Graphical Navigation in Solution Spaces using PlanPilot

Michelle Kornherr[1], Johannes K. Fichte[1], Dominik Rusovac[2], David Speck[3]
Sarah Gaggl[2], Augusto B. Corrêa[4], Markus Hecher[5], Daniel Gnad[1,6]

[1]Linköping University, Sweden; [2]TU Dresden, Germany; [3]University of Basel, Switzerland
[4]University of Oxford, UK; [5]Computer science Research Institute of Lens, France; [6]Heidelberg University, Germany
hicko047@student.liu.se, johannes.fichte@liu.se, dominik.rusovac@tu-dresden.de, davidjakob.speck@unibas.ch
sarah.gaggl@tu-dresden.de, augusto.blaascorrea@chch.ox.ac.uk, hecher@cril.fr, daniel.gnad@uni-heidelberg.de

Abstract

Many planning applications require not only a single solution but benefit substantially from having a set of possible plans from which users can select according to preferences. Surprisingly, planning research has primarily focused on quickly finding single plans for decades. Only recently have researchers started to investigate plan enumeration by top-k planning, offering more flexibility to the user. But simply enumerating the k best plans is far from targeted due to the time-consuming nature of enumeration, likely feeding many similar plans to the user, and forcing the user to define filters beforehand. In fact, in extensive search spaces, enumeration is hardly practical. We present an approach and a tool called `PlanPilot` to navigate solution spaces of planning tasks iteratively and interactively. We build on answer-set programming (ASP) to restrict the plan space. To that end, we employ *facets*, which are meaningful actions that appear in some, but not all plans. Enforcing or forbidding such facets allows for navigating even large plan spaces while ensuring desired properties quickly and step by step.

Introduction

Many planning applications call for multiple high-quality plans, which lead to the exploration of natural extensions of optimal classical planning in the community. A well-known technique is to find the k best plans by top-k planning (Katz et al. 2018; Speck, Mattmüller, and Nebel 2020; Katz and Sohrabi 2020) enabling post hoc restrictions for various applications (Boddy et al. 2005; Sohrabi et al. 2018). But enumeration results in major disadvantages. It is computationally costly and yields potentially many similar plans, which makes exploration entirely impractical. Interestingly, we can avoid enumeration in many cases, for example, when debugging for actions that unexpectedly never show up (Lin, Grastien, and Bercher 2023; Gragera et al. 2023), searching for sets of jointly achievable soft goals (Smith 2004), or asking for explanations of the absence of solutions that achieve the desired set of such soft goals (Eifler et al. 2020).

In this paper, we establish a practical approach to navigate plan spaces iteratively and interactively. We present an implementation, which we call `PlanPilot`, that enables systematic reasoning even with many plans. Similar to top-k planning, we consider the best plans of the given task, with the difference that we impose a bound on the plan length instead of the number of computed plans. We employ an existing ASP encoding for finding bounded plans and reasoning over them (Dimopoulos et al. 2019) and take advantage of *counting* and *facets* (Fichte, Gaggl, and Rusovac 2022). *Counting* enables us to reason about the plan space without enumerating solutions (Darwiche 2001). *Facets* have recently been introduced in planning (Speck et al. 2025) and provide meaningful actions that can be gradually restricted by the user enforcing or forbidding these actions. Facets are computationally easier and enable navigation even in large plan spaces while ensuring desired properties quickly and step by step. Our demo video illustrates the navigation.[1]

`PlanPilot` takes as input a planning task and constructs an ASP encoding of it. The ASP problem is passed to the `fasb` reasoner, which allows user interaction for navigating the plan space using facets. The user can query `PlanPilot` for the number of solutions of the task or list the available facets (i.e., meaningful actions). All these operations are fast, so the user can efficiently constrain the set of plans until a manageable set of solutions with the desired properties can be enumerated. `PlanPilot` is publicly available.[2]

Plan-Space Navigation using ASP

Our `PlanPilot` tool takes as input the PDDL description of a planning task and allows for interactive navigation of the solution space by the user. Figure 1 illustrates the interaction between the components of `PlanPilot`. First, we use `Fast Downward` (Helmert 2006) to find an optimal solution to obtain the horizon H, which acts as a plan-length bound. We pass this, together with the grounded task and two configurable parameters, encoding and step-type, to `plasp` to encode the planning task in ASP. The encoding type restricts plan length either *exactly* to H or let's H be the upper *bound*. The step type distinguishes between *concrete* and *abstract* time steps, which we will explain shortly. The ASP encoding is passed to `fasb` (and its internal solver `clingo`), which enables interactive reasoning with facets or plans.

