Rushing and Strolling among Answer Sets – Navigation Made Easy

Johannes Klaus Fichte[1], **Sarah Alice Gaggl**[2], **Dominik Rusovac**[2]

[1] Research Unit Database and Artificial Intelligence, TU Wien, Austria
[2] Logic Programming and Argumentation Group, TU Dresden, Germany
johannes.fichte@tuwien.ac.at, sarah.gaggl@tu-dresden.de, dominik.rusovac@tu-dresden.de

Abstract

Answer set programming (ASP) is a popular declarative programming paradigm with a wide range of applications in artificial intelligence. Oftentimes, when modeling an AI problem with ASP, and in particular when we are interested beyond simple search for optimal solutions, an actual solution, differences between solutions, or number of solutions of the ASP program matter. For example, when a user aims to identify a specific answer set according to her needs, or requires the total number of diverging solutions to comprehend probabilistic applications such as reasoning in medical domains. Then, there are only certain problem specific and handcrafted encoding techniques available to navigate the solution space of ASP programs, which is oftentimes not enough. In this paper, we propose a formal and general framework for interactive navigation toward desired subsets of answer sets analogous to faceted browsing. Our approach enables the user to explore the solution space by consciously zooming in or out of sub-spaces of solutions at a certain configurable pace. We illustrate that weighted faceted navigation is computationally hard. Finally, we provide an implementation of our approach that demonstrates the feasibility of our framework for incomprehensible solution spaces.

Introduction

Answer set programming (ASP) is a declarative programming paradigm, which has its roots in logic programming and nonmonotonic reasoning. It is widely used for knowledge representation and problem solving (Brewka, Eiter, and Truszczyński 2011). In ASP, a problem is encoded as a set of rules (logic program) and is evaluated under stable model semantics (Gelfond and Lifschitz 1988, 1991), using solvers such as `clingo` (Gebser et al. 2011, 2014), `WASP` (Alviano et al. 2015) or `DLV` (Alviano et al. 2017). Then, answer sets represent solutions to the modeled problem.

Oftentimes when modeling with ASP, the number of solutions of the resulting program can be quite high. This is not necessarily a problem when searching for a few solutions, e.g., optimal solutions (Gebser, Kaminski, and Schaub 2011; Alviano and Dodaro 2016a) or when incorporating preferences (Brewka 2004; Brewka et al. 2015; Alviano, Romero, and Schaub 2018). However, there are many situations where reasoning goes beyond simple search for one answer set, for example, planning when certain routes are gradually forbidden (Son et al. 2016), finding diverging solutions (Everardo 2017; Everardo et al. 2019), reasoning in probabilistic applications (Lee, Talsania, and Wang 2017), or debugging answer sets (Oetsch, Pührer, and Tompits 2018; Gebser et al. 2008).

Now, if the user is interested in more than a few solutions to gradually identify specific answer sets, tremendous solution spaces can easily become infeasible to comprehend. In fact, it might not even be possible to compute all solutions in reasonable time. Examples where we easily see large solution spaces are configuration problems (Soininen and Niemelä 1999; Soininen et al. 2001; Tiihonen et al. 2003), such as PC configuration, and planning problems (Dimopoulos, Nebel, and Koehler 1997; Lifschitz 1999; Nogueira et al. 2001). Consider the following example.

Example 1. *Consider an online shopping situation where we have a knowledge base on clothes and some rules specifying which combinations would suit well or not.*

$$
\begin{aligned}
&\{\{\mathit{outfit}(X,Y) : \mathit{clothes}(X,Y)\};\\
&\qquad\qquad\qquad\quad \leftarrow \mathit{outfit}(X,Y1),\\
&\qquad\qquad\qquad\qquad\ \mathit{outfit}(X,Y2), Y1 \neq Y2;\\
\mathit{occasion}(\mathit{vancouver}) &\leftarrow \mathit{outfit}(\mathit{jacket},\ldots);\\
\mathit{occasion}(\mathit{conference}) &\leftarrow \mathit{outfit}(\mathit{suit},Y), Y \neq \mathit{yellow};\\
\mathit{occasion}(\mathit{wistler}) &\leftarrow \mathit{outfit}(\mathit{boots},\ldots)\ldots\}
\end{aligned}
$$

Together with input facts from a clothes database like $\mathit{clothes}(\mathit{jacket,blue})$*;* $\mathit{clothes}(\mathit{shirt,red})$*;...* *one easily obtains more than a million answer sets. As we plan to travel to a conference in Vancouver, we want to find suitable outfits for this occasion. Say we zoom in on outfits including shorts, which leads to a rather small, but still incomprehensible sub-space of solutions. Say we want to inspect the most different outfits still remaining, then we aim to choose potential parts of our outfit that provide us with most diverse solutions. Now, we are almost good to go, seeking to find some final additions to our outfit quickly.*

Our example illustrates that different solutions in ASP programs can easily be hard to comprehend. Problem specific, handcrafted encoding techniques to navigate the solution space can be quite tedious.

Instead, we propose a formal and general framework for interactive navigation toward desired subsets of answer sets analogous to faceted browsing in the field of information retrieval (Tunkelang 2009). Our approach enables solution space exploration by consciously zooming in or out of subspaces of solutions at a certain configurable pace. To this end we introduce absolute and relative weights to quantify the size of the search space when reasoning under assumptions (facets). We formalize several kinds of search space navigation as goal-oriented and explore modes, and systematically compare the introduced weights regarding their usability for operations under natural properties splitting, reliability, preserving maximal sub-spaces (min-inline), and preserving minimal sub-spaces (max-inline). In addition, we illustrate the computational complexity for computing the weights. Finally, we provide an implementation on top of the solver `clingo` demonstrating the feasibility of our framework for incomprehensible solution spaces.

Related Work. Alrabbaa, Rudolph, and Schweizer (2018) proposed a framework in which solutions are systematically pruned with respect to facets (partial solutions). While this allows one to move within the answer set space, the user has absolutely no information on how big the effect of activating a facet is in advance, similar to assumptions in propositional satisfiability (Eén and Sörensson 2003). We go far beyond and characterize the *weight* of a facet. This is useful to comprehend the effect of navigation steps on the size of the solution space. Additionally, this allows for zooming into or out of the solution space at a configurable pace. Debugging in answer sets has widely been investigated (Oetsch, Pührer, and Tompits 2018; Dodaro et al. 2019; Vos et al. 2012; Shchekotykhin 2015; Gebser et al. 2008). However, we do not aim to correct ASP encodings. All answer sets which are reachable within the navigation are "original" answer sets, thus the adaptions we make during the navigation to the program, do not change the set of answer sets of the initial program. Justifications, which describe the support for the truth value of each atom, have been studied as a tool for reasoning and debugging (El-Khatib, Pontelli, and Son 2005). Probabilistic reasoning frameworks for logic programs were developed such as LP$^{\text{MLN}}$ (Lee, Talsania, and Wang 2017), which define notions of probabilities in terms of relative occurrences of stable models and their weights. Computing these probabilities (unless restricted to decision versions in terms of being different from zero) relates to counting probabilities under assumptions. Considering relative occurrences of stable models of weight one relates to search space exploration. However, probabilistic frameworks primarily address modeling conflicting information and reason about them. We assume large solution spaces and aim for navigating dynamically in the solution space.

Background

First, we recall basic notions of ASP, for further details on ASP we refer to standard texts (Calimeri et al. 2020; Gebser et al. 2012).

