Representative Answer Sets: Collecting Something of Everything

Elisa Böhl, Sarah Alice Gaggl and Dominik Rusovac

Logic Programming and Argumentation Group, TU Dresden, Germany
{elisa.boehl, sarah.gaggl, dominik.rusovac}@tu-dresden.de

Abstract. Answer set programming (ASP) is a popular problem solving paradigm with applications in planning and configuration. In practice, the number of answer sets can be overwhelmingly high, which naturally causes interest in a concise characterisation of the solution space in terms of representative answer sets. We establish a notion of representativeness that refers to the entropy of specified target atoms within a collection of answer sets. Accordingly, we propose different approaches for collecting such representative answer sets, based on answer set navigation. Finally, we conduct experiments using our prototypical implementation, which reveals promising results.

1 Introduction

Answer set programming (ASP) [27] is a declarative problem modeling and solving paradigm with applications in knowledge representation, artificial intelligence, planning, and many more. It is widely used to solve difficult search problems while allowing compact modeling [16]. In ASP, a problem is represented as a set of rules over a set of atoms, called *logic program*. Models of a program under the stable semantics [17, 18] form its solutions, so-called *answer sets*. ASP can even be used to solve optimization problems [15, 2], where specified optimization criteria lead the solver to the optimal solutions. However, combinatorial search problems such as product configuration [29] or test case generation and planning [30], can have an exponential number of solutions. In these scenarios a user might not have an optimization criterion in mind, but rather would like to see what kind of solutions are possible at all.

Example 1 *Suppose we need to schedule n integration tests by means of their ordering. Consider the following simple encoding:*

$$test(1 \dots n).$$
$$\{run(X, M) \ : \ M = 1 \dots n\} = 1 \leftarrow test(X).$$

Assuming $n = 10$ is quite optimistic, regarding the sheer number of functions and specifications usually found in real-world professional code bases, but such an unrealistically small problem instance already gives rise to 10^{10} possible test plans. Luckily, in practice, integration tests are subject to several dependencies and requirements, e.g., some tests have to run in parallel, others one after another or on separate threads, and so on, which naturally restricts the number of valid test plans. However, the search space will mostly still remain large, e.g., adding the constraint that no tests may run in parallel still admits $10! = 3628800$ answer sets.

As a consequence it becomes very hard to assess the scope of potential liabilities to test against. Running all test plans is not particularly productive though, and possibly infeasible. So, how can we efficiently choose a reasonably large selection of test plans to run, in order to possibly cover a crucial portion of potential issues in our code base?

In this paper we seek a way of collecting something of everything, that is, collecting answer sets that (S) represent all specified target atoms of a program, while (D) being as diverse as possible at the same time. In general terms, (S) and (D) resemble well-known NP-complete problems [21]. (S) stems from the *set cover* problem, which essentially asks for a set of subsets S of a set M s.t. $\bigcup S = M$ and $|S| \leq k \in \mathbb{N}$; (D) is the *set packing* problem, which asks for a set of pairwise disjoint subsets S of a set M s.t. $|S| \geq k \in \mathbb{N}$; and collecting something of everything, say (S+D), essentially corresponds to the *exact cover* problem, which asks for a partition $S \in 2^M$ of a set M. You can think of M being the set of atoms of a logic program and S being the answer sets.

Example 2 *For the sake of easier understanding, consider atoms consisting of symbols $\{\blacklozenge, \blacktriangle, \blacksquare, \bullet, \blacktriangledown\}$ and the following answer sets $\{\{\blacklozenge, \blacktriangle\}, \{\blacktriangledown, \bullet, \blacktriangle\}, \{\blacktriangledown, \blacksquare, \blacktriangle\}\}$. Every solution contains $\blacktriangle$, so they all have something in common. Collections c_1=$\{\{\blacklozenge, \blacktriangle\}, \{\blacktriangledown, \bullet, \blacktriangle\}\}$ and c_2=$\{\{\blacklozenge, \blacktriangle\}, \{\blacktriangledown, \blacksquare, \blacktriangle\}\}$ differ completely up to $\blacktriangle$, whereas c_3=$\{\{\blacktriangledown, \bullet, \blacktriangle\}, \{\blacktriangledown, \blacksquare, \blacktriangle\}\}$ and c_4=$\{\{\blacklozenge, \blacktriangle\}, \{\blacktriangledown, \bullet, \blacktriangle\}, \{\blacktriangledown, \blacksquare, \blacktriangle\}\}$ do not, as $\blacktriangledown$ occurs in two answer sets. While c_3 and c_4 are not completely diverse, c_1 and c_2 do not represent each symbol. Thus, due to the small number of answer sets, indeed it turns out that all answer sets $\{\{\blacklozenge, \blacktriangle\}, \{\blacktriangledown, \bullet, \blacktriangle\}, \{\blacktriangledown, \blacksquare, \blacktriangle\}\}$ form the desired collection in this minimal setting, as they cover something of everything, i.e., the entire scope of symbols is represented in the most diverse way possible.*

Now while there exist algorithms for solving the latter problems, such as Knuth's algorithm X [23] for the exact cover problem, they would require enumerating all answer sets, which in practice is often infeasible. For partial enumeration we are potentially missing out on huge parts of the input.

Recently, several ideas have been proposed to overcome the difficulty of comprehending large solution spaces [13, 1], which coined the concept of *answer set navigation*, a framework that aims at exploring answer sets with efficient methods, for several purposes, such as, for instance, finding diverse solutions [4, 5, 10]. Using and extending certain concepts of answer set navigation, we are convinced

that collecting something of everything contributes to (1) a better understanding of configuration problems as it reflects the scope of what is possible; and (2) allows for covering the variety of solutions in applications such as testing. The main contributions of this work are the following.

- We discuss and formalise a novel concept of representativeness of a collection of answer sets by means of entropy;
- we suggest several naive approaches and more sophisticated heuristics relying on reducing uncertainty and maximising information gain; and
- finally, we implemented the proposed methods and conducted preliminary experiments on novel suitable benchmarks.

Related Work. A closely related topic are so called *sequence covering arrays* (SCAs) [25, 24], which have applications in testing. An SCA is an array that consists of permutations of, say, m test cases, ensuring that any n out of m test cases, will be tested in every n-way order at least once. There exist several approaches on finding SCAs using ASP [3, 12], which apply to our setting. However, finding SCAs is a special case for us, as we are not primarily focused on finding all n-permutations over a specified set of m elements. Our approach is more generic, in the sense that we are aiming at collections of answer sets that cover anything possible, in as many different existing combinations, as possible. This then does not only apply to test planning or generation, but also to, for instance, configuration problems, where something of everything may give an impression of the scope of possibly billion configurations.

Diversity has also been studied in the context of Conjunctive Queries [28] and Constraint Satisfaction Problems [20].

2 Preliminaries

First, we recall basic notions of ASP. Then, we introduce fundamental notions of answer set navigation.