In interactive mode, the user can query `PlanPilot` for the number of available facets/plans or a list of these facets/-plans. In our ASP encoding, facets are meaningful actions that occur in some, but not all solutions. More specifically,

[1]**Demo video:** https://youtu.be/75UngGNr5bc
[2]**Code:** https://github.com/mischidream/PlanPilotpp

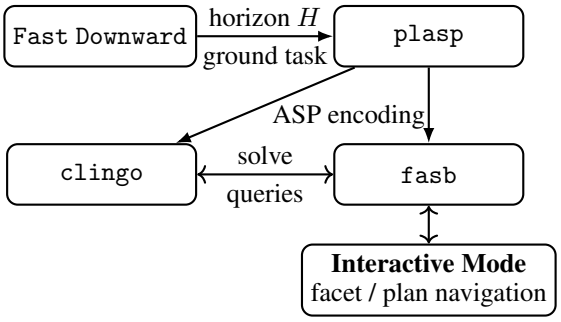

Figure 1: Illustration of PlanPilot's components.

we turn the occurrence of every action a at a *concrete* time step $1 \leq t \leq H$ into a facet $\texttt{occurs}(a, t)$. These facets are *enforcing*, which means that we restrict the plans by enforcing them to have action a at step t. Additionally, there are *prohibiting* facets, denoted $\sim\texttt{occurs}(a, t)$, which forbids the occurrence of action a at time step t. Both kinds of facets can be activated, which enables the respective restriction, and deactivated again, which removes the restriction. Using both kinds of facets, the user can iteratively and interactively refine the set of plans according to their requirements. At every step, i.e., after (de)activating a facet, the user can query PlanPilot for the number of remaining facets as well as the number of remaining plans that satisfy the enabled facets. Both, facets and plans, can also be enumerated at every step. The activation of facets reduces the freedom in the resulting plan space, which often *implies* certain action occurrences along a plan, as no other options exist. These implied actions can also be listed during navigation.

For use cases where the occurrence of an action is desired at *some* point in the plan, we leverage the expressiveness of ASP by adding the rule $\texttt{occurs\_sometime}(a) \leftarrow \texttt{occurs}(a, t), t > 0.$ to our encoding. We refer to this variant as the *abstract time steps* encoding. If such a facet is activated, then a must occur at some step in the plan, allowing for multiple occurrences of the same action. This is desirable in scenarios where the user is not interested in having a occur at a specific point, e.g., for domain debugging (Lin, Grastien, and Bercher 2023; Gragera et al. 2023).

Complexity Reasoning over facets is significantly easier than reasoning over plans (Rusovac et al. 2024; Speck et al. 2025). Therefore, it is advisable to opt for counting and enumeration of facets as planning tasks and horizons get larger. Moreover, enabling more facets limits the plan space considered by fasb, facilitating reasoning on the remaining space. Hence, the user can activate facets that correspond to important plan constraints to simplify the reasoning, such that plans can be counted and enumerated efficiently.

Navigation modes Besides enumerating the available facets, fasb can provide information on what it implies to activate a facet. This is done by showing the reduction in the number of facets as well as the number of remaining facets after activation, which allows for a more targeted interaction. If the user wants to find a small set of plans quickly, they can activate facets that result in high reduction, which implies that the set of remaining plans is maximally constrained. Alternatively, if the user desires a large, diverse set of plans, then selecting facets that lead to a low reduction in the remaining facet count are preferable. This allows the user to navigate the plan space according to their needs.

Technical aspects PlanPilot comes with a web-based interface to support plan space navigation. The frontend is implemented using Vue 3 with TypeScript and Pinia for state management, while the plans are visualized based on SVG. The frontend communicates with a Flask backend via REST endpoints, which in turn manages the execution of Fast Downward, plasp, and fasb. To avoid redundant computation, the backend caches results of previous reasoning steps. PlanPilot inherits the PDDL language support of plasp, which includes PDDL 3.1 without durative actions, numerical fluents, and preferences. We refer to the plasp repository for a full list of supported features.

Possible extensions All parts of the ASP encoding can be turned into facets in the same way as it is implemented for actions. In particular, this holds for state atoms. Thereby, the user can impose restrictions on the desired solutions not only by enforcing, or prohibiting, certain actions to occur on the plan, but also by requiring state atoms to be achieved in some state along the plan. This could be used, e.g., in oversubscription planning by encoding soft goals as facets, using PlanPilot to enforce a desired (set of) soft goals to be achieved in the goal or in some state along the plan.

