Interactive Exploration of Plan Spaces

Daniel Gnad[1,2], **Markus Hecher**[3], **Sarah Alice Gaggl**[4], **Dominik Rusovac**[4], **David Speck**[5], **Johannes K. Fichte**[2]
[1]Heidelberg University, Germany
[2]Linköping University, Sweden
[3]Université d'Artois, CRIL, CNRS, France
[4]TU Dresden, Germany
[5]University of Basel, Switzerland
firstname.lastname@{liu.se,tu-dresden.de}, hecher@cril.fr, davidjakob.speck@unibas.ch

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, for example, *when explaining plans*. For decades, research in classical AI planning has primarily focused on quickly finding single plans. Only recently researchers have started to investigate preferences, enumerate plans by top-k planning, or count plans to reason about the plan space. Unfortunately, reasoning about the plan space is computationally extremely hard and feeding many similar plans to the user is hardly practical. To circumvent computational shortcomings while still being able to reason about variability in plans, *faceted actions* have been introduced very recently. These are meaningful actions that can be used by some plan but are not required by all plans. Enforcing or forbidding such facets allows for navigating even large plan spaces while ensuring desired properties quickly and step by step. In this paper, we illustrate an industrial challenge, the Beluga logistics problem of Airbus, where reasoning with facets enables targeted plan space navigation. We present an approach to handle large plan spaces iteratively and interactively and present a tool that we call `PlanPilot`.

1 Introduction

Classical planning aims at finding a sequence of actions that transforms the initial state of a problem into a goal state (Bylander 1994; McDermott et al. 1998; Ghallab, Nau, and Traverso 2025). Since many planning applications call for multiple high-quality plans, natural extensions of optimal classical planning have been explored 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 due to extreme computational costs and very similar plans. Additionally, we need to define optimality properties beforehand, which makes exploration entirely impractical. To this end, additional *quantitative and qualitative reasoning techniques* on the plan space have very recently been introduced (Speck, Mattmüller, and Nebel 2020).

In this paper, we go a step further and aim at *navigating plan spaces interactively* to help users *explore*, *understand*, and *explain* solutions. Our goal is motivated by a logistics application in industry at Airbus SE, where parts are moved between several factories by a sequence of super transporter airplanes that received its nickname after the *Beluga* whale due to their exceptional shape. Engineers assemble the airplane parts at different locations, move them, and finally build an entire aircraft. Unsurprisingly, factories suffer from limited space and mobility resulting in numerous constraints, for example, Belugas carry certain parts in a fixed order, only one Beluga can be loaded at a time. As loading capacities need to be used most efficiently, engineers originally spend hours over hours to manually generate solutions for handling the disposition of Belugas. Due to the Beluga Competition (Tuples 2025) automated planning came into the picture reducing the labor-intensive process, however, putting the need to explain solutions. Then, humans interact with the system, ask questions about possible plan(s), generate answers to these, try to explain initial solution(s), or modify certain aspects of the solutions interactively.

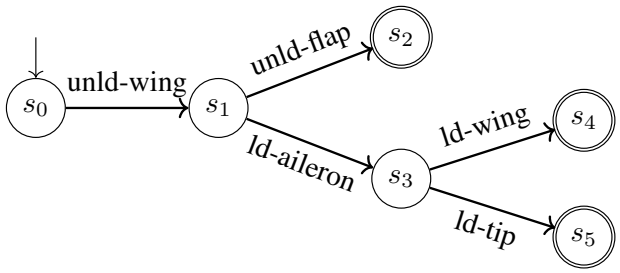


Figure 1: State space of our running example task Π_1. The initial state is denoted by s_0; the goal states are denoted by double corners.

Example 1. *Consider a planning task Π_1 consisting of a toy logistics scenario where we have to (un)load wings, flaps, ailerons, and wing tips. We illustrate the state space in Figure 1. The initial state is s_0 and we have multiple goal states s_2, s_4, and s_5. The edge labels identify the action being applied. Depending on external factors, we have three valid plans:*

(i) unload-wing; unload-flap,

(ii) unload-wing; load-aileron; load-wing, and

(iii) unload-wing; load-aileron; load-tip.

Clearly, the action unload-wing *has to occur every time making it a so-called* landmark *action. Then, we have a certain flexibility and may choose action* unload-flap *or* load-aileron*. Also, the actions* load-wing *and* load-tip *are not involved with all plans, but occur on some plan. Given this knowledge, we may forbid certain actions after we found one plan to obtain another plan or another direction.*

While we can easily enumerate or count all solutions in our example above, both are computationally extremely challenging when many solutions exist. Instead, when looking for alternative solutions and many other cases, we can employ reasoning between decisions and counting. This enables us to understand plans better while maintaining favorable complexity. A central tool are *facets*, which are actions or states that occur in some plan (relevant) but not all plans (dispensable). Reasoning with facets yields a more fine-grained understanding of variability and enables navigation.

Main Contribution.

1. We present a concrete use case where we aim to comprehend large solution spaces and make sense of alternative solutions. The use case originates in the 2025 Beluga AI Challenge.
2. We establish a practical approach to navigate plan spaces iteratively and interactively and make interactive plan space exploration more precise.
3. We demonstrate a practical tool, `PlanPilot`, that builds on multiple reasoning techniques allowing to interactively output plans, filter plans, restrict plans, restrict flexibility in plans, count plans, or count flexibility in actions, or estimate effects.

Related Works. Imposing additional constraints on solutions of planning tasks is a well-studied paradigm (Gerevini and Long 2005; Edelkamp 2006). Tool support is quite limited, though, and an interactive navigation of plans that satisfy the constraints has not been suggested. Querying planning spaces has been considered in the past, for example, 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). Planning systems that are based on answer-set programming (ASP) have been developed (Dimopoulos et al. 2019). These systems immediately support bounded plan enumeration (Gebser, Kaufmann, and Schaub 2009). In the context of ASP, *counting* and *facets* have been considered for navigating large solution spaces and the computational complexity classified (Fichte, Gaggl, and Rusovac 2022; Rusovac et al. 2024). Anytime and approximate counting techniques for ASP exists (Kabir et al. 2022; Fichte et al. 2024), but are limited in scalability. *Counting* enables detailed reasoning about the plan space without enumerating solutions (Darwiche 2001) and more fine-grained conditional reasoning (Fichte, Hecher, and Nadeem 2022). Knowledge compilation has been applied to conformant planning for plan validity checks (Palacios et al. 2005). Very recently, facets have been introduced for planning as a tool to understand the significance of an action in one step and the computational complexity classified (Speck et al. 2025). Fichte et al. (2025) introduced facets to abstract argumentation. (Eiter and Geibinger 2023) studied justifications for the presence, or absence, of an atom in the context of answer-set programming including so-called contrastive explanations. (Schmidt et al. 2025) considered complexity of facets in propositional abductive explanations.

