PlanPilot: Efficient Navigation in Plan Space

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

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

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, or 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

Classical planning aims at finding a sequence of actions that transforms the initial state of a problem into a goal state. 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 *et al.*, 2020] enabling post hoc restrictions for various applications [Boddy *et al.*, 2005; Sohrabi *et al.*, 2018]. There exist variants that aim to generating a diverse set of plans [Nguyen *et al.*, 2012; Katz and Sohrabi, 2020]. But enumeration results in major disadvantages. It is computationally costly, yields potentially many similar plans, and requires to define optimality properties beforehand, 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 *et al.*, 2023; Gragera *et al.*, 2023], searching for sets of jointly achievable soft goals [Smith, 2004], asking for explanations of the absence of solutions that achieve the desired set of such soft goals [Eifler *et al.*, 2020], or for goal recognition [Ramírez and Geffner, 2009; Pereira and Meneguzzi, 2016].

In this work, 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 *et al.*, 2022; Speck *et al.*, 2025]. *Counting* enables us to reason in details about the plan space without enumerating solutions [Darwiche, 2001]. *Facets* provide meaningful actions that can be gradually restricted, being enforced or forbidden by the user. Facets are computationally easier and enable navigation even in large plan spaces while ensuring properties quickly and step by step. We present a use case that allows to interactively navigate the plan space, by enforcing or forbidding actions.

Our `PlanPilot` tool[1] takes as input the PDDL description of a planning task and constructs an ASP encoding of it. The ASP problem is then passed to the `fasb` reasoner, which allows user interaction for navigating the set of plans 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. A graphical web interface has been successfully used to give explanations for a logistics application [Gnad *et al.*, 2025].

Background

We follow established definitions of lifted classical planning [Corrêa *et al.*, 2020], which capture a common fragment of PDDL. Comprehensive introductions on ASP are available [Gebser *et al.*, 2012; Calimeri *et al.*, 2020].

Lifted Classical Planning A planning task is a tuple $\Pi = \langle \mathcal{P}, O, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$, where $\mathcal{P}$ is a set of *predicate symbols*, O

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

is a set of *objects*, $\mathcal{A}$ is a set of *action schemas*, $\mathcal{I}$ is the *initial state*, and $\mathcal{G}$ is the *goal condition*. Each predicate $P \in \mathcal{P}$ has an arity n, and for an n-ary predicate and a n-tuple $\vec{t} = \langle t_1, \ldots, t_n \rangle$ consisting of variables or objects $o \in O$, $P(\vec{t})$ is called an *atom*. An atom $P(\vec{t})$ is *ground* if all of $\vec{t}$'s components are objects $o \in O$. A *state* s is a set of ground atoms, and both $\mathcal{I}$ and $\mathcal{G}$ are states. A state s that satisfies the goal condition, i.e., where $s \supseteq \mathcal{G}$, is a *goal state*.

An *action schema* $a[\Delta] \in \mathcal{A}$ is a triple $\langle \mathrm{pre}(a[\Delta]), \mathrm{add}(a[\Delta]), \mathrm{del}(a[\Delta]) \rangle$, consisting of the *precondition*, *add list*, and *delete list* of $a[\Delta]$, all of which are finite sets of atoms. Here, Δ is the set of free variables that occur in any atom of any component of $a[\Delta]$. We can ground an action schema $a[\Delta]$ by substituting the free variables Δ by objects in O, which results in a *ground* action a, or simply action. An action a is applicable in a state s if $\mathrm{pre}(a) \subseteq s$. When applying a in s, the successor state s' is defined as $s' := (s \setminus \mathrm{del}(a)) \cup \mathrm{add}(a)$. A sequence of actions $\langle a_1, \ldots, a_n \rangle$ is applicable in state s if each a_i is applicable in the state generated by applying $a_1, \ldots, a_{i-1}$ from s. The solution of a planning problem is a sequence of actions, called a *plan*, applicable to $\mathcal{I}$ that ends in a goal state.