Answer Set Programming. We fix a countable set $\mathcal{U}$ of *(domain) elements*, also called *constants*. An *atom* is an expression $p(t_1, \ldots, t_n)$, where p is a *predicate* of arity $n \geq 0$ and each t_i is either a variable or an element from $\mathcal{U}$. An atom is *ground* if it is free of variables. $B_\mathcal{U}$ denotes the set of all ground atoms over $\mathcal{U}$. A *(disjunctive) rule* r is of the form $a_0 \mid \ldots \mid a_n \leftarrow a_{n+1}, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_k$ where $0 \leq n \leq m \leq k$ and $a_0, \ldots, a_k$ are atoms and $\neg$ denotes *default negation*. We define $H(r) \coloneqq \{a_0, \ldots, a_n\}$, called *head* of rule r. The *body* of r consists of $B^+(r) \coloneqq \{a_{n+1}, \ldots, a_m\}$ and $B^-(r) \coloneqq \{a_{m+1}, \ldots, a_k\}$. A rule r is *ground* if no variable occurs in r. A *(normal disjunctive logic) program* Π is a finite set of rules. For any program Π, let U_Π be the set of all constants appearing in Π. $Gr(\Pi)$ is the set of rules $r\sigma$ obtained by applying, to each rule $r \in \Pi$, all possible substitutions σ from the variables in r to elements of U_Π. The set $at(r)$ consists of all ground atoms of r. By $at(\Pi) \coloneqq \bigcup_{r \in \Pi} at(r)$ we define the set of all ground atoms of Π. An *interpretation* $I \subseteq B_\mathcal{U}$ *satisfies* a ground rule $r \in \Pi$, iff $H(r) \cap I \neq \emptyset$ whenever $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$. I satisfies a ground program Π, if I satisfies each rule $r \in \Pi$. A non-ground rule r (resp., a program Π) is satisfied by an interpretation I iff I satisfies all groundings of r (resp., $Gr(\Pi)$). The *GL-reduct* Π^I is defined by $\Pi^I \coloneqq \{H(r) \leftarrow B^+(r) \mid I \cap B^-(r) = \emptyset, r \in \Pi\}$. I is an *answer set*, denoted by $I \in \mathcal{AS}(\Pi)$, if I satisfies Π^I and I is subset-minimal.

Answer Set Navigation. By the term *solution space* of Π we refer to $2^{\mathcal{AS}(\Pi)}$. An atom $a \in at(\Pi)$ induces a *facet* $f \in \{a, \neg a\}$, if $a \in F^+_\Pi \coloneqq \bigcup \mathcal{AS}(\Pi) \setminus \bigcap \mathcal{AS}(\Pi)$. A facet $f \in F_\Pi = F^+_\Pi \cup \{\neg a \mid a \in F^+_\Pi\}$ is activated by modifying Π to obtain $\Pi \cup \mathrm{ic}(f)$ with $\mathrm{ic}(f) = \{\leftarrow \neg a\}$, if $f = a$, and otherwise $\mathrm{ic}(f) = \{\leftarrow a\}$. We say $S \subseteq \mathcal{AS}(\Pi)$ satisfies f denoted by $S \models f$, whenever $S = \{s \in \mathcal{AS}(\Pi) \mid a \in s\}$, if $f = a$, and $S = \{s \in \mathcal{AS}(\Pi) \mid a \notin s\}$, if $f = \neg a$. A *route* $\delta = f_1 \wedge \cdots \wedge f_n$ is a finite sequence of facets separated by $\wedge$, which denotes n arbitrary navigation steps over Π by means of $\Pi^\delta \coloneqq \Pi \cup ic(f_1) \cup \cdots \cup ic(f_n)$. Δ^Π denotes all possible routes over Π and ϵ denotes the empty route, meaning, $\Pi^\epsilon = \Pi$. We say $S \subseteq \mathcal{AS}(\Pi)$ satisfies δ, if S satisfies each $f_i \in \delta$, denoted by $S \models \delta$. For more details we refer to [13].

3 Diversity and Soundness

In this section we will first of all tackle the problems of soundness (S) and diversity (D) separately, and then go on to present an approach for collecting something of everything (S + D). Next, we formally define notions to capture what we understand as diversity and soundness of answer sets.

Definition 1 *For a set of* target atoms $T \subseteq at(\Pi)$ *and its complement* $\overline{T} \coloneqq at(\Pi) \setminus T$*, we call collection* $S \subseteq \mathcal{AS}(\Pi)$ *(i) a* packing *over* T*, whenever for any pair* $s, s' \in S$ *we find that* $s \bigcap s' \subseteq \overline{T}$*; (ii)* sound *over* T*, whenever* $T \subseteq \bigcup S$*; and (iii)* perfect *over* T*, whenever* S *is a sound packing over* T*.*

The reason we take target atoms into consideration is that in some applications, such as for instance smoke testing, we may only be interested in having representative answer sets with regard to specific components of our problem, e.g., functionality that is affected by changes in a code base.

Example 3 *Consider program* Π_0 :

▲ | ◆ | ▼ ■ | ● ← ▲ ■ | ✖ ← ◆
✚ ← ▼ ● | ✖ ← ✚

which admits the following answer sets:

$s_1 = \{■, ▲\}$ $s_2 = \{●, ▲\}$ $s_3 = \{◆, ✖\}$
$s_4 = \{■, ◆\}$ $s_5 = \{✖, ▼, ✚\}$ $s_6 = \{●, ▼, ✚\}$.

Set $T = at(\Pi_0)$*. For instance, while* $S = \{s_1, s_2, s_4, s_5\}$ *is sound over* T*, it is not a packing over* T *as atoms* ▲ *and* ■ *appear twice. However,* S *is perfect over* $\{●, ◆, ✖, ▼, ✚\}$*, and by removing* s_1 *from* S*, we obtain a perfect collection over* T*.*

In fact, some solution spaces admit no perfect collection.

Example 4 *Consider Program* $\Pi_1 = \{■ \mid ◆; ∗ \mid ✖ \leftarrow ◆; ▲\}$ *which has* 3 *answer sets* $\{\{■, ▲\}, \{◆, ✖, ▲\}, \{◆, ∗, ▲\}\}$*. If we choose to collect over all atoms that induce a facet* $\{■, ◆, ∗, ✖\}$*, we see that the answer sets of* Π_1 *admit no perfect collection, as* ∗ *and* ✖ *both require* ◆*.*

Luckily, regardless of whether a perfect collection over target atoms exists, we can, in fact, reliably find packings and sound collections for any program that admits them, respectively, via a greedy approach, we would like to call "reap and sow". Essentially we repeatedly reap (collect) one seed solution s from a sub-space and then take a route (sow) towards a sub-space where every solution has nothing but a specified set of irrelevant atoms in common, as specified in the subroutine given in Algorithm 1.

Algorithm 1 Reaping and sowing.

Procedure: `reap_and_sow`
In: satisfiable program Π; target atoms $T \subseteq at(\Pi)$
Out: an answer set of Π and a route over Π

1: $\texttt{seed} \leftarrow$ **first found** $s \in \mathcal{AS}(\Pi)$;
2: **return** $(\texttt{seed}, \bigwedge_{a \in \texttt{seed} \setminus \overline{T}} \neg a)$;

Example 5 (Example 4 cont'd) *Let us reap and sow (again) targeting atoms* $\{\blacksquare, \blacklozenge, \ast, \maltese\}$*, until we have exhausted possible seed solutions. Suppose we first reap seed solution* $\{\blacksquare, \blacktriangle\} \in \mathcal{AS}(\Pi_1)$*. Next, we reap another solution from the sub-space* $\{\{\blacklozenge, \ast, \blacktriangle\}, \{\blacklozenge, \maltese, \blacktriangle\}\}$ *under route* $\neg\blacksquare$*, say,* $\{\blacklozenge, \ast, \blacktriangle\}$*. Since* $\neg\blacksquare \wedge \neg\blacklozenge \wedge \neg\ast$ *admits no answer set, we are done.*

Diverse Answer Sets. Next, we introduce a reap and sow variation, called D-Greedy search, as given in Algorithm 2, which ignores all non-target atoms, ultimately producing packings.