2 Background

We follow established definitions of classical planning (Bäckström and Nebel 1995; Helmert 2006), which capture a common fragment of PDDL. Comprehensive introductions on ASP are available (Janhunen and Niemelä 2016; Calimeri et al. 2020; Gebser et al. 2012).

Classical Planning. A *planning task* is a tuple $\Pi = \langle \mathcal{A}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$, where $\mathcal{A}$ is a finite set of propositional *state variables*. A *(partial) state* s is a total (partial) mapping $s : \mathcal{A} \to \{0, 1\}$. For a state s and a partial state p, we write $s \models p$ if s *satisfies* p, more formally, $p^{-1}(0) \subseteq s^{-1}(0)$ and $p^{-1}(1) \subseteq s^{-1}(1)$. $\mathcal{O}$ is a finite set of *actions*, where each action is a tuple $o = \langle \mathrm{pre}_o, \mathrm{eff}_o \rangle$ of partial states, called *preconditions* and *effects*. An action $o \in \mathcal{O}$ is *applicable* in a state s if $s \models \mathrm{pre}_o$. *Applying* action o to state s, $s[\![o]\!]$ for short, yields state s', where $s'(a) \coloneqq \mathrm{eff}_o(a)$, if $a \in \mathrm{dom}(\mathrm{eff}_o)$ and $s'(a) \coloneqq s(a)$, otherwise. Finally, $\mathcal{I}$ is the *initial state* of Π and $\mathcal{G}$ a partial state called *goal condition*. A state s_* is a *goal state* if $s_* \models \mathcal{G}$. Let Π be a planning task. A *plan* $\pi = \langle o_0, \ldots, o_{n-1} \rangle$ is a sequence of applicable actions that *generates* a sequence of states $s_0, \ldots, s_n$, where $s_0 = \mathcal{I}$, s_n is a goal state, and $s_{i+1} = s_i[\![o_i]\!]$ for every $i \in [n-1]$. Furthermore, we let $\pi(i) \coloneqq o_i$ and denote by $|\pi|$ the *length* of a plan π. We denote the set of all plans by $\mathrm{Plans}(\Pi)$ and the set of all plans of length at most ℓ by $\mathrm{Plans}_\ell(\Pi)$ and call it occasionally *plan space* as done in the literature (Russell and Norvig 1995). A plan π is *optimal* if there is no plan $\pi' \in \mathrm{Plans}(\Pi)$ where $|\pi'| < |\pi|$. The notion naturally extends to bounded-length plans. A *brave* action by $\mathcal{BC}_\ell(\Pi) \coloneqq \bigcup_{\pi \in \mathrm{Plans}_\ell(\Pi)} \bigtriangledown(\pi)$ and *cautious* action by $\mathcal{CC}_\ell(\Pi) \coloneqq \bigcap_{\pi \in \mathrm{Plans}_\ell(\Pi)} \bigtriangledown(\pi)$. The symbol $\bigtriangledown(\cdot)$ converts sequences into sets to drop time-points. Deciding or counting plans is computationally hard. The problem that asks to decide whether there exists a plan of length at most ℓ, is PSPACE-complete (Bylander 1994). We say that a plan is *polynomially bounded* if we restrict the length to be polynomial in the instance size, i.e., the length ℓ of Π is bounded by $\ell \leq \|\Pi\|^c$ for some constant c, where $\|\Pi\|$ is the encoding size of Π. In that case, the complexity drops, namely, plan existence is NP-complete (Bylander 1994).

Answer-Set Programming (ASP). We explain notions only for ground answer-set programs due to tight space constraints. Let m and n be non-negative integers and a, $b_1, \ldots, b_m, c_1, \ldots, c_n$ be distinct propositional atoms. A

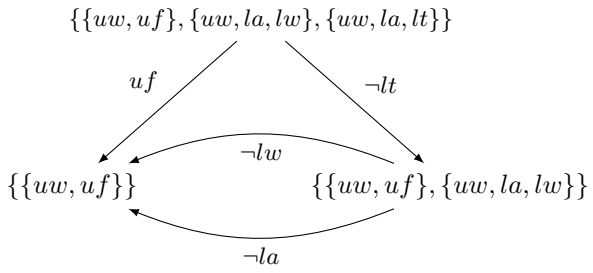


Figure 2: Navigating the plan space via selected navigation steps.

rule r is of the form $a \leftarrow b_1, \ldots, b_m, \sim c_1, \ldots, \sim c_n$ or $\leftarrow b_1, \ldots, b_m, \sim c_1, \ldots, \sim c_n$. Intuitively, the former means that a must be true if all atoms $b_1, \ldots, b_m$ are true and there is no evidence that c_1, ..., or c_n is true. The latter, called *constraint*, enforces that one of the atoms b_1, ..., b_m is false or one of the atoms c_1, ..., c_n is true. We let $H_r \coloneqq \{a\}$ or $\emptyset$, respectively, $B_r^+ \coloneqq \{b_1, \ldots, a_m\}$, and $B_r^- \coloneqq \{c_1, \ldots, c_n\}$. A *program* P consists of a set of rules. An interpretation $M \subseteq \mathrm{vars}(P)$ *satisfies* a rule r if $(H_r \cup B_r^-) \cap M \neq \emptyset$ or $B_r^+ \not\subseteq M$. M is a *model* of P if M satisfies every rule $r \in P$. The *(GL) reduct* of P with respect to M is defined as $P^M \coloneqq \{H_r \leftarrow B_r^+ \mid B_r^- \cap M = \emptyset\}$. Then, M is an *answer-set* of P if M is a model of P such that no interpretation $N \subsetneq M$ is a model of P^M (Gelfond and Lifschitz 1988). A popular tool is `clingo`[1], which allows to state non-ground programs, ground them and compute ground answer-sets (Gebser et al. 2019).

Solving Planning with ASP. We use the `plasp` tool in version 3.1.1[2] (Dimopoulos et al. 2019) to translate planning instances from PDDL to ASP. The tool implements a chain comprising of parsers and a default translator that constructs an ASP encoding of the planning task, which is then grounded and used to compute answer sets. Since we are primarily interested in the plan space on the ground level, we take advantage of the grounded ASP encoding. The ASP encoding is parameterized by a horizon H, which serves as a bound of the plan length. With this, `plasp` constructs an encoding which enables search for plans of exactly length H. We refer to this as the *exact* encoding. While `plasp` can compute plans itself as well, we employ it only to construct the ASP encoding. This approach enables us to employ ASP facets (encoding-dependent) to planning.

3 Navigating with Facets

We mainly employ facets to navigate solution spaces. Facets are known in ASP (Fichte, Gaggl, and Rusovac 2022) and have recently been introduced to classical planning (Speck et al. 2025). In the usual definition, a facet is part of a solution, but still enables freedom in the solution space and is therefore not contained in all solutions. We extend and adapt this idea to the plan space for a given planning task, as then we can navigate through the plan space by adding or removing facets. We formalize this concept below.