Answer Set Programming. By $\mathcal{A}(\Pi)$ we denote the set of (non-ground) *atoms* of a program Π. A literal is an atom $\alpha \in \mathcal{A}(\Pi)$ or its *default negation*, which refers to the absence of information, denoted by $\sim\alpha$. An atom α is a predicate $p(t_0, \ldots, t_n)$ of arity $n \geq 0$ where each t_i for $0 \leq i \leq n$ is a *term*, i.e., either a variable or a constant. We say an atom $\alpha \in \mathcal{A}(\Pi)$ is *ground* if and only if α is variable-free. By $Grd(\mathcal{A}(\Pi))$ we denote ground atoms. A (disjunctive) logic program Π is a finite set of rules r of the form $\alpha_0 \mid \ldots \mid \alpha_k \leftarrow \alpha_{k+1}, \ldots, \alpha_m, \sim\alpha_{m+1}, \ldots, \sim\alpha_n$ where $0 \leq k \leq m \leq n$ and each $\alpha_i \in \mathcal{A}(\Pi)$ for $0 \leq i \leq n$. For a rule r we denote the head by $H(r) := \{\alpha_0, \ldots, \alpha_k\}$, the body $B(r)$ consists of the positive body $B^+(r) := \{\alpha_{k+1}, \ldots, \alpha_m\}$, and the negative body $B^-(r) := \{\alpha_{m+1}, \ldots, \alpha_n\}$. If $B(r) = \emptyset$, we omit $\leftarrow$. A rule r where $H(r) = \emptyset$ is called *integrity constraint* and avoids that $B(r)$ is evaluated positively. By $grd(r)$ we denote the set of ground instances of some rule r, obtained by replacing all variables in r by ground terms. Accordingly, $Grd(\Pi) := \bigcup_{r \in \Pi} grd(r)$ denotes the ground instantiation of Π. Without any explicit contrary indication, throughout this paper, we use the term (logic) program to refer to ground disjunctive programs where $\mathcal{A}(\Pi) = Grd(\mathcal{A}(\Pi))$. An interpretation $X \subseteq \mathcal{A}(\Pi)$ satisfies a rule $r \in \Pi$ if and only if $H(r) \cap X \neq \emptyset$ whenever $B^+(r) \subseteq X$ and $B^-(r) \cap X = \emptyset$. X satisfies Π, if X satisfies each rule $r \in \Pi$. An interpretation X is a *stable model* (also called *answer set*) of Π if and only if X is a subset-minimal model satisfying the Gelfond-Lifschitz reduct of Π with respect to X, defined as $\Pi_X := \{H(r) \leftarrow B^+(r) \mid X \cap B^-(r) = \emptyset, r \in \Pi\}$. By $\mathcal{AS}(\Pi)$ we denote the answer sets of Π. For computing facets, we rely on two notions of consequences of a program, namely, *brave* consequences $\mathcal{BC}(\Pi) := \bigcup \mathcal{AS}(\Pi)$ and *cautious* consequences $\mathcal{CC}(\Pi) := \bigcap \mathcal{AS}(\Pi)$.

Faceted Navigation. *Faceted answer set navigation* is characterized as a sequence of navigation steps restricting the solution space with respect to partial solutions. Those partial solutions, called *facets*, correspond to ground atoms of a program Π that are not contained in each solution. We denote the *facets* of Π by $\mathcal{F}(\Pi) := \mathcal{F}^+(\Pi) \cup \mathcal{F}^-(\Pi)$ where $\mathcal{F}^+(\Pi) := \mathcal{BC}(\Pi) \setminus \mathcal{CC}(\Pi)$ denotes *inclusive facets* and $\mathcal{F}^-(\Pi) := \{\overline{\alpha} \mid \alpha \in \mathcal{F}^+(\Pi)\}$ denotes *exclusive facets* of Π. We say an interpretation $X \subseteq \mathcal{A}(\Pi)$ satisfies an inclusive facet $f \in \mathcal{F}^+(\Pi)$, if $f \in X$, which we denote by $X \models f$, and it satisfies an exclusive facet $f \in \mathcal{F}^-(\Pi)$, if $f \notin X$.

A navigation step is a transition from one program to another, obtained by adding some integrity constraint that enforces the atom refered to by some inclusive or exclusive facet to be present or absent, respectively, throughout answer sets. By $ic(f)$ we denote the function that translates a facet $f \in \{\alpha, \overline{\alpha}\} \subseteq \mathcal{F}(\Pi)$ into a singleton program that contains its corresponding integrity constraint:

$$ic(f) := \begin{cases} \{\leftarrow \sim\alpha\}, & \text{if } f = \alpha; \\ \{\leftarrow \alpha\}, & \text{otherwise.} \end{cases}$$

Accordingly, a navigation step from Π to Π' is obtained by modifying Π such that $\Pi' = \Pi \cup ic(f)$. Faceted navigation with respect to some program Π is possible as long as $\mathcal{F}(\Pi) \neq \emptyset$. Alrabbaa, Rudolph, and Schweizer (2018) established that if $f \in \mathcal{F}(\Pi)$, then $\Pi' := \Pi \cup ic(f)$ is satisfiable and $\mathcal{AS}(\Pi') = \{X \in \mathcal{AS}(\Pi) \mid X \models f\}$. When referring to

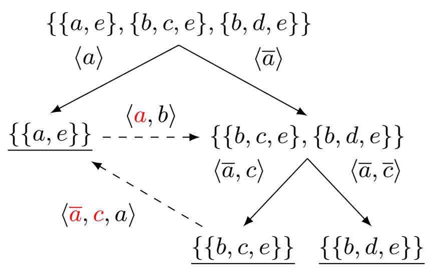

$\{\{a,e\},\{b,c,e\},\{b,d,e\}\}$
$\langle a\rangle$
$\langle\overline{a}\rangle$
$\{\{a,e\}\}$
$\langle a,b\rangle$
$\{\{b,c,e\},\{b,d,e\}\}$
$\langle\overline{a},c\rangle$
$\langle\overline{a},\overline{c}\rangle$
$\langle\overline{a},c,a\rangle$
$\{\{b,c,e\}\}$
$\{\{b,d,e\}\}$

Figure 1: Goal-oriented and free navigation on program Π_1.

$\mathcal{AS}(\Pi)$ as a solution space, we refer to the topological space induced by $2^{\mathcal{AS}(\Pi)}$ on $\mathcal{AS}(\Pi)$. Thus, answer set navigation means choosing among subsets of answer sets.

Routes and Navigation Modes

We introduce *routes* as a notion for characterizing sequences of navigation steps.

Definition 1. *A* route δ *is a finite sequence* $\langle f_1,\dots,f_n\rangle$ *of facets* $f_i \in \mathcal{F}(\Pi)$ *s.t.* $0 \leq i \leq n \in \mathbb{N}$, *denoting* n *arbitrary navigation steps over* Π. *We say* δ *is a* subroute *of* δ', *denoted by* $\delta \sqsubseteq \delta'$, *whenever if* $f_i \in \delta$, *then* $f_i \in \delta'$. *We define* $\Pi^\delta := \Pi \cup ic(f_1) \cup \dots \cup ic(f_n)$. *By* Δ^Π *we denote all possible routes over* $\mathcal{AS}(\Pi)$, *including the empty route* ϵ.

It is easy to see that any permutation of navigation steps of a fixed set of facets always leads to the same solutions. In general, different routes may lead to the same subset of answer sets. We say two routes $\delta, \delta' \in \Delta^\Pi$ are equivalent if and only if $\mathcal{AS}(\Pi^\delta) = \mathcal{AS}(\Pi^{\delta'})$. To ensure satisfiable programs, we aim to select so called *safe* routes. By $\Delta^\Pi_s := \{\delta \in \Delta^\Pi \mid \mathcal{AS}(\Pi^\delta) \neq \emptyset\}$ we define *safe routes* over $\mathcal{AS}(\Pi)$. Once an unsafe route is taken, some sort of *redirection*, which relates to the notion of *correction sets* (Alrabbaa, Rudolph, and Schweizer 2018), i.e., a route obtained by retracting conflicting facets, is required to continue navigation. For a program Π, $\delta \in \Delta^\Pi$ and $f \in \mathcal{F}(\Pi)$, we denote all *redirections* of δ with respect to f by $\mathcal{R}(\delta, f) := \{\delta' \sqsubseteq \delta \mid f \in \delta', \mathcal{AS}(\Pi^{\delta'}) \neq \emptyset\} \cup \{\epsilon\}$. The following example illustrates faceted navigation.

Example 2. *Consider program* $\Pi_1 = \{a \,|\, b; c \,|\, d \leftarrow b; e\}$. *It is easy to observe that the answer sets are* $\mathcal{AS}(\Pi_1) = \{\{a,e\}, \{b,c,e\}, \{b,d,e\}\}$. *Thus, we can choose from facets* $\mathcal{F}(\Pi_1) = \{a,b,c,d,\overline{a},\overline{b},\overline{c},\overline{d}\}$. *As illustrated in Figure 1, if we activate facet* a *we land at* $\mathcal{AS}(\Pi_1^{\langle a\rangle}) = \{\{a,e\}\}$. *Activating* b *on* $\langle a\rangle$ *gives* $\mathcal{AS}(\Pi_1^{\langle a,b\rangle}) = \emptyset$. *To redirect* $\langle a,b\rangle$ *we can choose from* $\mathcal{R}(\langle a,b\rangle, b) = \{\langle b\rangle\}$.