Discussion

We present PlanPilot, a tool to navigate plan spaces iteratively and interactively. We build on an ASP encoding for finding solutions and investigating the plan space. The ASP encoding enables us to consider restrictions on actions and specific times of these restrictions. We can, step by step, enforce or prohibit meaningful actions (facets) that are present or absent in a specific, or any step in the plan.

By combining facets and counting, we can navigate plan spaces in multiple ways. When *exploring*, we select facets that constrain the plan space the least. Whereas, when aiming for a particular *target*, we take facets that maximally constrain the plan space. Thereby, users can either obtain large sets of diverse plans, or quickly converge to very few plans, depending on application needs. Slightly extending the ASP encoding allows us to also reason about restrictions that happen at any *state* of the sequence, forcing the plan to achieve soft goals "on the way", or as usual in the final state.

We believe that beyond the current features, it is interesting to consider partial-order planning in the navigation, i.e., obtaining (minimal) partially ordered plans natively without adding implications to simulate all steps. We expect support for loopless plans to be beneficial for many applications, too. Both of these plan types are non-trivial to integrate, as it is unclear how to represent them on the ASP level.

In its current form and with many possible extensions, PlanPilot takes a major step towards making classical planning explainable to non-expert users. The interface is easily accessible and can connect to visualizations such as blocksworld, as shown in our video.

References

Boddy, M.; Gohde, J.; Haigh, T.; and Harp, S. 2005. Course of Action Generation for Cyber Security Using Classical Planning. In *Proc. ICAPS 2005*, 12–21.

Darwiche, A. 2001. Decomposable Negation Normal Form. *JACM*, 48(4): 608–647.

Dimopoulos, Y.; Gebser, M.; Lühne, P.; Romero, J.; and Schaub, T. 2019. plasp 3: Towards Effective ASP Planning. *Theory and Practice of Logic Programming*, 19(3): 477–504.

Eifler, R.; Cashmore, M.; Hoffmann, J.; Magazzeni, D.; and Steinmetz, M. 2020. A New Approach to Plan-Space Explanation: Analyzing Plan-Property Dependencies in Oversubscription Planning. In *Proc. AAAI 2020*, 9818–9826.

Fichte, J. K.; Gaggl, S. A.; and Rusovac, D. 2022. Rushing and Strolling among Answer Sets – Navigation Made Easy. In *Proc. AAAI 2022*, 5651–5659.

Gragera, A.; Fuentetaja, R.; Olaya, Á. G.; and Fernández, F. 2023. A Planning Approach to Repair Domains with Incomplete Action Effects. In *Proc. ICAPS 2023*, 153–161.

Helmert, M. 2006. The Fast Downward Planning System. *JAIR*, 26: 191–246.

Katz, M.; and Sohrabi, S. 2020. Reshaping Diverse Planning. In *Proc. AAAI 2020*, 9892–9899.

Katz, M.; Sohrabi, S.; Udrea, O.; and Winterer, D. 2018. A Novel Iterative Approach to Top-k Planning. In *Proc. ICAPS 2018*, 132–140.

Lin, S.; Grastien, A.; and Bercher, P. 2023. Towards Automated Modeling Assistance: An Efficient Approach for Repairing Flawed Planning Domains. In *Proc. AAAI 2023*, 12022–12031.

Rusovac, D.; Hecher, M.; Gebser, M.; Gaggl, S. A.; and Fichte, J. K. 2024. Navigating and Querying Answer Sets: How Hard Is It Really and Why? In *Proc. KR 2024*, In press.

Smith, D. E. 2004. Choosing Objectives in Over-Subscription Planning. In *Proc. ICAPS 2004*, 393–401.

Sohrabi, S.; Riabov, A. V.; Katz, M.; and Udrea, O. 2018. An AI Planning Solution to Scenario Generation for Enterprise Risk Management. In *Proc. AAAI 2018*, 160–167.

Speck, D.; Hecher, M.; Gnad, D.; Fichte, J. K.; and Corrêa, A. B. 2025. Counting and Reasoning with Plans. In *Proc. AAAI 2025*, 26688–26696.

Speck, D.; Mattmüller, R.; and Nebel, B. 2020. Symbolic Top-k Planning. In *Proc. AAAI 2020*, 9967–9974.