Definition 1. *Let* $\Pi = \langle \mathcal{A}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ *be a planning task. Then, for any action* $a \in \mathcal{O}$ *and integer* j*, let* $a@j$ *and* $\neg a@j$ *be a* facet of Π *if there is a* witnessing plan *where* a *is at the* j*-th position and some plan where* a *is not at* j*, respectively. We say* a *or* $\neg a$*, respectively, is a* facet of Π *if there is some plan containing* a *and some plan without* a*.*

Note that therefore facets are consequences of planning tasks that appear in some (but not all) plans. Indeed, it appears useless to navigate along non-facets, as those do not contribute to a better understanding of the plan space since adding or removing non-facets either does not change solutions or makes the instance unsatisfiable. To this end, we require the notion of routes for plan navigation. A *route* is a finite sequence of navigation steps, which keeps track of how we move within the solution space. We use a sequence instead of a set, as sequences preserve the order of steps, thus allowing to trace back or undo navigation steps.

Definition 2. *Let* $\Pi = \langle \mathcal{A}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ *be a planning task. A* route δ *is a finite sequence* $\langle f_1, \ldots, f_n \rangle$ *consisting of facets* $f_i \in \mathcal{F}(\Pi)$*, which denote* $n \in \mathbb{N}$ *navigation steps over* Π*. By* Δ^Π *we denote all possible routes over* Π*. The set* $\mathrm{Plans}(\Pi^\delta)$ *of plans for* Π *under* δ *contains only those plans in* $\mathrm{Plans}(\Pi)$ *that are witnessing every* f_i *in* δ*.*

Navigation enables notions to select plans of a planning task Π within the *plan space*. Facets *restrict plan spaces* under meaningful assumptions, i.e., those belonging to some but not to all plans. Indeed, facets are those consequences that are brave but not cautious and are therefore in $\mathcal{BC}(\Pi) \setminus \mathcal{CC}(\Pi)$. This retains planning flexibility, as we can find assumptions that still preserve plan diversity.

Example 2. *Consider Task* Π_1 *from Example 1 and recall that we had three solutions, namely:* $\{\langle$*unload-wing,unload-flaps*$\rangle$, $\langle$*unload-wing, load-aileron, load-wing*$\rangle$, $\langle$*unload-wing; load-aileron; load-tip*$\rangle\}$*. Due to space reasons, we abbreviate the actions by underlined characters, e.g.,* uw *instead of* unload-wing*. Observe that* uw *occurs on any plan and is therefore not a facet. However,* lt *is a facet and we can therefore navigate from the top to bottom right in Figure 2. If we then add* ¬lw*, we end up with a single plan (bottom left). Note that there is a faster route* ⟨uf⟩ *that is equivalent by a direct transition from the top to bottom left.*

3.1 A Practical Plan Navigation Tool

Our `PlanPilot`[3] tool takes as input a PDDL (McDermott 2000) or a SAS$^+$ description (Bäckström and Nebel 1995; Helmert 2009) of a planning task and allows for interactive navigation of the solution space by the user. Figure 3 illustrates the individual components of `PlanPilot` and their interaction. First, we employ `plasp` to encode the planning task in ASP. This encoding is parameterized by a horizon H, which restricts the plan length, and a parameter that allows switching between the exact encoding of `plasp` and a new *bounded* encoding. The latter enables plans of length *up to*

[1] https://github.com/potassco/clingo

[2] https://github.com/potassco/plasp

[3] https://github.com/abcorrea/planpilot

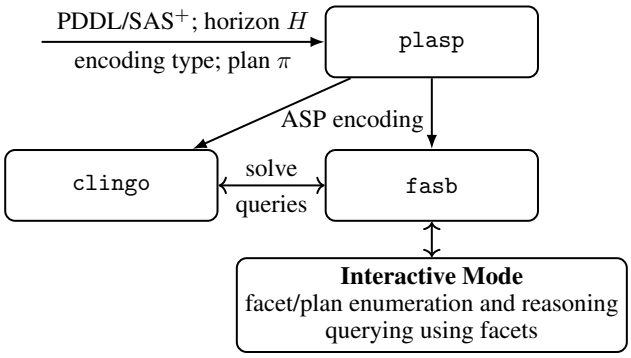


Figure 3: Illustration of PlanPilot's components.

H, which is useful if the user does not have specific requirements on how long plans should be. The ASP encoding is passed to fasb (and thereby to its internal solver clingo), which enables interactive facet reasoning.

In interactive mode, the user can query PlanPilot for the number of available facets or a list of these facets. In our ASP encoding, facets are meaningful actions that occur in some, but not all solutions. More specifically, the occurrence of every action $a \in \mathcal{A}$ at a given time step $1 \leq t \leq H$ is a possible 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.

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 following rule to our encoding:

$$\texttt{occurs\_sometime}(a) \leftarrow \texttt{occurs}(a, t), t > 0.$$

We refer to this variant as the *abstract time steps* encoding. If a facet $\texttt{occurs\_sometime}(a)$ 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, which could be beneficial in, e.g., domain debugging (Lin, Grastien, and Bercher 2023; Gragera et al. 2023).

Complexity. Reasoning over facets is significantly easier than reasoning over solutions (Fichte, Gaggl, and Rusovac 2022). Therefore, it is advisable to opt for counting and enumeration of facets, instead of plans, as the planning tasks and horizons get larger. At the same time, enabling more facets limits the solution space considered by fasb, making reasoning on that space more efficient. 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. We next exemplify two navigation modes that illustrate different strategies for interaction with PlanPilot.

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 gives high flexibility to the user to navigate the plan space according to their needs.

Technical aspects. PlanPilot is implemented as a Python script that controls all involved components, i.e., it creates the ASP encoding of the input planning task using plasp and passes it to fasb, which in turn solves the model using clingo, and starts the interactive mode. 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.

4 Navigating the Beluga Challenge

We next introduce the specifics of the Beluga application domain and describe important plan properties that are desired by the engineers to optimize the on-site logistics and robustness to unforeseen changes, for example in the flight schedule. Several criteria can be captured exactly using facet reasoning. We showcase how to employ facets to navigate and understand the plan space and give concrete explanations for a set of queries that address the most relevant properties.

4.1 The Beluga Problem

First, we summarize the planning encoding of the Beluga problem as required to follow our contribution. For full details, we refer to the competition description (Tuples 2025).