We consider two more notions for identifying routes that point to a unique solution. A set of facets is a delimitation, if any safe route constructible thereof leads to a unique answer set. This means that any further step would lead to an unsafe route.

Definition 2. *Let* Π *be a program and* $F, F' \subseteq \mathcal{F}(\Pi)$ *s.t.* $F := \{f_1,\dots,f_n\}$. *We define* $\tau(F)$ *as all permutations of* $\delta := \langle f_1,\dots,f_n\rangle$ *and say* F *is* delimiting *w.r.t.* Π, *if* $\tau(F) \subseteq \Delta^\Pi_s$ *and* $\forall F' \supset F : \tau(F') \not\subseteq \Delta^\Pi_s$. *By* $\mathcal{DF}(\Pi) \subset 2^{\mathcal{F}(\Pi)}$ *we denote the set of* delimitations *over* $\mathcal{F}(\Pi)$.

We call a route consisting of delimiting facets *maximal safe*.

Definition 3. *Let* Π *be a program,* $F \subseteq \mathcal{F}(\Pi)$ *and* $\delta \in \tau(F) \subseteq \Delta^\Pi$. *We call* δ maximal safe, *if and only if* $F \in \mathcal{DF}(\Pi)$. *By* Δ^Π_{ms} *we denote the set of maximal safe routes in* $\mathcal{AS}(\Pi)$.

In fact, each delimitation corresponds to a unique solution.

Lemma 1 ($\star$[1]). *Let* Π *be a program,* $F \subseteq \mathcal{F}(\Pi)$ *and* $\delta \in \tau(F) \subseteq \Delta^\Pi$. *If* $\delta \in \Delta^\Pi_{ms}$, *then* $|\mathcal{AS}(\Pi^\delta)| = 1$.

Theorem 1 ($\star$). $|\mathcal{AS}(\Pi)| = |\mathcal{DF}(\Pi)|$.

As mentioned, using routes and facets, there are several ways to explore solutions. A *navigation mode* is a function that prunes the solution space according to a search strategy that involves routes and facets.

Definition 4. *Let* $X_i \in 2^{\Delta^\Pi} \cup 2^{\mathcal{F}(\Pi)}$ *where* $0 \leq i \leq n \in \mathbb{N}$. *A* navigation mode *is a function* $\nu : X_0 \times \dots \times X_n \to 2^{\mathcal{AS}(\Pi)}$ *that maps an* n*-ary Cartesian product over subsets of routes over* Π *and facets of* Π *to answer sets of* Π.

The idea of *free* and *goal-oriented* navigation was mentioned by Alrabbaa, Rudolph, and Schweizer (2018). While free navigation follows no particular strategy, during goal-oriented navigation, we narrow down the solution space. Next, we formalize the goal-oriented navigation mode.

Definition 5. *We define the* goal-oriented *navigation mode* $\nu_{go} : \Delta^\Pi_s \times \mathcal{F}(\Pi) \to 2^{\mathcal{AS}(\Pi)}$ *by:*

$$\nu_{go}(\delta, f) := \begin{cases} \mathcal{AS}(\Pi^{\langle \delta, f\rangle}), & \text{if } f \in \mathcal{F}(\Pi^\delta);\\ \mathcal{AS}(\Pi^\delta), & \text{otherwise.}\end{cases}$$

As illustrated in Figure 1, while during goal-oriented navigation (indicated by solid lines) the space is being narrowed down, until some unique solution (indicated by underscores) is found, in free mode (indicated by both dashed and solid lines) unsafe routes are being redirected, as illustrated on route $\langle a,b\rangle$ where a is retracted. We call the effect of narrowing down the space *zooming in*, the inverse effect *zooming out* and any effect where the number of solutions remains the same, *slide* effect, e.g., activating a on route $\langle\overline{a},c\rangle$.

Weighted Faceted Navigation

During faceted navigation, we can zoom in, zoom out or slide. However, we are unaware of how big the effect of activating a facet will be. Recall that different routes can lead to the same unique solution. The activation of some facet may lead to a unique solution more quickly or less quickly than the activation of another facet, which means that during navigation one has no information on the length of a route. Our framework provides an approach for consciously zooming in on solutions. Introducing *weighted* navigation, we characterize a navigation step with respect to the extent to which it affects the size of the solution space, thereby we can navigate toward solutions at a configurable "pace" of navigation,

[1]Statements marked by "$\star$" are proven in the extended version: arXiv:2112.07596

which we consider to be the extent to which the current route zooms into the solution space.

The kind of parameter that allows for configuration is called the *weight* of a facet. Weights of facets enable users to inspect effects of facets at any stage of navigation, which allows for navigating more interactively in a systematic way. Any weight or pace is associated with a *weighting function* that can be defined in various ways, specifying the number of program-related objects, e.g., answer sets.

Definition 6. *Let* Π *be a program,* $\delta \in \Delta^{\Pi}$*,* $f \in \mathcal{F}(\Pi)$ *and* $\delta' \in \mathcal{R}(\delta, f)$*. We call* $\# : \{\Pi^{\delta} \mid \delta \in \Delta^{\Pi}\} \to \mathbb{N}$ *a* weighting function*, whenever* $\#(\Pi^{\delta}) > 0$*, if* $|\mathcal{AS}(\Pi^{\delta})| \geq 2$*. The weight* $\omega_{\#}$ *of* f *w.r.t.* $\#$*,* Π^{δ} *and* δ' *is defined as:*

$$\omega_{\#}(f, \Pi^{\delta}, \delta') := \begin{cases} \#(\Pi^{\delta}) - \#(\Pi^{\delta'}), & \text{if } \langle \delta, f \rangle \notin \Delta_s^{\Pi^{\delta}} \\ & \text{and } \delta' \neq \epsilon; \\ \#(\Pi^{\delta}) - \#(\Pi^{\langle \delta, f \rangle}), & \text{otherwise.} \end{cases}$$

The pace indicates the zoom-in effect of a route with respect to a weighting function.

Definition 7. *Let* Π *be a program s.t.* $|\mathcal{AS}(\Pi)| \geq 2$ *and* $\delta \in \Delta_s^{\Pi}$*. We define the pace* $\mathcal{P}_{\#}(\delta)$ *of* δ *w.r.t.* $\#$ *as* $\mathcal{P}_{\#}(\delta) := {(\#(\Pi) - \#(\Pi^{\delta}))}/{\#(\Pi)}$.

Before we instantiate weights with actual weighting functions, we identify desirable properties of weights. Most importantly, weights should indicate zoom-in effects of facets on safe routes, i.e., a weight should identify which facets lead to a proper sub-space of answer sets.

Definition 8. *We call a weight* $\omega_{\#}$ safe-zooming*, whenever if* $f \in \mathcal{F}(\Pi^{\delta})$*, then* $\omega_{\#}(f, \Pi^{\delta}, \epsilon) > 0$ *for* $\delta \in \Delta_s^{\Pi}$*.*

Essentially, whenever a weight is *safe-zooming* it is useful to to inspect zoom-in effects during goal-oriented navigation.

Definition 9. *We call a weight* $\omega_{\#}$ splitting*, if* $\#(\Pi^{\delta}) = \omega_{\#}(\alpha, \Pi^{\delta}, \delta') + \omega_{\#}(\overline{\alpha}, \Pi^{\delta}, \delta')$ *for* $\delta, \delta' \in \Delta_s^{\Pi}$ *and* $\alpha, \overline{\alpha} \in \mathcal{F}(\Pi^{\delta})$*.*

Splitting weights are useful during goal-oriented navigation, as any permissible route δ in ν_{go} is safe and if $\#(\Pi^{\delta})$ and the weight of a facet $f \in \mathcal{F}(\Pi^{\delta})$ for $\delta \in \Delta_s^{\Pi}$ are known, we can compute the weight of the respective inverse facet $f' \in \mathcal{F}(\Pi^{\delta})$ arithmetically and thus avoid computing $\#(\Pi^{\langle \delta, f' \rangle})$.

