) compared to using a classical domain-independent planner.

The plan library is a central component of a case-based planning system, which needs a policy to maintain the quality of the library as high as possible in order to be efficient. Even though this problem has been studied in the context of case-based reasoning (Aamodt and Plaza 1994; Smyth 1998; Smyth and McKenna 1999), comparable work in the planning context is still missing. In this paper, we formalize the problem of maintaining the plan library, and propose different policies for maintaining it. Such policies are experimentally evaluated in (Gerevini, Saetti, and Serina 2013).

Case Base Maintenance

Case-based planning is a type of case-based reasoning, exploiting the use of different forms of planning experiences concerning problems previously solved and organized in *cases* forming a case base or *plan library*. A library needs to represent as many various experiences the system has made as possible, while remaining of manageable size. The interplay between these two parameters has a significant impact on the observed performance of the case-based planner, because a too large case base requires a vast amount of time to be queried, whereas even well designed retrieval algorithm fails to provide a suitable case to the reuse procedure if such a case is not present in the case base.

There are two different kinds of maintenance policies — an *additive policy*, which considers inserting a new case into the case base when a new solution is found/provided, and a *removal policy*, which identifies cases that can be removed without decreasing the quality of the case base too much. Hence, we formalize the general maintenance problem as a two-decision problem.

Definition 1 (Case Base Maintenance Problem)**.**

- *Given a case base* $\mathcal{L} = \{c_i \mid c_i = \langle \Pi_i, \pi_i \rangle, i \in \{1, \ldots, n\}\}$*, decide for each* $i \in \{1, \ldots, n\}$ *whether the case* c_i *should be* removed *from* $\mathcal{L}$*.*
- *Given a new case* $c = \langle \Pi, \pi \rangle$*,* $c \notin \mathcal{L}$ *, decide whether* c *should be* added *to* $\mathcal{L}$*.*

In our work, we focus on the removal maintenance — in the absence of a policy to decide which elements to add, we can simply add every new case until the case base reaches a critical size, and then employ the removal maintenance policy to obtain a small case base of good coverage.

We start by considering when a case can address another problem, or rather *how well* it can do so. We interpret the notion of "addressability" using a distance between the solutions of the stored and new problem. Let $\wp$ denote the space of plans for a given planning domain. Then, a distance function $d_a : (\wp \times \wp) \rightarrow [0, 1]$ measures the distance of any two plans $\pi_i, \pi_j \in \wp$, where the greater distance indicates greater effort needed during the *adaptation phase*. We approximate $d_a(\pi_i, \pi_j)$ by the number of actions that are in π_i and not in π_j plus the number of actions that are in π_j and not in π_i, normalized over the total number of actions in π_i and π_j (Srivastava et al. 2007), i.e., $d_a(\pi_i, \pi_j) = \frac{|\pi_i - \pi_j| + |\pi_j - \pi_i|}{|\pi_i| + |\pi_j|}$.

Definition 2. *A case* $c_i = \langle \Pi_i, \pi_i \rangle$ *can be useful to solve problem* Π*, that is,* ***addresses****(*c_i, Π*), if there exists a solution plan* π *for* Π *that is close to* π_i*, that is,* $d_a(\pi_i, \pi) < \delta$ *for some finite* $\delta \in \mathbb{R}$*.*

Note that the definition of $addresses(c_i, \Pi)$ heavily relies on the distance between the solutions, and completely disregards the relation of the relative problems. However, also the structural properties of the problems play a considerable role, as the case retrieval step is based on the planning problem descriptions. Therefore, we also use a distance function d_r that is intended to reflect the similarity of the problems. Let $\mathcal{P}$ denote the space of problems in a given planning domain, $\Pi \in \mathcal{P}$ be a new problem, and $\Pi' \in \mathcal{P}$ be a problem previously solved. We define a problem distance function $d_r : (\mathcal{P} \times \mathcal{P}) \rightarrow [0, 1]$ as follows: $d_r(\Pi', \Pi) = 1 - \frac{|\mathcal{I}' \cap \mathcal{I}| + |\mathcal{G}' \cap \mathcal{G}|}{|\mathcal{I}'| + |\mathcal{G}|}$, where $\mathcal{I}$ and $\mathcal{I}'$ ($\mathcal{G}$ and $\mathcal{G}'$) are the initial states (sets of goals) of Π and Π', respectively (Serina 2010).

The smaller distance d_r between two problems is, the more similar they are; consequently, by assuming that the world is *regular* (i.e., similar problems have similar solutions), it is useful to retrieve from the case base the case for a problem that is mostly similar to the problem to solve. We can say that d_r *guides the retrieval phase* while d_a *estimates the plan adaptation effort*. The maintenance policy should consider both distances, in order not to remove important cases, but also to support the retrieval process. Therefore, we combine the two functions, obtaining distance function $d : ((\mathcal{P} \times \wp) \times (\mathcal{P} \times \wp)) \rightarrow [0, 1]$ measuring distance between cases. The combination of d_r and d_a allows us to assign different importance to the similarity of problems and their solutions, depending on the application requirements.

Maintenance Policies

If the world is regular and the problems recur sufficiently often, the system is likely to produce solutions similar to the previous ones by reusing those and, as the problems are also assumed to be similar, creating a case base with cases that are similar to each other. Therefore, it can be expected that the cases in the case base create groups of elements, similar to each other and that could be reduced to smaller groups without significant loss of information.

To design a procedure for the maintenance, we start by deciding which parameters of the case base define its quality, and so which criteria should guide the maintenance policy in determining which experiences to keep and which to discard. Obviously, an important criterion is the variety of problems the case base can address, which is also referred to as the case base *competence* (Smyth 1998), and its interplay with the size, or cardinality, of the case base. In our approach, instead of maximizing competence as an absolute property of a case base, the maintenance is guided by minimizing the amount of knowledge that is lost in the maintenance process, where removing a case from the library implies losing the corresponding knowledge, unless the same information is contained in some other case. The following notions of case covering and case base coverage are defined to capture this concept:

Definition 3. *Given a case base $\mathcal{L}$ and a case distance threshold $\delta \in \mathbb{R}$, we say that a case $c_i \in \mathcal{L}$ covers a case $c_j \in \mathcal{L}$, that is, **covers**(c_i, c_j), if $d(c_i, c_j) \leq \delta$.*
*Let $\mathcal{L}, \mathcal{L}'$ denote two case bases and let C denote the set of all cases in $\mathcal{L}$ that are covered by the cases in $\mathcal{L}'$, i.e., $C = \{c_i \in \mathcal{L} \mid \exists c'_i \in \mathcal{L}', covers(c'_i, c_i)\}$. The coverage of $\mathcal{L}'$ over $\mathcal{L}$, denoted **coverage**$(\mathcal{L}', \mathcal{L})$, is defined as $\frac{|C|}{|\mathcal{L}|}$.*

We can now formally define the outcome of a procedure addressing the plan library maintenance problem — it should be a case base $\mathcal{L}'$ that is smaller than the original case base $\mathcal{L}$, but that contains very similar experiences. Under such conditions, we say that $\mathcal{L}'$ reduces $\mathcal{L}$:

Definition 4. *Case base $\mathcal{L}'$ reduces case base $\mathcal{L}$, denoted **reduces**$(\mathcal{L}', \mathcal{L})$, if and only if $\mathcal{L}' \subseteq \mathcal{L}$ and $coverage(\mathcal{L}', \mathcal{L}) = 1$.*

In the previous definition, we may set additional requirements on $\mathcal{L}'$ to find a solution that is optimal in some ways. For example, instead of minimizing the size of $\mathcal{L}'$, we may try to maximize the quality of the coverage. The structure of the policy remains the same— it constructs $\mathcal{L}'$ by selecting the cases that satisfy a certain *condition optimizing* $\mathcal{L}'$. Such a condition corresponds to a specific criterion the maintenance policy attempts to optimize.

We developed four policies for maintaining a case base: the random policy, a policy guided by the distance between cases, and two policies guided by the coverage between the cases in the plan library.

Random Policy. This policy reduces the case base by randomly removing cases (Markovitch, Scott, and Porter 1993), which is fast and efficient (Smyth 1998). However, the coverage of the reduced case base $\mathcal{L}'$ over the original case base $\mathcal{L}$ cannot be guaranteed.

Distance-Guided Policy. The distance-guided policy identifies the cases to remove by exploiting the notion of *average minimum distance* δ_μ in the case base. Given a case $c_i \in \mathcal{L}$, the minimum distance case c_i^* of c_i is a case in $\mathcal{L}$ such that $d(c_i, c_i^*) < d(c_i, c_j),\ \forall c_j \in \mathcal{L} \setminus c_i^*$. *The distance guided policy keeps a case c_i in the case base if and only if $d(c_i, c_i^*) \geq \delta_\mu$*, where δ_μ is defined as follows: $\delta_\mu = \Sigma_{c_i \in \mathcal{L}} \frac{d(c_i, c_i^*)}{|\mathcal{L}|}$.

Coverage-Guided Policy. The distance-guided policy can preserve the knowledge in the case base better than the random maintenance does. However, it is not optimal, as some information is missed when only the minimum distance cases are considered. We generalize the approach by considering *all* the cases that may contain redundant information at once. For that we define the notion of neighborhood of a case c with respect to a certain similarity threshold δ, denoted $n_\delta(c)$. The idea of the case neighborhood is to group elements which contain redundant information and hence that can be reduced to a single case. The case neighborhood uses a value of δ in accordance with Def. 3.

Definition 5 (Case neighbourhood)**.** *Given a case base $\mathcal{L}$, a case $c \in \mathcal{L}$ and a similarity distance threshold $\delta \in \mathbb{R}$, the neighborhood of c is $n_\delta(c) = \{c_i \in \mathcal{L} \mid d(c, c_i) < \delta\}$.*

The Coverage-Guided policy is concerned with *finding a set $\mathcal{L}'$ of cases such that the union of all their neighborhoods covers all the elements of the given case base $\mathcal{L}$.*

There are many possible ways to reduce a case base in accordance with this policy, out of which some are more suitable than others. We observed that *minimizing the size* and *maximizing the quality of the coverage* of the reduced case base can significantly influence the performance of a case-based system adopting the coverage-guided policy. Considering the first criterion, the optimal result of the coverage-guided policy takes account of the number of elements in the reduced set:

Definition 6 (Cardinality Coverage-Guided Policy)**.** *Given a similarity threshold value $\delta \in \mathbb{R}$ and a case base $\mathcal{L}$, find a reduction $\mathcal{L}'$ of $\mathcal{L}$ with minimal cardinality.*

The quality of the case base coverage can intuitively be defined as the average distance from the removed cases to the closest kept case (*average coverage distance*). The quality measure to be optimized is based around a notion of *uncovered neighborhood*:

$$U_\delta(c) = \{c_j \in \mathcal{L} \,|\, c_j \in \{n_\delta(c) \cap \mathcal{L} \setminus \mathcal{L}'\} \cup \{c\}\}.$$

Then, we define the cost of a case c as a real function

$$v_\delta(c) = \left(\frac{\Sigma_{c_j \in U_\delta(c)} d(c, c_j)}{|U_\delta(c)|} + p \right).$$

The first term within the brackets indicates the average coverage distance of the uncovered neighbors; the second term, $p \in \mathbb{R}$, is a penalization value that is added in order to favorite reduced case bases with fewer elements and to assign a value different from 0 also to isolated cases in the case base. The sum of these costs for all the elements of a reduced set $\mathcal{L}'$ defines the cost $\mathcal{M}_\delta(\mathcal{L}')$ of $\mathcal{L}'$, i.e., $\mathcal{M}_\delta(\mathcal{L}') = \Sigma_{c \in \mathcal{L}'} v_\delta(c)$. The policy optimizing the quality of the case base coverage can then be defined as follows:

Definition 7 (Weighted Coverage-Guided Policy)**.** *Given a similarity threshold value $\delta \in \mathbb{R}$ and a case base $\mathcal{L}$, find a reduction $\mathcal{L}'$ of $\mathcal{L}$ that minimizes $\mathcal{M}_\delta(\mathcal{L}')$.*

Conclusions and Future Work

In this work, we have addressed the problem of maintaining a plan library (or planning case base) for case-based planning by proposing some maintenance policies of the case base. Such policies can optimize different quality criteria of the reduced case base. The random policy, that is also used in general case-based reasoning, does not optimize any criterion but is very fast to compute. We have introduced some better informed policies, the distance-guided and the coverage-guided policies, that attempt to generate reduced case bases of good quality for case-based planning.

Since computing the most informed policies can be computationally hard, in the extended version of this paper (Gerevini, Saetti, and Serina 2013), we proposed a greedy algorithm for effectively computing an approximation of them, and we showed that these approximated policies can be much more effective compared to the random policy, in term of quality of the reduced case base and CPU time required by a case-base planner using them.

There are several research directions to extend the work presented here. In future work, we intend to study in detail additional distance functions to assess the similarity between problems and solutions, to develop and compare additional policies, to investigate alternative methods for efficiently computing good policy approximations, and to conduct an extended experimental analysis with a large set of benchmarks. Moreover, current work includes determining the computational complexity of the two proposed (exact) coverage-guided policies, that we conjecture are both NP-hard.

References

Aamodt, A., and Plaza, E. 1994. Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Communications* 7(1):39–59.

Bäckström, C., and Nebel, B. 1996. Complexity results for SAS+ planning. *Computational Intelligence* 11:625–655.

Bäckström, C.; Chen, Y.; Jonsson, P.; Ordyniak, S.; and Szeider, S. 2012. The complexity of planning revisited – a parameterized analysis. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*. AAAI Press.

Bylander, T. 1991. Complexity results for planning. In *Twelfth International Joint Conference on Artificial Intelligence*.

Gerevini, A.; Saetti, A.; and Serina, I. 2012. Case-based planning for problems with real-valued fluents: Kernel functions for effective plan retrieval. In *20th European Conference on Artificial Intelligence*.

Gerevini, A.; Saetti, A.; and Serina, I. 2013. On the plan-library maintenance problem in a case-based planner. In *Twenty-first International Conference on Case-Based Reasoning*.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated planning - theory and practice*. Elsevier.

Liberatore, P. 2005. On the complexity of case-based planning. *Journal of Experimental & Theoretical Artificial Intelligence* 17(3):283–295.

Markovitch, S.; Scott, P. D.; and Porter, B. 1993. Information filtering: Selection mechanisms in learning systems. In *Tenth International Conference on Machine Learning*, 113–151. Morgan Kaufmann.

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

Serina, I. 2010. Kernel functions for case-based planning. *Artificial Intelligence* 174(16-17):1369–1406.

Smyth, B., and McKenna, E. 1999. Footprint-based retrieval. In *Third International Conference on Case-based Reasoning*, 343–357. Springer.

Smyth, B. 1998. Case-base maintenance. In *Eleventh International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*. Springer.

Spalazzi, L. 2001. A survey on case-based planning. *Artificial Intelligence Review* 16(1):3–36.

Srivastava, B.; Nguyen, T. A.; Gerevini, A.; Kambhampati, S.; Do, M. B.; and Serina, I. 2007. Domain independent approaches for finding diverse plans. In *Twentieth International Joint Conference on Artificial Intelligence*.

Post-planning Plan Optimization: Overview and Challenges

Asma Kilani and **Lukáš Chrpa**
School of Computing and Engineering
University of Huddersfield
{Asma.Kilani, l.chrpa}@hud.ac.uk

Abstract

Automated planning even in its simplest form, classical planning, is intractable (PSPACE-complete). Moreover, it has been proven that in some cases finding an optimal solution is intractable while finding any solution is tractable. With increasing involvement of automated planning in real-world applications, there is a need for techniques able to solve planning problems in little (or real) time in reasonable quality. Techniques based on greedy search often are able to obtain solutions quickly, but the quality of the solutions is usually low. Optimal planning techniques can guarantee the best solution quality, but runtime is usually high. A reasonable compromise seems to be using fast planning techniques for producing plans in a little time and then apply post-planning plan optimization techniques that can improve quality of these plans.

In this position paper, we present an overview of state-of-the-art plan optimization techniques, provide some interesting challenges and discuss our idea of a post-planning plan optimization system.

Introduction

AI Planning is the research area that studies the process of selecting and organising actions to achieve desired goals (Ghallab, Nau, and Traverso 2004). In classical sequential planning, the outcome of this process is a sequence of actions that modify the fully observable, static and deterministic environment from a given initial state to a state where all goal atoms are satisfied. In other words, AI planning engines generate plans, solutions to given planning problems, which can be passed to an autonomous agent (robot) which can execute these plans in order to achieve its desired goals.

AI Planning is an important area to study because the it directly contributes to scientific and engineering goals of AI. The scientific goal of AI planning is to provide a tool for autonomous agents which allows them to deliberately reason about action and change, so they can produce and execute plans in order to achieve their goals. The engineering goal of AI is to build autonomous intelligent machines (robots) where planning engines are embedded in such a way that the control loop of such a machine (robot) consists of sensing, planning and acting stages. In this respect, researchers have successfully applied automated planning in many applications including space exploration such as the Mars Rover (Estlin et al. 2003), manufacturing such as the software to plan sheet-metal bending operations (Gupta et al. 1998), and games such as Bridge Baron (Smith, Nau, and Throop 1998).

In general, classical planning is known to be computationally intractable (PSPACE-complete) (Bylander 1994). It has been proven that in many cases finding an optimal solution is NP-hard where finding any solution is tractable (i.e., solvable in polynomial time) (Helmert 2006). Many modern planning engines are "satisficing": that is they produce correct but not necessarily optimal solutions (the number of actions in plans are higher than necessary). This allows the planner to be more efficient which is especially useful in real time situations when any correct plan produced is better than no plan. LPG (Gerevini, Saetti, and Serina 2004) which performs greedy local search on planning graph is a good example of satisficing planner preferring to obtain solution quickly rather than in better quality. Other well known satisficing planners FF (Hoffmann and Nebel 2001) and LAMA (Richter and Westphal 2010) use enforced hill climbing or weighted A* search with an inadmissible but informed heuristic. In contrast to satisficing planning engines, optimal planning engines such as GAMER (Edelkamp and Kissmann 2008), which is based on exploring Binary Decision Diagrams, are focused on finding best plans (shortest). However, optimal planning is usually much more time consuming and, therefore, it might be inappropriate for real-time applications.

The motivation behind post-planning plan optimization techniques reflects situations in which we need to obtain a plan in a little time, for instance, when a robot is in imminent danger and must act quickly. However, when a plan is returned, there still may be some time to optimise it through post-planning analysis. We can therefore analyse solution plans and look for opportunities to shorten them. This post-processing step is very useful when compromising between the speed of the planning process and the quality of the solutions by improving the quality of the plans produced by satisficing planners. In this position paper, we present an overview of state-of-the-art plan optimization techniques, provide some interesting challenges and discuss our idea of a post-planning plan optimization system.

Preliminaries

The simplest form of planning is known as classical planning, *"Classical AI planning is concerned mainly with the generation of plans to achieve a set of pre-defined goals in situations where most relevant conditions in the outside world are known, and where the plans success is not affected by changes in the outside world."* (Yang 1997) In other words, classical planning requires complete knowledge about a deterministic, static and finite environment with restricted goals and implicit time.

The simplest representation for classical planning is a set-theoretic representation. In a set-theoretic representation, *atoms* that provide description for the environment are propositions. *States* are represented as sets of propositions. *Actions* are represented by three sets of propositions: preconditions to be met, negative and positive effects (e.g., $a = \{pre(a); eff(a); eff^+(a)\}$). An action a is *applicable* in a state s if and only if $pre(a) \subseteq s$. Application of a in s (if possible) results in a state $(s \setminus eff(a)) \cup eff^+(a)$.

The classical representation is more expressive representation which generalizes the set-theoretic representation using notation derived from first order logic. Atom are predicates. States are sets of predicates. A *planning operator* o, which can be understood as a generalized action (i.e. action is an instance of the operator), is a 4-tuple $o = (name(o), pre(o), eff(o), eff^+(o))$, where $name(o)$ consists of a unique name of the operator and a list of variable symbols (operator's arguments), and $pre(o), eff(o) and eff^+(o)$ are sets of (unground) predicates representing operator's precondition, negative and positive effects. A *planning domain* is specified by a set of predicates and a set of planning operators (alternatively propositions and actions). A (classical) planning problem is specified by a planning domain, initial state and set of goal atoms. A *plan* is a sequence of actions. A plan *solves* a planning problem if a consecutive application of actions in the plan staring in the initial state results in a state where all the goal atoms are present (the goal state). A plan solving a given planning problem is *optimal* if and only if there does not exist a plan solving the problem which is shorter (in terms of the number of actions in the action sequence). Note that besides sequential plans we can have parallel plans, sequences of sets of independent actions, or partial-order plans.

