Develop, Visualize and Test Classical Planning Descriptions in your Browser

Maurício Cecílio Magnaguagno, Ramon Fraga Pereira, Martin Duarte Móre
and **Felipe Meneguzzi**
Pontifical Catholic University of Rio Grande do Sul (PUCRS), Brazil
Graduate Program in Computer Science, School of Computer Science (FACIN)
{mauricio.magnaguagno, ramon.pereira, martin.more}@acad.pucrs.br
felipe.meneguzzi@pucrs.br

Abstract

Automated planning tools are complex pieces of software that take declarative domain descriptions and generate plans for complex domains. New users often find it challenging to understand the plan generation process, while experienced users often find it difficult to track semantic errors and efficiency issues. To simplify this process, in this paper, we develop a cloud-based planning tool with code editing, code verification and validation, and problem visualization capabilities. The code editor focuses on the domain, problem, and resulting plan and other textual information, helping the user see how such descriptions are connected without changing context across multiple screens. The visualization tool includes two alternative visualization schemes aimed at illustrating the explored state-space and dependencies between actions and predicates during plan execution.

Introduction

Classical planning algorithms typically require a declarative domain specification describing action schemata, which, in turn, define the dynamics of the underlying domain. Since the inception of the International Planning Competition (IPC), the standard specification language for classical planning is the Planning Domain Definition Language (PDDL) (Haslum et al. 2019). Given the declarative nature of PDDL, planning algorithm implementations are often opaque regarding the intermediate steps between reading the formalism and generating a plan. This creates a twofold problem for domain engineers that wish to use automated planning technology to solve any given planning instance: ensuring correctness, and optimizing the efficiency of a planning algorithm.

First, ensuring correctness of PDDL specifications may be a challenging task for new users even for simple domains, while detecting semantic mistakes in complex domains is always non-trivial. Even when the user successfully compiles and executes a planning instance with the chosen heuristic function the planner may fail to find a correct plan for the intended domain. In these cases, virtually no planning algorithm offers extra information, and the user only knows that either the domain has some kind of description error, or that specific problem supplied to the planner is unsolvable.

Second, practical applications of classical planners require not only a formalization of the domain in PDDL that is correct, but also exploit the search mechanisms employed by the underlying planners to find solutions efficiently. Most modern classical planning solvers (Richter and Westphal 2010; Hoffmann 2011) use heuristic functions to estimate which states are likely to be closer to the goal state and save time and memory during the planning process. Different heuristic functions may be more effective in solving problems in different planning domains within a reasonable time and a small memory footprint. Thus, key to understanding the efficiency of a domain formalization is its impact on the heuristic function used by the underlying planner.

In order to address these challenges, we developed WEB PLANNER (Magnaguagno et al. 2017), an online tool aimed at helping domain engineers to formalize classical planning descriptions and spot semantic errors in planning domains. Our tool, which we describe in the Architecture Section, includes a PDDL code editor with syntax highlighting and code auto-completion aimed at helping users to efficiently develop planning instances in a similar workflow to many popular Integrated Development Environments. Importantly, we integrate the editor to other debugging tools, described in the Capabilities Section, developed to help users cope with the declarative nature of PDDL and explore the effects of changes to the domain in solving concrete problems.

Architecture

We designed our tool envisioning a development process centered around two tasks by the domain developer. In the first task, the user aims to describe both domain and problem correctly. In the second task, the user tries to identify details of the description (in terms of predicate use) that impact performance and how these predicates occur during the planning process. The domain designer is free to move between these tasks and repeat until satisfied with the results. Once a planning instance is described it is possible to visualize the explored state-space, even when the planning process fails. When the planning process finishes, the user can visualize how each action adds or deletes predicates during plan execution. Such visualizations also help new users understand how states are represented and operated internally.

To avoid the considerable setup time of some planner implementations and maintain a consistent interface across platforms, we use a web interface. Planner and analysis tools are executed in the server, while the editor and visualization animations are executed in the client browser. Communication between client and server uses JSON[1] and is currently stateless, information sent to the server during each request is discarded once its resulting response is sent back to the user. We illustrate this architecture in Figure 1.