Definition 10. *We call a weight* $\omega_{\#}$ reliable*, whenever* $\omega_{\#}(f, \Pi^{\delta}, \epsilon) = \#(\Pi^{\delta})$ *if and only if* $\langle \delta, f \rangle \notin \Delta_s^{\Pi}$ *for* $\delta \in \Delta^{\Pi}$ *and* $f \in \mathcal{F}(\Pi)$*.*

The benefit of *reliable* weights, on the other hand, is that they indicate unsafe routes. Hence, reliability can be ignored during goal-oriented navigation, but appears to be useful during free navigation.

As we are focused on narrowing down the solution space, we want to know, whether the associated weighting function $\#$ of a weight detects maximal or minimal, respectively, zoom-in effects on safe routes.

Definition 11. *For a program* Π*,* $\delta \in \Delta^{\Pi}$ *and* $f \in F$*, then: (i)* f *is* maximal weighted*, denoted by* $f \in max_{\omega_{\#}}(\Pi^{\delta})$*, if* $\forall f' \in \mathcal{F}(\Pi^{\delta}) : \omega_{\#}(f, \Pi^{\delta}, \epsilon) \geq \omega_{\#}(f', \Pi^{\delta}, \epsilon)$*; and (ii)* f *is* minimal weighted*, denoted by* $f \in min_{\omega_{\#}}(\Pi^{\delta})$*, if* $\forall f' \in \mathcal{F}(\Pi^{\delta}) : \omega_{\#}(f, \Pi^{\delta}, \epsilon) \leq \omega_{\#}(f', \Pi^{\delta}, \epsilon)$*.*

A weight is min-inline, if every minimal weighted facet leads to a maximal sub-space of solutions. Analogously, a weight is max-inline, if every maximal weighted facet leads to a minimal sub-space.

Definition 12. *Let* Π *be a program,* $\delta \in \Delta_s^{\Pi}$ *and* $f \in \mathcal{F}(\Pi^{\delta})$*. We call a weight* $\omega_{\#}$ *(i)* min-inline*, whenever* $f \in min_{\omega_{\#}}(\Pi^{\delta})$ *if and only if for all* $f' \in \mathcal{F}(\Pi^{\delta}) \backslash min_{\omega_{\#}}(\Pi^{\delta}) : |\mathcal{AS}(\Pi^{\langle \delta, f \rangle})| > |\mathcal{AS}(\Pi^{\langle \delta, f' \rangle})|$*; and (ii)* max-inline*, whenever* $f \in max_{\omega_{\#}}(\Pi^{\delta})$ *if and only if for all* $f' \in \mathcal{F}(\Pi^{\delta}) \setminus max_{\omega_{\#}}(\Pi^{\delta}) : |\mathcal{AS}(\Pi^{\langle \delta, f \rangle})| < |\mathcal{AS}(\Pi^{\langle \delta, f' \rangle})|$*.*

Below, we introduce the *absolute* weight of a facet, which counts answer sets, and two so called *relative* weights, which seek for approximating the number of solutions to compare sub-spaces with respect to their actual size, while avoiding counting.

Absolute Weight

The most natural weighting function to identify the effect of a navigation step is based on the number of answer sets on a route. The absolute weight of a facet f is defined as the number of solutions by which the solution space grows or shrinks due to the activation of f.

Definition 13. *The* absolute weight $\omega_{\#\mathcal{AS}}$ *is defined by* $\#\mathcal{AS} : \Pi^{\delta} \mapsto |\mathcal{AS}(\Pi^{\delta})|$*.*

Example 3. *Let us inspect Figure 1 and the program* Π_1 *from Example 2. As stated by* $\omega_{\#\mathcal{AS}}(a, \Pi_1^{\langle \overline{a}, c \rangle}, \langle a \rangle) = 0$*, activating* a *on* $\langle \overline{a}, c \rangle$ *induces a slide.* $\omega_{\#\mathcal{AS}}(a, \Pi_1^{\langle a \rangle}, \langle b \rangle) = -1$*. This tells us that navigating toward* b *on* $\langle a \rangle$ *zooms out by one solution. In contrast,* $\omega_{\#\mathcal{AS}}(b, \Pi_1^{\langle c \rangle}, \langle \overline{a} \rangle) = 1$ *means that we zoom in by one solution.*

By definition, the absolute weight directly reflects the effect of a navigation step and satisfies all introduced properties.

Theorem 2 ($\star$)**.** *The absolute weight* $\omega_{\#\mathcal{AS}}$ *is safe-zooming, splitting, reliable, min-inline, and max-inline.*

Unfortunately, computing absolute weights is expensive.

Lemma 2 ($\star$)**.** *Outputting the absolute weight* $\omega_{\#\mathcal{AS}}$ *for a given program* Π *and route* δ *is* $\# \cdot \text{coNP}$*-complete.*

Relative Weights

Since computing absolute weights is computationally expensive (Lemma 2), we aim for less expensive methods that still retain the ability to compare sub-spaces with respect to their size. Therefore, we investigate two *relative weights*.

Facet Counting. One approach to manipulating the number of solutions and to keeping track of how the number changes over the course of navigation, is to count facets.

Definition 14. *The* facet-counting weight $\omega_{\#\mathcal{F}}$ *is defined by* $\#\mathcal{F} : \Pi^{\delta} \mapsto |\mathcal{F}(\Pi^{\delta})|$*.*

Next, we establish a positive result in terms of complexity. Recall $\Delta_3^P \subseteq \text{PH} \subseteq \text{P}^{\#\text{P}}$ (Stockmeyer 1976; Toda 1991).

Lemma 3 ($\star$)**.** *Outputting the facet-counting weight* $\omega_{\#\mathcal{F}}$ *for a given program* Π *and route* δ *is in* Δ_3^P*.*

Hence, assuming standard theoretical assumptions, counting facets is easier than counting solutions. However, below we show that counting facets has deficiencies, when it comes to comprehending the solution space regarding its size.

Lemma 4 ($\star$). $|\mathcal{AS}(\Pi)| \leq 1$ *if and only if* $|\mathcal{F}(\Pi)| = 0$.

From Lemma 4 and the fact that for program Π_1 from Example 2 we have $\omega_{\#\mathcal{F}}(c, \Pi_1^{\langle\overline{a}\rangle}, \epsilon) = |\mathcal{F}(\Pi_1^{\langle\overline{a}\rangle})|$, but $\langle\overline{a}, c\rangle \in \Delta_s^{\Pi_1}$, we conclude that $\omega_{\#\mathcal{F}}$ is not reliable. Furthermore, since therefore $\omega_{\#\mathcal{F}}(c, \Pi_1^{\langle\overline{a}\rangle}, \epsilon) + \omega_{\#\mathcal{F}}(\overline{c}, \Pi_1^{\langle\overline{a}\rangle}, \epsilon) \neq |\mathcal{F}(\Pi_1^{\langle\overline{a}\rangle})|$, $\omega_{\#\mathcal{F}}$ is not splitting either.

Corollary 1. *The facet-counting weight* $\omega_{\#\mathcal{F}}$ *is not reliable and not splitting.*

The reason for $\omega_{\#\mathcal{F}}$ not distinguishing between one and no solution is that we can interpret it as an indicator for how the diversity or similarity, respectively, of solutions changes by activating a facet. Accordingly, whenever a step leads to one or no solution, the thereby reached sub-space contains least-diverse or most-similar solutions, respectively.

Example 4. *Again consider* Π_1 *from Example 2. While for the absolute weights of* $\overline{a}$ *and* $\overline{c}$ *we have* $\omega_{\#\mathcal{AS}}(\overline{a}, \Pi_1, \epsilon) = 1 = \omega_{\#\mathcal{AS}}(\overline{c}, \Pi_1, \epsilon)$*, their respective relative weights differ with* $\omega_{\#\mathcal{F}}(\overline{a}, \Pi_1, \epsilon) = 4$ *and* $\omega_{\#\mathcal{F}}(\overline{c}, \Pi_1, \epsilon) = 2$*. The reason is that even though* $|\mathcal{AS}(\Pi_1^{\langle\overline{a}\rangle})| = |\mathcal{AS}(\Pi_1^{\langle\overline{c}\rangle})|$*, by activating* $\overline{c}$ *we can still navigate toward* $\mathcal{F}(\Pi_1^{\langle\overline{c}\rangle}) = \{a, b, d, \overline{a}, \overline{b}, \overline{d}\}$*, but activating* $\overline{a}$*, we can only navigate toward* $\mathcal{F}(\Pi_1^{\langle\overline{a}\rangle}) = \{c, d, \overline{c}, \overline{d}\}$*, i.e., answer sets that contain b.*

In other words, while $\#\mathcal{F}$ indicates how "far apart" solutions are, $\omega_{\#\mathcal{F}}$ indicates to what amount the solutions converge due to navigation steps.