Existing Techniques for Plan Optimization

Different techniques have been proposed for post-planning plan optimization, for instance, a plan optimization technique based on Genetic Programming (Westerberg and Levine 2001), exploring state space around the plan in order to find shorter (more optimal) plans (Nakhost and Müller 2010), identifying redundant actions that can be removed from the plan (Chrpa, McCluskey, and Osborne 2012a), replacing (sub)sequences of actions by shorter ones (Estrem and Krebsbach 2012; Chrpa, McCluskey, and Osborne 2012b).

Using Genetic Programming in post-planning plan optimization might provide some promising results (Westerberg and Levine 2001), however, it is unclear whether such an approach is domain-independent (i.e. whether it is required to hand code optimization policies for each domain), and, moreover, running time of such method might be high.

Nakhost and Müller (2010) proposed two methods: Action Elimination (AE) and Plan Neighbourhood Graph Search (PNGS). AE is an algorithm which identifies and removes some redundant actions from the plan. AE derives from the concept of Greedily Justified Actions (Fink and Yang 1992) - a greedily justified action in a plan is, informally said, such an action which if it and actions dependent on it are removed from the plan, the plan becomes invalid. AE, on the other hand, tries to remove a given action and only the first action in the rest of the plan which becomes inapplicable. AE iteratively checks actions for 'relaxed greedily justification' and if removing the involved actions does not affect validity of the plan, the plan is updated (optimized); otherwise the removed actions are restored, and the plan remains unchanged. PNGS creates a Plan Neighbourhood Graph (PNG) by expanding a limited number of states (nodes) around each state along the plan, which is often a small subset of the original state space, then applies Dijsktra's algorithm to find a shorter path in the neighbourhood graph, which might lead towards better (shorter) plans. Figure 1 shows the example of constructing a neighbourhood graph. Note that states can be expanded by using breadth-first or best-first search. Although, both methods can work as standalone methods, combination of both works better because AE is able to identify and remove some redundant actions efficiently which provides shorter plans for PNGS. PGNS is extremely useful for local improvement of plan quality, however, it might not work well if some 'optimizable' actions lies far from each other in the plan although in some plan's permutation can be adjacent.

AIRS (Estrem and Krebsbach 2012) improves plans by identifying 'optimizable' subsequences of actions according to heuristic estimation, and tries to find shorter (optimal) ones by using more expensive (optimal) planning technique, and eventually replace the longer sequence by the shorter one. The heuristic estimation is used for estimating a distance between given pairs of states. If states seem to be closer than they are in the plan, then an optimal or nearly-optimal planner is used to re-plan between these states. Figure 2 depicts the example of a plan obtained after the refinement process. Three pairs of states (A, B), (C, D), and (E, F) were selected by AIRS as candidates for refinement. Grey solid lines between (A, B) and (C, D) represent the final subplans and the dashed lines represent the initial (lower quality) subplans. AIRS continues to select pairs of states in the current solution plans and refine the corresponding subplans until the given time runs out.

PNGS and AIRS are useful for local enhancement, however, they do not take into account plans structure, for instance, some actions may be far from each other in the plan, but can be close (adjacent) in some permutation of the plan. Chrpa, McCluskey, and Osborne (2012b) proposed a method which explores plan structures based on analysing action dependencies and independencies (Chrpa and Barták 2008) in order to identify redundant actions or non-optimal subplans. Firstly, the method identifies and removes all the actions

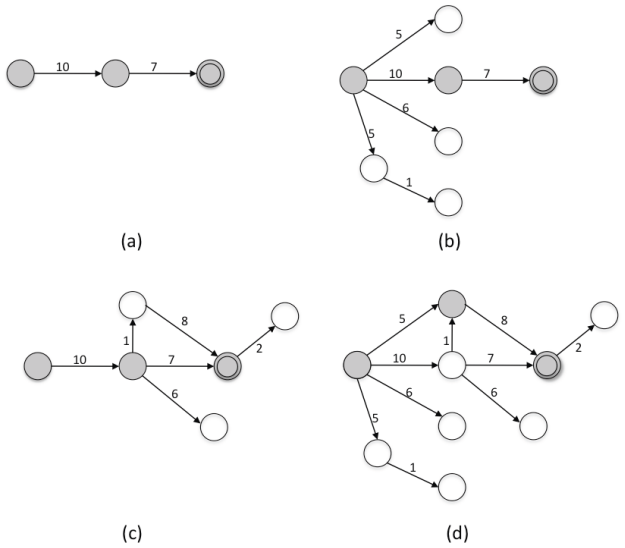


Figure 1: Taken from (Nakhost and Müller 2010). (a) The input plan. (b) and (c) Expansion of only one node does not reveal anything useful. (d) An improved plan can be found in the neighbourhood graph.

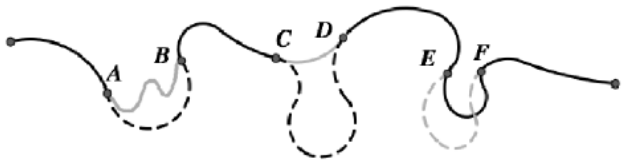


Figure 2: Taken from (Estrem and Krebsbach 2012). An intuitive example of a plan obtained after the refinement process done by AIRS is complete. Solid lines depict the final plan, with dashed lines showing either replaced subplans or situations when replanning does not produce better subplans.

on which the goal is not dependent, which basically corresponds to Backward Justified actions (Fink and Yang 1992). Secondly, all pairs of inverse actions reverting each other effects are checked for redundancy, which depends only on actions placed in between the pair of inverse actions, and eventually removed. An extension allowing to remove grouped nested pairs of inverse actions, which covers situations when we can only remove all the pairs together and not step by step, has been introduced in (Chrpa, McCluskey, and Osborne 2012a). Thirdly, the method identifies pairs of weakly adjacent actions, actions that can be adjacent in some permutation of the plan, and if possible replaces them by a single action. Weak adjacency of action is determined by using the action independence relation which allows swapping adjacent actions without affecting plans validity. An algorithm for determining weak adjacency of actions in plans has been also used for learning macro-operators (Chrpa 2010). This approach has presented some positive aspects such as being able to reasonably optimize plans in a little time. It is efficient in combination with fast satisfcing planners (e.g. LPG). Although this method is quite restricted to specific cases (e.g. inverse actions, non-optimal subplans of length two), it might be very useful as a preprocessing step to some other method (e.g. PNGS).

It is good to mention an approach which optimises parallel plans (Balyo, Barták, and Surynek 2012). This approach improves plans locally where (parallel) subplans of a predefined length k are possibly replaced by shorter ones. After all the subplans are processed, k is incremented and the optimization process is performed until a length of subplans reaches (or exceeds) a given limit.

Recently, an approach for optimal planning with inadmissible heuristics has been proposed (Karpas and Domshlak 2012). Shortcut rules, which were introduced there, are learnt applied during the planning process, concretely, they used for deriving existential optimal landmarks and for removing some redundant actions from partial plans. In general, shortcut rules can be used to replace some action sequences (totally or partially ordered) by shorter (or less expensive) action sequences without violating the correctness of plans. These rules could be obtained by several sources including learning them online, during the planning process, for example, plan rewrite rules (Nedunuri, Cook, and Smith 2011).

Challenges

Post-planning plan optimization is still a challenging area of AI planning because even determining a maximum set of redundant actions in a plan is generally intractable (Fink and Yang 1992). Clearly, guaranteeing optimal plans refined from suboptimal ones is intractable as well. Hence, there is a need for developing new sophisticated (tractable) techniques as well as for studying theoretical properties of planning domains/problems and plans in order to determine whether a given optimization technique can guarantee optimality of plans. We discuss several challenges which are from our point of view most interesting.

Optimal Planning with Macro-operators

In some cases, we enhance domains by macro-operators, operators that encapsulate sequences of (primitive) operators. It has been shown that while using macro-operators in the search we can produce plans much more quickly but we might lose the optimality of solution plans (unless we use action costs). It was discussed in (Chrpa, McCluskey, and Osborne 2012b) that in some cases we might be able to refine an optimal plan from (sub-optimal) plans with macro-operators retrieved by optimal planning engines. In the Depots domain we can generate macro-operators lift-load and unload-drop. If we have a problem where a crate has to be moved from one stack to another within the same depot, then even using an optimal planning engine may provide a solution plan ⟨lift-load,unload-drop⟩. This plan can be unfolded into ⟨lift,load,unload,drop⟩. It is obviously not an optimal plan (the optimal plan is ⟨lift,drop⟩). We may easily observe that the actions load and unload are inverse and can be identified as redundant and removed (Chrpa, McCluskey, and

Osborne 2012a). However, if there is more than one hoist in the depot these actions might not be inverse (a different hoist is used to unload the crate from the truck). In this case, we cannot simply remove the actions load and unload from the plan because the precondition of drop will not be satisfied. To fix this we have to modify an argument referring to the hoist of the drop action to correspond with the lift action. In the Zeno domain we can generate a macro-operator refuel-fly. This can cause that, even if using an optimal planning engines, solution plans (unfolded) may contain more refuel actions than necessary. Similarly with the previous case, removing the refuel action from the plan causes inapplicability of the following fly (or zoom) actions. Therefore, their arguments referring to fuel levels must be updated.

Hence, it seems to be reasonable to somehow classify potential situations (not limited to those discussed above) which may occur while planning with macro-operators using optimal planning engines (without action costs). Developing or applying existing (tractable) optimization methods for these classes which guarantee optimality is an interesting challenge.

Characteristic Domains

In the well known BlocksWorld domain, one source of non-optimality of plans is Sussman Anomaly (depicted in figure 3). For example, having three blocks A, B and C such that the initial state is on(C, A) and onTable(B) and the goal situation is on(A,B) and on(B,C). There are two possibilities in which order the goal atoms can be achieved, i.e. , on(A,B) then on(B,C), or vice-versa. We can obtain following plans: i) ⟨unstack(C,A), putdown(C), **pickup(A), stack(A,B) , unstack(A,B), putdown(A)**, pickup(B), stack(B,C) , pickup(A), stack(A,B)⟩, and ii) ⟨**pickup(B), stack(B,C), unstack(B,C), putdown(B)**, unstack(C,A), putdown(C), pickup(A), stack(A,B), unstack(A,B), putdown(A), pickup(B), stack(B,C) , pickup(A), stack(A,B)⟩. For these situations we can apply a technique for determining redundant actions (Chrpa, McCluskey, and Osborne 2012a) which can provide optimized plans which are optimal. This technique should work as discussed in literature in more complex cases where, for example, after stacking A to B we might be moving blocks between different stacks and after that we may unstack A from B. Therefore, we might ask whether such a technique can guarantee optimal plans.

In the Gold-miner domain we have to navigate through the maze in order to collect gold. In the maze, we have to unblock cells either by laser or by bomb, however, a cell containing gold can be unblocked only by the bomb. However, when unblocking the cells by bomb, the bomb is consumed and has to be re-collected. Hence, the optimal strategy is to pick the laser unblock all the necessary cells and use the bomb only for the cell containing gold. However, some planners might prefer using bomb rather than laser which leads to non-optimal plans. In this case, a large part of the plan has to be replaced by an optimal one. The best option might be to use AIRS (Estrem and Krebsbach 2012) which can use an optimal (or nearly optimal) planner to re-plan that part of the plan.

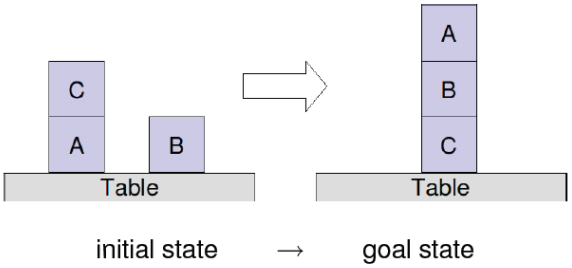


Figure 3: A simple Sussman Anomaly Problem

In the Zeno domain we might, for instance, use more refuel actions than necessary. However, as mentioned before removing even unnecessary refuel action will cause the plan to become invalid. To make the plan valid again we have to modify fuel level arguments of corresponding fly or zoom actions. Perhaps, an adaptation of the approach for determining redundant actions (Chrpa, McCluskey, and Osborne 2012b) might handle this issue.

In the Depots domain, we may observe that moving a crate from one truck to another is not optimal. Instead of two actions unload and load we can have one extra drive action. However, we cannot simply replace the unload and load actions by the drive action because in that case we might lose validity of the plan. Such a change has to be propagated, i.e., the crate will not be unloaded from the second truck and the first truck will be at a different location. For such a situation PNGS (Nakhost and Müller 2010) might be the right approach, however, it might not work well if these actions are placed far from each other (but can be in some permutation of the plan).

Learning Shortcut Rules

Shortcut rules (Karpas and Domshlak 2012) can be understood as a mapping between causal structures of planning operators (or actions) such that structures are mapped to structures with smaller cost (determined by the number of operators or a sum of action costs of the operators). A specific case of shortcut rules are plan rewrite rules (Nedunuri, Cook, and Smith 2011) which are used for replacing sub-plans (subsequences of actions in plans) by shorter (or cheaper) subsequences of actions. AIRS (Estrem and Krebsbach 2012) is based on identifying potentially non-optimal sub-plans and replacing them by optimal (or nearly optimal) ones. Work (Chrpa, McCluskey, and Osborne 2012b) identifies pairs of weakly adjacent actions (i.e. actions that can be adjacent in some permutation of the plan) which can be replaced by a single action. Replaceability in this case is defined in such a way that an action which is going to replace another action (or action sequence) must have a weaker (or equal) precondition, weaker (or equal) negative effects and stronger (or equal) positive effects. However, in some cases this might be a too strong assumption, since in some cases we might be able to satisfy stronger preconditions etc. Therefore, we might have to distinguish between problem-independent and problem- or plan-specific shortcut

rules.

Of course, shortcut rules can encapsulate more complex causal structures of planning operators. The key question is how shortcut rules can be effectively learnt. One possibility is to learn them online (when optimizing plans). However, the main shortcoming of such an approach might be its time consumption since we have to blindly check a very high number of operator causal structures. Another possibility is to identify possible non-optimal operator causal structures by investigating operator schema. It might give us an opportunity to identify ways how a certain atom or atoms can be achieved. However, there are some limitations which might prevent learning problem-specific shortcut rules. For example, in the Depots domain two drive actions can be replaced by a single one (e.g. drive(t,a,b), drive(t,b,c) can be replaced by drive(t,a,c)). However, assuming an extension of the domain where there must be a road between locations in order to allow the truck to drive between them the drive(t,a,c) action is applicable only if there is a road between the locations a and c.

Applying shortcut rules might be tractable, however, it is debatable whether the degree of the polynomial might not be too high. Replacing a pair of actions by a single one (where possible) can take at worst $\mathcal{O}(l^2)$ (l is the number of actions placed in between the pair) (Chrpa, McCluskey, and Osborne 2012b). However, for more complex shortcut rules the complexity may rise. Considering situations in which actions involved in some shortcut rules are not adjacent replacing these actions might not be straightforward since the 'new' actions may influence or be influenced by actions placed in between the 'old' actions. Therefore, we have to also study how computational complexity of applying shortcut rules rises with their size (i.e. involving more complex causal structures of operators).

Proposal for a System Architecture for a Post Planning Plan Optimization

Our high-level proposal of a system for post-planning plan optimization is shown in figure 4, this system takes the non-optimal plan, planning domain, and planning problem descriptions as inputs. The system analyses them in order to identify some redundant actions, which can be removed, then it learns a shortcut library, which consists of some non-optimal sub-plan and their optimal alternatives, then the systems applies these shortcut rules and produce a new (optimal) plan as output. To implement this system we must address these issues: How can be redundant actions or non-optimal sub-plans easily identified ? How we can easily obtain shortcut rules ? How we can easily apply shortcut rules ? Where does the system guarantee optimality of the plan ?

Concept

The idea of our system is based on an idea of composing different approaches (e.g. using AE and PGNS together (Nakhost and Müller 2010)). In our case we consider identifying and removing (some) redundant actions and learning and applying shortcut rules.

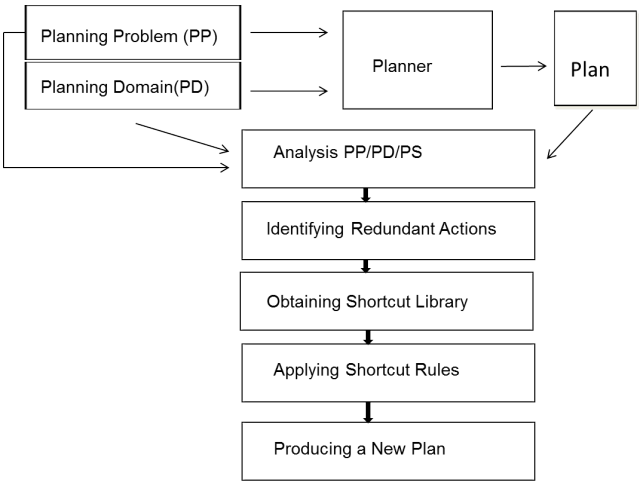


Figure 4: A Proposal for a System Architecture for a Post Planning Plan Optimization

Analysing structures of plans, planning domains and planning problems will help to obtain useful knowledge base, for instance, inverse actions, common non-optimalities in plans etc. Inverse actions which can be considered as 'good candidates' for being redundant are identified easily (Chrpa, McCluskey, and Osborne 2012a). Moreover, removing actions that are not Greedily Justified might improve the plans as well, however, such a technique require a cubical time (with respect to plans length) (Fink and Yang 1992). Identifying common non-optimalities might be done by analysing structures of planning operators, for instance, by investigating which predicates are achieved by which operator. Investigating some ideas of regression based planning (McDermott 1996) might be also very useful. We expect that such sort of analysis can be used for learning some simpler shortcut rules. Learning and applying specific shortcut rules where subplans of length two were replaced by single actions (if possible) has already been studied (Chrpa, McCluskey, and Osborne 2012b). However, worst case computational complexity seems to be high (up to$\mathcal{O}(n^4)$. Learning and applying more complex shortcut rules seems to be counterproductive given the fact that the computational complexity can be very high. From this perspective it might be more efficient to use techniques such as PNGS or AIRS rather than complex shortcut rules. Nevertheless, efficient generation and application of shortcut rules will be our future work. As we discussed before it is also necessary to study theoretical properties such as computational complexity which can somehow indicate which shortcut rules are easy or difficult to learn and apply.

Lessons learnt from existing approaches shows that it seems to be very reasonable to apply a different plan optimization techniques ordered from computationally easy ones to computationally hard ones. Hence, more expensive techniques are applied on shorter plans. This also allows to optimize plans at least slightly if a given time is very small.

Conclusions

This position paper presents an overview of the state-of-the-art approaches for post-planning plan optimization, discusses some interesting challenges in this area, and propose an idea of a plan optimization system. Clearly, there has been significant progress during the last 3 years in this area, however, post-planning plan optimization remains a challenge. Given the advantages and disadvantages of existing techniques and their key concepts we can get inspirations in adapting current techniques or developing new ones. Of course, we have to be aware of certain limitations (e.g. complexity issues) which may hinder applicability of such techniques because their runtime might be higher than when using optimal (or nearly) optimal planning engines.

In future we plan to develop a system for post-planning plan optimisation as discussed in the paper. This task will require to study and extend current techniques, for example, determining redundant actions, and develop new techniques, for example learning and applying shortcut rules. Besides this, one the very important aspects in to study theoretical properties of these techniques such as completeness and complexity. Even though the proposed system is planned to support only classical (STRIPS) planning, we plan to extend it in further future for non-classical planning (ADL, durative actions etc.).

References

Balyo, T.; Barták, R.; and Surynek, P. 2012. Shortening plans by local re-planning. In *Proceedings of ICTAI*, 1022–1028.