Algorithm 2 D-Greedy search.

In: program Π; target atoms $T \subseteq at(\Pi)$
Procedure: `d_greedy`
Out: collection of answer sets

1: $S \leftarrow \emptyset$; $\delta \leftarrow \epsilon$;
2: **while** $\mathcal{AS}(\Pi^\delta) \neq \emptyset$ **do**
3: $\quad (s, \delta_s) \leftarrow \texttt{reap\_and\_sow}(\Pi^\delta, T)$;
4: $\quad S \leftarrow S \cup \{s\}$; $\delta \leftarrow \delta \wedge \delta_s$;
5: **return** S;

In fact, as shown next, via D-Greedy we can always find a packing over answer sets of a program admitting at least one packing.

Observation 1 *If the input program* Π *admits packings* $\{S_1, \ldots, S_n\}$ *over target atoms* T*, then* D*-Greedy search returns* $S \in \{S_1, \ldots, S_n\}$.

Proof Observing Algorithm 2, we always reap from a sub-space $\mathcal{AS}(\Pi^\delta)$ where $\delta = \neg a_1 \wedge \cdots \wedge \neg a_n$ s.t. $a_i \in \bigcup S \setminus \overline{T}$ for $i = 1 \ldots n$, which means that, if $\mathcal{AS}(\Pi^\delta) \neq \emptyset$, any answer set $s \in \mathcal{AS}(\Pi^\delta)$ added to our final collection S will satisfy $\bigcup S \cap s \subseteq \overline{T}$, and otherwise $S = \emptyset$. □

Sound Answer Sets. To obtain a sound collection we can reap and sow solutions until we have covered the entire scope of facets of a solution space, as described in Algorithm 3.

Algorithm 3 S-Greedy search.

In: program Π; target atoms $T \subseteq at(\Pi)$
Out: collection of answer sets

1: $S \leftarrow \emptyset$;
2: **while** $T \neq \emptyset$ **do**
3: $\quad$ **guess** $a \in T$;
4: $\quad$ **if** $\mathcal{AS}(\Pi^a) \neq \emptyset$ **then**
5: $\quad\quad s \leftarrow$ **first solution found in** $\mathcal{AS}(\Pi^a)$;
6: $\quad\quad S \leftarrow S \cup \{s\}$; $T \leftarrow T \setminus s$;
7: $\quad$ **else**
8: $\quad\quad$ **return** S;
9: **return** S;

Observation 2 *If the input program* Π *admits sound collections* $\{S_1, \ldots, S_n\}$ *regarding target atoms* T*, then* S*-Greedy search returns* $S \in \{S_1, \ldots, S_n\}$.

Proof Suppose Π admits collections $\{S_1, \ldots, S_n\}$ that are sound regarding T. In Algorithm 3, we can observe that S-Greedy search collects answer sets S until each target atom $t \in T$ has occurred at least once, so that $S \in \{S_1, \ldots, S_n\}$. □

4 Representing Something of Everything

While S-Greedy search will provide us with a sound collection, diversity among the resulting solutions is not guaranteed. Conversely, D-Greedy search will not guarantee soundness. Typically, soundness will be at the expense of rather similar answer sets, and, vice versa, pairwise disjunct answer sets will rarely be sound out of the box. In an ideal setting, we can detect a perfect collection over target atoms specified by the user, using Algorithm X [23] to detect an exact cover on the incidence matrix of all the answer sets of a program. However, as already mentioned, in practice, most of the time there are simply too many answer sets to enumerate and exact approaches may not scale for real world problems. Moreover, whenever there exists no perfect collection, we would have to settle for "less" anyways.

As a matter of fact, we are not interested in solving a decision problem, but rather aim at giving a concise and neutral representation of the solution space with efficient methods. *Representativeness*, as we put it, is meant to convey an idea of how target atoms mesh across answer sets. The set of all answer sets of a program is a collection of all possible combinations of parts of a problem that form a solution to the problem. This, indeed, represents all the ways atoms mesh. However, (1) due to its sheer size, potentially interesting insights will be hard to comprehend; (2) when focusing on specific target atoms, looking into all answer sets causes noise, which might not very well represent how certain atoms in question mesh; and (3) most importantly, regardless of what we are focusing on, in general, smaller collections might just be as representative as a collection of all answer sets. Now, how can we express this conception of representativeness?

4.1 How to Express Representativeness?

We aim at a *well-mixed* aggregation of *all* target atoms by means of a collection of answer sets. Soundness relates to covering all atoms and diversity relates to well-mixedness. There is not so much to the concept of soundness as we present it: we need to see each target atom at least once. Diversity, however, is a bit more complicated, as there is a broad spectrum on the meaning of diversity of objects. In terms of how close a collection is to a perfect collection (exact cover), diversity can be seen as a measure quantifying over the number of pairwise disjunct answer sets; indirectly this relates to the number of uniquely appearing atoms. In a perfect collection each target atom appears once and only once, so there is no bias towards any atom or thereby induced facet or sub-space of answer sets, which indicates well-mixedness in terms of the atoms at hand: it is equally surprising for any atom to show up in an answer set. In terms of entropy, a perfect collection is well-mixed, because the expected value of self-information of an atom within the collection, is maximized, but so is any other collection that induces a uniform distribution on target atoms showing up in answer sets. Therefore, we propose entropy as a notion of diversity. In the following, we pursue this idea along the lines of [26].

Let $S|_T^m$ be a finite multiset denoted by $\{a_1^{m_1}, \ldots, a_n^{m_n}\}$ where m_i corresponds to the frequency of atom $a_i \in T = \{a_1, \ldots, a_n\} \subseteq at(\Pi)$ within a collection $S \subseteq \mathcal{AS}(\Pi)$ of answer sets of program Π. Let $\mathbf{T}|_S$ be a discrete random variable that takes values in T and is distributed according to $p_S(a_i) = \frac{m_i}{\sum_{j=1}^{n} m_j} \in [0,1]$ with $\sum_{i=1}^{n} p_S(a_i) = 1$. In short, distribution p_S maps target atom a to its relative frequency of showing up within answer sets belonging to collection S. p_S is *uniform*, if $p_S(a_i) = \frac{1}{n}$ for $i = 1 \ldots n$. The *Shannon entropy* of $\mathbf{T}|_S$ is $\mathrm{H}[\mathbf{T}|_S] \coloneqq \sum_{a \in T} p_S(a) \log_2 \frac{1}{p_S(a)}$ where $\log_2 \frac{1}{p_S(a)}$ can be regarded as the surprise at observing a within S, i.e., the information gained by observing a is being described. Accordingly, entropy is a measure for the average amount of information gained per observation. The more p_S concentrates on a single element, the less the entropy of p_S. Accordingly the entropy peaks, if p_S is uniform and results in $\mathrm{H}[\mathbf{T}|_S] = |T| \frac{1}{|T|} \log_2 \frac{1}{1/|T|} = \log_2 |T|$. We define $\log_2 \frac{1}{0} = 0$ and $\log_2 0 = \infty$.