Theorem 3 ($\star$). *Weight* $\omega_{\#\mathcal{F}}$ *is safe-zooming.*

Due to Theorem 3, we know that $\#\mathcal{F}$ can be used to determine the pace of safe navigation. In fact the facet-counting pace $\mathcal{P}_{\#\mathcal{F}}$ emphasizes that $\omega_{\#\mathcal{F}}$ is not directly related to the size of the solution space.

Example 5. *Consider* Π_1 *from Example 2. While* $|\mathcal{AS}(\Pi_1^{\langle\overline{c}\rangle})| = 2$ *and* $|\mathcal{AS}(\Pi_1)| = 3$*, which means that activating* $\overline{c}$ *on* Π_1 *we lose 1 of 3 solutions so that* $\mathcal{P}_{\#\mathcal{AS}}(\langle\overline{c}\rangle) = 1/3$*, we have* $\mathcal{P}_{\#\mathcal{F}}(\langle\overline{c}\rangle) = 1/4$*.*

From Lemma 4, we immediately conclude:

Corollary 2. $\mathcal{P}_{\omega_{\#\mathcal{F}}}(\delta) = 1$ *if and only if* $\delta \in \Delta_{ms}^{\Pi}$*. In contrast, for all* $\delta \in \Delta_s^{\Pi}$ *we have* $\mathcal{P}_{\#\mathcal{AS}}(\delta) \leq \frac{|\mathcal{AS}(\Pi)|-1}{|\mathcal{AS}(\Pi)|}$*.*

Corollary 2 states that, in contrast to $\mathcal{P}_{\#\mathcal{AS}}$, the facet counting pace $\mathcal{P}_{\omega_{\#\mathcal{F}}}$ detects whether users sit on a unique solution. More importantly it is the better option to find a viable implementation of the pace of navigation for our framework. While in that sense using the relative weight $\omega_{\#\mathcal{F}}$ is beneficial, unfortunately it is not *min-inline*.

Example 6. *We consider* $\Pi_2 = \{a \,|\, b \,|\, c;\ d \,|\, e \leftarrow b;\ f \leftarrow c\}$ *where* $\mathcal{AS}(\Pi_2) = \{\{a\}, \{b, d\}, \{b, e\}, \{c, f\}\}$*. While* $\overline{a} \in min_{\omega_{\#\mathcal{F}}}(\Pi_2)$ *and* $\overline{c} \notin min_{\omega_{\#\mathcal{F}}}(\Pi_2)$*, we have* $|\mathcal{AS}(\Pi_2^{\langle\overline{a}\rangle})| = |\mathcal{AS}(\Pi_2^{\langle\overline{c}\rangle})|$*. Hence, the relative weight* $\omega_{\#\mathcal{F}}$ *is not min-inline.*

We suspect that the property max-inline is not satisfied by the weight $\omega_{\#\mathcal{F}}$ as we observed in our experiments that the activation of some facets, which had no maximal $\omega_{\#\mathcal{F}}$ weight, lead to smaller answer set spaces than the activation of facets which had maximal $\omega_{\#\mathcal{F}}$ weight. An actual counterexample is still open.

Supported Model Counting. Another approach to comparing sub-spaces with respect to their size, while avoiding answer set counting, is to count supported models. An interpretation X is called *supported model* (Apt, Blair, and Walker 1988; Alviano and Dodaro 2016b) of Π if X satisfies Π and for all $\alpha \in X$ there is a rule $r \in \Pi$ such that $H(r) \cap X = \{\alpha\}$, $B^+(r) \subseteq X$ and $B^-(r) \cap X = \emptyset$. By $\mathcal{S}(\Pi)$ we denote the supported models of Π. It holds that $\mathcal{AS}(\Pi) \subseteq \mathcal{S}(\Pi)$ (Marek and Subrahmanian 1992), but the converse does not hold in general. We define *supp weights*, by which in short we refer to supported model counting weights, accordingly as follows.

Definition 15. *The supp weight* $\omega_{\#\mathcal{S}}$ *is defined by* $\#\mathcal{S} : \Pi^{\delta} \mapsto |\mathcal{S}(\Pi)|$.

The *positive dependency graph* of program Π is $G(\Pi) := (\mathcal{A}(\Pi), \{(\alpha_1, \alpha_0) \mid \alpha_1 \in B^+(r), \alpha_0 \in H(r), r \in \Pi\})$. Π is called *tight*, if $G(\Pi)$ is acyclic. If Π is tight, then its supported models and answer sets coincide (Fages 1994).

Corollary 3. *If* Π *is tight, then for all* $f \in \mathcal{F}(\Pi^{\delta})$ *we have that* $\omega_{\#\mathcal{AS}}(f, \Pi^{\delta}, \delta') = \omega_{\#\mathcal{S}}(f, \Pi^{\delta}, \delta')$*.*

Due to the fact that unsatisfiable programs may have supported models (Marek and Subrahmanian 1992), $\omega_{\#\mathcal{S}}$ is not reliable. Moreover the following example shows that $\omega_{\#\mathcal{S}}$ is neither min-inline, nor max-inline.

Example 7. *We consider* $\Pi_3 = \{a;\ b \leftarrow a, {\sim}c;\ c \leftarrow {\sim}b, {\sim}d;\ d \leftarrow d\}$ *with* $\mathcal{S}(\Pi_3) = \{\{a, b\}, \{a, c\}, \{a, b, d\}\}$ *and* $\mathcal{AS}(\Pi_3) = \{\{a, b\}, \{a, c\}\}$*. The facets of* Π_3 *are given by* $\mathcal{F}(\Pi_3) = \{b, c, \overline{b}, \overline{c}\}$*. Then, the facets b and* $\overline{c}$ *both have supp weight 1 and thus are minimal weighted, and the facets c and* $\overline{b}$ *have supp weight 2 and thus are maximal weighted. As* $|\mathcal{AS}(\Pi_3^{\langle b\rangle})| = |\mathcal{AS}(\Pi_3^{\langle c\rangle})| = 1$ *we see that both the minimal and the maximal weighted facets w.r.t. supp weights have the same number of answer sets. Hence,* $\omega_{\#\mathcal{S}}$ *is neither min-inline, nor max-inline.*

Although $\omega_{\#\mathcal{S}}$ does not satisfy min-inline and max-inline, it shares some properties with $\omega_{\#\mathcal{AS}}$ and $\omega_{\#\mathcal{F}}$.

Lemma 5 ($\star$). *For program* Π *and* $\delta \in \Delta_s^{\Pi}$*, if* $f \in \mathcal{F}(\Pi^{\delta})$*, then* $\mathcal{S}(\Pi^{\langle\delta, f\rangle}) = \{X \in \mathcal{S}(\Pi^{\delta}) \mid X \models f\} \subset \mathcal{S}(\Pi^{\delta})$*.*

Theorem 4 ($\star$). *Weight* $\omega_{\#\mathcal{S}}$ *is safe-zooming and splitting.*

Computing supp weights is computationally easier.

Lemma 6 ($\star$). *Outputting the supp weight* $\omega_{\#\mathcal{S}}$ *for a given program* Π *and route* δ *is* $\#\mathrm{P}$*-complete.*

However, recalling Lemma 3, note that counting facets is still the least expensive method.