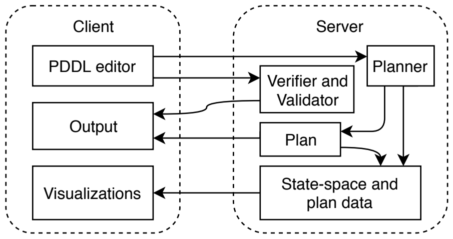

Figure 1: Overview of the WEB PLANNER architecture.

Capabilities

Our implementation uses three distinct components to assist planning: editor, verifier and validator, and visualizations.

Editor

The text editor interface uses a split-view of a domain and problem to make clear to the user that both descriptions are separate files but are often developed simultaneously. The user is able to see both domain, problem and textual outputs (such as the plan) without context switching. The editor's syntax-highlighter helps users identify missing elements, such as parentheses, while keyboard shortcuts allow the user to generate common PDDL structures, such as actions.

Verifier and Validator

Plan output alone is not enough to identify errors in a planning description. The declarative nature of PDDL obscures the intermediate structures of the planner for novice users (or users without working knowledge of planner implementation), requiring further modification of the chosen planner to log such information. Common mistakes can be identified by a verifier, which tests both domain and problem for repeated or undeclared elements (action names, parameters, preconditions, effects), contradictions, invalid characters, etc.

A syntactically valid PDDL description can be verified for errors, but one may ask if this is enough to obtain a valid representation of the environment they are simulating. To semantically validate one must execute plans. The validator applies each plan action, testing if such action exists and is applicable, and with their effects generate each intermediate state, validating a final state that satisfies the goal. Verifiers and validators are often separate tools (Howey, Long, and Fox 2004) outside the knowledge of novice users that could greatly benefit from automated analysis of their descriptions.

[1]JSON (JavaScript Object Notation) is an open-standard format for structuring data.

Visualizations

Textual outputs describe details about the intermediate states explored by a planner, but only visualizations can give an idea of the whole state-space and plan in a single look. We currently support two visualizations: an interactive state-space tree and a puzzle-like plan metaphor (Magnaguagno, Pereira, and Meneguzzi 2016). Such visualizations are useful to explain high-level concepts, such as a heuristic function forcing search to explore selected branches of the state-space tree or which previous actions satisfy preconditions to apply other actions and achieve the goal state. By interacting with these two visualizations, state-space tree and plan, we expect users to obtain insights about the complexity and relations between domain actions.

Conclusions

In this paper, we report on the WEB PLANNER demo, which consists of a PDDL editor to formalize planning domains and problems, a verifier and a validator to report common errors, and visualizations to help understand the impact of heuristic usage in state-space search. Our small scale user testing with undergraduate students shows that such tool can substantially mitigate the burden of complex installation and usage instructions before required for new users to formalize planning problems in PDDL. WEB PLANNER is available at `https://web-planner.herokuapp.com`.

Acknowledgements: Felipe acknowledges support from CNPq under project numbers 407058/2018-4 and 305969/2016-1, and FAPERGS process number 18/2551-0000500-2. We acknowledge support from HP Brasil using incentives of Brazilian Informatics Law n^o 8.2.48 of 1991. We also acknowledge the support given by CAPES under Finance Code 001 and the Pro-Alertas project (88887.115590/2015-01).

References

Haslum, P.; Lipovetzky, N.; Magazzeni, D.; and Muise, C. 2019. An introduction to the planning domain definition language. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 13(2):1–187.

Hoffmann, J. 2011. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *CoRR* abs/1106.5271.

Howey, R.; Long, D.; and Fox, M. 2004. VAL: automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *ICTAI*, 294–301.

Magnaguagno, M. C.; Pereira, R. F.; Móre, M. D.; and Meneguzzi, F. 2017. Web planner: A tool to develop classical planning domains and visualize heuristic state-space search. In *Proceedings of UISP*, 32–38.

Magnaguagno, M. C.; Pereira, R. F.; and Meneguzzi, F. 2016. DOVETAIL - An Abstraction for Classical Planning Using a Visual Metaphor. In *Proceedings of FLAIRS*, 74–79.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR* 39(1):127–177.