Bylander, T. 1994. The computational complexity of propositional strips planning. *Artificial Intelligence* 69:165–204.

Chrpa, L., and Barták, R. 2008. Towards getting domain knowledge: Plans analysis through investigation of actions dependencies. In *Proceedings of FLAIRS 2008*, 531–536.

Chrpa, L.; McCluskey, T. L.; and Osborne, H. 2012a. Determining redundant actions in sequential plans. In *Proceedings of ICTAI*, 484–491.

Chrpa, L.; McCluskey, T. L.; and Osborne, H. 2012b. Optimizing plans through analysis of action dependencies and independencies. In *Proceedings of ICAPS*. 338–342.

Chrpa, L. 2010. Generation of macro-operators via investigation of action dependencies in plans. *Knowledge Engineering Review* 25(3):281–297.

Edelkamp, S., and Kissmann, P. 2008. Gamer: Bridging planning and general game playing with symbolic search. In *Proceedings of the sixth IPC*.

Estlin, T.; Castano, R.; Anderson, R.; Gaines, D.; Fisher, F.; and Judd, M. 2003. Learning and planning for mars rover science. In *In Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*. Morgan Kaufmann Publishers.

Estrem, S. J., and Krebsbach, K. D. 2012. Airs: Anytime iterative refinement of a solution. In *Proceedings of FLAIRS*, 26–31.

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.

Gerevini, A.; Saetti, A.; and Serina, I. 2004. Planning in pddl2.2 domains with lpg-td. In *Proceedings of the fourth IPC*.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated planning, theory and practice*. Morgan Kaufmann Publishers.

Gupta, S. K.; Bourne, D. A.; Kim, K. H.; and Krishnan, S. S. 1998. Automated process planning for sheet metal bending operations. *Journal of Manufacturing Systems* 17:338–360.

Helmert, M. 2006. New complexity results for classical planning benchmarks. In *Proceedings of ICAPS 2006*, 52–62.

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

Karpas, E., and Domshlak, C. 2012. Optimal search with inadmissible heuristics. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling*.

McDermott, D. V. 1996. A heuristic estimator for means-ends analysis in planning. In *Proceedings of AIPS*, 142–149.

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

Nedunuri, S.; Cook, W. R.; and Smith, D. R. 2011. Cost-based learning for planning. In *Workshop of Planning and Learning*.

Richter, S., and Westphal, M. 2010. The lama planner: guiding cost-based anytime planning with landmarks. *Journal Artificial Intelligence Research (JAIR)* 39:127–177.

Smith, S. J. J.; Nau, D.; and Throop, T. 1998. Computer bridge: A big win for ai planning. *AI Magazine* 19:93–105.

Westerberg, C. H., and Levine, J. 2001. Optimising plans using genetic programming. In *Proceedings of ECP*, 423–428.

Yang, Q. 1997. *Intelligent Planning: A Decomposition and Abstraction Based Approach*. Springer-Berlin.

Knowledge Engineering Tools in Planning: State-of-the-art and Future Challenges

M.M.S. Shah and **L. Chrpa** and **F. Jimoh** and **D. Kitchin**

T.L. McCluskey and **S. Parkinson** and **M. Vallati**
School of Computing and Engineering
University of Huddersfield
United Kingdom

Abstract

Encoding a planning domain model is a complex task in realistic applications. It includes the analysis of planning application requirements, formulating a model that describes the domain, and testing it with suitable planning engines. In this paper we introduce a variety of new planning domains, and we then use and evaluate three separate strategies for knowledge formulation, encoding domain models from a textual, structural description of requirements using (i) the traditional method of a PDDL expert and text editor (ii) a leading planning GUI with built in UML modelling tools (iii) a hierarchical, object-based notation inspired by formal methods.

We distill lessons learned from these experiences. The results of the comparison give insights into strengths and weaknesses of the considered approaches, and point to needs in the design of future tools supporting PDDL-inspired development.

Introduction

Knowledge Engineering for automated planning is the process that deals with acquisition, formulation, validation and maintenance of planning knowledge, where a key product is the *domain model*. The field has advanced steadily in recent years, helped by a series of international competitions[1], the build up of experience from planning applications, along with well developed support environments. It is generally accepted that effective tool support is required to build domain models and bind them with planning engines into applications. There have been reviews of such knowledge engineering tools and techniques for AI Planning (Vaquero, Silva, and Beck 2011). While these surveys are illuminating, they tend not to be founded on practice-based evaluation, in part, no doubt, because of the difficulty in setting up evaluations of methods themselves. Given a new planning domain, there is little published research to inform engineers on which method and tools to use in order to effectively *engineer a planning domain model*. This is of growing importance, as domain independent planning engines are now being used in a wide range of applications, with the consequence that operational problem encodings and domain models have to be developed in a standard language such as PDDL (Ghallab et al. 1998).

[1] for the most recent see http://icaps12.poli.usp.br/icaps12/ickeps

In this paper we explore the deployment of automated planning to assist a variety of real world applications: machine tool calibration, road traffic accident management, and urban traffic control. In introducing these new planning domains, we take the opportunity to employ and hence evaluate three separate methods for knowledge formulation (i) the traditional method of hand-coding by a PDDL expert, using a text editor and relying on dynamic testing for debugging (ii) itSIMPLE (Vaquero et al. 2007), an award-winning GUI, utilising a method and tool support based on UML (iii) a rigorous method utilising a hierarchical, object-based notation OCL_h, with the help of tool support from GIPO (Simpson, Kitchin, and McCluskey 2007). Evaluating these three approaches gives a range of interesting insights into their strengths and weaknesses for encoding new domains, and point to needs in the design of future tools supporting PDDL-inspired development. Evaluation measures used are based on several criteria that describes the quality of the engineering process and the quality of the product.

This paper is organized as follows. We first provide an overview of existing KE tools for supporting the task of encoding planning domain models. Next we introduce the real-world domains that have been considered in this analysis. Then we introduce the features that are used for comparing the different encoding methods. Finally, we summarize the lessons learned and we provide some guidelines for future tools.

Overview of existing KE tools

In this section we will provide an overview of KE tools that can be used for producing planning domain models. Tools are listed in alphabetical order.

EUROPA

The Extensible Universal Remote Operations Planning Architecture (EUROPA) (Barreiro et al. 2012), is an integrated platform for AI planning & scheduling, constraint programming and optimisation. The main goal of this platform is to deal with complex real-world problem. It is able to handle two representation languages, NDDL and ANML (Smith, Frank, and Cushing 2008). The latter has been used in various missions by NASA. EUROPA provides modelling support, result visualisation and an interactive planning process.

GIPO
The Graphical Interface for Planning with Objects (GIPO) (McCluskey and Simpson 2006; Simpson, Kitchin, and McCluskey 2007) is based on its own object-centred languages *OCL* and OCL_h. These formal languages exploit the idea that the universe of potential states of objects are defined first, before operator definition (McCluskey and Kitchin 1998). GIPO is centered on the precise definition of a planning state as an amalgam of object's individual states. This gives the concept of a *world state* as one being made up of a set of states of objects, satisfying certain types of constraints. Operator schemas are constrained to be consistent with respect to the state, giving the opportunity for using tools to do consistency checking. GIPO uses a number of consistency check, like if the object's class hierarchy is consistent, object state descriptions satisfy invariants, predicate structures and operator schema are mutually consistent and task specifications are consistent with the domain model. Such consistency checking guarantees that several classes of errors are prevented, in contrast to ad hoc methods such as hand crafting.

itSIMPLE
itSIMPLE (Vaquero et al. 2007; 2012) provides an environment that enables knowledge engineers to model a planning domain using the Unified Modelling Language (UML) standard (OMG 2005). itSIMPLE focuses on the initial phases of a disciplined design cycle, facilitating the transition of requirements to formal specifications. Requirements are gathered and modeled using UML to specify, visualize, modify, construct and document domains in an object-oriented approach. A second representation is automatically generated from the UML model, and it is used to analyze dynamic aspects of the requirements such as deadlocks and invariants. Finally, a third representation in PDDL is generated in order to input the planning domain model and instance into an automated planner.

JABBAH
JABBAH (González-Ferrer, Fernández-Olivares, and Castillo 2009) is an integrated domain-dependent tool that aims to develop process transformation to be represented in a corresponding HTN planning domain model. The system mainly deals with business processes and workflows. The processes are represented as Gantt charts or by using an open source workflow engine. The tool provides support for transforming Business Process Management Notation (BPMN) (graphical notation) to HTN-PDDL. Such HTN-PDDL (Castillo et al. 2006) domain model is used in HTN planners to obtain a solution task network.

MARIO
Mashup Automation with Runtime Invocation and Orchestration (MARIO) (Bouillet et al. 2009; Feblowitz et al. 2012) is an integrated framework for composing workflow for multiple platforms, such as Web Services and Enterprise Service Bus. This tool provides a tag-based knowledge representation language for composition of planning problems and goals. It also provides a web-based GUI for AI planning system so that the user can provide software composition goals, views and generated flow with parameter to deploy them into other platform.

PDDL Studio
PDDL Studio (Plch et al. 2012) is a recent PDDL editor that allows the user to write and edit PDDL domain and problem files. The main goal of the tool is to provide knowledge engineers the functionality to edit and inspect PDDL code, regardless of how the PDDL code was created. The tool supports the user by identifying syntactic errors, highlighting PDDL components and integrating planners. PDDL Studio does not require the user to draw any diagram, it is more like writing traditional programming language code by using an Integrated Development Environment (IDE). The current version of this tool can help editing basic PDDL and also provides error checking.

VIZ
VIZ (Vodrážka and Chrpa 2010), is a knowledge engineering tool inspired by GIPO and itSIMPLE. It shares many characteristics of those systems (GIPO and itSIMPLE) with the addition of a simple, user friendly GUI by allowing inexperienced knowledge engineers to produce PDDL domain models. This tool uses an intuitive design process that makes use of transparent diagrams to produce a PDDL domain model. The tool does not support any third party planner integration. However, the tool is still being developed.

Considered applications

We considered three real-world domains that have been encoded in planning domain models. Namely, the machine tool calibration (Parkinson et al. 2012), the road traffic accident management (Shah, McCluskey, and Chrpa 2012), and the urban traffic control (Jimoh et al. 2012)

Machine tool calibration
Engineering companies working with machine tools will often be required to calibrate those machines to international standards. The requirement to manufacture more accurate parts and minimise manufacturing waste is resulting in the continuing requirement for machine tools which are more accurate. To determine a machine's accuracy, frequent calibration is required. During calibration, the machine will not be available for normal manufacturing use. Therefore, reducing the time taken to perform a calibration is fundamental to many engineering companies.

The calibration process requires various errors in the machine to be measured by a skilled expert. In addition to conducting the tests, the engineer must also plan the order in which the tests should take place, and also which instruments should be used to perform each test. It is critical to find the optimal sequence of measurements so that the machine is not out of service for too long.

An example PDDL2.2 (Edelkamp and Hoffman 2004) operator taken from the machine tool calibration domain model can be seen in Figure 1. This operator can be considered as

```
(:durative-action setup
:parameters
(?er - error ?ax - axis ?in - instrument)
:duration (= ?duration (setup-time ?in ?ax))
:condition
(and (over all (not (blocked ?in ?ax)))
    (over all (axis-error ?ax ?er))
    (over all (measures   ?in ?er))
    (over all
        (forall (?a - axis ?i - instrument)
           (imply (setup ?i ?a) (= ?a ?ax))))
    (over all (forall (?i - instrument)
           (imply (operating ?i)
            (compatible ?i ?in))))
    (at start (<= (using ?in) 0))
    (at start (>= (using ?in) 0))
    (at start (not (measured ?ax ?er)))
    (at start (not (operating ?in)))
    (over all (>= (working-range ?in)
        (travel-length ?ax)))
    (over all (working-day))
  )
 :effect
   (and
(at end   (setup      ?in ?ax))
(at end   (setup-for  ?in ?er))
(at end   (operating  ?in))
(at start (increase (using ?in) 1))
   )
)
```

Figure 1: A sample planning operator from the machine tool calibration domain model, encoded in PDDL 2.2.

one of the most complex that we have dealt with in this work. It includes quantification, timed-initial literal and ADL features of PDDL.

Figure 2 illustrates an excerpt taken from a valid, optimal calibration plan produced from using the PDDL2.2 domain and LPG-td planner (Gerevini, Saetti, and Serina 2006). From the excerpt it can be seen that the planner has scheduled measurements that use the same equipment together, and that where possible, measurements are taken concurrently. This plan allows the calibration process to be completed as quickly as possible, minimizing the down-time of the machine.

Road traffic accident management

Accidents cause traffic congestion, injury, increase environment pollution, and cost millions of pounds every year because of delay and damage. This has lead to highway agencies needing more appropriate solutions to manage accidents. Accidents are a particular type of road traffic incident which can be defined as irregular or unplanned events that reduce road capacity, increase congestion and travel time. Incidents increase traveler delay which may lead to more serious problems such as further accidents (Owens et al. 2000). The consequence of this problem is often severe since accidents limit the operation of the road networks and put all road users at risk. The main

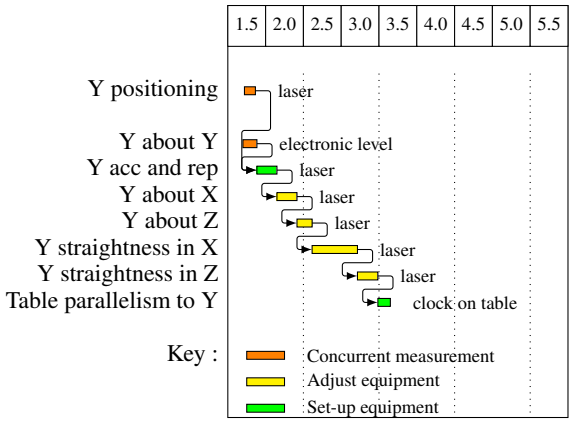


Figure 2: Excerpt from a produced calibration plan from the machine tool calibration application

responsibility for managing and dealing with the incident lies with the highway agencies which are serving on that area. It is a top priority for highway agencies around the world to manage accidents more effectively, efficiently and as fast as possible to save time, money and most crucially life. Utilising automated planning capabilities in real applications is a current topic with great potential to help in speed, accuracy, and co-ordination of tasks to be carried out.

Urban Traffic Control

Traffic in urban areas (e.g. town centers) tends to be dense, especially in rush hours, which often leads to traffic jams which can significantly increase travel time. Therefore, a need for efficient Urban Traffic Control in such exposed areas is becoming more important. It is necessary to minimize travel time by efficiently navigating road vehicles throughout the road network, while avoiding road congestion and diverting traffic when a road is blocked. Traditional Urban Traffic Control methods are based on reactive acting, they operate, for instance, on the basis of adaptive green phases and flexible co-ordination in road (sub)networks based on measured traffic conditions (Dusparic and Cahill 2009; sheng Yang et al. 2005; Salkham et al. 2008; Daneshfar et al. 2009; Bazzan 2005). However, these approaches are still not very efficient during unforeseen situations such as road incidents, when changes in traffic are requested in a short time interval (Dusparic and Cahill 2012). The role of AI planning in this case is to come with deliberative reasoning. In contrary to traditional reactive control we can reason about the road network globally that is useful while dealing with unexpected situations. Consequently, by exploiting AI planning in traffic control, we can reduce cost and pollution which is often a serious issue in town/city centers.

Criteria for evaluating approaches

We identified several criteria that are useful for evaluating the considered approaches for encoding domain models.

Operationality. How efficient are the models produced? Is

```
(:durative-action DRIVE
   :parameters (?r - road ?n - num)
   :duration (= ?duration (length ?r))
   :condition
    (and
      (at start (>= (head ?r) (val ?n)))
      (over all (operational ?r))
    )
   :effect
    (and
     (at start (decrease (head ?r) (val ?n)))
     (at end (increase (tail ?r) (val ?n)))
    )
)
```

Figure 3: A sample planning operator from the Urban Traffic Control application encoded in PDDL 2.1.

the method able to improve the performances of planners on generated models and problems?

Collaboration. Does the method/tool help in team efforts? Is the method/tool suitable for being exploited in teams or is it focused on supporting the work of a single user?

Maintenance. How easy is it to come back and change a model? Is there any type of documentation that is automatically generated? Does the tool induce users to produce documentation?

Experience. Is the method/tool indicated for inexperienced planning users? Do users need to have a good knowledge of PDDL? Is it able to support users and to hide low level details?

Efficiency. How quickly are acceptable models produced?

Debugging. Does the method/tool support debugging? Does it cut down the time needed to debug? Is there any mechanism for promoting the overall quality of the model?

Support. Are there manuals available for using the method/tools? Is it easy to receive support? Is there an active community using the tool?

Evaluation of the approaches with respect to stated criteria

In this paper we employ and hence evaluate three separate methods for knowledge formulation: (A) the traditional method of hand-coding by a PDDL expert, using a text editor and relying on dynamic testing for debugging; (B) using state-of-the-art KE tool itSIMPLE; (C) using transparency and consistency checkers in GIPO III.

In the following we will evaluate each method with respect to the criteria stated in the previous Section.

Method A

This method involves a PDDL expert that uses a text editor (in this paper, Gedit) for generating a planning domain model, given the description of the real world domain. For illustration, a handcoded planning operator is depicted in Figure 3.

Operationality. Even if this is the most exploited method for generating new planning domain models, there is no evidence that it leads to models that are more efficient than those generated by other methods. The quality of models depends on the expertise of the person that encodes it, which is very hard to predict a-priori. In fact, we have experimentally observed that often, the models generated by this method reduce the performances of planning algorithms.

Collaboration. This method does not support any type of collaboration. Usually the model is produced by a single expert, that eventually discusses issues or improvements with domain experts rather than with other planning experts.

Maintenance. It is usually easy, for the expert that encoded the domain model, to come back and maintain or modify it. On the other hand, models are usually not documented. This means that the maintenance is potentially hard, with regard to the complexity of the model, for people that were not involved in the encoding process.

Experience. This method is applicable only for PDDL experts. PDDL experts know the ways for handling some common issues and are able to interpret the planners output in order to identify bugs.

Efficiency. Usually, the first version of the model is quickly produced. This leads users to perceive this method as a very efficient one. On the other hand, the first version requires a lot of dynamic tests and improvements to become acceptable.

Debugging. Debugging while hand-coding a model is a critical task. The only way for debugging is dynamic testing. This involves the use of one (or more) planners for solving some toy instances. The produced plans are then analysed for identifying bugs that can be fixed by modifying the model. This cycle is repeated until no bugs are found. Omitting some important constraints is often a source of bugs for this encoding method.

Support. Since this is a traditional method, there are many guides available online for generating new domain models. However, these guides are usually technically written and are very difficult to follow for non-experts of automated planning.

Method B

This method involves a user that exploits itSIMPLE for generating new planning domain models. The steps of the method follow the use of UML in software engineering: (i) design of class diagrams; (ii) definition of state machines; (iii) translation to PDDL; (iv) generation of problem files.

Operationality. From our experience, it seems that domain models generated by itSIMPLE can often improve the performances of planners. This is probably due to a domain description that is less constrained than the one developed by exploiting method A. The quality of the models generated does not depend on the expertise of the users; itSIMPLE guides users in the design process.

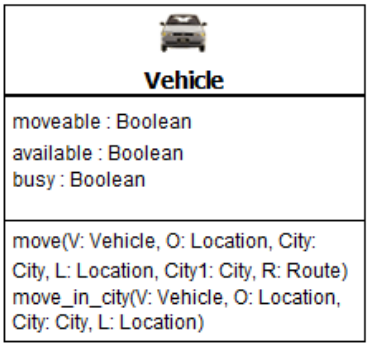


Figure 4: An example of a Class Diagram designed in itSIMPLE.

```
...
Checking method carry_direct(P,O,D)
found an unrecognised decomposition item: unload_subject(P,D,V)'.
Check failed
Checking method carry_direct(P,O,D)
found an unrecognised decomposition item: unload_subject(P,D,V)'.
Check failed
Checking method transport(Subject,Org,Dest)
The static predicate in_region(Org,Region) has no prototype
The static predicate in_region(Dest,Region) has no prototype
Check failed
Doing task checks.
....
```

Figure 5: Part of output from GIPO: here the transparency of HTN methods is checked and found to fail, with the likely faulty components identified.