In summary, we can characterize and compare the introduced weights as given in Table 1. Every weight has its advantages that should be used to leverage performance, or characterize the solution space and its sub-spaces. While

	saf	rel	spl	min	max
$\omega_{\#\mathcal{AS}}$	✓	✓	✓	✓	✓
$\omega_{\#\mathcal{F}}$	✓	✗	✗	✗	?
$\omega_{\#\mathcal{S}}$	✓	✗	✓	✗	✗

Table 1: Comparing weights regarding `saf`: is safe-zooming, `spl`: is splitting, `rel`: is reliable, `min`: is min-inline and `max`: is max-inline.

counting solutions is the most desirable choice, computing $\omega_{\#\mathcal{AS}}$ is hard. Our results show that, when narrowing down the space by strictly pruning the max./min. number of solutions, at least for tight programs, $\omega_{\#\mathcal{S}}$ is the best choice, as it coincides with $\omega_{\#\mathcal{AS}}$ while remaining less expensive. In general, in contrast to $\omega_{\#\mathcal{AS}}$, relative weights come with different use cases regarding their interpretation. Even though $\omega_{\#\mathcal{F}}$ has deficiencies, it satisfies the most essential property, namely being safe-zooming, and provides information on the similarity/diversity of solutions with respect to a route. To conclude, while facet-counting is the most promising method for distinguishing zoom-in effects of facets regarding computational feasibility, counting supported models of tight programs is precise about zoom-in effects.

Weighted Navigation Modes

In the following, we introduce two new navigation modes, called *strictly goal-oriented* and *explore*. They can be understood as special cases of goal-oriented navigation.

Definition 16. *Let* Π *be a program,* $\delta \in \Delta_s^\Pi$ *and* $f \in \mathcal{F}(\Pi)$. *The* strictly goal-oriented *mode* $\nu_{sgo}^{\#}$ *and the* explore *mode* $\nu_{expl}^{\#}$ *are defined by:*

$$\nu_{sgo}^{\#}(\delta, f) := \begin{cases} \mathcal{AS}(\Pi^{\langle\delta,f\rangle}), & \text{if } f \in max_{\omega_\#}(\Pi^\delta); \\ \mathcal{AS}(\Pi^\delta), & \text{otherwise.} \end{cases}$$

$$\nu_{expl}^{\#}(\delta, f) := \begin{cases} \mathcal{AS}(\Pi^{\langle\delta,f\rangle}), & \text{if } f \in min_{\omega_\#}(\Pi^\delta); \\ \mathcal{AS}(\Pi^\delta), & \text{otherwise.} \end{cases}$$

Corollary 4. $\nu_{sgo}^{\#}$ *and* $\nu_{expl}^{\#}$ *avoid unsafe routes, hence we can use the restriction* $\omega_\#|_X$ *of* $\omega_\#$ *where* $X := \{(f, \delta, \epsilon) \mid f \in \mathcal{F}(\Pi), \delta \in \Delta_s^\Pi\}$.

While in strictly goal-oriented mode the objective is to "rush" through the solution space, navigating at the highest possible pace in order to reach a unique solution as quick as possible, explore mode keeps the user off one unique solution as long as possible, aiming to provide her with as many solutions as possible to explore while "strolling" between sub-spaces. As a consequence, regardless of whether absolute or relative weights are used, during weighted navigation some (partial) solutions may be unreachable.

Example 8. *Consider* Π_2 *from Example 6 where we can choose from facets* $\mathcal{F}(\Pi_2) = \{a, b, c, d, e, f, \overline{a}, \overline{b}, \overline{c}, \overline{d}, \overline{e}, \overline{f}\}$ *and* $max_{\omega_{\#\mathcal{AS}}}(\Pi_2) = \{a, c, d, e, f\} = max_{\omega_{\#\mathcal{F}}}(\Pi_2)$. *Thus, any solution* $X \in \mathcal{AS}(\Pi_2) = \{\{a\}, \{b, d\}, \{b, e\}, \{c, f\}\}$ *s.t.* $b \in X$ *is unreachable in* $\nu_{sgo}^{\#\mathcal{AS}}$ *and* $\nu_{sgo}^{\#\mathcal{F}}$. *Since* $\omega_{\#\mathcal{AS}}$ *is splitting, it follows that* $min_{\#\mathcal{AS}}(\Pi_2) = \{\overline{a}, \overline{c}, \overline{d}, \overline{e}, \overline{f}\}$. *Hence, navigating in* $\nu_{expl}^{\#\mathcal{AS}}$, *one has to sacrifice either partial solution* a, *or* c *and* f *right in the beginning. Furthermore, since* $min_{\omega_{\#\mathcal{F}}}(\Pi_2) = \{\overline{a}, \overline{d}, \overline{e}\}$, *right in the beginning of navigating in* $\nu_{expl}^{\#\mathcal{F}}$, *one has to sacrifice partial solution* a, d, *or* e.

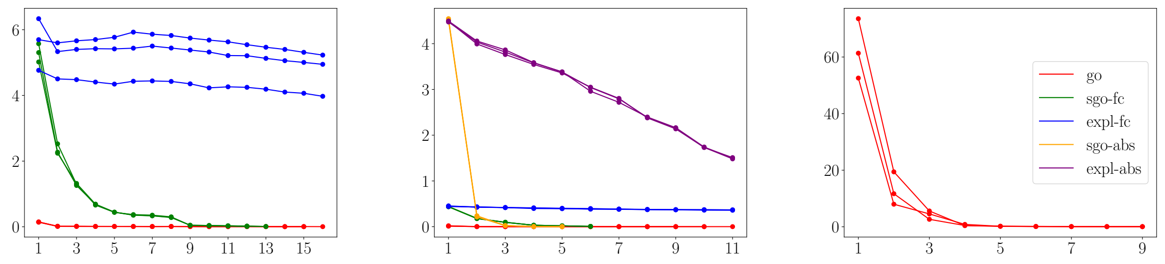
Implementation and Evaluation

To study the feasibility of our framework, we implemented the *faceted answer set browser* (`fasb`) on top of the `clingo` solver. In particular, we conducted experiments on three instance sets that range from large solution spaces to complex encodings in order to verify the following two hypotheses: (**H1**) weighted faceted navigation can be performed in reasonable time in an incomprehensible solution space associated with product configuration; and (**H2**) the feasibility of our framework depends on the complexity of the given problem, i.e., program. The implementation and experiments are publicly available (Fichte, Gaggl, and Rusovac 2021a,b). We follow standard guidelines for empirical evaluations (van der Kouwe et al. 2018; Fichte et al. 2021a)

Environment. `fasb` is designed for desktop systems, enabling users to practicably explore the solution space in an interactive way. Hence, runtime was limited to 600 seconds and the experiments were run on an eight core Intel i7-10510U CPU 1.8 GHz with 16 GB of RAM, running Manjaro Linux 21.1.1 (kernel 5.10.59-1-MANJARO). Runtime was measured in elapsed time by timers in `fasb` itself.

Design of Experiment. Currently, we miss data on real user behavior. Thus, we run three iterations of random navigation steps in each of the implemented modes, to simulate a user and avoid bias regarding the choice of steps. For *go*, *sgo-fc*, and *sgo-abs*, we use the `--random-safe-walk` call, which in the provided mode performs random steps until the current route is maximal safe, e.g., in *sgo-fc* and *sgo-abs* it computes maximal weighted facets and then chooses one of them to activate randomly. Since, in practice, using *expl-fc* and *expl-abs*, we do not necessarily aim to arrive at a unique solution, we use `--random-safe-steps` for *expl-fc* and *expl-abs* and provide the maximum number n of steps among iterations in *go*, which performs n random steps in the provided mode. We measure the elapsed time for a mode to filter current facets according to its strategy, then, using the mentioned calls, we randomly select a facet thereof to activate, until we reach a unique solution or took n steps. For any mode except *go*, we ignore the elapsed time of `--activate`, for *go* we solely measure elapsed time of the `--activate` call, which in the case of *go* includes runtime of computing facets. `fasb` computes the initial facets at startup, which are used throughout further computations, in particular when performing a first step. Thus, we add elapsed time, due to startup, to the first result in each mode.

Instances. To study (**H1**), we inspect product configuration (Gorczyca 2020) where users may configure PC components over a large solutions space until a full configuration is obtained. To verify (**H2**), we select instances from *abstract argumentation* using the ASPARTIX encodings (Egly,



(a) PC configuration. (b) Stable extensions. (c) Preferred extensions.