Answer-Set Programming (ASP) We explain notions only for ground answer-set programs for conciseness. 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 *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 $c_1, \ldots$, or c_n is true. Similarly, the latter, called *integrity constraint*, is a constraint enforcing that one of the atoms $b_1, \ldots, b_m$ is false or there is evidence that one of the atoms $c_1, \ldots, c_n$ is true. We let $H_r := \{a\}$ or $\emptyset$, respectively, $B_r^+ := \{b_1, \ldots, a_m\}$, and $B_r^- := \{c_1, \ldots, c_n\}$. 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 $P^M := \{H_r \leftarrow B_r^+ \mid B_r^- \cap M = \emptyset\}$. 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]. The *search space* of a program P consists of all its possible interpretations and we let the set $\mathrm{AS}(P)$ consist of all answer-sets of P. We define the *brave* consequences by $\mathcal{BC}(P) := \bigcup_{M \in \mathrm{AS}(P)} M$ and *cautious* consequences by $\mathcal{CC}(P) := \bigcap_{M \in \mathrm{AS}(P)} M$. A popular tool to compute answer sets is `clingo`[2], which allows to state non-ground programs, ground them and compute answer-sets on the ground level [Gebser *et al.*, 2019].

Facets and Counting ASP navigation provides concepts and notions to select answer-sets of a program P within the *solution space* $2^{\mathrm{AS}(P)}$. *Facets* restrict assumptions to (literals over) atoms of a program P that are meaningful, i.e., atoms belonging to some but not to all answer-sets [Fichte *et al.*, 2022; Speck *et al.*, 2025]. Thus, we let $\mathcal{F}^+(P) := \mathcal{BC}(P) \setminus \mathcal{CC}(P)$ be the *enforcing* facets and $\mathcal{F}^-(P) := \{\neg a \mid a \in \mathcal{F}^+(P)\}$ the *prohibiting* facets. By counting answer sets, we simply mean computing the number $|\mathrm{AS}(P)|$ of answer-sets. An interactive tool to navigate along answer-sets using quantitative and qualitative facet reasoning and counting is `fasb`. It can enforce or prohibit facets, answer declarative queries, and uses a state-of-the-art approach to quantitative reasoning with facets. We use `fasb` in version 0.2.2[3].

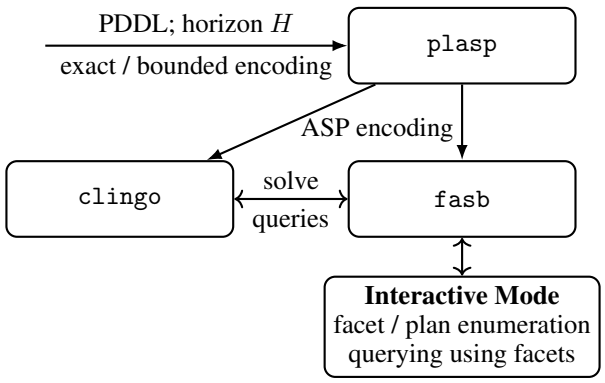


Figure 1: Illustration of `PlanPilot`'s components.

Solving Planning with ASP We use the `plasp` tool in version 3.1.1[4] [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 on 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.

The notion of cautious consequences is closely related to that of landmarks in planning [Porteous *et al.*, 2001; Hoffmann *et al.*, 2004]. Both cautious consequences and landmarks appear in every solution.

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 shows the 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* 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.

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

[3]https://github.com/drwadu/fasb/releases/tag/v0.2.2

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

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 plans. More specifically, we turn the occurrence of every action $a \in \mathcal{A}$ at a given time step $1 \leq t \leq H$ into a facet $\texttt{occurs}(a, t)$. These facets are *enforcing*, which means that we restrict the possible 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 call this variant 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 action. This is desirable in scenarios where the user is not interested in a occurring at a specific point, e.g., for domain debugging [Lin *et al.*, 2023; Gragera *et al.*, 2023].

Complexity

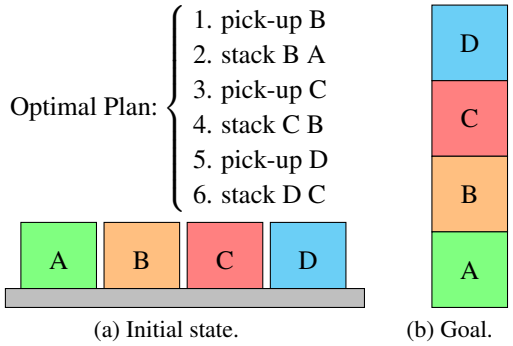
Reasoning over facets is significantly easier than reasoning over plans [Fichte *et al.*, 2022; Speck *et al.*, 2025]. Therefore, it is advisable to opt for counting and enumeration of facets, rather than plans, as planning tasks and horizons get larger. At the same time, enabling more facets limits 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.