Example 6 (Example 3 cont'd) *Consider $S' = \mathcal{AS}(\Pi_0)$, giving $S'|_T^m = \{\blacklozenge^2, \blacksquare^2, \bullet^2, \blacktriangle^2, \maltese^2, \clubsuit^2, \blacktriangledown^2\}$ and therefore yielding a uniform distribution p_S with $p_S(a) = \frac{1}{7}$ as each atom $a \in T$ occurs twice.*

Formally we define *diversity* by means of entropy as the function $\mathrm{D}(\mathbf{T}|_S) \coloneqq 2^{\mathrm{H}[\mathbf{T}|_S]}$, which is also known as the *perplexity* of distribution p_S [22] or the order-1 diversity of atoms T within $S|_T^m$ [26]. In a nutshell, the diversity of a collection S with respect to target atoms T is expressed by the average rarity (self-information) of an atom $a \in T$ belonging to an answer set in S. So, essentially high diversity stands for a well-mixed collection of answer sets.

Example 7 (Example 6 cont'd) *We can observe that $\mathrm{D}(\mathbf{T}|_{S \setminus \{s_1\}}) = \mathrm{D}(\mathbf{T}|_{S'}) = 7$. We can also consider how well-mixed a collection is w.r.t. specified ignored atoms, e.g., $\mathrm{D}(\mathbf{T}'|_S) = 5$ for $T' = \{\blacklozenge, \bullet, \maltese, \clubsuit, \blacktriangledown\}$, whereas $\mathrm{D}(\mathbf{T}|_S) \approx 6.61$.*

We are now in position to express *representativeness* of a collection S over a set of non-empty[1] target atoms T by means of

$$\mathrm{R}(\mathbf{T}|_S) \coloneqq \frac{\mathrm{D}(\mathbf{T}|_S)}{|T|}.$$

We establish the following observation to motivate $\mathrm{R}(\cdot)$.

Observation 3 *For any program Π, collection $S \subseteq \mathcal{AS}(\Pi)$ and target atoms $T \subseteq at(\Pi)$*

(a) $\mathrm{R}(\mathbf{T}|_S) = 1$ *iff p_S over $S|_T^m$ is uniform; and*
(b) $\mathrm{R}(\mathbf{T}|_S) = 2^{\mathrm{H}[\mathbf{T}|_S] - \log_2 |T|} \in [0,1]$.

Proof (a) follows from the fact that $2^{\mathrm{H}[\mathbf{T}|_S]}$ is an effective number, meaning, $2^{\mathrm{H}[\mathbf{T}|_S]} = |T|$ for any $|T| \in \mathbb{N}$ with uniform distribution p_S over T [26]. (b) can be derived as follows:

$\mathrm{R}(\mathbf{T}|_S) = {}^{\mathrm{D}(\mathbf{T}|_S)}/_{|T|} = {}^{2^{\mathrm{H}[\mathbf{T}|_S]}}/_{2^{\log_2 |T|}} = 2^{\mathrm{H}[\mathbf{T}|_S] - \log_2 |T|}$ □

What Observation 3 is meant to convey is that representativeness is effectively describing the similarity between the actual distribution p_S over target atoms T within S and any hypothetical uniform distribution of $|T|$ objects by means of the base-2 exponential of the observed error $\mathrm{H}[\mathbf{T}|_S] - \log_2 |T|$. For uniform distributions the error equals $\log_2 |T| - \log_2 |T| = 0$, hence the representativeness $\mathrm{R}(\mathbf{T}|_S)$ peaks at 1.

[1] From now on we implicitly assume non-empty target atoms.

Example 8 (Example 7 cont'd) $\mathrm{R}(\mathbf{T}'|_S) = 1$ *tells that S is representative regarding T', as it induces a uniform distribution on target atoms T'. But in general, when interested in all atoms as specified by T, we have $\mathrm{R}(\mathbf{T}|_S) \approx 0.94$, because $T \setminus T' = \{\blacktriangle, \blacksquare\}$ forms the answer set $s_1 \in S$, whose atoms could be placed in answer sets s_2, s_4 respectively to balance the distribution.*

So, essentially when it comes to representativeness we seek for a collection that induces a distribution over target atoms that is as close as possible to a uniform distribution. Thus, in order to maximize representativeness, we need to maximize entropy.

4.2 Increasing Representativeness

Our approach for increasing representativeness revolves around a heuristic that depends on two advances in (faceted) answer set navigation, which we discuss next.

Filtering Relevant Information. When collecting representative answer sets regarding target atoms T, we are only interested in a certain portion of the solution space, namely those answer sets, which contain at least some relevant information $a \in T$. Until now, the way (faceted) answer set navigation is conceived, routes describe conjunctions of literals (facets). That way the corresponding sub-spaces emerge from respective intersections over the set of all answer sets. While this is useful to sharply bundle answer sets $S \subseteq \mathcal{AS}(\Pi)$ around an atom in terms of $a \in \bigcap S$ iff $S \models a$ or $a \notin \bigcup S$ iff $S \models \neg a$, we lack expressiveness in terms of grouping answer sets by means of unions, in other words, a disjunction of facets. Accordingly, we introduce a new navigation step operation $\vee$ with lower precedence than $\wedge$, which gives $\delta ::= \epsilon \mid f \in F_\Pi \mid \delta \wedge \delta \mid \delta \vee \delta$. For the sake of an intuition of how we filter answer sets, think of $\wedge$ and $\vee$ in terms of classical conjunction and disjunction. We say S satisfies $a_1 \vee \cdots \vee a_k \vee \neg a_{k+1} \vee \cdots \vee \neg a_n \wedge \delta$, if S satisfies δ, and every $s \in S$ satisfies $s \cap \{a_1, \ldots, a_k\} \neq \emptyset$ or $\{a_{k+1}, \ldots, a_n\} \setminus s \neq \emptyset$.

To satisfy a route δ that contains a disjunction over atoms it is sufficient to look into the part of the solution space where the route δ has been activated, i.e. we only need to look into $\mathcal{AS}(\Pi^\delta)$.

Theorem 1 *Let $\delta = a_1 \vee \cdots \vee a_k \vee \neg a_{k+1} \vee \cdots \vee \neg a_n \wedge \delta_\wedge$ be a route over program Π where $\delta_\wedge \in \Delta^\Pi$ and $a_i \in at(\Pi)$ for $1 \leq i \leq n$, and let $\Pi^\delta \coloneqq \Pi^{\delta_\wedge} \cup \{\leftarrow \neg a_1, \ldots, \neg a_k, a_{k+1}, \ldots, a_n\}$. The route δ can be satisfied within $\mathcal{AS}(\Pi)$ iff $\mathcal{AS}(\Pi^\delta)$ satisfies δ.*

Proof Assume $\delta = \delta_\vee \wedge \delta_\wedge$ where

$$\delta_\vee = a_1 \vee \cdots \vee a_k \vee \neg a_{k+1} \vee \cdots \vee \neg a_n$$
$$\delta_\wedge = a_{n+1} \wedge \ldots a_m \wedge \neg a_{m+1} \wedge \cdots \wedge \neg a_\ell.$$

($\Rightarrow$): As δ is satisfied within $\mathcal{AS}(\Pi)$, it is known that $\mathcal{AS}(\Pi^{\delta_\wedge}) \subseteq \mathcal{AS}(\Pi)$ with $\{a_{n+1}, \ldots, a_m\} \subseteq \bigcap \mathcal{AS}(\Pi^{\delta_\wedge})$ and $\{a_{m+1}, \ldots, a_\ell\} \cap \bigcup \mathcal{AS}(\Pi^{\delta_\wedge}) = \emptyset$. Suppose $\mathcal{AS}(\Pi^{\delta_\wedge}) \neq \emptyset$. Adding $\leftarrow \neg a_1, \ldots, \neg a_k, a_{k+1}, \ldots, a_n$ to $\Pi^{\delta_\wedge}$, enforces that it cannot be that all of $\{a_1, \ldots, a_k\}$ are false and all of $\{a_{k+1}, \ldots, a_n\}$ are true. In other words, each answer set $s \in \mathcal{AS}(\Pi^\delta)$ contains at least one atom in $\{a_1, \ldots, a_k\}$ or omits at least one atom in $\{a_{k+1}, \ldots, a_n\}$. Thus, every $s \in \mathcal{AS}(\Pi^\delta)$ satisfies $s \cap \{a_1, \ldots, a_k\} \neq \emptyset$ or $\{a_{k+1}, \ldots, a_n\} \setminus s \neq \emptyset$, so $\mathcal{AS}(\Pi^\delta)$ satisfies δ.