Figure 2: Comparing random steps in several navigation modes. The x-axis refers to the respective navigation step, the y-axis refers to the execution time in seconds. Colors in Figure 2a and 2b follow the legend as given in Figure 2c.

Gaggl, and Woltran 2010; Gaggl et al. 2015; Dvořák et al. 2020) `stable.lp` and `preferred-cond-disj.dl`. There, brave and cautious reasoning in abstract argumentation is of higher complexity for preferred semantics than for the stable semantics (Baroni, Caminada, and Giacomin 2011). For the *stable* argumentation semantics, the problems can be encoded as normal programs. Whereas for preferred, one needs disjunctive programs. As input instance, we used the abstract argumentation framework `A/3/ferry2.pfile-L3-C1-06.pddl.1.cnf.apx` from the benchmark set of (ICCMA'17) (Gaggl et al. 2020). There, solutions of both semantics coincide with exactly 7696 answer sets.

Observations and Results. In the beginning of PC configuration, we choose from 340 facets resulting in on average 15 steps in *go* and 13 steps in *sgo-fc* to reach a uniqe solution. Taking 16 steps in *expl-fc*, throughout all iterations the facet-counting pace of the obtained route is 9%. The number of solutions for the respective generated benchmark `pc_config` remains unknown. Running `clingo` for over 9 hours resulted in more than $1.3 \cdot 10^9$ answer sets. As expected, for more than a billion solutions, *sgo-abs* and *expl-abs* timed out in the first step. Inspecting Figure 2a, we see that *sgo-fc* execution time drops significantly from Step 1 to 5, which originates in the fact that Steps 1 to 5 throughout all iterations on the average decreased the number of remaining facets by 35%. Consequently, it reduces the number of facets to compute weights for and leads to shorter execution times. In *expl-fc*, on the other hand, throughout all iterations each step decreases the facet-count by 2. Except for one outlier, this leads to slowly decreasing, but in general, similar execution times. Figures 2b and 2c illustrate the execution times for navigation steps in the argumentation instances. As expected, we see no timeouts when navigating through 7696 stable extensions. Whereas exploring 7696 preferred extensions, works only in mode *go*. Computing cautious consequences was most expensive when considering the execution time of processes at startup for preferred extensions, which emphasizes (**H2**). From Figure 2b, we see that *go*, *sgo-fc*, and *expl-fc* show a similar trend to Figure 2a. While *go* and *expl-fc* remain rather steady in execution time, *sgo-fc* drops in the first steps. Moreover, we observe that the execution time of *expl-abs*, in contrast to *expl-fc*, decreases noticeably with every step indicating that counting less answer sets in each step becomes easier, whereas counting facets does not. Throughout all iterations, while *sgo-fc* needs 6 steps, *sgo-abs* only needs 5 steps to reach a unique solution. The significant drop between Step 1 and 2 in *sgo-abs* originates in zooming in by 93%, pruning 7152 out of 7696 solutions.

Summary. In general (**H2**) the feasibility of weighted navigation depends on the complexity of the given problem. Regarding product configuration, associated with a large and incomprehensible solution space (**H1**), weighted navigation can be performed in reasonable time using `fasb`.

Conclusion and Future Work

We provide a formal, dynamic, and flexible framework for navigating through subsets of answer sets in a systematic way. We introduce absolute and relative weights to quantify the size of the search space when reasoning under assumptions (facets) as well as natural navigation operations. In a systematic comparison, we prove which weights can be employed under the search space navigation operations. In addition, we illustrate the computational complexity for computing the weights. Our framework is intended as an additional layer on top of a solver, adding functionality for systematically manipulating the size of the solution space during (faceted) answer set navigation. Our implementation, on top of the solver `clingo`, demonstrates feasibility of our framework for an incomprehensible solution space.

For future work, we believe that an interesting question is to research relative weights which preserve the properties min-inline and max-inline. Our framework may have interesting applications for characterizing features of a configuration space (Sundermann, Thüm, and Schaefer 2020). We are interested in visual support for navigating the ASP solution space by extending the work of (Yang, Gaggl, and Rudolph 2020). Furthermore, we aim to investigate whether supported model counting is in fact practically feasible using recent developments in propositional model counting (Fichte, Hecher, and Hamiti 2021; Fichte et al. 2021b; Fichte, Hecher, and Roland 2021; Korhonen and Järvisalo 2021), ASP (Fichte and Hecher 2019), or approximate counting (Kabir et al. 2022).

Acknowledgements

The authors are stated in alphabetic order. This research was partially funded by the DFG through the Collaborative Research Center, Grant TRR 248 (see https://perspicuous-computing.science project ID 389792660), the Bundesministerium für Bildung und Forschung (BMBF), Grant 01IS20056_NAVAS, a Google Fellowship at the Simons Institute, and the Austrian Science Fund (FWF), Grant Y698. Work has partially been carried out while Johannes Fichte was visiting the Simons Institute for the Theory of Computing.

References

Alrabbaa, C.; Rudolph, S.; and Schweizer, L. 2018. Faceted Answer-Set Navigation. In Benzmüller, C.; Ricca, F.; Parent, X.; and Roman, D., eds., *Proc. of RuleML+RR'18*, 211–225. Springer.

Alviano, M.; Calimeri, F.; Dodaro, C.; Fuscà, D.; Leone, N.; Perri, S.; Ricca, F.; Veltri, P.; and Zangari, J. 2017. The ASP System DLV2. In Balduccini, M.; and Janhunen, T., eds., *Proc. of LPNMR'17*, volume 10377 of *LNCS*, 215–221. Cham: Springer. ISBN 978-3-319-61660-5.

Alviano, M.; and Dodaro, C. 2016a. Anytime answer set optimization via unsatisfiable core shrinking. *TPLP*, 16(5-6): 533—551.

Alviano, M.; and Dodaro, C. 2016b. Completion of Disjunctive Logic Programs. In Kambhampati, S., ed., *Proc. of IJCAI'16*, 886–892. IJCAI/AAAI Press.

Alviano, M.; Dodaro, C.; Leone, N.; and Ricca, F. 2015. Advances in WASP. In Calimeri, F.; Ianni, G.; and Truszczynski, M., eds., *Proc. of LPNMR'15*, 40–54. Springer.

Alviano, M.; Romero, J.; and Schaub, T. 2018. Preference Relations by Approximation. In Thielscher, M.; and Toni, F., eds., *Proc. of KR'18*, 2–11.

Apt, K. R.; Blair, H. A.; and Walker, A. 1988. Towards a theory of declarative knowledge. In *Foundations of deductive databases and logic programming*, 89–148. Elsevier.

Baroni, P.; Caminada, M.; and Giacomin, M. 2011. An introduction to argumentation semantics. *Knowledge Eng. Review*, 26: 365–410.

Brewka, G. 2004. Complex Preferences for Answer Set Optimization. In Dubois, D.; Welty, C. A.; and Williams:, M.-A., eds., *Proc. of KR'04*, 213–223. The AAAI Press.

Brewka, G.; Delgrande, J.; Romero, J.; and Schaub, T. 2015. asprin: Customizing answer set preferences without a headache. In *Proc. of AAAI'15*.

Brewka, G.; Eiter, T.; and Truszczyński, M. 2011. Answer set programming at a glance. *Communications of the ACM*, 54(12): 92–103.

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. *TPLP*, 20(2): 294–309.

Dimopoulos, Y.; Nebel, B.; and Koehler, J. 1997. Encoding planning problems in nonmonotonic logic programs. In Steel, S.; and Alami, R., eds., *Proc. of ECP'97*, 169–181. Springer.

Dodaro, C.; Gasteiger, P.; Reale, K.; Ricca, F.; and Schekotihin, K. 2019. Debugging Non-ground ASP Programs: Technique and Graphical Tools. *TPLP*, 19(2): 290–316.

Dvořák, W.; Gaggl, S. A.; Rapberger, A.; Wallner, J. P.; and Woltran, S. 2020. The ASPARTIX System Suite. In Prakken, H.; Bistarelli, S.; Santini, F.; and Taticchi, C., eds., *Proc. of COMMA'20*, volume 326 of *FAIA*, 461–462. IOS Press.

Eén, N.; and Sörensson, N. 2003. An Extensible SAT-solver. In Giunchiglia, E.; and Tacchella, A., eds., *Proc. of SAT'03*, 502–518. Springer.