An instance of the Beluga problem consists of a series of *Beluga* cargo airplanes $\langle B_1, \ldots, B_n \rangle$ that arrive at the facility in the given order. Every Beluga delivers a (possibly empty)

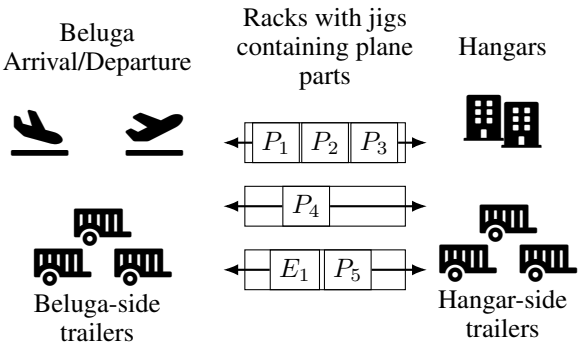


Figure 4: Visualization of the Beluga domain, where incoming and outgoing Beluga planes are loaded with airplane parts. These parts are moved around on jigs, which can be transported by trailers to the storage racks or to the production lines in the hangars. Only the outermost jigs on the left or right side of a rack can be accessed. In the example shown, the jig with part P_4 on the middle rack can be accessed from both sides. However, to access the jig with part P_2 on the upper rack, one must first remove the jig with part P_1 or the jig with part P_3 from the rack.

set of airplane parts that are needed at the production lines in which airplanes are assembled. Every part is placed on a *jig* J that has a size $|J|$. The jigs delivered by a Beluga B have to be unloaded in a specific order $\langle J_1^B, \ldots, J_k^B \rangle$ into *trailers* that transport the jigs to a set of storage racks $\{R_1, \ldots, R_m\}$. Each *rack* R has a fixed *capacity* $|R|$ to fit a number of jigs, such that at every point $\sum_{J \in R} |J| \leq |R|$. The content of a rack can be accessed from both sides, but only the two outermost jigs can be taken out. Jigs that have just been unloaded from a Beluga are placed on the *Beluga side* of a rack. On the other side, the *factory side*, jigs are transported by trailers to the production lines that are located in several *hangars* $\{H_1, \ldots, H_o\}$. Jigs that have been *emptied* in a hangar are transported back to the racks and eventually loaded into a Beluga that ships them to another facility. The planning encoding distinguishes trailers on the Beluga side from those on the factory side and trailers cannot switch sides.

We illustrate a Beluga problem in Figure 4, with the sequence of Beluga flights on the left, the racks in the middle, and the hangars with the production lines on the right. We distinguish full jigs P_x and empty jigs E_x. An important observation is that every jig arrives non-empty with a Beluga, is transported to a hangar, where it is emptied. Afterwards, the empty jig is transported back to a Beluga and leaves the site. So the shortest path for individual jigs moves full ones exclusively from left to right in the illustration, and empty ones from right to left. In an ideal scenario, these shortest path for every jig are combined into an optimal plan. Due to space restrictions and depending on the order of arrival of jigs, this ideal scenario might not be feasible.

The on-site logistics are constrained by the capacity of the racks and the number of trailers on each side of the racks. Since jigs are transported in both directions, and jigs in the racks can only be taken from either end of a rack, it is essential to place the jigs appropriately to prevent unnecessary *swaps*, where a jig is temporarily taken out of a rack and put back into the same or another rack on the same side, to free space or gain access to otherwise blocked jigs. Every swap increases the plan length from the optimum, where all jigs are moved on shortest paths, by two. For enhancing the robustness in case the flight schedule changes unexpectedly, it is desirable to keep one rack empty at all times. Moreover, a rack R might be inaccessible due to maintenance operations, in which case the remaining jigs $J \in R$ can be taken out of the rack, but no new jigs can be placed in it. Some soft constraints that engineers have on the plans are that smaller jigs are to be put into smaller racks, empty jigs are placed on the Beluga side of a rack and full jigs on the factory side, which facilitates logistics.

4.2 Beluga Explanation Challenge

With the tight space constraints at the assembly facilities and limited mobility of the jigs, plans obtained from an automated planning system are unlikely to be robust enough to ensure executability under uncertain flight schedules. Fully representating the above constraints in the planning model is not feasible, either. Thus, the practical approach is to generate a solution with a planner, ask for explanations for certain decisions, i.e., actions taken in the plan or the implications of alternatives, and possibly refine the plan accordingly.

We adopt the setting from the Beluga challenge where a solution has been computed by an automated planner and is passed to the plan explanation tool (explainer) along with a query that should be answered by the system. The following queries have been identified as relevant for the application, which we adopt from the challenge as well:

Q1 Why is jig x loaded on rack A instead of another rack B?

Q2 Why load jig B on rack D instead of loading jig C on rack A?

Q3 Why not load jig C on rack A before loading jig B on rack D?

Q4 How can we reduce the number of swaps?

Q5 What is the impact of removing rack A for maintenance?

Q6 How can we keep one rack empty all the time?

The task of the explainer is to (1) determine the feasibility of the alternative solution, (2) give an insight into the consequences that the alternative solution would entail, and (3) present a comparison of the chosen plan with alternative scenarios that were considered but not selected.

We group the queries into three sets, where Queries 1–3 ask about concrete changes in the given plan, Query 4 is specific to swaps, and Queries 5–6 deal with rack removal.

4.3 Q1–Q3: Feasibility of Alternative Solutions

We start by our approach to deal with Queries 1–3. Along with the query, we get a concrete plan $\pi = \langle a_1, \ldots, a_n \rangle$ that allows us to identify the action(s) in question. For Query 1, for example, it is an action that loads jig x into rack A on one side of the rack. This can be ambiguous in case of swaps or if the jig is placed on rack A in both its full and empty state. Since the challenge instructions do not provide any further details, we opt for the first such load action that occurs on the

plan. Queries 2–3 ask for a change in order of two occurring actions. Again, we resolve potential ambiguities by choosing the first time the respective actions occur on the plan.

To answer the query, we enforce the plan prefix $\pi_p = \langle a_1, \ldots, a_{i-1} \rangle$ up to (excluding) the first action a_i in question by navigating along the route $\delta_p = \langle a_1@1, \ldots, a_{i-1}@i-1 \rangle$. Then, we check if the alternative action a' for step i is a facet, i.e., if $\delta_p^i = \delta_p \circ a'@i$ is a route. If that is the case, then taking the alternative action is possible and we obtain a new solution by completing the navigation until a plan is found. To analyze the resulting solution space, we can also count the number of plans $|\operatorname{Plans}(\Pi^{\delta_p^i})|$, which provides insights in the flexibility for re-planning due to unforeseen events.

To stick as close to the original plan as possible, we can, before performing navigation step $a'@i$, do steps $a_k@k$ for all $k > i$ where a_k does not directly conflict with a_i or a'. Here, conflicting actions are those that operate on the same racks and can no longer be applied in their original step. For Q1, for example, this is an action unloading jig x from rack A on the opposite size, as the jig is now on rack B.