Collaboration. itSIMPLE has not been designed for team work. This means that usually, the model is developed by a single user. However, it is possible to import models (projects) developed by different users. This helps to exchange ideas and comments among users. Moreover, the UML diagrams generated are useful for sharing and discussing issues with experts.

Maintenance. The itSIMPLE tool is designed for supporting a disciplined design cycle. The UML diagrams can also be used as documentation. From this point of view it is easy to maintain a generated domain model also for people that were not involved in the design process. However, if a model generated by itSIMPLE is modified using a different tool (or even a text editor), then it is not possible to import it back to itSIMPLE.

Experience. The typical itSIMPLE user does not have to be a PDDL expert. However, he should have some basic experience in software engineering and especially in UML.

Efficiency. Most of the time is spent in designing classes of objects and defining legal interactions between them in UML. After that, only a short time is required for debugging. This method is usually slower than method A, but faster than method C, to generate a first version model.

Debugging. Even if itSIMPLE provides dynamic analysis by simulation of Petri Nets created from UML models, most debugging initiates through dynamic testing done by running planners on some toy instances. While the UML description of the models helps in development and maintenance, and "designs out" some sources of error, it is the failure of a planning engine to solve a given problem that, in most cases, alerts the user to bug presence.

Support. itSIMPLE provides a complete documentation which includes a description of the tool, a tutorial and an online FAQ section. It is easy to find information and solutions for most of the common issues.

Method C

We focus on domain models encoded in OCL_h with the help of tools in GIPO-III, using basic consistency checkers as well as the more complex transparency property checker. Hierarchies of classes are used to capture state: for example, in the Road Traffic Accident domain, in a particular state an ambulance may have a position, be in service and available. The set of constraints are added using GIPO-III to encode the behaviour of each of the dynamic object classes, that is, the range of states that each object can occupy.

Operationality. The size of the OCL_h model is larger than the size of the PDDL models, because state constraints are encoded explicitly, and HTN methods are specified in addition to primitive ones. The structure of the solutions is similar to the structure of solutions generated by the LPG planner on the PDDL model developed in itSIMPLE. However, given the different nature of OCL_h, it seems to be impossible to provide a rigorous comparison with PDDL.

Collaboration. GIPO has been designed for a single user. However, it is possible to import models (projects) developed by different users. This helps the exchange of ideas and comments among users.

Maintenance. Domain models generated by GIPO are generally easier to maintain than the hand-encoded ones, due to the consistency checking opportunities during each stage of the modelling process.

Experience. The typical user is not required to be an OCL expert, but he should have some basic knowledge of the language or the tool.

Efficiency. GIPO does not require creating UML diagram like itSIMPLE, but it requires to explicitly encode the constraints of the domain as transition diagrams, that are then used to create operators. This method is usually slower then the others, but the first model generated is very close to the final one.

Debugging. The creation of a dynamic hierarchy of object classes encoded constraints of the domain explicitly, and this is what is used by GIPO's tool support to check operator schema, states, predicates, etc., and identify bugs prior to dynamic testing. Part of the output of the consistency checking tools is shown in Figure 5.

Support. GIPO comes with complete documentation which includes a user manual, tutorial and OCL manual. It is easy to find information and solutions for most of the common issues.

Summary of lessons learned

We can now summarize the lessons that we learned by using the three outlined methods for formulating requirements into domain models.

We observed that creating different models does not take very different amounts of time (taking into account the developers expertise) while exploiting method A and B. Method C usually requires significantly more time resource than the others, because in addition the users have to provide object hierarchies and invariants (for consistency check), and also users have to encode HTN methods. As mentioned before, in method C creation of a dynamic hierarchy of object classes encoded constraints of the domain explicitly, and this is what is used by GIPO's tool support to check operator schema, states, predicates, etc., and identify bugs prior to dynamic testing. While one could argue that dynamic testing will pick bugs up anyway, there may be behaviours that have not been picked up in the tests done in method A and B, that result from hidden bugs. On the other hand the UML description of the domain, that is required by itSIMPLE, helps to prevent many unwanted behaviours. Method A is the most sensitive to bugs, and the quality of the produced model completely depend on the expertise of the user.

Considering the maintenance of the generated models, method B provides the better instruments for changing a model. The UML description provides a sort of documentation that can be exploited for quickly understanding the domain and for applying changes. An issue that we noticed while working with itSIMPLE is that it is not possible to import a model that, even if originally generated by using itSIMPLE, has been slightly modified with a different tool. This force users to make several steps in the framework also for very small changes. In method C, the complex representation of domain models makes it more difficult for a non-expert user to come back and maintain the model.

Regarding the generated models, there are several interesting aspects to consider. Models generated by method A are usually very compact in terms of numbers of lines, predicates and types, but they are usually over-constrained in order to avoid unwanted behaviours. The iterative process of analyzing produced plans, identifying bugs and removing them from the model leads to incrementally add constraints in the form of pre- and/or post- conditions. The structured and principled process of encoding the requirements of Method B usually leads to domain encodings that are clear and easy to understand, even if less compact (around 10% longer) then the ones generated by method A. It is worth mentioning that the good quality of the encoded domains leads to good quality generated plans. The quality of models generated by method C is harder to understand due to the different language used. The models are significantly larger than PDDL ones (about two times longer in terms of number of lines) and need a HTN planner to solve corresponding problems. We observed that the structures of the solutions are similar to solutions generated by domain-independent planners on models developed by other methods. We believe that a possible advantage of this approach might be the better scalability than the PDDL models. However, current OCL_h techniques do not support durative actions, which makes it less interesting in domains where time optimization is critical.

Needs in the design of future tools

The comparison of the three methods for encoding different domain models was fruitful. It gave us the opportunity for understanding the strengths and weaknesses of them, and to highlight the needs that future tools should meet.

Expertise

A main issue of current KE approaches for encoding domain models is that they require a specific expertise. Method A (and some approaches based on existing KE tools such as PDDL Studio) requires a PDDL expert, Method B requires some expertise in UML language, which is common knowledge mainly in software engineering. Finally, method C requires some expertise in the OCL_h language, which is not a widely known language in the AI Planning community. This requirement might significantly reduce the number of potential users of the KE tools. Since users with different research background usually do not have the required expertise, they are not able to exploit existing approaches for encoding domain models. They require an expert that, due to his limited knowledge of the real world domain, will introduce some noise in the encoding. Moreover, given the hardness of generating domain models for planning, many users are not exploiting automated planning but use easier approaches, even if they are less efficient. It is also worth considering that KE tools for encoding domain models are, usually, not very well known outside the planning community. This, again, reduces the number of potential users that could exploit them.

Team work

Current KE tools are designed for a single user. This is usually fine; actually, the generated domain models are encoding easy domains or significantly simplified versions of complex domains. On the other hand, the number of efficient KE tools is growing, especially in the last few years. Hopefully, in coming years, we will be able to encode very accurate models of complex real-world domains. In this scenario, it seems reasonable that many experts will have to cooperate for generating a domain model. From this perspective it is straightforward to consider the need of tools explicitly designed for team work as a critical requirement for future KE tools.

Maintenance

Users are not supported by existing KE tools in writing documentation related to the generated model. As a result, users are usually not writing any sort of documentation. Given this, it is often quite hard to change an existing domain model a few months after its generation. Providing support for writing documentation would make changes

easier and would also help the users while encoding the model. The process of describing what has been done is a first test for the model. Furthermore, some tools are not able to handle domain models that have been changed manually, or by using a different tool. This limits the support that such tools could give to the life cycle of domain models.

Debugging
We noticed that the checking tools provided by GIPO are very helpful for minimizing the time spent on dynamic debugging. Moreover, exploiting the automatic debugging is a strategy for reducing the number of bugs that remain in the domain model, since many problems are usually not easy to find by dynamic debugging. A significant improvement in the techniques for automatic debugging of static/dynamic constraints will lead to significantly better encoded domain models.

Language support
Finally, existing KE tools for generating domain models for planning have a very limited support of the features of PDDL language. Most of them are supporting only PDDL, while a few of them are also able to handle some structures of PDDL2.1 (Fox and Long 2001). It is noticeable that the latest versions of PDDL have some features (e.g. durative actions, actions costs, ...) that are fundamental for a correct encoding of real world domains. Furthermore, none of the existing tools support PDDL+ (Howey, Long, and Fox 2004). PDDL+ provides features for dealing with continuous planning, which is needed in systems working in real-time and that must be able to react to unexpected events. This is the case for the machine tool calibration domain that we considered in this paper. In addition, during the implementation of this domain it was noticed that it can be difficult to model and debug multiple, interacting equations in PDDL. The only way of currently evaluating the implementation is by using VAL with the domain, problem and solution. A KE tool with support for PDDL+ with strong numeric domains would be highly beneficial.

Conclusions

In this paper we have presented the state-of-the-art of Knowledge Engineering tools for encoding planning domain models. We introduced three real world domains that we have encoded: the machine tool calibration, the road traffic accident management and the urban traffic control. We used and evaluated three different strategies for knowledge formulation, encoding domain models from a textual, structural description of requirements using: (i) the traditional method of a PDDL expert and text editor (ii) a leading planning GUI with built in UML modelling tools (iii) a hierarchical, object-based notation inspired by formal methods. We evaluate these methods using a set of criteria, i.e., operationality, collaboration, maintenance, experience, efficiency, debugging and support. We observed that creating different models does not take very different amounts of time. We highlighted weaknesses of existing methods and tools and we discussed the needed in the design of future tools support for PDDL-inspired development.

Future work will involve a simulation framework for evaluating plan execution, where we can couple model design and plan generation more tightly. We are also interested in improving the KE tools comparison by considering also other existing tools and a larger set of features to compare, such as quality of the solutions found and runtimes of different planners on generated domain models.

Acknowledgements

The research was funded by the UK EPSRC Autonomous and Intelligent Systems Programme (grant no. EP/J011991/1).

References

Barreiro, J.; Boyce, M.; Do, M.; Jeremy Frank, M. I.; Kichkayloz, T.; Morrisy, P.; Ong, J.; Remolina, E.; Smith, T.; and Smithy, D. 2012. EUROPA: A Platform for AI Planning, Scheduling, Constraint Programming, and Optimization. In *Proceedings of the 22nd International Conference on Automated Planning & Scheduling (ICAPS-12) – The 4th International Competition on Knowledge Engineering for Planning and Scheduling*.

Bazzan, A. L. 2005. A distributed approach for coordination of traffic signal agents. *Autonomous Agents and Multi-Agent Systems* 10(1):131–164.

Bouillet, E.; Feblowitz, M.; Feng, H.; Ranganathan, A.; Riabov, A.; Udrea, O.; and Liu, Z. 2009. Mario: middleware for assembly and deployment of multi-platform flow-based applications. In *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '09, 26:1–26:7. New York, NY, USA: Springer-Verlag New York, Inc.

Castillo, L.; Fdez-olivares, J.; scar Garca-prez; and Palao, F. 2006. Efficiently handling temporal knowledge in an htn planner. In *In Sixteenth International Conference on Automated Planning and Scheduling, ICAPS*, 63–72. AAAI.

Daneshfar, F.; RavanJamjah, J.; Mansoori, F.; Bevrani, H.; and Azami, B. Z. 2009. Adaptive fuzzy urban traffic flow control using a cooperative multi-agent system based on two stage fuzzy clustering. In *Vehicular Technology Conference, 2009. VTC Spring 2009. IEEE 69th*, 1–5.

Dusparic, I., and Cahill, V. 2009. Distributed w-learning: Multi-policy optimization in self-organizing systems. In *Proceedings of the 2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, SASO '09, 20–29. Washington, DC, USA: IEEE Computer Society.

Dusparic, I., and Cahill, V. 2012. Autonomic multi-policy optimization in pervasive systems: Overview and evaluation. *ACM Trans. Auton. Adapt. Syst.* 7(1):11:1–11:25.

Edelkamp, S., and Hoffman, J. 2004. PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition. Technical report, Albert-Ludwigs-Universität Freiburg.

Feblowitz, M. D.; Ranganathan, A.; Riabov, A. V.; and Udrea, O. 2012. Planning-based composition of stream processing applications. *and Exhibits* 5.

Fox, M., and Long, D. 2001. PDDL2.1: An extension to PDDL for expressing temporal planning domains . In *Technical Report, Dept of Computer Science, University of Durham*.

Gerevini, A.; Saetti, A.; and Serina, I. 2006. An Approach to Temporal Planning and Scheduling in Domains with Predictable Exogenous Events. *JAIR* 25:187–231.

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.

González-Ferrer, A.; Fernández-Olivares, J.; and Castillo, L. 2009. JABBAH: A java application framework for the translation between business process models and htn. In *Working notes of the 19th International Conference on Automated Planning & Scheduling (ICAPS-09) – Proceedings of the 3rd International Competition on Knowledge Engireeng for Planning and Scheduling(ICKEPS)*, 28–37.

Howey, R.; Long, D.; and Fox, M. 2004. Automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *Proceedings of the Sixteenth International Conference on Tools with Artificial Intelligence*, 294 – 301.

Jimoh, F.; Chrpa, L.; Gregory, P.; and McCluskey, T. 2012. Enabling autonomic properties in road transport system. In *The 30th Workshop of the UK Planning And Scheduling Special Interest Group, PlanSIG 2012*.

McCluskey, T. L., and Kitchin, D. E. 1998. A tool-supported approach to engineering htn planning models. In *In Proceedings of 10th IEEE International Conference on Tools with Artificial Intelligence*.

McCluskey, T. L., and Simpson, R. 2006. Combining constraint-based and classical formulations for planning domains: GIPO IV. In *Proceedings of the 25th Workshop of the UK Planning and Scheduling SIG (PLANSIG-06)*, 55–65.

Owens, N.; Armstrong, A.; Sullivan, P.; Mitchell, C.; Newton, D.; Brewster, R.; and Trego, T. 2000. Traffic Incident Management Handbook. Technical Report Office of Travel Management, Federal Highway Administration.

Parkinson, S.; Longstaff, A.; Crampton, A.; and Gregory, P. 2012. The application of automated planning to machine tool calibration. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012*.

Plch, T.; Chomut, M.; Brom, C.; and Barták, R. 2012. Inspect, edit and debug pddl documents: Simply and efficiently with pddl studio. *ICAPS12 System Demonstration* 4.

Salkham, A.; Cunningham, R.; Garg, A.; and Cahill, V. 2008. A collaborative reinforcement learning approach to urban traffic control optimization. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 02*, WI-IAT '08, 560–566. Washington, DC, USA: IEEE Computer Society.

Shah, M.; McCluskey, T.; and Chrpa, L. 2012. Symbolic representation of road traffic domain for automated planning to manage accidents. In *The 30th Workshop of the UK Planning And Scheduling Special Interest Group, PlanSIG 2012*.

sheng Yang, Z.; Chen, X.; shan Tang, Y.; and Sun, J.-P. 2005. Intelligent cooperation control of urban traffic networks. In *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, volume 3, 1482–1486 Vol. 3.

Simpson, R.; Kitchin, D. E.; and McCluskey, T. 2007. Planning domain definition using gipo. *Knowledge Engineering Review* 22(2):117–134.

Smith, D. E.; Frank, J.; and Cushing, W. 2008. The anml language. *Proceedings of ICAPS-08*.

Vaquero, T. S.; Romero, V.; Tonidandel, F.; and Silva, J. R. 2007. itSIMPLE2.0: An integrated tool for designing planning domains. In *Proceedings of the 17th International Conference on Automated Planning & Scheduling (ICAPS-07)*, 336–343. AAAI Press.

Vaquero, T. S.; Tonaco, R.; Costa, G.; Tonidandel, F.; Silva, J. R.; and Beck, J. C. 2012. itSIMPLE4.0: Enhancing the modeling experience of planning problems. In *System Demonstration – Proceedings of the 22nd International Conference on Automated Planning & Scheduling (ICAPS-12)*.

Vaquero, T. S.; Silva, J. R.; and Beck, J. C. 2011. A brief review of tools and methods for knowledge engineering for planning & scheduling. In *Proceedings of the Knowledge Engineering for Planning and Scheduling workshop – The 21th International Conference on Automated Planning & Scheduling (ICAPS-11)*.

Vodrážka, J., and Chrpa, L. 2010. Visual design of planning domains. In *KEPS 2010: Workshop on Knowledge Engineering for Planning and Scheduling*.

A timeline, event, and constraint-based modeling framework for planning and scheduling problems

Gérard Verfaillie and **Cédric Pralet**
ONERA - The French Aerospace Lab, F-31055, Toulouse, France
{Gerard.Verfaillie,Cedric.Pralet}@onera.fr

Abstract

This paper presents a framework dedicated to the modeling of deterministic planning and scheduling problems. This framework is based on the notions of timelines (evolutions of state variables and of resource levels), events which impact timelines, and constraints on and between events. Building it has been motivated by the observation that many real planning and scheduling problems involve a finite set of events and that the main question when solving them is to assign values to event parameters (presence in the plan, position in the event sequence, precise date, specific type-dependent parameters) by satisfying a finite set of constraints on these parameters and optimizing some function of some of these parameters. This framework, we refer to as TECK for *Timelines, Events, and Constraints Knowledge*, makes it possible to express four kinds of knowledge that must be combined when planning and scheduling: knowledge about events (event presence and parameters), time (event positions and dates), state (state variable evolutions), and resources (discrete or continuous resource evolutions).

Introduction

In classical scheduling problems, in Operations Research (OR) (Baptiste, Pape, and Nuijten 2001), the basic notions are tasks, time, and resources. The goal is to schedule a finite set of tasks by satisfying a set of temporal and resource constraints and by optimizing a given criterion, for example the time at which all tasks are performed (makespan). On the contrary, in classical planning problems, in Artificial Intelligence (AI) (Ghallab, Nau, and Traverso 2004), the basic notions are state variables, actions, and action preconditions and effects. The goal is to build a finite sequence of actions that leads the system from a given initial state to a state that satisfies some goal conditions. Similar notions of state variables, events, event preconditions (guards) and effects (transitions) are at the basis of the modeling of discrete event dynamic systems (Cassandras and Lafortune 2008) (automata, Petri nets, Markov processes). On the other hand, in more basic combinatorial optimization frameworks (linear programming, constraint programming, boolean satisfiability), the basic notions are variables and constraints. The goal is to assign values to variables in such a way that all constraints are satisfied and a given criterion is optimized.

It is well recognized that neither classical scheduling, nor classical planning frameworks can correctly model real-world problems and that most of these problems combine scheduling and planning features: need for reasoning on time and resources as in scheduling and need for reasoning on states and event preconditions and effects as in planning. In fact when looking at the planning and scheduling literature and at the many real-world problems we had to face, we get convinced that four kinds of knowledge must be expressed and handled: knowledge about *events* (how events can or must be activated), about *time* (how events are ordered and at which time they occur), about *state* (how events change state values or evolutions), and about *resources* (how events impact resource levels or evolutions). Moreover, whereas classical planning considers that there is no predefined bound on the number of actions in a plan, we can claim that many real-world planning and scheduling problems involve a *predefined finite set of events* and that the main question when solving them is to assign values to event parameters (presence in the plan, position in the event sequence, precise date, specific type-dependent parameters) by satisfying a finite set of constraints on these parameters and optimizing some function of some of these parameters.

These observations led us to the definition of a new framework dedicated to the modeling of deterministic planning and scheduling problems (certain initial state and event effects). Its main ingredients are: static variables, dynamic state variables, event types, events, constraints on events and states, and some optimization criterion.

In this paper, we present this framework, we refer to as TECK for *Timelines, Events, and Constraints Knowledge*, by using as an illustrative example the Petrobras planning and scheduling problem of ship operations for petroleum ports and platforms, which has been one of the challenges of the ICKEPS competition in 2012 (International Competition on Knowledge Engineering for Planning and Scheduling, http://icaps12.poli.usp.br/icaps12/ickeps). See (Vaquero et al. 2012) for a problem description and modeling, and (Toropila et al. 2012) for three modeling and solving approaches.

We start with a short presentation of the illustrative example we use. Then, we show how a planning domain and a planning problem are defined and what an optimal solution is in the TECK framework. Before concluding, we discuss

complexity and solving issues, and subsumed and related frameworks.