Egly, U.; Gaggl, S. A.; and Woltran, S. 2010. Answer-set programming encodings for argumentation frameworks. *Argument & Computation*, 1(2): 147–177.

El-Khatib, O.; Pontelli, E.; and Son, T. C. 2005. Justification and Debugging of Answer Set Programs in ASP. In *Proc. of AADEBUG'05*, 49–58. ACM. ISBN 1595930507.

Everardo, F. 2017. Towards an automated multitrack mixing tool using answer set programming. In *14th Sound and Music Computing Conf.*

Everardo, F.; Janhunen, T.; Kaminski, R.; and Schaub, T. 2019. The Return of xorro. In Balduccini, M.; Lierler, Y.; and Woltran, S., eds., *Proc. of LPNMR'19*, 284–297. Springer.

Fages, F. 1994. Consistency of Clark's completion and existence of stable models. *Journal of Methods of logic in computer science*, 1(1): 51–60.

Fichte, J. K.; Gaggl, S. A.; and Rusovac, D. 2021a. Rushing and Strolling among Answer Sets - Navigation Made Easy (Experiments). https://doi.org/10.5281/zenodo.5780050.

Fichte, J. K.; Gaggl, S. A.; and Rusovac, D. 2021b. Rushing and Strolling among Answer Sets - Navigation Made Easy (Faceted Answer Set Browser fasb). https://doi.org/10.5281/zenodo.5767981.

Fichte, J. K.; and Hecher, M. 2019. Treewidth and Counting Projected Answer Sets. In Balduccini, M.; Lierler, Y.; and Woltran, S., eds., *Proc. of LPNMR'19*, volume 11481 of *LNCS*, 105–119. Philadelphia, PA, USA: Springer.

Fichte, J. K.; Hecher, M.; and Hamiti, F. 2021. The Model Counting Competition 2020. *ACM Journal of Experimental Algorithmics*, 26(13).

Fichte, J. K.; Hecher, M.; McCreesh, C.; and Shahab, A. 2021a. Complications for Computational Experiments from Modern Processors. In Michel, L. D., ed., *Proc. of CP'21*, volume 210 of *LIPIcs*, 25:1–25:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Fichte, J. K.; Hecher, M.; and Roland, V. 2021. Parallel Model Counting with CUDA: Algorithm Engineering for Efficient Hardware Utilization. In Michel, L. D., ed., *Proc. of CP'21*, volume 210 of *LIPIcs*, 24:1–24:20. Dagstuhl, Germany: Dagstuhl Publishing. ISBN 978-3-95977-211-2.

Fichte, J. K.; Hecher, M.; Thier, P.; and Woltran, S. 2021b. Exploiting Database Management Systems and Treewidth for Counting. *TPLP*, 1–30.

Gaggl, S. A.; Linsbichler, T.; Maratea, M.; and Woltran, S. 2020. Design and results of the Second International Competition on Computational Models of Argumentation. *Artif. Intell.*, 279.

Gaggl, S. A.; Manthey, N.; Ronca, A.; Wallner, J. P.; and Woltran, S. 2015. Improved Answer-Set Programming Encodings for Abstract Argumentation. *TPLP*, 15(4-5): 434–448.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2012. Answer set solving in practice. *Synthesis lectures on artificial intelligence and machine learning*, 6(3): 1–238.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2014. Clingo = ASP + Control: Preliminary Report. *CoRR*, abs/1405.3694.

Gebser, M.; Kaminski, R.; König, A.; and Schaub, T. 2011. Advances in gringo series 3. In Delgrande, J. P.; and Faber, W., eds., *Proc. of LPNMR'11*, 345–351. Springer.

Gebser, M.; Kaminski, R.; and Schaub, T. 2011. Complex optimization in answer set programming. *TPLP*, 11(4-5): 821–839.

Gebser, M.; Pührer, J.; Schaub, T.; and Tompits, H. 2008. A Meta-Programming Technique for Debugging Answer-Set Programs. In *Proc. of AAAI '08*.

Gelfond, M.; and Lifschitz, V. 1988. The Stable Model Semantics For Logic Programming. In Kowalski, R. A.; and Bowen, K. A., eds., *Proc. of ICLP/SLP'88*, volume 2, 1070–1080. MIT Press.

Gelfond, M.; and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Comput.*, 9(3/4): 365–386.

Gorczyca, P. 2020. Configuration Problem ASP Encoding Generator. https://doi.org/10.5281/zenodo.5777217.

Kabir, M.; Everardo, F.; Shukla, A.; Fichte, J. K.; Hecher, M.; and Meel, K. 2022. ApproxASP – A Scalable Approximate Answer Set Counter. In *Proc. of AAAI'22*. The AAAI Press.

Korhonen, T.; and Järvisalo, M. 2021. Integrating Tree Decompositions into Decision Heuristics of Propositional Model Counters. In Michel, L. D., ed., *Proc. of CP'21*, volume 210 of *LIPIcs*, 8:1–8:11. Dagstuhl Publishing. ISBN 978-3-95977-211-2.

Lee, J.; Talsania, S.; and Wang, Y. 2017. Computing LPMLN using ASP and MLN solvers. *TPLP*, 17(5-6): 942–960.

Lifschitz, V. 1999. Action languages, answer sets, and planning. In *The Logic Programming Paradigm*, 357–373. Springer.

Marek, W.; and Subrahmanian, V. 1992. The relationship between stable, supported, default and autoepistemic semantics for general logic programs. *Theor. Comput. Sci.*, 103(2): 365–386.

Nogueira, M.; Balduccini, M.; Gelfond, M.; Watson, R.; and Barry, M. 2001. An A-Prolog decision support system for the Space Shuttle. In Ramakrishnan, I. V., ed., *Proc. of PADL'01*, 169–183. Springer.

Oetsch, J.; Pührer, J.; and Tompits, H. 2018. Stepwise debugging of answer-set programs. *TPLP*, 18(1): 30–80.

Shchekotykhin, K. M. 2015. Interactive Query-Based Debugging of ASP Programs. In Bonet, B.; and Koenig, S., eds., *Proc. of AAAI'15*, 1597–1603. AAAI Press.

Soininen, T.; and Niemelä, I. 1999. Developing a declarative rule language for applications in product configuration. In Gupta, G., ed., *Proc. of PADL'99*, 305–319. Springer.

Soininen, T.; Niemelä, I.; Tiihonen, J.; and Sulonen, R. 2001. Configuration Knowledge With Weight Constraint Rules. In Provetti, A.; and Son, T. C., eds., *Proc. of ASP'01*, volume 1.

Son, T. C.; Sabuncu, O.; Schulz-Hanke, C.; Schaub, T.; and Yeoh, W. 2016. Solving Goal Recognition Design Using ASP. In *Proc. of AAAI'16*, 3181–3187.

Stockmeyer, L. J. 1976. The polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1): 1–22.

Sundermann, C.; Thüm, T.; and Schaefer, I. 2020. Evaluating# SAT solvers on industrial feature models. In *Proc. of the 14th Int. Working Conf. on Variability Modelling of Software-Intensive Systems*, 1–9.

Tiihonen, J.; Soininen, T.; Niemelä, I.; and Sulonen, R. 2003. A practical tool for mass-customising configurable products. In *Proc. of ICED'03*.

Toda, S. 1991. PP is as Hard as the Polynomial-Time Hierarchy. *SIAM J. Comput.*, 20(5): 865–877.

Tunkelang, D. 2009. Faceted Search. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 1(1).

van der Kouwe, E.; Andriesse, D.; Bos, H.; Giuffrida, C.; and Heiser, G. 2018. Benchmarking Crimes: An Emerging Threat in Systems Security. *CoRR*, abs/1801.02381.

Vos, M. D.; Kisa, D. G.; Oetsch, J.; Pührer, J.; and Tompits, H. 2012. Annotating answer-set programs in Lana. *TPLP*, 12(4-5): 619–637.

Yang, M.; Gaggl, S. A.; and Rudolph, S. 2020. Neva - Extension Visualization for Argumentation Frameworks. In Prakken, H.; Bistarelli, S.; Santini, F.; and Taticchi, C., eds., *Proc. of COMMA 2020*, volume 326 of *FAIA*, 477–478. IOS Press.