Navigation modes

Besides enumerating the 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 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 maximally constrains the set of remaining plans. Alternatively, a large, diverse set of plans is obtained by selecting facets that lead to a low reduction in the number of remaining facets. This gives 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.



(a) Initial state. (b) Goal.

Figure 2: Blocks World task from our use case.

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.

Use Case

We next show an interactive run of `PlanPilot`, using an instance of the Blocks World domain. The task is depicted in Figure 2, it has 4 blocks and an optimal plan with 6 actions. `PlanPilot` takes as input the PDDL files 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 length of the optimal one:

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

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

```
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",const("b"),const("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",const("c"),const("b"))),9)
occurs(action(("pick-up",constant("d"))),10)
occurs(action(("stack",const("d"),const("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, rather than activating some facets first.

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
                   --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) ...
```

Conclusion

We present `PlanPilot`, a tool to navigate plan spaces iteratively and interactively. We build on an ASP encoding that transforms the PDDL planning task into ground-ASP, on which we investigate the plan space. This enables us to consider restrictions on actions that occur in a plan. 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.

Based on combinations of facets and counting, we can construct reasoning modes that allow for navigating 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 `PlanPilot` marks a significant step towards explainable AI planning (XAIP) [Chakraborti *et al.*, 2020]. It allows the user to navigate the plan space interactively, directly showing the consequences when restricting the possible solutions by activating facets. This leads to a better understanding of the plans for a given task, which can aid model debugging. In contrast to many existing XAIP systems, `PlanPilot` does not provide post-hoc explanations for a solution computed by a planner, but enables a mechanism to explore, restrict, and select a set of plans.

An interesting direction for future research is to find out how non-expert users interact with our tool and how useful the given explanations are [Miller, 2023].

References

[Boddy *et al.*, 2005] Mark Boddy, Johnathan Gohde, Tom Haigh, and Steven Harp. Course of action generation for cyber security using classical planning. In Susanne Biundo, Karen Myers, and Kanna Rajan, editors, *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005)*, pages 12–21. AAAI Press, 2005.

[Calimeri *et al.*, 2020] Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Marco Maratea, Francesco Ricca, and Torsten Schaub. ASP-Core-2 input language format. 20(2):294–309, 2020.

[Chakraborti *et al.*, 2020] Tathagata Chakraborti, Sarath Sreedharan, and Subbarao Kambhampati. The emerging landscape of explainable automated planning & decision making. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI 2020)*, pages 4803–4811. IJCAI, 2020.

[Corrêa *et al.*, 2020] Augusto B. Corrêa, Florian Pommerening, Malte Helmert, and Guillem Francès. Lifted successor generation using query optimization techniques. In J. Christopher Beck, Erez Karpas, and Shirin Sohrabi, editors, *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)*, pages 80–89. AAAI Press, 2020.

[Darwiche, 2001] Adnan Darwiche. Decomposable negation normal form. *Journal of the ACM*, 48(4):608–647, 2001.

[Dimopoulos *et al.*, 2019] Yannis Dimopoulos, Martin Gebser, Patrick Lühne, Javier Romero, and Torsten Schaub. plasp 3: Towards effective ASP planning. *Theory and Practice of Logic Programming*, 19(3):477–504, 2019.

[Eifler *et al.*, 2020] Rebecca Eifler, Michael Cashmore, Jörg Hoffmann, Daniele Magazzeni, and Marcel Steinmetz. A new approach to plan-space explanation: Analyzing plan-property dependencies in oversubscription planning. In Vincent Conitzer and Fei Sha, editors, *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*, pages 9818–9826. AAAI Press, 2020.

[Fichte *et al.*, 2022] Johannes Klaus Fichte, Sarah Alice Gaggl, and Dominik Rusovac. Rushing and strolling among answer sets – navigation made easy. In Vasant Honavar and Matthijs Spaan, editors, *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI 2022)*, pages 5651–5659. AAAI Press, 2022.

[Gebser *et al.*, 2012] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer set solving in practice*. Morgan & Claypool Publishers, 2012.

[Gebser *et al.*, 2019] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Multi-shot ASP solving with clingo. *Theory and Practice of Logic Programming*, 19(1):27–82, 2019.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP'88*, pages 1070–1080, August 1988.