Illustrative example

The Petrobras planning and scheduling problem is a kind of logistics problem which involves a set of ports and platforms, a set of vessels, and a set of items to be delivered from ports to platforms. The goal is to deliver all items by satisfying constraints on vessel capacities and speeds and constraints on docking, undocking, loading, unloading, and refueling operations at ports or platforms, and by optimizing some combination of three criteria (makespan, fuel consumption, and docking cost). Its data is:

- a set **Po** of ports;
- a set **Pf** of platforms;
- a set **Ve** of vessels;
- a set **It** of items to be delivered;
- a set **Wa** of waiting areas;
- a subset **Pfr** $\subseteq$ **Pf** of platforms where refueling is possible; refueling is possible at all the ports and at some platforms, but not at the waiting areas;
- for each pair of locations $l_1, l_2 \in$ **Po**$\cup$**Pf**$\cup$**Wa**, a distance $\mathbf{Di}_{l_1,l_2}$ from l_1 to l_2;
- for each vessel $v \in$ **Ve**:
  - a capacity $\mathbf{Ci}_v$ (resp. $\mathbf{Cf}_v$) in terms of item weight (resp. of fuel);
  - a speed $\mathbf{Sp}_v$: distance per time unit;
  - a fuel consumption rate $\mathbf{Rfe}_v$ (resp. $\mathbf{Rfc}_v$) when v is empty (resp. not empty): fuel per distance unit;
  - a docking/undocking duration $\mathbf{Dpo}_v$ (resp. $\mathbf{Dpf}_v$) at a port (resp. at a platform);
  - a loading/unloading rate $\mathbf{Ri}_v$ at a port or platform: weight per time unit;
  - a refueling rate $\mathbf{Rpo}_v$ (resp. $\mathbf{Rpf}_v$) at a port (resp. at a platform): fuel per time unit;
  - an initial waiting area $\mathbf{Il}_v \in$ **Wa**;
  - an initial level of fuel $\mathbf{If}_v$;
- for each item $i \in$ **It**:
  - a loading location $\mathbf{Ll}_i \in$ **Po** $\cup$ **Pf**;
  - an unloading location $\mathbf{Ul}_i \in$ **Po** $\cup$ **Pf**;
  - a weight $\mathbf{We}_i$;
- a docking cost **Co** per hour at a port;
- a maximum number **Ns** of steps in the sequence of visits of each vessel: at each step, a vessel visits a port ot a platform; a port or platform can be visited several times; initial and final steps in waiting areas are excluded.

It is assumed that all vessels are initially empty in their initial waiting areas. The goal is that all items be delivered and that all vessels go empty in a waiting area, with a level of fuel that allows them at least to reach a refueling location.

For each waiting area $l \in$ **Wa** and each vessel $v \in$ **Ve**, let $\mathbf{Mf}_{l,v} = (\min_{l' \in \mathbf{Po} \cup \mathbf{Pfr}} \mathbf{Di}_{l,l'}) \cdot \mathbf{Rfe}_v$ be the minimum level of fuel allowing v to reach a refueling station from l.

Planning domain definition

We adopt the usual distinction in planning between planning domain and planning problem. In the TECK framework, a planning domain is defined by:

- a finite set **Vs** of static variables;
- a finite set **Vd** of dynamic variables;
- a finite set **De** of dependencies between dynamic variables;
- a finite set **Et** of event types.

Static variables A static variable represents a planning domain parameter that does not evolve due to successive events. To model the Petrobras problem, we introduce two sets of static variables:

- for each vessel $v \in$ **Ve**, the number $n_v \in [\![0; \mathbf{Ns}]\!]$ of steps in its sequence of visits, initial and final steps in waiting areas excluded;
- for each item $i \in$ **It**, the vessel $v_i \in$ **Ve** that transports it (it is assumed that any item is transported by a unique vessel from its loading to its unloading location, directly or not).

These variables must not be mistaken for problem data: they are really variables whose value must be decided by the planning process.

Dynamic variables A dynamic variable represents a planning domain parameter that evolves due to successive events. We distinguish dynamic variables whose value remains constant between two successive events (piecewise constant evolution) and those whose value evolves continuously with time between two successive events (linearly or not; non piecewise constant evolution). See Figure 1 for an illustration of the two kinds of evolution. To model the Petrobras problem, we introduce four sets of dynamic variables:

- for each vessel $v \in$ **Ve**, the current location $l_v \in$ **Po**$\cup$**Pf**$\cup$ **Wa**;
- for each vessel $v \in$ **Ve**, the current charge $c_v \in [0; \mathbf{Ci}_v]$ in terms of item weight;
- for each vessel $v \in$ **Ve**, the current level $f_v \in [0; \mathbf{Cf}_v]$ of fuel;
- for each item $i \in$ **It**, its status $s_i \in \{\mathbf{W}, \mathbf{T}, \mathbf{D}\}$ (**W** for waiting, **T** for transit, and **D** for delivered).

The evolution of all these variables is piecewise constant. We do not need to model the continuous evolution of the level of fuel in each vessel. However, if we should model it, the current level of fuel would be an example of dynamic variable whose evolution is not piecewise constant (linear in this case).

Dependencies Dependencies allow functional dependencies between dynamic variables to be represented: the fact that some variables functionally depend on other static or dynamic variables. In the Petrobras problem, the current

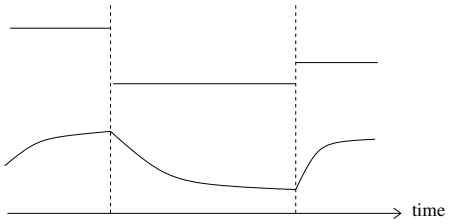


Figure 1: Piecewise constant evolution of a dynamic variable (above) and non piecewise constant evolution (below).

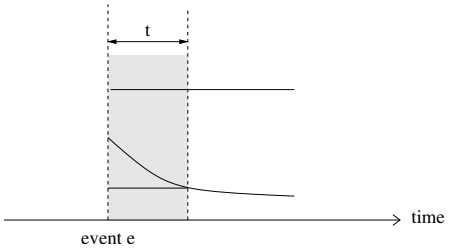


Figure 2: Value of a dynamic variable after an event in case of a piecewise constant evolution (above) and of a non piecewise constant evolution (below).

charge of a vessel v is the sum of the weights of the items that are assigned to v and currently in transit:

$$\forall v \in \mathbf{Ve} :\ c_v = \sum_{i \in \mathbf{It}} \mathbf{We}_i \cdot (v_i = v) \cdot (s_i = \mathbf{T}) \quad (1)$$

Event types With each event type *et*, are associated a finite set $\mathbf{par}(et)$ of parameters, a finite set $\mathbf{pre}(et)$ of preconditions, and a finite set $\mathbf{eff}(et)$ of effects.

Each parameter is a variable.

Each precondition is a constraint on a subset of $\mathbf{Vs} \cup \mathbf{par}(et) \cup \mathbf{Vd}$, where **Vd** represents the value of the dynamic variables just before the event.

Each effect is either (1) a functional constraint on a subset of $\mathbf{Vs} \cup \mathbf{par}(et) \cup \mathbf{Vd} \cup \mathbf{Vd'}$, where **Vd'** represents the value of the dynamic variables just after the event, which expresses the value of a dynamic variable after the event, in case of a piecewise constant evolution, or (2) a functional constraint on a subset of $\mathbf{Vs} \cup \mathbf{par}(et) \cup \mathbf{Vd} \cup \mathbf{Vd'} \cup \{t\}$, where t represents the time that went on from the event, which expresses how a dynamic variable evolves continuously after the event, in case of a non piecewise constant evolution. See Figure 2 for an illustration. Obviously, a dynamic variable cannot be the target of several dependencies or effects (unique definition) and the graph induced by dependencies and effects must be acyclic (no loop in definitions). If a variable appears in no effect of an event, its value (in case of a piecewise constant evolution) or its evolution function (in case of a non piecewise constant evolution) remains unchanged.

To model the Petrobras problem, we use six event types **StDo**, **StUl**, **StLo**, **StRf**, **StUd**, and **StTr**:

- event type **StDo** represents the starting of a docking operation at a port or platform; it has two parameters: a vessel $\mathbf{v} \in \mathbf{Ve}$ and a location $\mathbf{l} \in \mathbf{Po} \cup \mathbf{Pf}$; it has neither precondition, nor effect;
- event type **StUl** represents the starting of an unloading operation; it has three parameters: a vessel $\mathbf{v} \in \mathbf{Ve}$, a location $\mathbf{l} \in \mathbf{Po} \cup \mathbf{Pf}$, and a set of items $\mathbf{Iu} \subseteq \mathbf{It}$ to be unloaded; it has two preconditions which respectively express that one unloads all the items that are in transit in **v** and whose unloading location is **l**, and that at least one item is unloaded:

  $$\mathbf{Iu} = \{i \in \mathbf{It} \,|\, (s_i = \mathbf{T}) \wedge (v_i = \mathbf{v}) \wedge (\mathbf{Ul}_i = \mathbf{l})\} \quad (2)$$
  $$\mathbf{Iu} \neq \emptyset \quad (3)$$

  It has a set of effects, expressing that all the unloaded items are now delivered:

  $$\forall i \in \mathbf{Iu} :\ s'_i = \mathbf{D} \quad (4)$$

  The charge c_v does not appear in event effects because its new value can be inferred by dependency (see Eq. 1).
- event type **StLo** represents the starting of a loading operation; it has three parameters: a vessel $\mathbf{v} \in \mathbf{Ve}$, a location $\mathbf{l} \in \mathbf{Po} \cup \mathbf{Pf}$, and a set of items $\mathbf{Il} \subseteq \mathbf{It}$ to be loaded; it has three preconditions which respectively express that one loads only items that are waiting in **l** for loading in **v**, that at least one item is loaded, and that the vessel capacity cannot be exceeded:

  $$\mathbf{Il} \subseteq \{i \in \mathbf{It} \,|\, (s_i = \mathbf{W}) \wedge (v_i = \mathbf{v}) \wedge (\mathbf{Ll}_i = \mathbf{l})\} \quad (5)$$
  $$\mathbf{Il} \neq \emptyset \quad (6)$$
  $$c_{\mathbf{v}} + \sum_{i \in \mathbf{Il}} \mathbf{We}_i \leq \mathbf{Ci_v} \quad (7)$$

  It has a set of effects, expressing that all the loaded items are now in transit:

  $$\forall i \in \mathbf{Il} :\ s'_i = \mathbf{T} \quad (8)$$

- event type **StRf** represents the starting of a refueling operation; it has three parameters: a vessel $\mathbf{v} \in \mathbf{Ve}$, a location $\mathbf{l} \in \mathbf{Po} \cup \mathbf{Pfr}$, and a quantity $\mathbf{f} \in [0; \mathbf{Cf_v}]$ of fuel; it has two preconditions which respectively express that the quantity of fuel must be strictly positive, and that the vessel capacity cannot be exceeded:

  $$\mathbf{f} > 0 \quad (9)$$
  $$f_{\mathbf{v}} + \mathbf{f} \leq \mathbf{Cf_v} \quad (10)$$

  It has one effect on the current level of fuel:

  $$f'_{\mathbf{v}} = f_{\mathbf{v}} + \mathbf{f} \quad (11)$$

- event type **StUd** represents the starting of an undocking operation from a port or platform; it has two parameters: a vessel $\mathbf{v} \in \mathbf{Ve}$ and a location $\mathbf{l} \in \mathbf{Po} \cup \mathbf{Pf}$; it has neither precondition, nor effect;
- finally, event type **StTr** represents the starting of a transit from a location to another one; it has three parameters: a vessel $\mathbf{v} \in \mathbf{Ve}$, a starting location $\mathbf{l} \in \mathbf{Po} \cup \mathbf{Pf} \cup \mathbf{Wa}$, and a target location $\mathbf{tl} \in \mathbf{Po} \cup \mathbf{Pf} \cup \mathbf{Wa}$; it has six preconditions

which express that the starting location **l** must be the current one, the target location **tl** must be different from the starting one, there must be enough fuel to reach the target location, and, when the target location **tl** is a waiting area, all the items assigned to **v** must have been delivered and **tl** will be reached with enough fuel to reach a refueling station:

$$\mathbf{l} = l_{\mathbf{v}} \tag{12}$$
$$\mathbf{tl} \neq \mathbf{l} \tag{13}$$
$$(c_{\mathbf{v}} = 0) \rightarrow (f_{\mathbf{v}} \geq \mathbf{Di}_{\mathbf{l},\mathbf{tl}} \cdot \mathbf{Rfe}_{\mathbf{v}}) \tag{14}$$
$$(c_{\mathbf{v}} > 0) \rightarrow (f_{\mathbf{v}} \geq \mathbf{Di}_{\mathbf{l},\mathbf{tl}} \cdot \mathbf{Rfc}_{\mathbf{v}}) \tag{15}$$
$$(\mathbf{tl} \in \mathbf{Wa}) \rightarrow (\sum_{i \in \mathbf{It}} ((s_i \neq \mathbf{D}) \wedge (v_i = \mathbf{v})) = 0) \tag{16}$$
$$(\mathbf{tl} \in \mathbf{Wa}) \rightarrow (f_{\mathbf{v}} \geq \mathbf{Di}_{\mathbf{l},\mathbf{tl}} \cdot \mathbf{Rfe}_{\mathbf{v}} + \mathbf{Mf}_{\mathbf{tl},\mathbf{v}}) \tag{17}$$

On the other hand, it has three effects on the current location and the current level of fuel:

$$l'_{\mathbf{v}} = \mathbf{tl} \tag{18}$$
$$(c_{\mathbf{v}} = 0) \rightarrow (f'_{\mathbf{v}} = f_{\mathbf{v}} - \mathbf{Di}_{\mathbf{l},\mathbf{tl}} \cdot \mathbf{Rfe}_{\mathbf{v}}) \tag{19}$$
$$(c_{\mathbf{v}} > 0) \rightarrow (f'_{\mathbf{v}} = f_{\mathbf{v}} - \mathbf{Di}_{\mathbf{l},\mathbf{tl}} \cdot \mathbf{Rfc}_{\mathbf{v}}) \tag{20}$$

To model the Petrobras problem, we consider only events associated with the starting of operations. Operation durations will be taken into account via temporal constraints between events (see the next section).

Planning problem definition

In the TECK framework, a planning problem is defined by:

- a planning domain $\mathbf{Pd} = \langle \mathbf{Vs}, \mathbf{Vd}, \mathbf{De}, \mathbf{Et} \rangle$;
- an initial state **I**;
- a temporal horizon **H**;
- a finite set **E** of events;
- a finite set **Cs** of constraints on static variables;
- a finite set **Ce** of constraints on events;
- a finite set **Cd** of constraints on states;
- a criterion **Cr** to be optimized.

Initial state The initial state is defined by a finite set of initial effects. As with event effects, each initial effect is either (1) a functional constraint on a subset of **Vs** $\cup$ **Vd'**, where **Vd'** represents the value of the dynamic variables just after initialization, which expresses the value of a dynamic variable after initialization in case of a piecewise constant evolution, or (2) a functional constraint on a subset of **Vs** $\cup$ **Vd'** $\cup \{t\}$, where t represents the time that went on from initialization, which expresses how a dynamic variable evolves continuously after initialization in case of a non piecewise constant evolution. Obviously, a dynamic variable cannot be the target of several dependencies or effects (unique definition) and the graph induced by dependencies and effects must be acyclic (no loop in definitions). An initial effect must be defined for every dynamic variable that is not the target of a dependency.

In the Petrobras problem, the initial state is defined by the following equations:

$$\forall v \in \mathbf{Ve} : \ (l'_v = \mathbf{Il}_v) \wedge (f'_v = \mathbf{If}_v) \tag{21}$$
$$\forall i \in \mathbf{It} : \ s'_i = \mathbf{W} \tag{22}$$

Temporal horizon The temporal horizon is an interval of reals or integers, defined by an initial time **Ts** and a final time **Te**, equal to $+\infty$ in case of unbounded horizon.

In the Petrobras problem, the temporal horizon is the interval of reals $[0; +\infty[$.

Events Let $\mathbf{Ne} = |\mathbf{E}|$ be the finite number of events to be considered. Each event e is defined by its name $\mathbf{nam}(e)$, its type $\mathbf{typ}(e)$, its presence $\mathbf{pres}(e)$ (events may be either present or absent), its position $\mathbf{pos}(e)$ in the event sequence (all events are totally ordered), its date $\mathbf{dat}(e)$, and the values of its parameters $\mathbf{par}(e)$. Name and type are constant, whereas presence, position, date, and parameter values are variables. Type $\mathbf{typ}(e)$ is an element of **Et**. Presence $\mathbf{pres}(e)$ is a boolean (1 in case of presence and 0 otherwise). Position $\mathbf{pos}(e)$ is an integer between 1 and **Ne**. Date $\mathbf{dat}(e)$ is an element of **H**. $\mathbf{par}(e)$ is a copy of $\mathbf{par}(\mathbf{typ}(e))$: for each parameter, same name, same type, and same domain of value. When an event is absent, its position, date, and parameters take default non significant values.

To model the Petrobras problem, we associate with each vessel $v \in \mathbf{Ve}$ an event $stTr_{v,0}$ of type **StTr** which represents the first transit. Moreover, with each vessel $v \in \mathbf{Ve}$ and each step $j \in [1..\mathbf{Ns}]$, we associate six events $stDo_{v,j}$, $stUl_{v,j}$, $stLo_{v,j}$, $stRf_{v,j}$, $stUd_{v,j}$, and $stTr_{v,j}$ of respective types **StDo**, **StUl**, **StLo**, **StRf**, **StUd**, and **StTr**, which represent respectively the starting of docking, unloading, loading, refueling, undocking, and transit.

Constraints on static variables A constraint on static variables is a constraint on any subset of **Vs**.

In the Petrobras problem, we have a set of constraints on static variables that express that, if a vessel must deliver items, it must visit at least one port or platform:

$$\forall v \in \mathbf{Ve} : \ (\sum_{i \in \mathbf{It}} (v_i = v) > 0) \leftrightarrow (n_v > 0) \tag{23}$$

Constraints on events A constraint on events is a constraint on any subset of **Vs** $\cup$ $(\cup_{e \in \mathbf{E}} (\{\mathbf{pres}(e), \mathbf{pos}(e), \mathbf{dat}(e)\} \cup \mathbf{par}(e)))$. In the Petrobras problem, we have the following set of constraints on events.