In case the alternative action or order is not feasible, we can increase the horizon beyond the length of the given solution to see if a longer plan exists that satisfies the requested change. This is for example the case if the change implies more swaps, which introduces redundant actions that move a jig around. To avoid re-computation, we can increase the horizon for the facet reasoning right away by 2x the number of swaps we are willing to sacrifice for the alternative plan.

The main property that captures solution quality is the number of swaps. Like just described, we can impose a hard limit on the number of swaps an alternative action introduces using the horizon. We describe a more fine-grained approach in Section 4.4. When the aim is to keep a rack free at all time, we can identify an initially free rack R and perform navigation steps $\neg a$ for all actions a that load a jig into R. Denote the set of these action by $\mathcal{O}^{lR}$. We select the smallest initially empty rack R_{min} and check during navigation if one such action becomes cautious, i.e., $\mathcal{CC}_\ell(\Pi) \cap \mathcal{O}^{lR} \neq \emptyset$. If that is the case, then no solution is feasible (with the given horizon l) that keeps any rack empty at all times. The case of rack maintenance, as well as placing only jigs of a certain size on a rack, can be analyzed analogously.

All proposed reasoning steps are efficiently solvable with a single execution of `fasb` and a series of navigation steps.

4.4 Q4: Minimizing the Number of Swaps

A swap is a sequence of actions, possibly interleaved by other actions, that takes a jig out of a rack to make another jig accessible and either puts it back into the same rack afterwards or places it in another rack on the same side. This adds redundant actions to a plan, which are expensive to execute on site and should be avoided if possible. Swaps can be detected effectively by checking for the occurrence of specific actions in the plan. If a full jig is taken out of a rack on the Beluga side, then this implies a swap because such jigs will have to be transported to a hangar at some point, taking them out on the Beluga side is only required to make another jig accessible. The second type of action that implies a swap is an empty jig being taken out of a rack on the hangar side, for analogue reasons. In general, every swap can be detected by the occurrence of a single action in the plan. We denote the set of these actions that imply a swap by $\mathcal{O}^{\text{swap}}$.

With single actions identifying swaps, we can employ facet reasoning to check if swaps can be prevented, or are implied by some actions taken in a plan. In the last section we explained an approach that reasons about the minimum number of swaps by checking if the planning task is solvable with different horizons. While this is a viable approach, it involves solving several planning tasks. We suggest to use facet reasoning instead, using a slightly higher horizon than necessary to fit the given plan. This can not only be more efficient, but also provides the flexibility to stay close to the given plan and explore alternatives via navigation.

Given the original plan π, we can either fix the actions up to the first swap action, as described before, or start from scratch without keeping actions from π. In both cases, we navigate the plan space systematically by performing navigation steps $\neg a$ for the actions $a \in \mathcal{O}^{\text{swap}}$. Once a swap action becomes cautious, i.e., $\mathcal{CC}_\ell(\Pi) \cap \mathcal{O}^{\text{swap}} \neq \emptyset$, we know that a swap is required for the current route. This search over routes can be performed in a branch-and-bound style, where we check for the number of swaps B in plans iteratively and skip completing routes when more than B are already implied. With this, we can stop the search once we found a plan without swaps, and still analyze other plan properties or give preference to desired actions during navigation.

4.5 Q5/Q6: Handling Rack Removal

We approach the increase of redundancy in plans by rack removal similar to before by a pre-processing step that identifies the actions that put some jig into one of the respective racks. Then, we forbid these actions using facets.

As before, in every navigation step, we check if such an action becomes a cautious consequence. If this is the case, we know that at least one action is needed that puts a jig into that rack. Then, we do not need to search over facets. Thereby, we can answer the question on impact of removing rack as well as how to keep one rack empty all the time. This is more flexible than, for example, removing the rack from the task description and checking if a plan exists.

5 Experiments

We considered the 48 solvable instances of the Beluga explainability challenges. In line with the challenge, we tested the interaction and explainability based on a given plan. To do this, we used an optimal planner to determine an optimal plan for each instance. More precisely, we ran symbolic bidirectional search (Torralba et al. 2017) with the `SymK` planner (Speck, Seipp, and Torralba 2025). We also tried other search algorithms, such as A* (Hart, Nilsson, and Raphael 1968) with the landmark cut (LM-Cut) heuristic (Helmert and Domshlak 2009) using `FastDownward` (Helmert 2006). However, these alternatives performed worse than symbolic search for finding a single optimal plan. In total, we managed to find an optimal plan for 31 instances, which, together with the induced plan and cost bound, formed our benchmark set.

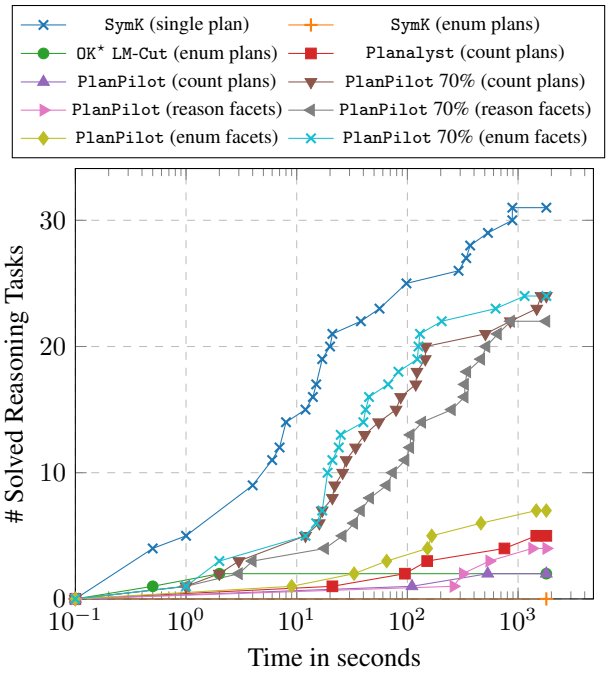


Figure 5: Inverted cactus plot showing the runtime necessary to solve or reason over a given number of instances. The x-axis shows runtime in seconds on a log scale. The number of solved instances is on the y-axis. Planners and configurations are coded by color.

For all experiments, we used a memory and time limit of 3.5 GB and 30 min per instance. Source code, benchmarks, and data are available online (Gnad et al. 2025).

To test PlanPilot, we invoked it with a query to count the plans, enumerate the facets, and report the significance of the facets. As the input model for PlanPilot, we used the ground SAS$^+$ model produced by the FastDownward translator (Helmert 2009). Furthermore, for each instance, we ran all modes of PlanPilot with a different degree of freedom in the plan space. This was achieved by enforcing in the model that some actions of the known optimal plan had already been selected by a user. In other words, the configurations PlanPilot X% indicated that X percent of the time steps had no assigned action. These configurations simulate a user interacting and navigating in the plan space to study the practical reasoning capabilities of PlanPilot on the Beluga domain. For comparison, we considered two top-k planners in their default and recommended configurations: Symk (Speck, Mattmüller, and Nebel 2020) and OK* (Katz et al. 2018), which enumerate all solutions, and Planalyst (Speck et al. 2025), which generates a d-DNNF (Darwiche and Marquis 2002) representing the plan space, allowing for subsequent reasoning, such as counting. However, both existing paradigms, top-k planning and d-DNNF compilation, require a costly precomputation phase before any reasoning over the plan space becomes feasible.