[Gnad *et al.*, 2025] Daniel Gnad, Markus Hecher, Sarah Gaggl, Dominik Rusovac, David Speck, and Johannes K. Fichte. Interactive exploration of plan spaces. In Renata Wassermann Magdalena Ortiz and Torsten Schaub, editors, *Proceedings of Twenty-Second International Conference on Principles of Knowledge Representation and Reasoning (KR 2025)*. IJCAI Organization, 2025.

[Gragera *et al.*, 2023] Alba Gragera, Raquel Fuentetaja, Ángel García Olaya, and Fernando Fernández. A planning approach to repair domains with incomplete action effects. In Sven Koenig, Roni Stern, and Mauro Vallati, editors, *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)*, pages 153–161. AAAI Press, 2023.

[Hoffmann *et al.*, 2004] Jörg Hoffmann, Julie Porteous, and Laura Sebastia. Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22:215–278, 2004.

[Katz and Sohrabi, 2020] Michael Katz and Shirin Sohrabi. Reshaping diverse planning. In Vincent Conitzer and Fei Sha, editors, *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*, pages 9892–9899. AAAI Press, 2020.

[Katz *et al.*, 2018] Michael Katz, Shirin Sohrabi, Octavian Udrea, and Dominik Winterer. A novel iterative approach to top-k planning. In Mathijs de Weerdt, Sven Koenig, Gabriele Röger, and Matthijs Spaan, editors, *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS 2018)*, pages 132–140. AAAI Press, 2018.

[Lin *et al.*, 2023] Songtuan Lin, Alban Grastien, and Pascal Bercher. Towards automated modeling assistance: An efficient approach for repairing flawed planning domains. In Yiling Chen and Jennifer Neville, editors, *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence (AAAI 2023)*, pages 12022–12031. AAAI Press, 2023.

[Miller, 2023] Tim Miller. Explainable AI is dead, long live explainable ai!: Hypothesis-driven decision support using evaluative AI. In *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency, FAccT 2023, Chicago, IL, USA, June 12-15, 2023*, pages 333–342. ACM, 2023.

[Nguyen *et al.*, 2012] Tuan Anh Nguyen, Minh Binh Do, Alfonso Gerevini, Ivan Serina, Biplav Srivastava, and Subbarao Kambhampati. Generating diverse plans to handle unknown and partially known user preferences. *Artificial Intelligence*, 190:1–31, 2012.

[Pereira and Meneguzzi, 2016] Ramon Fraga Pereira and Felipe Meneguzzi. Landmark-based plan recognition. In Gal A. Kaminka, Maria Fox, Paolo Bouquet, Eyke Hüllermeier, Virginia Dignum, Frank Dignum, and Frank van Harmelen, editors, *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, volume

285 of *Frontiers in Artificial Intelligence and Applications*, pages 1706–1707. IOS Press, 2016.

[Porteous *et al.*, 2001] Julie Porteous, Laura Sebastia, and Jörg Hoffmann. On the extraction, ordering, and usage of landmarks in planning. In Amedeo Cesta and Daniel Borrajo, editors, *Proceedings of the Sixth European Conference on Planning (ECP 2001)*, pages 174–182. AAAI Press, 2001.

[Ramírez and Geffner, 2009] Miquel Ramírez and Hector Geffner. Plan recognition as planning. In Craig Boutilier, editor, *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 1778–1783, 2009.

[Smith, 2004] David E. Smith. Choosing objectives in over-subscription planning. In Shlomo Zilberstein, Jana Koehler, and Sven Koenig, editors, *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*, pages 393–401. AAAI Press, 2004.

[Sohrabi *et al.*, 2018] Shirin Sohrabi, Anton V. Riabov, Michael Katz, and Octavian Udrea. An AI planning solution to scenario generation for enterprise risk management. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI 2018)*, pages 160–167. AAAI Press, 2018.

[Speck *et al.*, 2020] David Speck, Robert Mattmüller, and Bernhard Nebel. Symbolic top-k planning. In Vincent Conitzer and Fei Sha, editors, *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*, pages 9967–9974. AAAI Press, 2020.

[Speck *et al.*, 2025] David Speck, Markus Hecher, Daniel Gnad, Johannes K. Fichte, and Augusto B. Corrêa. Counting and reasoning with plans. In Julie Shah and Zico Kolter, editors, *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2025)*, pages 26688–26696. AAAI Press, 2025.