Case of the unused vessels; either they do not move, or they transit to another waiting area:

$$\begin{aligned} \forall v \in \mathbf{Ve} : \quad & (n_v = 0) \rightarrow \\ & ((\mathbf{pres}(stTr_{v,0}) = 0) \vee \\ & ((\mathbf{pres}(stTr_{v,0}) = 1) \wedge \\ & (\mathbf{dat}(stTr_{v,0}) = 0) \wedge (\mathbf{v}(stTr_{v,0}) = v) \wedge \\ & (\mathbf{l}(stTr_{v,0}) = \mathbf{Il}_v) \wedge (\mathbf{tl}(stTr_{v,0}) \in \mathbf{Wa}))) \end{aligned} \tag{24}$$

Case of the used vessels; they first transit to a port or platform:

$$\forall v \in \mathbf{Ve}: \quad (n_v > 0) \rightarrow \tag{25}$$
$$((\mathbf{pres}(stTr_{v,0}) = 1) \wedge$$
$$(\mathbf{dat}(stTr_{v,0}) = 0) \wedge (\mathbf{v}(stTr_{v,0}) = v) \wedge$$
$$(\mathbf{l}(stTr_{v,0}) = \mathbf{Il}_v) \wedge (\mathbf{tl}(stTr_{v,0}) \in \mathbf{Po} \cup \mathbf{Pf}))$$

For each vessel, absence of events beyond the number of steps:

$$\forall v \in \mathbf{Ve}, \tag{26}$$
$$\forall j \in [\![1; \mathbf{Ns}]\!]: \quad (j > n_v) \rightarrow$$
$$((\mathbf{pres}(stDo_{v,j}) = 0) \wedge$$
$$(\mathbf{pres}(stUl_{v,j}) = 0) \wedge$$
$$(\mathbf{pres}(stLo_{v,j}) = 0) \wedge$$
$$(\mathbf{pres}(stRf_{v,j}) = 0) \wedge$$
$$(\mathbf{pres}(stUd_{v,j}) = 0) \wedge$$
$$(\mathbf{pres}(stTr_{v,j}) = 0))$$

For each vessel, presence of events of type docking, undocking, and transit until the number of steps; at each step, docking, undocking, and transit are mandatory, but unloading, loading, and refueling are not:

$$\forall v \in \mathbf{Ve}, \tag{27}$$
$$\forall j \in [\![1; \mathbf{Ns}]\!]: \quad (j \leq n_v) \rightarrow$$
$$((\mathbf{pres}(stDo_{v,j}) = 1) \wedge$$
$$(\mathbf{pres}(stUd_{v,j}) = 1) \wedge$$
$$(\mathbf{pres}(stTr_{v,j}) = 1))$$

For each vessel, presence of events of type unloading, loading, and refueling until the number of steps; at least one operation between unloading, loading, and refueling is present (one does not visit a location to do nothing):

$$\forall v \in \mathbf{Ve}, \tag{28}$$
$$\forall j \in [\![1; \mathbf{Ns}]\!]: (j \leq n_v) \rightarrow$$
$$(\mathbf{pres}(stUl_{v,j}) + \mathbf{pres}(stLo_{v,j}) + \mathbf{pres}(stRf_{v,j}) > 0)$$

For each vessel and each present event, event parameters (vessel and location):

$$\forall v \in \mathbf{Ve}, \tag{29}$$
$$\forall j \in [\![1; \mathbf{Ns}]\!]: (j \leq n_v) \rightarrow$$
$$((\mathbf{v}(stDo_{v,j}) = v) \wedge$$
$$((\mathbf{pres}(stUl_{v,j}) = 1) \rightarrow (\mathbf{v}(stUl_{v,j}) = v)) \wedge$$
$$((\mathbf{pres}(stLo_{v,j}) = 1) \rightarrow (\mathbf{v}(stLo_{v,j}) = v)) \wedge$$
$$((\mathbf{pres}(stRf_{v,j}) = 1) \rightarrow (\mathbf{v}(stRf_{v,j}) = v)) \wedge$$
$$(\mathbf{v}(stUd_{v,j}) = v) \wedge$$
$$(\mathbf{v}(stTr_{v,j}) = v))$$

$$\forall v \in \mathbf{Ve}, \tag{30}$$
$$\forall j \in [\![1; \mathbf{Ns}]\!]: (j \leq n_v) \rightarrow$$
$$((\mathbf{l}(stDo_{v,j}) = \mathbf{tl}(stTr_{v,j-1})) \wedge$$
$$((\mathbf{pres}(stUl_{v,j}) = 1) \rightarrow (\mathbf{l}(stUl_{v,j}) = \mathbf{tl}(stTr_{v,j-1})) \wedge$$
$$((\mathbf{pres}(stLo_{v,j}) = 1) \rightarrow (\mathbf{l}(stLo_{v,j}) = \mathbf{tl}(stTr_{v,j-1}))) \wedge$$
$$((\mathbf{pres}(stRf_{v,j}) = 1) \rightarrow (\mathbf{l}(stRf_{v,j}) = \mathbf{tl}(stTr_{v,j-1})) \wedge$$
$$(\mathbf{l}(stUd_{v,j}) = \mathbf{tl}(stTr_{v,j-1})) \wedge$$
$$(\mathbf{l}(stTr_{v,j}) = \mathbf{tl}(stTr_{v,j-1})))$$

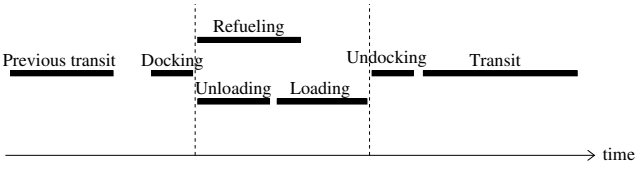


Figure 3: Temporal constraints between operations at any step.

Constraints on event dates: docking must follow transit from the previous location (we assume that one can wait at sea before docking); if present, unloading must immediately follow docking and, if present, loading must immediately follow unloading whereas, if present, refueling must immediately follow docking and be performed concurrently with unloading and loading; undocking is performed as soon as unloading, loading, and refueling are finished in order to limit docking costs; finally, transit to the next location must immediately follow undocking; see Figure 3 for a global view of temporal constraints; we present these constraints only for the starting dates of docking and unloading; the other ones can be written similarly.

Docking starting date, taking into account transit time:

$$\forall v \in \mathbf{Ve}, \tag{31}$$
$$\forall j \in [\![1; \mathbf{Ns}]\!]: \quad (j \leq n_v) \rightarrow$$
$$(\mathbf{dat}(stDo_{v,j}) \geq \mathbf{dat}(stTr_{v,j-1}) +$$
$$\mathbf{Di}_{\mathbf{l}(stTr_{v,j-1}),\mathbf{tl}(stTr_{v,j-1})}/\mathbf{Sp}_v)$$

Unloading starting date, taking into account docking time:

$$\forall v \in \mathbf{Ve}, \tag{32}$$
$$\forall j \in [\![1; \mathbf{Ns}]\!]: \quad ((j \leq n_v) \wedge (\mathbf{pres}(stUl_{v,j}) = 1)) \rightarrow$$
$$(((\mathbf{l}(stDo_{v,j}) \in \mathbf{Po}) \rightarrow$$
$$(\mathbf{dat}(stUl_{v,j}) = \mathbf{dat}(stDo_{v,j}) + \mathbf{Dpo}_v)) \wedge$$
$$((\mathbf{l}(stDo_{v,j}) \in \mathbf{Pf}) \rightarrow$$
$$(\mathbf{dat}(stUl_{v,j}) = \mathbf{dat}(stDo_{v,j}) + \mathbf{Dpf}_v)))$$

Transit target locations; one transits to a waiting area only at the last step:

$$\forall v \in \mathbf{Ve}, \forall j \in [\![1; \mathbf{Ns}]\!]: \quad (j < n_v) \rightarrow \tag{33}$$
$$(\mathbf{tl}(stTr_{v,j}) \in \mathbf{Po} \cup \mathbf{Pf})$$
$$\forall v \in \mathbf{Ve}: \quad \mathbf{tl}(stTr_{v,n_v}) \in \mathbf{Wa} \tag{34}$$

At any time, no more than one vessel docked at a platform:

$$\forall v, \forall v' \in \mathbf{Ve} \,|\, v \neq v', \tag{35}$$
$$\forall j, \forall j' \in [\![1; \mathbf{Ns}]\!]: \quad ((j \leq n_v) \wedge (j' \leq n_{v'}) \wedge$$
$$(\mathbf{l}(stDo_{v,j}) \in \mathbf{Pf}) \wedge$$
$$(\mathbf{l}(stDo_{v',j'}) = \mathbf{l}(stDo_{v,j}))) \rightarrow$$
$$((\mathbf{dat}(stTr_{v,j}) \leq \mathbf{dat}(stDo_{v',j'})) \vee$$
$$(\mathbf{dat}(stTr_{v',j'}) \leq \mathbf{dat}(stDo_{v,j})))$$

At any time, no more than two vessels docked at a port:

$$\forall v \in \mathbf{Ve}, \quad (36)$$
$$\forall j \in [\![1; \mathbf{Ns}]\!] : ((j \leq n_v) \wedge (\mathbf{l}(stDo_{v,j}) \in \mathbf{Po})) \rightarrow$$
$$((\sum_{v' \in \mathbf{Ve}, j' \in [\![1;\mathbf{Ns}]\!] \mid v \neq v'} ((j' \leq n_{v'}) \wedge$$
$$(\mathbf{l}(stDo_{v',j'}) = \mathbf{l}(stDo_{v,j})) \wedge$$
$$(\mathbf{dat}(stDo_{v',j'}) \leq \mathbf{dat}(stDo_{v,j}) < \mathbf{dat}(stTr_{v',j'}))))$$
$$\leq 1)$$

Constraints on states Let **Ss** be the finite sequence of system states, function of the finite sequence of events (see the following section). For each dynamic variable $v \in \mathbf{Vd}$ and each state $s \in \mathbf{Ss}$, let $\mathbf{val}(v, s)$ be the value of v in s. A constraint on states is a constraint on any subset of $\mathbf{Vs} \cup \{\mathbf{val}(v, s) | s \in \mathbf{Ss}, v \in \mathbf{Vd}\}$. However, to model the Petrobras problem, we do not need to use any constraint on states.

Criterion The criterion **Cr** is a function of any subset of $\mathbf{Vs} \cup \{\mathbf{val}(v, s) | s \in \mathbf{Ss}, v \in \mathbf{Vd}\} \cup (\cup_{e \in \mathbf{E}}(\{\mathbf{pres}(e), \mathbf{pos}(e), \mathbf{dat}(e)\} \cup \mathbf{par}(e)))$ in a totally ordered set. In the Petrobras problem, the criterion (to be minimized) may be any combination of three criteria, which are all only function of events.

Consumed fuel quantity:

$$\sum_{v \in \mathbf{Ve}} \sum_{j=1}^{\mathbf{Ns}} \mathbf{pres}(stRf_{v,j}) \cdot \mathbf{f}(stRf_{v,j}) \quad (37)$$

Makespan (date of arrival of the last vessel at a waiting area):

$$\max_{v \in \mathbf{Ve}} \quad \mathbf{pres}(stTr_{v,n_v}) \cdot \quad (38)$$
$$(\mathbf{dat}(stTr_{v,n_v}) + \mathbf{Di}_{\mathbf{l}(stTr_{v,n_v}),\mathbf{tl}(stTr_{v,n_v})}/\mathbf{Sp}_v)$$

Docking cost:

$$\sum_{v \in \mathbf{Ve}} \sum_{j=1}^{\mathbf{Ns}} \quad \mathbf{Co} \cdot (j \leq n_v) \cdot (\mathbf{l}(stDo_{v,j}) \in \mathbf{Po}) \cdot \quad (39)$$
$$(\mathbf{dat}(stTr_{v,j}) - \mathbf{dat}(stDo_{v,j}))$$

Planning problem solution

In this section, we define the framework semantics, that is what an assignment, a solution, and an optimal solution of a planning problem are in TECK.

Assignment An assignment **A** of a planning problem $\mathbf{P} = \langle\langle \mathbf{Vs}, \mathbf{Vd}, \mathbf{De}, \mathbf{Et}\rangle, \mathbf{I}, \mathbf{H}, \mathbf{E}, \mathbf{Cs}, \mathbf{Ce}, \mathbf{Cd}, \mathbf{Cr}\rangle$ is a pair made of:

- an assignment **As** of the static variables in **Vs**;
- an assignment **Ae** of the events in **E**.

An assignment **As** of **Vs** assigns to every static variable $v \in \mathbf{Vs}$ a value from its domain. An assignment **Ae** of **E** assigns, for every event $e \in \mathbf{E}$, a value to $\mathbf{pres}(e)$ (0 or 1). If $\mathbf{pres}(e) = 1$ (present event), it assigns to $\mathbf{pos}(e)$, to $\mathbf{dat}(e)$, and to every parameter in $\mathbf{par}(e)$ a value from their respective domains.

Sequence of events We assume the following default constraints on event presences, positions, and dates:

Total order on present events:

$$\forall e \in \mathbf{E} : \quad 1 \leq \mathbf{pos}(e) \leq \sum_{e \in \mathbf{E}} \mathbf{pres}(e) \quad (40)$$
$$\forall e, e' \in \mathbf{E} \,|\, e \neq e' : \quad \mathbf{pos}(e) \neq \mathbf{pos}(e') \quad (41)$$

Consistency of dates with regard to positions:

$$\forall e, e' \in \mathbf{E} \,|\, e \neq e' : \; (\mathbf{pos}(e) < \mathbf{pos}(e')) \rightarrow (\mathbf{dat}(e) \leq \mathbf{dat}(e')) \quad (42)$$

As it is assumed in basic modeling frameworks for discrete event dynamic systems, such as automata or Petri nets, and differently from what is often assumed in classical planning, events may occur at exactly the same time, but are totally ordered.

As a consequence, an assignment **Ae** of **E** induces a finite sequence **Es** of present events, of the form $[e_1, \ldots, e_i, \ldots, e_{\mathbf{ne}}]$, where $\mathbf{ne} = \sum_{e \in \mathbf{E}} \mathbf{pres}(e)$ is the number of present events.

Sequence of states Due to initialization and event effects, an assignment **A** of a planning problem induces a finite sequence **Ss** of states, of the form $[s'_0, s_1, s'_1, \ldots, s_i, s'_i, \ldots, s_{\mathbf{ne}}, s'_{\mathbf{ne}}]$ if $\mathbf{Te} = +\infty$, and of the form $[s'_0, s_1, s'_1, \ldots, s_i, s'_i, \ldots, s_{\mathbf{ne}}, s'_{\mathbf{ne}}, s_{\mathbf{ne}+1}]$ otherwise. s'_0 is the state after initialization. For each $i \in [\![1; \mathbf{ne}]\!]$, s_i is the state just before event e_i and s'_i the state just after. If $\mathbf{Te} \neq +\infty$, $s_{\mathbf{ne}+1}$ is the state at the end of the temporal horizon. Distinguishing the state before and after an event is necessary in case of non piecewise constant evolutions of dynamic variables.

Solution An assignment $\mathbf{A} = \langle \mathbf{As}, \mathbf{Ae}\rangle$ of a planning problem $\mathbf{P} = \langle\langle \mathbf{Vs}, \mathbf{Vd}, \mathbf{De}, \mathbf{Et}\rangle, \mathbf{I}, \mathbf{H}, \mathbf{E}, \mathbf{Cs}, \mathbf{Ce}, \mathbf{Cd}, \mathbf{Cr}\rangle$ is solution if and only if:

- **As** satisfies all the constraints in **Cs**;
- **Ae** satisfies all the constraints in **Ce**, including the default constraints on event presences, positions, and dates;
- the sequence **Ss** of states induced by **A** satisfies all the constraints on domains of value of dynamic variables, all the event preconditions, and all the constraints in **Cd**.

Optimal solution A solution is optimal if and only if there is no other solution inducing a better value of **Cr**.

Complexity and subsumed frameworks

It can be shown that the decision problem associated with the TECK framework (existence or not of a solution plan) is NP-complete. It belongs to NP because checking that a given plan is solution is obviously polynomial. It is NP-complete because the CSP problem is NP-complete and any CSP p can be transformed into a TECK problem with a static part which is a copy of p and an empty dynamic part.

It can be shown that the TECK framework allows existing problems and frameworks to be modeled, such as for example:

- the RCPSP problem (*Resource Constrained Project Scheduling Problem* (Baptiste, Pape, and Nuijten 2001));
- the HTN framework (*Hierarchical Task Networks* (Nau et al. 2003)), if we assume that the tree of possible decompositions is finite;
- the STRIPS framework (Fikes and Nilsson 1971), provided that the number of actions in a plan be bounded (one does not require that the number of actions be fixed; we only need that a maximum number be defined).

Proofs of subsumption are omitted for space reasons.

Solving issues

Although this paper focuses on modeling issues, we cannot ignore that any modeling framework is the result of a difficult tradeoff between modeling issues (one wants to model as precisely as possible physical systems and user requirements) and solving issues (one wants to maintain the solving complexity as low as possible).

One can first observe that any TECK problem can be transformed into a CSP problem, more precisely into a form of dynamic CSP problem (Mittal and Falkenhainer 1990), because of the presence or absence of events. So, any constraint solver can be used to solve planning and scheduling problems expressed in the TECK framework.

This is an option, but not necessarily the best one, because it may not correctly exploit the particular structure of TECK problems. This why we are currently working on efficient local search algorithms which could be used to perform offline or online planning. Such algorithms, based on additions or removals of events and changes in event parameters, could be non chronological (choices not performed following a chronological order) and be consequently less blind and more heurisitically informed than the classical chronological planning algorithms.

Discussion and related frameworks

If we look at the TECK framework, we can first notice that it allows two kinds of constraints on plans to be expressed: (i) event preconditions and effects which are Markovian because they connect only the current state, the current event, and the next state, and (ii) constraints on events which may be non Markovian because they can connect any subset of events in the sequence of events. The first kind of constraints is usual in planning and, more generally, in the modeling of discrete event dynamic systems. The second one is less usual in planning, although it is very useful to express physical constraints, user requirements, or heuristics on plans. In the modeling of discrete event dynamic systems, temporal logics (Emerson 1990) is often used to model this kind of constraints and several works already proposed to introduce temporal logics in planning in order to express constraints on plans (Bacchus and Kabanza 2000; Kvarnström and Doherty 2001). In the TECK framework, this kind of constraints is expressed by using constraints on event parameters. Whereas event preconditions allow "what can be" to be expressed, constraints on event parameters also allow "what must be" to be expressed (for example events that must be present, possibly due to the presence of other events).

The introduction of static variables and constraints is another originality of the TECK framework. They are very useful when modeling planning problems, as it is clear in the Petrobras example. They are strangely unusual in planning, although the idea is already present in the discrete event dynamic systems community (Cimatti, Palopoli, and Ramadian 2008) with the notion of parametric timed automata.

On the other hand, it is clear that the need to associate a maximum number of events with each event type (we assume a finite number of events, present or not, each one with a fixed type) is the main limitation of the TECK framework. However, our experience in the modeling of many planning and scheduling problems in the aerospace domain and beyond allows us to claim that, in many real-world problems, it is possible to define reasonable bounds on the number of events.

With regard to classical scheduling (Baptiste, Pape, and Nuijten 2001) and to constraint programming tools that have been built to deal with scheduling problems (IBM ILOG), the TECK framework adds states, events, and state transitions. However, (Serra, Nishioka, and Marcellino 2012) uses IBM ILOG to model a Petrobras planning problem which is different from, but close to the 2012 ICKEPS competition challenge, used in this paper.

With regard to classical planning (Ghallab, Nau, and Traverso 2004) and usual planning languages (Fox and Long 2003), it adds static variables and constraints, and constraints on events. With regard to SAT or CSP-based planning approaches (Kautz and Selman 1992), it allows the presence and the position of events to be variable: for example, in the Petrobras problem, for each vessel v, at any step beyond the variable number n_v of steps, all events are absent and, at any step until n_v, docking, undocking, and transit events are present, but unloading, loading, and refueling events may be present or not (see Eqs. 26 and 27); moreover, the dates and thus the relative positions of events on different vessels are not predetermined. Finally, with regard to planning systems that are based on the common notion of timelines such as IxTeT, Europa, or APSI (Ghallab and Laruelle 1994; Frank and Jónsson 2003; Fratini, Pecora, and Cesta 2008), it offers simplicity and clear semantics.

The TECK framework inherits some features from frameworks we previously proposed. However, the CNT framework (*Constraint Networks on Timelines* (Verfaillie, Pralet, and Lemaître 2010)), based on the notion of dynamic constraint which connects a variable number of variables, was certainly too general. Moreover, it did not make any distinction between states and events. In the CTA framework (*Constraint Timed Automata* (Pralet and Verfaillie 2012)), the so-called activation constraints (events activating other events) were not clearly formalized.

Some choices may be arguable in the definition of the TECK framework, such as for example the choice of event rather than durative action as the elementary building block in the framework. We think that such a choice makes for the maximum flexibility (it is always preferable to use the most

basic elements as elementary building blocks) and that it does not prevent from building more complex blocks by using elementary ones (a durative action can be easily defined as a pair of events with the following constraints: presence (resp. absence) of action implies presence (resp. absence) of both events and presence implies a temporal distance between starting and ending events).

Finally, we think that the case where dynamic variables represent (numeric) resource levels, with events which add (subtract) quantities to (from) resource levels or slopes, represents a very important sub-framework, for which more efficient algorithms can be designed. This is what we started doing with (Pralet and Verfaillie 2013).

Conclusion and perspectives

In this paper we proposed a framework we think to be adequate for the modeling of the real-world planning and scheduling problems we encountered in the aerospace domain and beyond. Based on the notions of timelines, events, and constraints, it allows knowledge about events (event presence and parameters), time (event positions and dates), state (state variable evolutions), and resources (discrete or continuous resource evolutions) to be expressed. The next steps will consist in:

- verifying that many diverse planning and scheduling problems can effectively be modeled in the TECK framework;
- defining first useful constructs on top of the TECK basic framework, such as durative actions, temporal and resource constraints;
- exploring several solving alternatives, inspired from the background in combinatorial optimization, but exploiting the specific structure of TECK problems, with the objective to get efficient either optimal, or approximate anytime algorithms.