5.1 Overall Performance

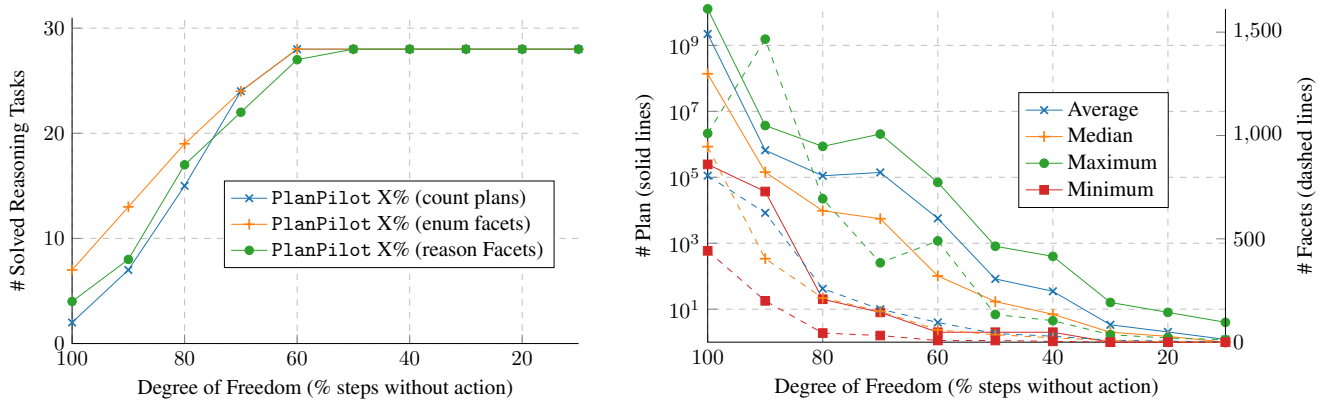
Figure 5 shows the number of problems solved over time. At one extreme, we see that finding a single optimal solution SymK (single plan) is empirically the most performant, which is not surprising. At the other extreme, enumerating all optimal plans is extremely challenging for the Beluga domain. SymK (enum plans) fails to enumerate all optimal plans for any of the problems, and OK* with the LM-Cut heuristic only succeeds in two. This is due to the vast number of optimal plans in the Beluga domain, which we discuss in Section 5.2.

Counting the number of optimal plans is more feasible, as shown by the "(count plan)" configurations. Here, we can see that the d-DNNF transformation of Planalyst performs favorably over our PlanPilot approach based on ASP overall. However, when considering the Planalyst configurations (enum facets/reason facets), we can see that the proposed approach enables interaction with the plan space for more problems than enumerating or counting all optimal plans. Furthermore, when a partial plan is present and fixed, interaction with the plan space via facets becomes feasible in many more instances, as shown by the Planpilot 70% configuration (here, 30% of the time steps have determined actions, and 70% of the time steps can be determined by a user). Counting the remaining plans while taking the partial plans into account also becomes more feasible.

Interestingly, in some instances, it is possible to count the plans and enumerate the facets, but reasoning about the significance of the facets is not feasible within the set resource limits. This performance difference occurs because determining the significance of a particular facet f requires counting the facets in the current situation and then counting them again after fixing facet f. Consequently, computing the significance of all facets can incur notable overhead compared to merely enumerating them. Furthermore, counting the number of optimal plans is more efficient in scenarios with a small number of remaining plans. Figure 6a provides a more detailed picture of this behavior by visualizing the number of problems for which Planpilot can count plans, enumerate facets, or reason about the significance of facets for different degrees of freedom in the plan space.

5.2 Number of Facets and Plans

Figure 6b shows the average, median, maximum, and minimum number of facets and plans across the Beluga instances for different degrees of freedom (i.e., X% of time steps without assigned action), as found by PlanPilot (and, for X = 100%, by all planners). Note that the number of solved tasks increases as the degrees of freedom decrease, which explains the non-monotonic behavior of the curves. Overall, we can see that the number of plans is vast for high degrees of freedom. However, this number decreases significantly once actions are scheduled at certain time points. By comparison, the number of facets is multiple orders of magnitude lower because they do not represent optimal plans, i.e., sequences of actions. They indicate whether, for a particular position in the plan, there exists at least one plan that includes a given action at that position and at least one plan that does not. The significance value of a facet indicates how much flexibility



(a) The number of instances (y-axis) for which `PlanPilot` was able to count plans, list facets, or reason about the facet significance.

(b) The number of plans (left y-axis) and facets (right y-axis) as determined by our experiments.

Figure 6: Visualization of the performance of `PlanPilot` (left) and the number of plans and facets (right) for a decreasing degree of freedom on the x-axis.

remains in subsequent choices once that particular facet (i.e., whether to take a specific action at a specific time) is selected. The comparatively small number of facets makes interacting with and navigating in the plan space more accessible and understandable than if users were overwhelmed with millions or even billions of plans. Furthermore, the enumeration of those large numbers of plans is challenging, as shown above.

6 Conclusion

We present a solution to the 2025 Beluga AI Explainability Challenge. This concrete planning-in-the-wild use case, illustrates a successful deployment of KR techniques to a practical AI planning application. Our `PlanPilot` tool advances the state-of-the-art solving systems to comprehend large plan spaces allowing to make sense of alternative solutions or ensuring alternative solutions to increase resilience. `PlanPilot` builds on multiple reasoning techniques allowing to interactively output plans, filter plans, restrict plans, restrict flexibility in plans. Moreover, `PlanPilot` also supports counting plans, counting flexibility in actions, and estimate effects iteratively. In our system, we take advantage of the underlying ground-ASP encoding, which 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. Thereby, we establish a practical approach to navigate plan spaces iteratively and interactively and make interactive plan space exploration more precise.

In the Beluga use case, we primarily rely on facets. However, `PlanPilot` supports additional techniques to understand the plan space, which we believe will be useful for other settings. In more detail, we can combine facets and counting plans. This allows us to construct reasoning modes that allow for navigating plan spaces in multiple ways: *exploring* and *targeting*. 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.