($\Leftarrow$): We have that $\mathcal{AS}(\Pi^\delta) \subseteq \mathcal{AS}(\Pi)$, so it is rather easy to see that each answer set $s \in \mathcal{AS}(\Pi^\delta)$ that satisfies δ is also contained in $\mathcal{AS}(\Pi)$, and thus δ is satisfied in $\mathcal{AS}(\Pi)$ as well. □

Answer sets that do not contain any target atoms cannot improve the representativeness of a collection. Thus, in order to collect representative answer sets with respect to a set of target atoms T, we can ignore all answer sets that do not contain any target atoms. This means that we only need to search within the part of the solution space that fulfills the or-constraint over the target atoms.

Proposition 1 *For any $T \subseteq at(\Pi)$ with $T = \{t_1, \ldots, t_n\}$, let $\delta^t = t_1 \vee \cdots \vee t_n$ and any $S \subseteq \mathcal{AS}(\Pi)$. If $S' \in \operatorname{argmax}_{S'' \subseteq \mathcal{AS}(\Pi^{\delta^t})} \mathrm{R}(\mathbf{T}|_{S''})$, then $\mathrm{R}(\mathbf{T}|_S) \leq \mathrm{R}(\mathbf{T}|_{S'})$.*

Proof Suppose S be any collection of answer sets within $\mathcal{AS}(\Pi)$ and $S' \in \operatorname{argmax}_{S'' \subseteq \mathcal{AS}(\Pi^{\delta^t})} \mathrm{R}(\mathbf{T}|_{S''})$. We can make the following distinction of cases.

1. Assume $S \subseteq \mathcal{AS}(\Pi^{\delta^t})$. Then, if S has maximal representativeness, it follows that $\mathrm{R}(\mathbf{T}|_S) = \mathrm{R}(\mathbf{T}|_{S'})$, and otherwise $\mathrm{R}(\mathbf{T}|_S) < \mathrm{R}(\mathbf{T}|_{S'})$.
2. Assume $S \not\subseteq \mathcal{AS}(\Pi^{\delta^t})$.
   (a) If $S \cap \mathcal{AS}(\Pi^{\delta^t}) = \emptyset$, then $\mathrm{R}(\mathbf{T}|_S) = 0 \leq \mathrm{R}(\mathbf{T}|_{S'})$.
   (b) If $S \cap \mathcal{AS}(\Pi^{\delta^t}) \neq \emptyset$, then $\mathrm{R}(\mathbf{T}|_{S'}) = \mathrm{R}(\mathbf{T}|_{S' \cup \{s\}})$ for any $s \in S \setminus \mathcal{AS}(\Pi^{\delta^t})$, therefore $\mathrm{R}(\mathbf{T}|_S) \leq \mathrm{R}(\mathbf{T}|_{S'})$. □

Quantifying Uncertainty. The concept of *weighting* facets [13] can express several quantities, such as the answer set count or the facet count.

Essentially, the answer set count can be used to derive (joint or conditional) probabilities of events as specified by facets. Next, we suggest how to derive meaning from *counting facets*. Increasing representativeness is about increasing entropy, a quantity that is, among other interpretations, being understood as a measure of *uncertainty*. With this in mind, we can think of the concept of a facet as a source of uncertainty in the sense that a facet is an atom whose truth value is not fixed to true or false within a set of answer sets, but rather *unknown* or simply not yet *certain* based on given information. In turn, activating a facet causes a certain amount of uncertainty reduction or information gain, as previously undetermined truth values of atoms become certain. Here we are interested in relating facet counts to sets of target atoms. Thus, we define the *facet-counting weight* of a facet $f \in F_\Pi$ regarding target atoms $T \subseteq at(\Pi)$ and a set of answer sets $S \subseteq \mathcal{AS}(\Pi)$ by $\omega|_S(f, T) := 1 - \frac{|F^T_{Sf}|}{|F^T_S|}$ where $F^T_S := \bigcup S \setminus (\bigcap S \cup \overline{T})$ are the inclusive facets among T within S and $F^T_{Sf} := \bigcup S^f \setminus (\bigcap S^f \cup \overline{T})$ with $S^f \models f$ are the inclusive facets among T within the subset of S that satisfies f.

Example 9 *Consider Example 3 and let $T = \{\blacksquare, \maltese\}$. For example, uncertainty regarding T can be reduced by 50% with ✚, as the truth value of ✖ is unknown and ■ is false. ◆ and ●, for instance, both erase uncertainty, as $\omega|_{\mathcal{AS}(\Pi_0)}(\blacklozenge, T) = \omega|_{\mathcal{AS}(\Pi_0)}(\bullet, T) = 1$; however, for different reasons. While ◆ entails T, so that ■ and ✖ are true, no target atoms occur together with ● in $\mathcal{AS}(\Pi_0)$, meaning, ■ and ✖ are false.*

We assume that reducing uncertainty by counting facets is a useful tool to assist the search for representative answer sets. In the next section, we propose heuristics that incorporate relevant information filtering and uncertainty reduction.

4.3 Heuristics for Collecting Representative Answer Sets

We consider several heuristics to efficiently collect highly representative answer sets relative to a specified set of target atoms.

S-Greedy-based Heuristics. As a naive approach, we consider S-Greedy search as a baseline strategy to find representative answer sets, as the idea is straightforward: collect all target atoms across answer sets, if possible. We can observe that this approach, in theory, issues at most $|T|$ calls to a consistency check oracle, which returns an answer set, whenever there is one. It is well-known that the consistency check for disjunctive programs is a Σ^{P}_2-complete problem [11]. To filter relevant information right away, we additionally consider the *S-Greedy-sieve* heuristic, which is S-Greedy search on route $\bigvee_{i=1} t_i$ over the input program; thus constraining the search space to answer sets that contain at least one target atom among $\{t_1, \ldots, t_n\}$ each.

D-Greedy-based Heuristics. D-Greedy-based heuristic are more involved as they revolve around reducing uncertainty, which relies on facet counting. Weighting facets by means of counting facets of disjunctive programs is in Δ^{P}_3 [13]. Naturally, the following approaches are computationally more challenging than S-Greedy-based methods, as, in theory, they require calls to a Δ^{P}_3-oracle. In particular,

Algorithm 4 Filter facet with maximal information gain.