References

Bacchus, F., and Kabanza, F. 2000. Using Temporal Logics to Express Search Control Knowledge for Planning. *Artificial Intelligence* 16:123–191.

Baptiste, P.; Pape, C. L.; and Nuijten, W. 2001. *Constraint-based Scheduling: Applying Constraint Programming to Scheduling Problems*. Kluwer Academic Publishers.

Cassandras, C., and Lafortune, S. 2008. *Introduction to Discrete Event Systems*. Springer.

Cimatti, A.; Palopoli, L.; and Ramadian, Y. 2008. Symbolic Computation of Schedulability Regions using Parametric Timed Automata. In *Proc. of RTSS-08*, 80–89.

Emerson, E. 1990. Temporal and Modal Logic. In van Leeuwen, J., ed., *Handbook of Theoretical Computer Science, Volume B:Formal Models and Semantics*. Elsevier. 995–1072.

Fikes, R., and Nilsson, N. 1971. STRIPS: a New Approach to the Application of Theorem Proving. *Artificial Intelligence* 2:189–208.

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.

Frank, J., and Jónsson, A. 2003. Constraint-Based Attribute and Interval Planning. *Constraints* 8(4):339–364.

Fratini, S.; Pecora, F.; and Cesta, A. 2008. Unifying Planning and Scheduling as Timelines in a Component-based Perspective. *Archives of Control Sciences* 18(2):5–45.

Ghallab, M., and Laruelle, H. 1994. Representation and Control in IxTeT: a Temporal Planner. In *Proc. of AIPS-94*, 61–67.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

IBM ILOG. IBM ILOG CPLEX Optimization Studio. http://www-01.ibm.com/software/integration/optimization/cplex-optimization-studio/.

Kautz, H., and Selman, B. 1992. Planning as Satisfiability. In *Proc. of ECAI-92*, 359–363.

Kvarnström, J., and Doherty, P. 2001. TALplanner: A Temporal Logic Based Forward Chaining Planner. *Annals of Mathematics and Artificial Intelligence* 30:119–169.

Mittal, S., and Falkenhainer, B. 1990. Dynamic Constraint Satisfaction Problems. In *Proc. of AAAI-90*, 25–32.

Nau, D.; Au, T.; Ilghami, O.; Kuter, U.; Murdock, W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research* 20:379–404.

Pralet, C., and Verfaillie, G. 2013. Dynamic Online Planning and Scheduling using a Static Invariant-based Evaluation Model. In *Proc. of ICAPS-13*.

Pralet, C., and Verfaillle, G. 2012. Combining Static and Dynamic Models for Boosting Forward Planning. In *Proc. of CP-AI-OR-12*, 322–338.

Serra, T.; Nishioka, G.; and Marcellino, F. 2012. The Offshore Resources Scheduling Problem: Detailing a Constraint Programming Approach. In *Proc. of CP-12*, 823–839.

Toropila, D.; Dvoràk, F.; Trunda, O.; Hanes, M.; and Bartàk, R. 2012. Three Approaches to Solve the Petrobras Challenge. In *Proc. of ICTAI-12*, 191–198.

Vaquero, T.; Costa, G.; Tonidandel, F.; Igreja, H.; Silva, J.; and Beck, C. 2012. Planning and Scheduling Ship Operations on Petroleum Ports and Platforms. In *Proc. of the ICAPS-12 Workshop on "Scheduling and Planning Applications" (SPARK-12)*.

Verfaillie, G.; Pralet, C.; and Lemaître, M. 2010. How to Model Planning and Scheduling Problems using Timelines. *The Knowledge Engineering Review* 25(3):319–336.

Using Static Graphs in Planning Domains to Understand Domain Dynamics*

Gerhard Wickler
Artificial Intelligence Applications Institute
University of Edinburgh
Edinburgh, Scotland

Abstract

This paper describes a method for analyzing STRIPS-like planning domains by identifying static graphs that are implicit in the set of operators defining a planning domain. A graph consisting of nodes and possibly directed edges is a common way to construct representations for many problems, including computational problems and problems of reasoning about action. Furthermore, there may be objects or properties related to the nodes of such a graph that may be modified by the operators in ways restricted by the graph. The formal definition of *shift operators* over *static graphs* as a domain feature is the main contribution of this paper.

Such an analysis could be used for verification and validation of the planning domain when it is presented to the domain author who may or may not agree with the result of the analysis. The method described relies on domain features that can also be extracted automatically, and it works on domains rather than problems, which means the result is problem-independent. However, if problems are given further analysis may be perfomed. The method has been evaluated using a small number of planning domains drawn from the international planning competition.

Introduction

Specifying a planning domain and a planning problem in a formal description language defines a search space that can be traversed by a state-space planner to find a solution plan. It is well known that this specification process, also known as *problem formulation* [Russell and Norvig, 2003], is essential for enabling efficient problem-solving though search [Amarel, 1968]. However, the most efficient representation is often hard to understand, verify and maintain. One way to ensure the correctness of a problem specification is to enforce consistency. Obviously this does not guarantee correctness, but it may highlight problems to the knowledge engineer.

Consistency can be enforced if the representation contains some redundancy. We have described a set of domain features [Wickler, 2011] that can be used to assist during the verification and validation of planning domains by exploiting information implicit in the planning domain. The features are: *domain types*, *relation fluency*, *inconsistent effects* and *reversible actions*. These features can be efficiently and automatically extracted from a planning domain. If the planning domain also contains an explicit specification of these features then these represent redundant information that can be compared and used to enforce consistency, by which we mean that the feature values specified by the knowledge engineer should be the same as the ones that can be automatically extracted. Hopefully, this will lead to a planning domain that is in line with what the knowledge engineer intended to represent.

*This work has been sponsored by the Engineering and Physical Sciences Research Council (UK) under grant number EP/J011800/1. The University of Edinburgh and research sponsors are authorized to reproduce and distribute reprints and online copies for their purposes notwithstanding any copyright annotation hereon.

Knowledge Engineering

Knowledge engineering (KE) for planning domains is a topic that has received relatively little attention in the AI planning community and much work is still needed in the area. However, with planners becoming more efficient, the problems they can solve in reasonable time are becoming larger, and so the planning domains may be larger and more complex to engineer.

KE Methodology

The design process for planning domain models is similar to the knowledge engineering process followed for other types of software models. The baseline phases for planning domains described in [Vaquero *et al.*, 2011] are the following:

1. requirements specification
2. knowledge modeling
3. model analysis
4. deployment to planner
5. plan synthesis
6. plan analysis and post-design

The work described in this paper focusses on the third phase, model analysis, which includes the verification and validation of the planning domain model.

One of the most advanced systems in this area is GIPO [Simpson, 2007] which includes support for the complete KE process, including model analysis. GIPO goes well beyond simple syntactic checks, verifying the consistent use

of a type hierarchy and predicate templates, as well as more advanced features such as invariants. It also includes a tool that visualizes domain dynamics for the knowledge engineer. Another graphical system supporting KE for planning domains is itSIMPLE [Vaquero *et al.*, 2007]. Static support for model analysis is mostly visual, using multiple views which can also be interpreted as a kind of redundancy.

The target output in most KE systems for planning is the Planning Domain Definition Language (PDDL) [Fox and Long, 2003], which has become a de-facto standard for specifying STRIPS-like planning domains and problems with various extensions. PDDL allows for the specification of some auxiliary information about a domain, such as types, but this information is optional.

Domain Features

Amongst the features mentioned above, domain types have received significant attention in the planning literature. A rigorous method for problem formulation in the case of planning domains was presented in [McCluskey and Porteous, 1997]. In the second step of their methodology types are extracted from an informal description of a planning domain. Types have been used as a basic domain feature in TIM [Fox and Long, 1998]. Their approach exploits functional equivalence of objects to derive a hierarchical type structure. This work has later been extended to infer generic types such as mobiles and resources that can be exploited to optimize plan search [Coles and Smith, 2006].

The distinction between rigid and fluent relations [Ghallab *et al.*, 2004] is common in AI planning and will be discussed below. Inconsistent effects of different actions are exploited in the GraphPlan algorithm [Blum and Furst, 1995] to define the mutex relation. However, this is applied to pairs of actions (i.e. fully ground instances of operators) rather than operators. Reversible actions, as a domain feature, are not related to regression of goals, meaning this feature is unrelated to the direction of search (forward from the initial state or regressing backwards from the goal). A formal treatment of reversibility of actions (or operators) does not appear to feature much in the AI planning literature, despite the fact that reversible actions are common in planning domains. However, in generic search problems they are a common technique used to prune search trees [Russell and Norvig, 2003].

Preprocessing of planning domains is a technique that has been used to speed up the planning process [Dawson and Siklossy, 1977]. Perhaps the most common preprocessing step is the translation of the STRIPS (function-free, first-order) representation into a propositional representation. An informal algorithm for this is described in [Ghallab *et al.*, 2004, section 2.6]. A conceptual flaw in this algorithm (highlighted by the analysis of inconsistent effects) was described in [Wickler, 2011].

Static and Fluent Relations

A domain feature that is useful for the analysis of planning domains concerns the relations that are used in the definition of the operators. The set of predicates used here can be divided into static (or rigid) relations and fluent (or dynamic) relations, depending on whether atoms using this predicate can change their truth value from state to state.

Definition 1 (static/fluent relation) *Let $O = \{O_1, \ldots, O_{n(O)}\}$ be a set of operators and let $P = \{P_1, \ldots, P_{n(P)}\}$ be a set of all the predicate symbols that occur in these operators. A predicate $P_i \in P$ is **fluent** iff there is an operator $O_j \in O$ that has an effect that uses the predicate P_i. Otherwise the predicate is **static**.*

The algorithm for computing the sets of fluent and static predicate symbols is trivial and hence, we will not list it here.

There are at least two ways in which this information can be used in the validation of planning problems. Firstly, if the domain definition language allowed the domain author to specify whether a relation is static or fluent then this could be verified when the domain is parsed. This might highlight problems with the domain. This use requires only a planning domain to be provided. Secondly, if a planning problem that uses additional relations is given, these could be highlighted or simply removed from the initial state.

Type Information

Many planning domains include explicit type information. In PDDL the `:typing` requirement allows the specification of typed variables in predicate and operator declarations. In problem specifications, it allows the assignment of constants or objects to types. If nothing else, typing tends to greatly increase the readability of a planning domain. However, it is not necessary for most planning algorithms to work, and the analysis techniques described here require neither a problem nor explicit types to be given.

Type Consistency The simplest kind of type system often used in planning is one in which the set of all constants C used in the planning domain and problem is divided into disjoint types T. That is, each type corresponds to a subset of all constants and each constant belongs to exactly one type. This is the kind of type system we will look at here.

Definition 2 (type partition) *A **type partition** $\mathcal{P}$ is a tuple $\langle C, T, \tau \rangle$ where:*

- *C is a finite set of $n(C) \geq 1$ constant symbols $C = \{c_1, \ldots, c_{n(C)}\}$,*
- *T is a set of $n(T) \leq n(C)$ types $T = \{t_1, \ldots, t_{n(T)}\}$, and*
- *$\tau : C \rightarrow T$ is a function defining the type of a given constant.*

A type partition divides the set of all constants that may occur in a planning problem into a set of equivalence classes. However, constants are only necessary to define the types' extension. As we shall show, the intension is implicit in the operators that constitute the planning domain alone. The availability of a type partition can be used to limit the space of world states that may be searched by a planner. In general, a world state in a planning domain can be any subset of the powerset of the set of ground atoms over predicates P with arguments from C.

Definition 3 (type function) *Let $P = \{P_1, \ldots, P_{n(P)}\}$ be a set of $n(P)$ predicate symbols with associated arities $a(P_i)$ and let $T = \{t_1, \ldots, t_{n(T)}\}$ be a set of types. A **type function** for predicates is a function*

$$arg_P : P \times \mathbb{N} \to T$$

which, for a given predicate symbol P_i and argument number $1 \leq k \leq a(P_i)$ gives the type $arg_P(P_i, k) \in T$ of that argument position.

This is the kind of type specification we find in PDDL domain definitions as part of the definition of predicates used in the domain, provided that the typing extension of PDDL is used. The type function is defined by enumerating the types for all the arguments of each predicate.

Definition 4 (type consistency) *Let $\langle C, T, \tau \rangle$ be a type partition. Let $P_i \in P$ be a predicate symbol and let $c_1, \ldots, c_{a(P_i)} \in C$ be constant symbols. The ground first-order atom $P_i(c_1, \ldots, c_{a(P_i)})$ is **type consistent** iff $\tau(c_k) = arg_P(P_i, k)$. A world state is **type consistent** iff all its members are type consistent.*

Thus, for a given predicate P_i there are $|C|^{a(P_i)}$ possible ground instances that may occur in world states. Clearly, the set of type consistent world states is a subset of the set of all world states. The availability of a set of types can also be used to limit the actions considered by a planner.

Definition 5 (type function) *Let $O = \{O_1, \ldots, O_{n(O)}\}$ be a set of $n(O)$ operator names with associated arities $a(O_i)$ and let $T = \{t_1, \ldots, t_{n(T)}\}$ be a set of types. A **type function** for operators is a function*

$$arg_O : O \times \mathbb{N} \to T$$

which, for a given operator symbol O_i and argument number $1 \leq k \leq a(O_i)$ gives the type $arg_O(O_i, k) \in T$ of that argument position.

Again, this is exactly the kind of type specification that may be provided in PDDL where the function is defined by enumeration of all the arguments with their types for each operator definition.

Definition 6 (type consistency) *Let $\langle C, T, \tau \rangle$ be a type partition. Let $O_i(v_1, \ldots, v_{a(O_i)})$ be a* STRIPS *operator defined over variables $v_1, \ldots, v_{a(O_i)}$ with preconditions precs(O_i) and effects effects(O_i), where each precondition/effect has the form $P_j(v_{P_j,1}, \ldots, v_{P_j,a(P_j)})$ or $\neg P_j(v_{P_j,1}, \ldots, v_{P_j,a(P_j)})$ for some predicate $P_j \in P$. The operator O_i is **type consistent** iff:*

- *all the operator variables $v_1, \ldots, v_{a(O_i)}$ are mentioned in the positive preconditions of the operator, and*
- *if $v_k = v_{P_j,l}$, i.e. the kth argument variable of the operator is the same as the lth argument variable of a precondition or effect, then the types must also be the same: $arg_O(O_i, k) = arg_P(P_j, l)$.*

The first condition is often required only implicitly (see [Ghallab *et al.*, 2004, chapter 4]) to avoid the complication of "lifted" search in forward search.

Derived Types The above definitions assume that there is an underlying type system that has been used to define the planning domain (and problems) in a consistent fashion. We shall continue to assume that such a type system exists, but it may not have been explicitly specified in the PDDL definition of the domain. We shall now define a type system that is derived from the operator descriptions in the planning domain.

Definition 7 (type name) *Let $O = \{O_1, \ldots, O_{n(O)}\}$ be a set of* STRIPS *operators. Let P be the set of all the predicate symbols used in all the operators. A **type name** is a pair $\langle N, k \rangle \in (P \cup O) \times \mathbb{N}$.*

A type name (a predicate or operator name and an argument number) can be used to refer to a type in a derived type system. There usually are multiple names to refer to the same type. The basic idea behind the derived types is to partition the set of all type names into equivalence classes. If a planning problem is given, the constants occurring in such a problem can then be assigned to the different equivalence classes, thus treating each equivalence class as a type.

Definition 8 (O-type) *Let $O = \{O_1, \ldots, O_{n(O)}\}$ be a set of* STRIPS *operators over operator variables $v_1, \ldots, v_{a(O_i)}$ with conds(O_i) = precs$(O_i) \cup$ effects(O_i) and all operator variables mentioned in the positive preconditions. Let P be the set of all the predicate symbols used in all the operators. An **O-type** is a set of type names. Two type names $\langle N_1, i_1 \rangle$ and $\langle N_2, i_2 \rangle$ are in the same O-type, denoted $\langle N_1, i_1 \rangle \equiv_O \langle N_2, i_2 \rangle$, iff one of the following holds:*

- *$N_1(v_{1,1}, \ldots, v_{1,a(N_1)})$ is an operator with precondition or effect $N_2(v_{2,1}, \ldots, v_{2,a(N_2)}) \in$conds$(N_1)$ which share a specific variable: $v_{1,i_1} = v_{2,i_2}$,*
- *$N_2(v_{2,1}, \ldots, v_{2,a(N_2)})$ is an operator with precondition or effect $N_1(v_{1,1}, \ldots, v_{1,a(N_1)}) \in$conds$(N_2)$ which share a specific variable: $v_{1,i_1} = v_{2,i_2}$, or*
- *there is a type name $\langle N, j \rangle$ such that $\langle N, j \rangle \equiv_O \langle N_1, i_1 \rangle$ and $\langle N, j \rangle \equiv_O \langle N_2, i_2 \rangle$.*

Definition 9 (O-type partition) *Let (s_i, g, O) be a* STRIPS *planning problem. Let C be the set of all constants used in s_i. Let $T = \{t_1, \ldots, t_{n(T)}\}$ be the set of O-types derived from the operators in O. Then we can define the function $\tau : C \to T$ as follows:*
$\tau(c) = t_i : \forall R(c_1, \ldots, c_{a(R)}) \in s_i : (c_j = c) \Rightarrow \langle R, j \rangle \in t_i$

Note that $\tau(c)$ is not necessarily well-defined for every constant mentioned in the initial state, e.g. if a constant is used in two relations that would indicate different derived types (which rely only on the operator descriptions). In this case the O-type partition cannot be used as defined above. However, if appropriate unions of O-types are taken then this results in a new type partition for which $\tau(c)$ is defined. In the worst case this will lead to a type partition consisting of a single type. Given that this approach is always possible, we shall now assume that $\tau(c)$ is always defined.

Definition 10 *Let $T = \{t_1, \ldots, t_{n(T)}\}$ be the set of O-types for a given set of operators O and let $P = \{P_1, \ldots, P_{n(P)}\}$ be the predicates that occur on operators from O. We can easily define type functions arg_P and arg_O as follows:*

$$arg_P(P_i, k) = t_i : \langle P_i, k\rangle \in t_i \text{ and}$$
$$arg_O(O_i, k) = t_i : \langle O_i, k\rangle \in t_i$$

Proposition 1 *Let (s_i, g, O) be a* STRIPS *planning problem and let $\langle C, T, \tau\rangle$ be the O-type partition derived from this problem. Then every state that is reachable from the initial state s_i is type consistent.*

To show this we first show that the initial state is type consistent. Since the definition of τ is based on the argument positions in which they occur in the initial state, this follows trivially.

Next we need to show that every action that is an instance of an operator in O is type consistent. All operator variables must be mentioned in the positive preconditions according to the definition of an O-type. Furthermore, if a precondition or effect share a variable with the operator, these must have the same type since $\equiv_O$ puts them into the same equivalence class.

Finally we can show that, if action a is applicable in a type consistent state s, the resulting state $\gamma(s, a)$ must also be type consistent. Every atom must come either from s in which case it must be type consistent, or it comes from a positive effect, which, given the type consistency of a means it must also be type consistent. ■

This shows that the type system derived from the operator definitions is indeed useful as it creates a state space of type consistent states. Note that the definition of the type system does not require the initial state that is part of the problem, but uses only the operators. The initial state is only necessary for the proposition, as is spans a state-space about which we can make a claim in the above proposition.

Advanced Features

In this section we shall define some more abstract features that can be used to achieve an understanding of a planning domain that is, perhaps, more human-like. The formal definition of these features represent the main contribution of this paper.

Static Graphs

We have now formally defined a type system that we can derive from a planning domain, and we have defined what it means for a predicate used in a planning domain to be static. Together, these two features form the basis for the static graphs that we will identify in a given planning domain. We shall use the dock worker robots (DWR) domain defined in [Ghallab *et al.*, 2004] to illustrate the concepts defined in this section.

Many planning domains fall into the general category of transportation domains. That is, they define a network of locations that are connected by paths that can be traversed by vehicles. Often there are other types of movable objects that need to be brought into a given configuration, defined by the goal of a planning problem over such a domain. The path network that is part of the planning problem is usually fixed, meaning it cannot be changed by actions in the domain. It forms a graph that can be analyzed independent from the state of the world, i.e. the location of vehicles and other movable objects.