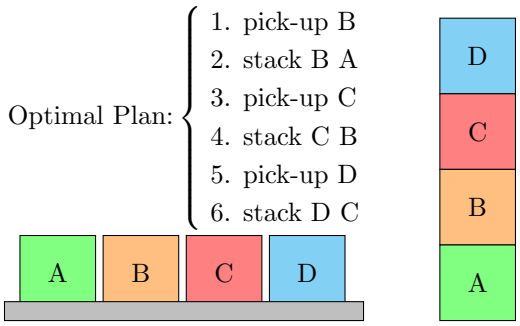
We believe that beyond the time-stamped or global action facets, it is interesting to consider partial-order planning in the navigation, i.e., obtaining (minimal) partially ordered plans natively without adding implications to simulate all steps. We expect support for loopless plans to be beneficial 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. We are interested in evaluating our tool on a larger set of practical instances and investigate limitations of facet-based reasoning as well as constructing visual navigation tools that take advantage of `PlanPilot`.

A Blocks World Demo

We show an interactive run of `PlanPilot`, using an instance of the Blocksworld domain. Blocksworld (Slaney and Thiébaux 2001) is a well-known toy problem in the planning community. It asks to arrange blocks in towers on a table, no block may be on top of itself or on/under two blocks, as illustrated in Figure 7. The example has 4 blocks and an optimal plan with 6 actions. `PlanPilot` takes as input a PDDL task together with the horizon H (the bound on the plan length), and the type of encoding (exact or bounded). In our example we use $H = 12$ with the bounded encoding, which allows all plans up to twice the optimal plan length:

```
In: ./planpilot.py -d domain.pddl
                   -i probBLOCKS-4-0.pddl
                   --horizon 12
                   --plasp-translate
                   --encoding bounded
```

Once the interactive mode is started, we can run a series of queries or commands. For example, the commands `#!` and `#?` ask for the number of plans, respectively facets:



(a) Initial state. (b) Goal.

Figure 7: Blocksworld task.

```
In: #!
18697
In: #?
360
```

We can also ask for a list of all facets using `?`:

```
In: ?
occurs(action(("pick-up",constant("a"))),1)
occurs(action(("put-down",constant("c"))),7)
   ...
```

This list can be augmented by information on how much the activation of each facet restricts the remaining facets, e.g., this shows a reduction of 32.78% and 242 remaining facets when activating the first listed facet:

```
In: #??
0.3278 242
    occurs(action(("pick-up",constant("a"))),1)
0.0444 344
   ~occurs(action(("pick-up",constant("a"))),1)
0.6667 120
    occurs(action(("put-down",constant("c"))),7)
0.0056 358
   ~occurs(action(("put-down",constant("c"))),7)
...
```

The command `#!!` behaves similarly, but shows the reduction in the number of remaining *plans* for each facet.

Alternatively, we can also iteratively query how many facets and how many plans remain, after activating a facet. A facet is activated via `+ FACET-NAME`:

```
In: + occurs(action(("pick-up",constant("a"))),1)
In: + occurs(action(("put-down",constant("c"))),7)
In: #?
32
In: #!
25
```

Finally, with the command `!` we can output all plans that remain, analyze them and select from them.

```
In: !
solution 1:
occurs(action(("pick-up",constant("a"))),1)
occurs(action(("put-down",constant("a"))),2)
occurs(action(("pick-up",constant("b"))),3)
occurs(
   action(("stack",constant("b"),constant("a"))),4)
occurs(action(("pick-up",constant("c"))),6)
occurs(action(("put-down",constant("c"))),7)
occurs(action(("pick-up",constant("c"))),8)
occurs(
   action(("stack",constant("c"),constant("b"))),9)
occurs(action(("pick-up",constant("d"))),10)
occurs(
  action(("stack",constant("d"),constant("c"))),11)
solution 2:
...
```

After activating the two facets above, all resulting plans will have action *pick-up*(a) as their first step and action *put-down*(c) as their seventh step. We can continue to enforce (or prohibit) more facets to further condition the set of plans. Note that in some plans time steps are skipped to allow for plans shorter than the bound. For example, the plan shown has only ten actions and time step 5 is empty.

Overall, all queries and facet (de)activations are fast. In our example, all operations only take split seconds. The runtime of our tool is usually dominated by the time it takes `clingo` to compute the requested plans. For some domains this can be a challenge (Dimopoulos et al. 2019).

In particular, `clingo` may need much more time if we ask it to enumerate *all plans* up to a given bound.

If we add an additional command line argument, we can utilize the abstract time steps encoding:

```
In: ./planpilot.py -d domain.pddl
                   -i probBLOCKS-4-0.pddl
                   --horizon 12
                   --plasp-translate
                   --encoding bounded
                   --abstract-time-steps
```

It turns out that if the first action is *pick-up*(a) and some action of the plan is *stack*(d, a), there is only a single plan left. This can be easily discovered by our tool:

```
In: + occurs(action(("pick-up",constant("a"))),1)
In: #!
1469
In: #??
1.0000 0 occurs_sometime(action(("stack",constant("
    d"),constant("a"))))
...
```

We show the remaining plan after requiring that the plan stacks d on a, which concludes our use case:

```
In: +  occurs_sometime(action(("stack",constant("d"
    ),constant("a"))))
In: #!
1
In: !
!
solution 1:
occurs(action(("pick-up",constant("a"))),1) occurs(
    action(("put-down",constant("a"))),2) occurs(
    action(("pick-up",constant("d"))),3) occurs(
    action(("stack",constant("d"),constant("a")))
    ,4) occurs(action(("unstack",constant("d"),
    constant("a"))),5) occurs(action(("put-down",
    constant("d"))),6) occurs(action(("pick-up",
    constant("b"))),7) occurs(action(("stack",
    constant("b"),constant("a"))),8) occurs(action
    (("pick-up",constant("c"))),9) occurs(action((
    "stack",constant("c"),constant("b"))),10)
    occurs(action(("pick-up",constant("d"))),11)
    occurs(action(("stack",constant("d"),constant(
    "c"))),12) ...
found 1
```

Acknowledgments

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, and by TAILOR, a project funded by the EU Horizon 2020 research and innovation programme under grant agreement no. 952215. The work has received funding from the Swiss National Science Foundation (SNSF) as part of the project "Unifying the Theory and Algorithms of Factored State-Space Search" (UTA), the Austrian Science Fund (FWF), grants 10.55776/J4656, and ELLIIT funded by the Swedish government. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

References

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS^+ planning. *Computational Intelligence* 11(4):625–655.

Boddy, M.; Gohde, J.; Haigh, T.; and Harp, S. 2005. Course of action generation for cyber security using classical planning. In Biundo, S.; Myers, K.; and Rajan, K., eds., *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005)*, 12–21. AAAI Press.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence* 69(1–2):165–204.

Calimeri, F.; Faber, W.; Gebser, M.; Ianni, G.; Kaminski, R.; Krennwallner, T.; Leone, N.; Maratea, M.; Ricca, F.; and Schaub, T. 2020. ASP-Core-2 input language format. *Theory Pract. Log. Program.* 20(2):294–309.

Darwiche, A., and Marquis, P. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research* 17:229–264.

Darwiche, A. 2001. Decomposable negation normal form. *Journal of the ACM* 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.