Procedure: `max`
In: program Π; route δ; atoms to filter from $A \neq \emptyset$; target atoms T;
Out: filtered facet $f \in A$

```
1: min ← |T|; f ← guess any atom from A;
2: if |A| > 1
3:   for f' ∈ A do
4:     uncertainty ← |T ∩ F_{Π^{δ∧f'}}|;
5:     if uncertainty ≤ min
6:       if uncertainty = 0
7:         return f';
8:       min ← uncertainty; f ← f';
9: return f;
```

filtering procedures to determine facets that reduce uncertainty the most are required. The `max`-filter as given by Algorithm 4 identifies the facet (-inducing atom) among atoms A that reduces uncertainty regarding target atoms the most. At most $|A|$ oracle-calls are required, as, due to Line 6, we terminate earlier whenever a facet admits no uncertainty. We consider another filter, namely, the $\texttt{max}_+$-filter, which in addition to the remaining facets, also counts the number of target atoms that are present by activating a facet. The idea is to find the facet, which omits the least number of target atoms in resulting answer sets. The $\texttt{max}_+$-filter is realised by changing line 4 in Algorithm 4 to `uncertainty` $\leftarrow |T \setminus (F_{\Pi^{\delta \wedge f'}} \cup \bigcap \mathcal{AS}(\Pi^{\delta \wedge f'}))|$;. In theory, this adds no complexity, as we get $\bigcap \mathcal{AS}(\Pi^{\delta \wedge f'})$ for free, when computing $F_{\Pi^{\delta \wedge f'}}$. Algorithm 5 describes two D-Greedy-based heuristics, each of which use the `max`-filter (D-Greedy-`max`) and $\texttt{max}_+$-filter (D-Greedy-$\texttt{max}_+$), respectively. D-Greedy-`max` and D-Greedy-$\texttt{max}_+$ solely differ in their choice of filters. The idea is to identify a facet, based on which to perform D-Greedy search, using one of the aforementioned filters. This requires at most $|T|^2$ oracle calls. We take two more heuristics into consideration in our experiments, that is, D-Greedy-`max` and D-Greedy-$\texttt{max}_+$, but with Line 4 changed to $f \leftarrow$ `filter`$(\Pi, \delta, F^+_\Pi, T)$;, which means that we will

Algorithm 5 Heuristic using D-Greedy and filter.

In: program Π; target atoms $T \subseteq at(\Pi)$; $\texttt{filter} \in \{\texttt{max}, \texttt{max}_+\}$
Out: collection $S \subseteq \mathcal{AS}(\Pi)$

```
1: S ← ∅; δ ← ⋁_{t∈T} t;
2: while T ≠ ∅ and AS(Π^δ) ≠ ∅ do
3:     if |T| > 1 then
4:         f ← filter(Π, δ, T, T);
5:         S' ← d_greedy(Π^{δ∧f}, T);
6:         S ← S ∪ S'; T ← T \ ⋃S'; δ ← ⋁_{t∈T} t;
7:     else
8:         s ← first found in AS(Π^t) where t ∈ T;
9:         S ← S ∪ {s};
10: return S;
```

look for suitable facets among all facet inducing atoms of the input program, instead of constraining the filter to target atoms. We call them D-Greedy-all-max and D-Greedy-all-$\max_+$, respectively. This then requires at most $|T| \cdot |F^+_\Pi|$ oracle calls. However, what could cause larger runtimes in practice, is that the number of atoms to go through in the filters is fixed to $|F^+_\Pi|$ and not decreasing with $|T|$.

To summarize, we propose the following 6 Greedy (abbreviated by G) heuristics for collecting representative answer sets, all of which will be evaluated empirically in the next section:

- S-G (Algorithm 3)
- S-G-sieve (Algorithm 3 on constrained search space)
- D-G-max (Algorithm 5 with max-filter)
- D-G-$\max_+$: (Algorithm 5 with $\max_+$-filter)
- D-G-all-max: (Algorithm 5 with max-filter on all facets)
- D-G-all-$\max_+$: (Algorithm 5 with $\max_+$-filter on all facets)

5 Experiments

To demonstrate scalability of the proposed approaches on real-world problems, while providing acceptable results, we generated benchmarks [6] in the realm of smoke testing (scenario S1) and claim-augmented argumentation (scenario S2), respectively, and conducted experiments on them using an implementation of our approach, called `soe`.

Scenario S1. We envision a scenario in which certain parts of a code base have changed and thus require smoke testing to ensure that the made changes did not break anything. In particular, as is often the case in practice, a subset of tests is running in parallel, while other tests are subject to constraints that prohibit parallel execution. To realize this, we used 8 popular open source projects written in the Rust programming language and encoded test and module relations within the respective code base by $\texttt{test}(i, j)$ where $i, j \in \mathbb{N}$ stand for the module id i and the test id j. We further added rules to choose at least one module and one test k within the module to run, as expressed by $\texttt{run\_test}(k)$. We added constraints to either always or never run certain tests or modules in parallel, and included from 10% to 30% of all tests and modules in such constraints at random, respectively. This resulted in 25 different logic programs per code base (Rust-project). Finally, we generated 10 files per Rust-project that contain from 10% to 100% of possible target atoms $\texttt{run\_test}(\ell)$, which express that features tested in test ℓ have changed, thus being subject to smoke tests. Additionally, 10% of modules to run were added to the target atoms. Representative answer sets then correspond to test plans that represent several ways of running tests that include features that have recently changed. In total we generated 2000 pairs of ASP instances and target atoms.

Scenario S2. S2 consists of 195 argumentation framework (AF) instances based on benchmarks A and B from ICCMA 2017 [14] together with the ASP encodings for stable argumentation semantics [9], where each argument is associated to a claim, as common in claim-augmented argumentation frameworks CAFs [8] under the inherited semantics. We envision a scenario in which a user wants to get an impression of how arguments assigned to chosen target claims mesh. From the ICCMA instances, we only included instances which were able to produce at least 10 answer sets within 10 minutes of solving time. The number of claims is set to 20% of the number of arguments. Claims are assigned at random, but each claim belongs to at least two arguments. Target claims were chosen out of facets at random by a 50% chance, for each instance 10 target atom mappings were generated, resulting in 1950 AF/ target claim pairs. Representative answer sets in this scenario then correspond to those stable extensions (sets of arguments) that are representative with respect to the target claims.

Setup & Design. Determining representative answer sets should be feasible on desktop systems, enabling users, or test frameworks that use our approach in their back-end, to practicably explore, or determine, how atoms mesh, instead of enumerating all or only or a certain specified number of answer sets, which is typically rather time consuming. As a consequence, runtime was limited to 300 seconds and the experiments were run on a single core AMD EPYC 7513, 2.6 GHz with 16 GB of RAM inside a VM (Debian 11, rust 1.68). We ran our prototypical implementation of the 6 proposed heuristics on each instance (logic program and input target atoms) of S1 and S2, respectively, Runtime was measured using `perf 5.10`.

Expectations. Based on the conception of the heuristics under evaluation, we expect the following:

(E1) The S-G-based approaches will be significantly faster than the D-G-based approaches;
(E2) in contrast to other methods, methods using the $\max_+$-filter will produce a small number of rather big answer sets;
(E3) the additional measures imposed in D-G-all-* heuristics (which in theory increase runtime) will not improve R, as reducing uncertainty by filtering from target atoms is sufficient; and
(E4) the smaller the number of target atoms, the easier the problem, in terms of observable low run times and high R-values.