The following definition attempts to capture this notion:

Definition 11 (static graph relation) *Let $O = \{O_1, \ldots, O_{n(O)}\}$ be a set of* STRIPS *operators. Let $P = \{P_1, \ldots, P_{n(P)}\}$ be a set of $n(P)$ predicate symbols with associated arities $a(P_i)$ used in all the operators. Then we say that P_i is a* ***static graph relation*** *if and only if:*

- *P_i is a static relation;*
- *P_i is a binary relation: $a(P_i) = 2$; and*
- *the two arguments of P_i are of the same (derived) type: $arg_{P_i}(1) = arg_{P_i}(2)$.*

Note that this definition relies only on information that can be computed from the planning domain specification, i.e. no planning problem and no hint from the knowledge engineer as to what relation might define a graph is required. In the DWR domain, the only relation that satisfies these conditions is the `adjacent`-relation, and this the relation that defines the network of locations between which the robots can move. Given a static graph relation, it is straight-forward to define the graph that is defined by this relation.

Definition 12 (static graph) *Let (s_i, g, O) be a* STRIPS *planning problem and let $\langle C, T, \tau\rangle$ be the O-type partition derived from this problem. Let P_i be a static graph relation for the set of operators O. Then P_i defines a* ***static graph*** *$G_{P_i} = (V, E)$ consisting of nodes (vertices) V and directed edges E, where:*

- *$V = \{c \in C | \tau(c) = arg_{P_i}(1) = arg_{P_i}(2)\}$; and*
- *$E = \{(c, c') | P_i(c, c') \in s_i\}$.*

Thus, in the DWR example, the `adjacent`-relation defines a graph that consists of nodes that correspond to instances of the type `location` and the edges are defined by the initial state.

Note that the definition of a static graph applies to a planning problem. However, the analysis which relations can define a static graph is problem independent, and therefore does not have to be re-computed for each problem. It can be computed from the set of operators defining the domain.

To exploit this analysis for verification and validation, the knowledge engineer would have to explicitly specify which relations are meant to represent static graphs. Furthermore, a relation specified as a static graph relation could have more properties that can be easily verified: it could be reflexive, symmetric, and/or transitive, or it could even be specified to represent a tree structure.

Finally, note that the idea of static graphs is based on location networks as found in the DWR domain, but static graph could represent many things, and thus this technique is more general and not only applicable to transportation domains.

Node-Fixed Types

Given a static graph, it is often possible to identify other types that represent objects or properties in the domain which have a fixed relation to nodes in a static graph.

Definition 13 (node-fixed type) *Let $O = \{O_1, \ldots, O_{n(O)}\}$ be a set of* STRIPS *operators and P_i be a static graph relation for this domain, where t_i is the*

node type for this relation. A type $t_j \neq t_i$ represents a ***node-fixed type*** *if and only if:*

- *there exists a static binary relation P_j; and*
- *P_j has one argument of type t_i and the other of type t_j.*

The intuition for node-fixed types should be fairly obvious: these are objects that cannot move between nodes in a static graph. In the DWR example, where the only static graph is defined by the `adjacent`-relation and the nodes are of type `location`, there are two node-fixed types: a `crane` belongs to a location and `pile` (of containers) is attached to a location. Thus, any operator that has a crane or a pile as one of its parameters is implicitly located by these objects.

In general, one might expect the relation P_j to be functional if the node-fixed type represents a physical object that can only be at one node. This is not necessarily the case, however, and the node-fixed type may represent a property (e.g. a colour) in which case instances can be associated with multiple nodes. Similarly, the relation P_j may be functional in the other direction, e.g. if there is only ever one crane at a given location. Once a node-fixed type has been identified the functional properties of the defining relation can be computed fairly easily for a given planning problem, which will define the graph. Also, since the relation must be static (by definition), these functional properties do not change in the state space spanned the problem. However, since this requires a planning problem to be given we shall not go into it.

Finally, the intuition behind node-fixed types may location-based, but nothing in the definition requires such a view, and thus this technique may be applied to any type of static graph.

Shift Operators

Given a static graph relation that defines a static graph for a given planning problem, there are often operators that shift objects or properties from one node in the static graph to a neighbouring node. This is a basic way in which the underlying state can be changed. Of course, there is usually more to such an operator than the simple shifting of an object or property.

Definition 14 (shift operator) *Let O be a planning operator with positive preconditions $p^p_1, \ldots, p^p_{n(p^p)}$, positive effects $e^p_1, \ldots, e^p_{n(e^p)}$ and negative effects $e^n_1, \ldots, e^n_{n(e^n)}$, where each precondition and positive/negative effect is a first-order atom. Let P_i be a static graph relation for the domain containing O, where t_i is the node type for this relation. Then we say that O is a* ***shift operator wrt. node type*** *t_i if and only if:*

- *O has a precondition $P_i(v, v')$;*
- *O has a precondition p^p_s that has an argument v (or v');*
- *O has a negative effect that is equal to this precondition p^p_s; and*
- *O has a positive effect that is equal to the precondition except where the argument v (or v') occurs, where the effect must have the value v' (or v respectively).*

Again, note that the definition of a shift operator relies solely on the definition of the operator, i.e. no planning problem is required. However, once a planning problem is given, this analysis step can be used to compute which objects or properties can be shifted to which other node in a static graph.

To illustrate this definition, we shall look at the `move` operator defined for the DWR domain. This operator is defined as follows:

```
(:action move
  :parameters (?r ?fr ?to)
  :precondition (and (adjacent ?fr ?to)
    (at ?r ?fr) (not (occupied ?to)))
  :effect (and (at ?r ?to) (occupied ?to)
    (not (occupied ?fr)) (not (at ?r ?fr))))
```

The intended meaning should be fairly obvious (to a human): the operator moves `?r`, a `robot`, from the `location ?fr` to the `location ?to`. Note that the operator has two parameters that have the node type (`location`) for the static graph defined by the `adjacent`-relation. The verify that `move` is a shift operator with respect to node type `location`, we simply have to find preconditions and effects according to the definition:

- the precondition `(adjacent ?fr ?to)` uses the predicate that defines the static graph, thus defining the edge along which this operators shifts;
- the precondition `(at ?r ?fr)` uses the variable `?fr` as its second argument, thus defining the node from which the shifting takes place;
- `(at ?r ?fr)` is also a negative effect of the operator; and finally
- the positive effect `(at ?r ?to)` is equal to the deleted precondition except for the position of the second argument which is replaced by the other node given in the first precondition, thus defining the node to which a shift takes place.

The algorithm that identifies shift operators follows the same procedure and simply applies the definition, attempting to find preconditions and effects that satisfy all the conditions. This is expressed in the following pseudo code.

function is-shift-op(O, P_i)
  for every $P_i(v, v') \in p^p_1, \ldots, p^p_{n(p^p)}$ **do**
    for every $P_j(v_1, \ldots, v_k) \in p^p_1, \ldots, p^p_{n(p^p)}$ **do**
      $i_v \leftarrow i_v$ such that $v_{i_v} = v$
      if i_v is undefined **continue**
      if $P_j(v_1, \ldots, v_k) \notin e^n_1, \ldots, e^n_{n(e^n)}$ **continue**
      for every $P_j(x_1, \ldots, x_k) \in e^p_1, \ldots, e^p_{n(e^p)}$ **do**
        for $i_e \in 1 \ldots k$ **do**
          if $i_v = i_e \wedge x_{i_e} \neq v'$ **next** $P_j(x_1, \ldots, x_k)$
          if $i_v \neq i_e \wedge x_{i_e} \neq v x_{i_e}$ **next** $P_j(x_1, \ldots, x_k)$
        return true

The algorithm takes two parameters, an operator and a static graph relation. It returns true if and only if the given operator is a shift operator wrt. the argument type of the

given predicate. Note that this requires the type to be known and well-defined, but as shown above, this kind of type can be derived from the planning domain. The algorithm then loops over all the precondition to find one that represents an edge in the graph. Then it loops over the preconditions again to find one that represents a candidate for a shifted property. A necessary condition here is that the node from which we are shifting occurs in the property precondition. Given such a candidate, the algorithm tests whether the property is deleted by the operator, another necessary condition. Finally, the algorithm tests whether a positive effect exists that represents the shifted property, which is true if it agrees with the property precondition in all arguments except for the one representing the static graph node, which must be the node to which we are shifting for the effect.

Domain Analysis

The above definitions and algorithm show how we can understand a planning domain in terms of static graphs that will be encoded in the planning problem. As already mentioned, the analysis of the domain (without the problem) lets us identify which relations represent edges and which types represent nodes in such a static graph. Given the planning problem itself, we may perform an analysis of the graph itself, where the results of this analysis will be valid for every world state that is reachable from the initial state.

One specific property that might be interesting in such a static graph in light of the shift operators just defined, is whether the graph is fully connected, or if it is not, which nodes are reachable from a given node. If we know that a node is reachable from a node in the initial state, and we know that a property that we can shift with a given operator holds in the initial state, then we know that this property can be achieved in any node reachable from the node in the initial state. In other words, we have a reachability condition that can be evaluated in constant time. Of course, planning graph analysis [Blum and Furst, 1995; Hoffmann and Nebel, 2001] gives us the same information and more, but at a much higher computational cost. This type of analysis is very useful to guide search, but whether it contributes to a better understanding of the problem is a different question, and a better understanding is what aids knowledge engineering.

There is another important difference between our analysis and the planning graph techniques just mentioned. While the latter give very good information to a heuristic search planner, it is hard to understand the information contained in a planning graph from a knowledge engineering point of view. The technique described in this paper is specifically aimed at the knowledge engineer, supporting a more intuitive way of understanding how a given set of operators can manipulate a world state.

In fact, the analysis can be used to help the knowledge engineer even more. Once an operator has been identified as shifting a property across a network of nodes, the next question is what other conditions there are that make the generic problem difficult. Clearly, if the shifting was all that is going on in a domain, this would not be a hard problem. So the remaining preconditions and effects of a shift operator must somehow encode the difficult part.

For example, the the `move` operator defined for the DWR domain constitutes a shift operator that shifts the location of the robot as given by the `at` along edges defined by the `adjacent relation`. Thus, for a given problem, it is easy to compute all the possible locations at which a robot may be in the state space. Also, paths to these locations are easy to compute. However, the other conditions defining the `move` operator specify that movement is only possible to an unoccupied location. Thus, the problem becomes hard, because an optimal solution involves collision avoidance in time. Interestingly, if one continued this line of reasoning it should be easy to see that problems involving one robot are easy, as the `occupied` relation can simply be disregarded, that is, it can dropped from the planning domain. This type of reasoning is very much in line with the kind of analysis shown in [Amarel, 1968], and this is what we hope to (eventually) achieve with this work.

Evaluation

The methodology for evaluating the technique described above was chosen as follows. We used the DWR domain [Ghallab *et al.*, 2004] to develop the technique in terms of definitions and algorithms. This was possible because the DWR domain does encode a static network of locations along which a robot can move. There are also other operators that do not represent shift operations. Thus, this domain was used to provide a first correctness check.

To properly evaluate the technique we have applied it to a small number of other planning domains. To avoid any bias we used only planning domains that were available from third parties, namely from the international planning competition. Since the algorithm works on domains and the results have to be interpreted manually only a limited number of experiments was possible. Note that a knowledge engineer using the approach described here to ensure the consistency of the domain they are developing would not need to perform a manual analysis. This manual analysis is only necessary for this evaluation as the features we are looking at were not defined with the domains. Random domains are not suitable as they cannot be expected to encode meaningful knowledge.

The domains used for the evaluation were the simple STRIPS versions of the following domains: movie, gripper, logistics, mystery, mprime, and grid. The first step towards an analysis of these domains was an analysis identifying static and fluent relations and derived types, as none of the domains had explicit given types as part of the domain definition. However, this will not be described here.

Static Graphs

The first domain in our analysis, the movie domain, is a very simple domain that is almost propositional. All the predicates used are unary, effectively specifying the types of objects that exist. Clearly, there is no static graph encoded here and indeed, our algorithm does not identify any static graph relation.

The second domain, gripper, is more promising as it contains a robot that can move objects between room. Thus

the same intuition as for the DWR domain applies, and one might expect to find at least a network of locations as a static graph in this domain. However, the domain is defined to allow movement of the robot from any room to another; the path taken by the robot is abstracted away. No other static graph is identified by our algorithm in this domain.

The same issue occurs in the logistics domain. Again, the connections between the different types of places are not explicit, meaning no static graph can be identified. This is unfortunate as the domain has different types of locations (cities and airports), which would have provided an interesting challenge for our intuition.

The next domain, mystery, is perhaps the most interesting case here. This domain is not based on robots that have to transport objects between locations, and it is indeed a mystery what is going on in this domain at first glance. Our domain analysis identifies three different predicates that define static graphs for this domain. The first predicate, `orbits` is perhaps the most obvious as it can be seen as location-related. The other static graph relations identified are `eats` and `attacks`, which are somewhat similar and can be interpreted in a meaningful way. Note that none of the three relations identified here would be expected to be symmetric, so this is quite different from the adjacency used as the intuition behind the approach.

The mprime domain is really a variation of the mystery domain and the analysis of static graph relations yields the same three relations as a result.

The final domain used for the evaluation is the grid domain which is a classic transportation domain in which the network of locations is explicitly specified. And indeed, our domain analysis finds one static graph relation for this domain, namely the relation `conn` which, not surprisingly, corresponds directly to the adjacency relation in the DWR domain.

Node-Fixed Types

The movie domain which does not contain a static graph, cannot contain node-fixed types. The same is true for the gripper and the logistics domain.

More interesting is the mystery domain, which had three static graph relations. The first of these, `orbits`, gives us one node-fixed type. Looking at the type definition, we can see that this is the type used for both arguments of the `orbits`-relation, which must hold by definition. Another place where this type is used is the unary `plant` predicate, which gives us an idea of type of object this is meant to be. Finally, the type is also used as the second argument of the `harmony`-relation. The other static graph relations, `eats` and `attacks`, also define one node-fixed type each: `eats` gives us a type that is also used in the unary predicate `food`, whereas `attacks` reveals a node-fixed type that is not used in a unary predicate, but only as the second argument in the `locale`-relation.

The node-fixed types for the mprime domain are the same as the ones for the mystery domain. What is perhaps interesting here is that there is a surprising amount of static information in both these domains, which might make a domain analysis before planning useful.

Finally, the grid domain is less surprising here, rendering just one node-fixed type which is used in, amongst other, two unary relations, namely `at-robot` and `place`. The latter can be interpreted as the node type, of course.

Shift Operators

Of course, none of the first three domains, movie, gripper and logistics contains shift operators since there also do not contain static graphs.

The mystery domain is again interesting in that two of the static graph relations have operators that shift properties along the edges. Firstly, the operator `succumb` shifts `harmony(?v,?s1)` along `orbits(?s1,?s2)` to `harmony(?v,?s2)`, and secondly, the operator `feast` shifts `craves(?v,?n1)` along `eats(?n1,?n2)` to `craves(?v,?n2)`.

And this is where the mprime domain differs from the mystery domain significantly. In the mprime domain there is a shift operator also for the final static graph relation: operator `drink` shifts `locale(?n2,?l21)` along `attacks(?l21,?l22)` to `locale(?n2,?l22)`.

The last domain, the grid domain, shows itself as being similar to the DWR domain again. It is the move operator that can be used to shift the robot from one place to another. What is perhaps interesting here is that the property that is shifted in this domain is represented by a unary predicate: `at-robot`. This is of course a valid representational choice and, perhaps fortunately, the definitions of static graphs and shift operators are sufficiently general to catch this special case.

Conclusions

The work described in this paper builds on the domain analysis in terms of features described in [Wickler, 2011]. The general approach in which this work can be used is similar to the previous analysis: knowledge engineering could be given the option to specify additional, redundant knowledge that can then be used by the automatic analysis to ensure consistency of the representation. As for previous work, the analysis is of the domain, that is, a set of operators, not of a planning problem. Thus, the result of the analysis is valid for all problems referring to the analyzed domain and will not have to be repeated.

Further analysis of a planning problem, based on the results of the domain analysis, may be performed and may give further insights into the problem. In fact, it is often the case that those aspects of the initial state of a planning problem that are given by the static relations have a degree of reusability and may not change across a range of problems. For example, in a real dock the topology is fixed and will not change from one problem to the next.

The major difference between previous work and the technique described here is that the analysis is in terms of a feature that may or may not be present in a planning domain, namely, static graphs that are more or less explicit in the representation. GIPO and itSIMPLE are systems that support the whole KE process, but neither performs an analysis in terms of shift operators based on static graphs as we have defined

it here. A graph consisting of nodes and edges is a very generic way of describing an aspect of a problem, and thus we can hope to find this feature in many domains, but if not present, this analysis obviously cannot aid the knowledge engineering for such a domain.

Also, the analysis algorithms and the underlying definitions rely on certain representational choices that are commonly used when formally representing knowledge, but there may be alternatives that we have not considered that may make the analysis fail despite the presence of static graphs.

The DWR domain that was used to develop the ideas underlying this work highlights some of the issues with the representation that interfere with our analysis. For example, our algorithm does not find that the move operator also shifts the `occupied`-relation together with the `at`-relation. This is simple because the domain uses occupied as a negative precondition, which is not captured by the definition. The definitions could of course be adapted to this case, but it also raises the question why this representational choice was made and whether its negation, the `free`-relation, would not be a better choice for the domain.

Similarly, the exploitation of invariants may lead to domain specifications for which our analysis fails.

Another interesting result from the evaluation is that most static graphs were identified in domains that are not classic transportation domains. While this shows that the analysis does indeed apply to other classes of problems, it is not clear what these classes are. However this was never our aim; our approach is inspired by one problem class, but the definitions do not make reference to specific concepts associated with this class.

What the type of domain analysis presented in this paper is trying to achieve is a more human-like understanding of a planning domain without reverting to using the names of symbols as a clue to their meaning or even the comments in a PDDL. A graph may be just one way to achieve this. If we could identify other sub-problems that are common in planning domains and allow for a fast (polynomial time) analysis, then it may just be possible to identify what exactly it is that makes a planning problem that uses the domain difficult, or which interaction of features is the hard part of the generic problem. This is future work, of course.

References

Saul Amarel. On representations of problems of reasoning about actions. In Donald Michie, editor, *Machine Intelligence 3*, pages 131–171. Elsevier/North-Holland, 1968.

Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. In *Proc. 14th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1636–1642. Morgan Kaufmann, 1995.

Andrew Coles and Amanda Smith. Generic types and their use in improving the quality of search heuristics. In *Proc. 25th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2006)*, 2006.

Clive Dawson and Laurent Siklossy. The role of preprocessing in problem-solving systems. In *Proc. 5th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 465–471. Morgan Kaufmann, 1977.

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

Maria Fox and Derek Long. PDDL2.1 : An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning*. Morgan Kaufmann, 2004.

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

T.L. McCluskey and J.M. Porteous. Engineering and compiling planning domain models to promote validity and efficiency. *Artificial Intelligence*, 95:1–65, 1997.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2003.

R.M. Simpson. Structural domain definition using GIPO IV. In *Proc. 2nd Int. Competition on Knowledge Engineering for Planning and Scheduling*, 2007.

Tiago Stegun Vaquero, V. Romero, F. Tonidandel, and J.R. Silva. itSIMPLE 2.0: An integrated tool for designing planning environments. In Mark Boddy, Maria Fox, and Sylvie Thiébaux, editors, *Proc. 17th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 336–343, 2007.

Tiago Stegun Vaquero, José Reinaldo Silva, and J. Christopher Beck. A brief review of tools and methods for knowledge engineering for planning and scheduling. In Roman Barták, Simone Fratini, Lee McCluskey, and Tiago Stegun Vaquero, editors, *Proc. Knowledge Engineering for Planning and Scheduling (KEPS)*, pages 7–14, 2011.

Gerhard Wickler. Using planning domain features to facilitate knowledge engineering. In Roman Barták, Simone Fratini, Lee McCluskey, and Tiago Stegun Vaquero, editors, *Proc. Knowledge Engineering for Planning and Scheduling (KEPS)*, pages 39–46, 2011.