Edelkamp, S. 2006. On the compilation of plan constraints and preferences. In Long, D.; Smith, S. F.; Borrajo, D.; and McCluskey, L., eds., *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006)*, 374–377. AAAI Press.

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 Conitzer, V., and Sha, F., eds., *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*, 9818–9826. AAAI Press.

Eiter, T., and Geibinger, T. 2023. Explaining answer-set programs with abstract constraint atoms. In Elkind, E., ed., *Proceedings of the 32nd International Joint Conference on Artificial Intelligence (IJCAI 2023)*, 3193–3202.

Fichte, J. K.; Gaggl, S. A.; Hecher, M.; and Rusovac, D. 2024. IASCAR: incremental answer set counting by anytime refinement. *Theory Pract. Log. Program.* 24(3):505–532.

Fichte, J. K.; Fröhlich, N.; Hecher, M.; Lagerkvist, V.; Mahmood, Y.; Meier, A.; and Persson, J. 2025. Facets in argumentation: A formal approach to argument significance. In Kwok, J., ed., *Proceedings of the 34th International Joint Conference on Artificial Intelligence (IJCAI 2025)*.

Fichte, J. K.; Gaggl, S. A.; and Rusovac, D. 2022. Rushing and strolling among answer sets – navigation made easy. In Honavar, V., and Spaan, M., eds., *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI 2022)*, 5651–5659. AAAI Press.

Fichte, J. K.; Hecher, M.; and Nadeem, M. A. 2022. Plausibility reasoning via projected answer set counting - A hybrid approach. In Raedt, L. D., ed., *Proceedings of the 31st International Joint Conference on Artificial Intelligence (IJCAI 2022)*, 2620–2626. ijcai.org.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2012. *Answer set solving in practice*. Morgan & Claypool Publishers.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2019. Multi-shot ASP solving with clingo. *Theory and Practice of Logic Programming* 19(1):27–82.

Gebser, M.; Kaufmann, B.; and Schaub, T. 2009. Solution enumeration for projected boolean search problems. In van Hoeve, W. J., and Hooker, J. N., eds., *Proceedings of the 6th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2009)*, 71–86. Springer-Verlag.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In Kowalski, R., and Bowen, K., eds., *Proceedings of International Logic Programming Conference and Symposium (ICLP/SLP 1988)*, 1070–1080. MIT Press.

Gerevini, A. E., and Long, D. 2005. Plan constraints and preferences in PDDL3. Technical Report R. T. 2005-08-47, University of Brescia, Department of Electronics for Automation.

Ghallab, M.; Nau, D.; and Traverso, P. 2025. *Acting, Planning, and Learning*. Cambridge University Press. To appear.

Gnad, D.; Hecher, M.; Gaggl, S.; Rusovac, D.; Speck, D.; and Fichte, J. K. 2025. Code, benchmarks and data for the KR 2025 paper "Interactive Exploration of Plan Spaces". https://doi.org/10.5281/zenodo.16563394.

Gragera, A.; Fuentetaja, R.; Olaya, Á. G.; and Fernández, F. 2023. A planning approach to repair domains with incomplete action effects. In Koenig, S.; Stern, R.; and Vallati, M., eds., *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS 2023)*, 153–161. AAAI Press.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 162–169. AAAI Press.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence* 173:503–535.

Janhunen, T., and Niemelä, I. 2016. The answer set programming paradigm. *AI Magazine* 37(3):13–24.

Kabir, M.; Everardo, F. O.; Shukla, A. K.; Hecher, M.; Fichte, J. K.; and Meel, K. S. 2022. ApproxASP – a scalable approximate answer set counter. In Honavar, V., and Spaan, M., eds., *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI 2022)*, 5755–5764.

Katz, M., and Sohrabi, S. 2020. Reshaping diverse planning. In Conitzer, V., and Sha, F., eds., *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*, 9892–9899. AAAI Press.

Katz, M.; Sohrabi, S.; Udrea, O.; and Winterer, D. 2018. A novel iterative approach to top-k planning. In de Weerdt, M.; Koenig, S.; Röger, G.; and Spaan, M., eds., *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS 2018)*, 132–140. AAAI Press.

Lin, S.; Grastien, A.; and Bercher, P. 2023. Towards automated modeling assistance: An efficient approach for repairing flawed planning domains. In Chen, Y., and Neville, J., eds., *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence (AAAI 2023)*, 12022–12031. 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 – Version 1.2. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, Yale University.

McDermott, D. 2000. The 1998 AI Planning Systems competition. *AI Magazine* 21(2):35–55.

Palacios, H.; Bonet, B.; Darwiche, A.; and Geffner, H. 2005. Pruning conformant plans by counting models on compiled d-DNNF representations. In Biundo, S.; Myers, K. L.; and Rajan, K., eds., *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS 2005)*, 141–150. AAAI Press.

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 Marquis, P.; Ortiz, M.; and Pagnucco, M., eds., *Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning (KR 2024)*, 642–653.

Russell, S., and Norvig, P. 1995. *Artificial Intelligence — A Modern Approach*. Prentice Hall.

Schmidt, J.; Maizia, M.; Lagerkvist, V.; and Fichte, J. K. 2025. Complexity of faceted explanations in propositional abduction. *Theory Pract. Log. Program.* In press.

Slaney, J., and Thiébaux, S. 2001. Blocks World revisited. *Artificial Intelligence* 125(1–2):119–153.

Smith, D. E. 2004. Choosing objectives in over-subscription planning. In Zilberstein, S.; Koehler, J.; and Koenig, S., eds., *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*, 393–401. AAAI Press.

Sohrabi, S.; Riabov, A. V.; Katz, M.; and Udrea, O. 2018. An AI planning solution to scenario generation for enterprise risk management. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI 2018)*, 160–167. AAAI Press.

Speck, D.; Hecher, M.; Gnad, D.; Fichte, J. K.; and Corrêa, A. B. 2025. Counting and reasoning with plans. In Walsh, T.; Shah, J.; and Kolter, Z., eds., *Proceedings of the 39th AAAI Conference on Artificial Intelligence (AAAI 2023)*, 26688–26696.

Speck, D.; Mattmüller, R.; and Nebel, B. 2020. Symbolic top-k planning. In Conitzer, V., and Sha, F., eds., *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*, 9967–9974. AAAI Press.

Speck, D.; Seipp, J.; and Torralba, Á. 2025. Symbolic search for cost-optimal planning with expressive model extensions. *Journal of Artificial Intelligence Research* 82:1349–1405.

Torralba, Á.; Alcázar, V.; Kissmann, P.; and Edelkamp, S. 2017. Efficient symbolic search for cost-optimal planning. *Artificial Intelligence* 242:52–79.

Tuples. 2025. Beluga™ ai challenge. https://tuples.ai/competition-challenge/.