Observations & Results. Table 1 affirms expectation (E1) for S1, and even more so time consumption reflects our considerations regarding the complexity of the heuristics. The only exception is S-G-sieve, terminating faster than S-G, which is reasonable, as adding the or-constraint to filter relevant information in S-G-sieve causes almost no overhead while reducing the search space. The timing pattern is also reflected in the relative number of solved instances, as faster heuristics solve more instances.

The mean R-value varies between the heuristics, indicating different representativeness of their output. Obtaining a general ranking regarding R is rather difficult. The reason is that we are missing out on potential R-values of timed-out data points. For S1, the Rust-project-based ranking in Table 2 of the R-value reveals that (O1) S-G-sieve mostly outperforms S-G regarding R; and (O2) D-G-max (resp. D-G-$\max_+$) outperforms D-G-all-max (resp. D-G-all-$\max_+$). Also for

	heuristic	rank	mean R	mean time [s]	solved
S1	S-G	5	0.713	< 0.01	1.00
	S-G-sieve	1	**0.826**	$<$ **0.01**	1.00
	D-G-max	3	0.772	0.21	0.80
	D-G-max$_+$	2	0.804	0.21	0.80
	D-G-all-max	6	0.555	0.55	0.48
	D-G-all-max$_+$	4	0.760	0.61	0.44
S2	S-G	5	0.967	1.88	1.00
	S-G-sieve	6	0.958	**1.58**	1.00
	D-G-max	3	0.972	11.00	0.97
	D-G-max$_+$	4	0.969	11.07	0.90
	D-G-all-max	2	0.976	46.31	0.86
	D-G-all-max$_+$	1	**0.979**	54.23	0.83

Table 1: Results regarding scenario S1 and S2, rank regarding R, solved is the fraction of instances which terminated within timeout. Heuristics are arranged from lowest to highest complexity.

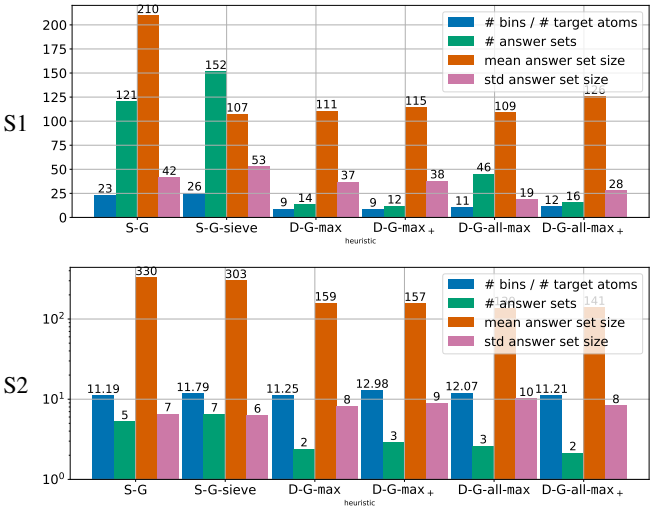

Figure 1: Statistics of output in scenario S1 and S2, including # bins / # target atoms, which stands for the number of different frequencies of target atoms across the output divided by the number of target atoms.

the two extensive Rust-projects G and H with a higher number of potential target atoms, the more sophisticated heuristics were not able to produce any output within the timeout.

Addressing collection properties, Figure 1 (S1) reveals that (O3) S-G-based approaches produce the largest collections; (O4) D-G-based approaches using the max$_+$-filter produce small collections,

	A	B	C	D	E	F	G	H
# tests	52	251	227	204	406	282	1141	1932
# modules	16	61	94	159	4	159	840	193
S-G	5	5	4	4	1	2	1	1
S-G-sieve	1	1	3	1	2	1	1	1
D-G-max	3	3	2	2	4	1	1	-
D-G-max$_+$	2	2	1	2	3	1	1	-
D-G-all-max	4	6	4	3	5	3	-	-
D-G-all-max$_+$	2	4	3	2	4	2	-	-

Table 2: Ranking heuristics based on mean R for the 8 Rust-projects (A to H) in S1. - means 100% timeout. Heuristics share a rank if their data's p-value passed 0.05 in a standard T-Test for independent samples.

affirming (E2) ; and (O5) D-G-based approaches using the max-filter have a tendency to produce outputs where most target atoms will occur with the same frequency. Regarding (E4), conducting linear regression analysis on the number of target atoms against time and R indicates a consistent tendency for the time to increase and R to decline for rising numbers of target atoms.

For scenario S2 the data shows a rather homogeneous picture as seen in Table 1 (S2) and Figure 1 (S2). Expectations (E1) and (E3) are supported, (E2) is partially supported and (E4) cannot be tested on S2 due to the structure of the instance pairs. The quality of time consumption compares to S1, but all R-values are nearly identical, due to the heterogeneous nature of the scenario: instead of 8 basic instances (Rust-projects), 195 basic instances were used. Therefore (O3) can be validated and (O4) is partially supported. Also the collection properties do not vary as much as in S1. Noticeable are the large answer sets, especially for the S-G-based heuristics.

Summary. Our experiments demonstrated that using more complex heuristics pays off, whenever the quality of the output matters in terms of small size and high representativeness. However, as a baseline approach, ignoring the output size, less complex methods like S-G-sieve are the number one choice, as they achieve comparably high representativeness in less than a second. We, thus, suggest that S-G-sieve is a good choice for generating highly representative test plans in a short amount of time, whereas D-G-based approaches are in general the preferred choice, when it comes to exploration scenarios, where the size of the collection matters.

6 Conclusion

We discussed the concept of diversity of solutions (answer sets), involving entropy as a reasonable measure that is independent of the number solutions, which finally evolved into a novel formal concept of representativeness of solutions that solely depends on the distribution of partial solutions (atoms). Further, we suggested several methods to produce representative collections of answer sets, ranging from naive to complex heuristics, of which some proved to be useful to quickly generate a highly representative collection of several test plans in a real-world smoke test planning scenario. Others appeared to be more useful in a more exploration-centric scenario, when the size of the output has to be small. We used novel suitable benchmarks to conduct experiments, one instance set of which stems from claim-augmented argumentation. The other instance set was retrieved from 8 popular open source projects using the Rust programming language.

Future Work. Taking the insights of this work into consideration, we assume that starting the search with S-G-based approaches and using more elaborate heuristics, like D-G-based approaches, towards the end of the search, when most of the target atoms have already been covered, could be promising. Further, developing efficient methods to translate representative collections into visual representations thereof, potentially using tools such as `clingraph` [19] or NEXAS [7], is interesting future work.

$\delta \leftarrow \bigwedge\{\neg a : a \in T \cap \bigcup S\}$

$\delta \leftarrow \bigwedge_{a \in T \cap \bigcup S} \neg a$

Acknowledgements

The authors are stated in alphabetic order. This work was supported by the Bundesministerium für Bildung und Forschung (BMBF) in project 01IS20056_NAVAS.

References

[1] Christian Alrabbaa, Sebastian Rudolph, and Lukas Schweizer, 'Faceted answer-set navigation', in *Proc. of the 2nd Int. Joint Conf. on Rules and Reasoning (RuleML+RR'18)*, eds., Christoph Benzmüller, Francesco Ricca, Xavier Parent, and Dumitru Roman, pp. 211–225. Springer, (2018).

[2] Mario Alviano and Carmine Dodaro, 'Anytime answer set optimization via unsatisfiable core shrinking', *Theory and Practice of Logic Programming*, **16**(5-6), 533–551, (2016).

[3] Mutsunori Banbara, Naoyuki Tamura, and Katsumi Inoue, 'Generating event-sequence test cases by answer set programming with the incidence matrix', in *TC of the 28th Int. Conf. on Logic Programming (ICLP 2012)*, eds., Agostino Dovier and Vítor Santos Costa, volume 17 of *LIPIcs*, pp. 86–97. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, (2012).

[4] Julien Baste, Michael R. Fellows, Lars Jaffke, Tomás Masarík, Mateus de Oliveira Oliveira, Geevarghese Philip, and Frances A. Rosamond, 'Diversity of solutions: An exploration through the lens of fixed-parameter tractability theory', *Artif. Intell.*, **303**, 103644, (2022).

[5] Elisa Böhl and Sarah Alice Gaggl, 'Tunas - fishing for diverse answer sets: A multi-shot trade up strategy', in *Proc. of the 16th Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR 2022)*, eds., Georg Gottlob, Daniela Inclezan, and Marco Maratea, volume 13416 of *LNCS*, pp. 89–102. Springer, (2022).

[6] Elisa Böhl, Sarah Alice Gaggl, and Dominik Rusovac. Representative Answer Sets: Collecting Something of Everything (Tool & Instances), 2023.

[7] Raimund Dachselt, Sarah Alice Gaggl, Markus Krötzsch, Julián Méndez, Dominik Rusovac, and Mei Yang, 'NEXAS: A visual tool for navigating and exploring argumentationsolution spaces', in *Proc. of the 9th Int. Conf. on Computational Models of Argument (COMMA 2022)*, ed., Francesca Toni, volume 220146 of *FAIA*, pp. 116–127. IOS Press, (2022).

[8] Wolfgang Dvorák and Stefan Woltran, 'Complexity of abstract argumentation under a claim-centric view', *Artif. Intell.*, **285**, 103290, (2020).

[9] Wolfgang Dvořák, Sarah Alice Gaggl, Anna Rapberger, Johannes Peter Wallner, and Stefan Woltran, 'The ASPARTIX system suite', in *Proc. of the 8th Int. Conf. on Computational Models of Argument (COMMA'20)*, eds., Henry Prakken, Stefano Bistarelli, Francesco Santini, and Carlo Taticchi, volume 326 of *FAIA*, pp. 461–462. IOS Press, (2020).

[10] Thomas Eiter, Esra Erdem, Halit Erdogan, and Michael Fink, 'Finding similar/diverse solutions in answer set programming', *Theory and Practice of Logic Programming*, **13**(3), 303–359, (2013).

[11] Thomas Eiter and Georg Gottlob, 'On the computational cost of disjunctive logic programming: Propositional case', *Annals of Mathematics and Artificial Intelligence*, **15**(3), 289–323, (1995).

[12] Esra Erdem, Katsumi Inoue, Johannes Oetsch, Jörg Pührer, Hans Tompits, and Cemal Yılmaz, 'Answer-set programming as a new approach to event-sequence testing', in *Proc. of the 3rd Int. Conf. on Advances in System Testing and Validation Lifecycle (VALID'11)*, pp. 25–34. IARIA, (2011).

[13] Johannes Klaus Fichte, Sarah Alice Gaggl, and Dominik Rusovac, 'Rushing and strolling among answer sets–navigation made easy', in *Proc. of the 36th AAAI Conference on Artificial Intelligence (AAAI 2022)*, pp. 5651–5659. AAAI Press, (2022).

[14] Sarah Alice Gaggl, Thomas Linsbichler, Marco Maratea, and Stefan Woltran, 'Design and results of the second international competition on computational models of argumentation', *Artificial Intelligence*, **279**, (February 2020).

[15] Martin Gebser, Roland Kaminiski, and Torsten Schaub, 'Complex optimization in answer set programming', *Theory and Practice of Logic Programming*, **11**(4-5), 821–839, (2011).

[16] Martin Gebser, Benjamin Kaufmann, and Torsten Schaub, 'Conflict-driven answer set solving: From theory to practice', *Artif. Intell.*, **187-188**, 52–89, (2012).

[17] Michael Gelfond and Vladimir Lifschitz, 'The stable model semantics for logic programming', in *Proc. of the 5th Int. Conf. and Symposium on Logic Programming (ICLP/SLP'88)*, eds., Robert A. Kowalski and Kenneth A. Bowen, volume 2, pp. 1070–1080. MIT Press, (1988).

[18] Michael Gelfond and Vladimir Lifschitz, 'Classical negation in logic programs and disjunctive databases', *New Generation Comput.*, **9**(3/4), 365–386, (1991).

[19] Susana Hahn, Orkunt Sabuncu, Torsten Schaub, and Tobias Stolzmann, 'Clingraph: ASP-based visualization', in *Proc. of the 16th Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR 2022)*, eds., Georg Gottlob, Daniela Inclezan, and Marco Maratea, volume 13416 of *LNCS*, pp. 401–414. Springer, (2022).

[20] Linnea Ingmar, Maria Garcia de la Banda, Peter J. Stuckey, and Guido Tack, 'Modelling diversity of solutions', in *Proc. of the 34th AAAI Conference on Artificial Intelligence (AAAI 2020)*, pp. 1528–1535. AAAI Press, (2020).

[21] Richard M Karp, 'Reducibility among combinatorial problems', in *Complexity of computer computations*, 85–103, Springer, (1972).

[22] Dietrich Klakow and Jochen Peters, 'Testing the correlation of word error rate and perplexity', *Speech Communication*, **38**(1-2), 19–28, (2002).

[23] Donald E Knuth, 'Dancing links', *arXiv preprint cs/0011047*, (2000).

[24] D Richard Kuhn, James M Higdon, James F Lawrence, Raghu N Kacker, and Yu Lei, 'Combinatorial methods for event sequence testing', in *Proc. of the 5th Int. Conf. on Software Testing, Verification and Validation (ICST 2012)*, eds., Giuliano Antoniol, Antonia Bertolino, and Yvan Labiche, pp. 601–609. IEEE Computer Society, (2012).

[25] D Richard Kuhn, Raghu N Kacker, Yu Lei, et al., 'Practical combinatorial testing', *NIST special Publication*, **800**(142), 142, (2010).

[26] Tom Leinster, *Entropy and diversity: the axiomatic approach*, Cambridge university press, 2021.

[27] Victor W. Marek and Mirosław Truszczyński, 'Stable models and an alternative logic programming paradigm', in *The Logic Programming Paradigm: a 25-Year Perspective*, Artificial Intelligence, 375–398, Springer, (1999).

[28] Timo Camillo Merkl, Reinhard Pichler, and Sebastian Skritek, 'Diversity of answers to conjunctive queries', in *Proc. of the 26th International Conference on Database Theory (ICDT 2023)*, eds., Floris Geerts and Brecht Vandevoort, volume 255 of *LIPIcs*, pp. 10:1–10:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, (2023).

[29] Seemran Mishra, 'Product configuration in answer set programming', *Electronic Proceedings in Theoretical Computer Science*, **345**, 296–304, (2021).

[30] Tobias Philipp, Valentin Roland, and Lukas Schweizer, 'Smoke test planning using answer set programming', *Int. J. Interact. Multim. Artif. Intell.*, **6**(5), 57–65, (2021).