Tunas - Fishing for Diverse Answer Sets: A Multi-Shot Trade up Strategy

Elisa Böhl[0000−0002−1237−163X] and Sarah Alice Gaggl[0000−0003−2425−6089]

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

Abstract. Answer set programming (ASP) solvers have advanced in recent years, with a variety of different specialisation and overall development. Thus, even more complex and detailed programs can be solved. A side effect of this development are growing solution spaces and the problem of how to find those answer sets one is interested in. One general approach is to give an overview in form of a small number of highly diverse answer sets. By choosing a favourite and repeating the process, the user is able to leap through the solution space. But finding highly diverse answer sets is computationally expensive. In this paper we introduce a new approach called *Tunas* for *Trade Up Navigation for Answer Sets* to find diverse answer sets by reworking existing solution collections. The core idea is to collect diverse answer sets one after another by iteratively solving and updating the program. Once no more answer sets can be added to the collection, the program is allowed to trade answer sets from the collection for different answer sets, as long as the collection grows and stays diverse. Elaboration of the approach is possible in three variations, which we implemented and compared to established methods in an empirical evaluation. The evaluation shows that the Tunas approach is competitive with existing methods, and that efficiency of the approach is highly connected to the underlying logic program.

Keywords: multi-shot answer set programming · navigation · diverse answer sets

1 Introduction

Answer set programming (ASP) is a rising declarative programming paradigm based on logic programming and non-monotonic reasoning [12]. ASP is particularly well suited to model and solve combinatorial search problems such as scheduling [22, 1], planning [8, 11], and product configuration [24, 25]. Over the last decade ASP solvers such as `clingo` [16], `WASP` [4] and `dlv` [2] have been improved and further developed to solve and enumerate answer sets faster [17], allow for more control over the grounding and solving process [14] and even enhanced with theory reasoning capabilities [21].

Due to these developments ASP finds more and more it's way into industrial applications [19, 10]. Which reveals the issue that for real world applications the

solution space can be extremely large. Recently Fichte et al. [13] introduced a framework that allows to navigate the solution space by introducing weights on atoms that occur in some but not all answer sets. Another way to tackle this problem is to search for optimal solutions by formulating preferences [6, 5] or optimisation criteria [15, 3]. However, the user might not have a particular preference in mind. For example consider the product configuration domain, where the producer models the product to configure in terms of ASP. This kind of combinatorial problem usually allows for many combinations of parts of the product, together with a rather low number of constraints, leading to a combinatorial explosion of solutions (answer sets). For the potential customer it is not possible to inspect all (possibly several million) configurations. Therefore, it would be beneficial to be able to provide a small collection of highly diverse configurations, such that the customer obtains an overview on the different characteristics of the potential products. Eiter et al. [9] introduced several approaches how to compute similar and diverse answer sets, ranging from post processing enumerations over parallel solving to iterative solving. The approaches diverge in behaviour, so is the parallel solving complete but becomes infeasible quickly while the fast iterative solving often leads to suboptimal results.

In this work we revisit four problems formulated in [9] and propose a novel approach for computing diverse answer sets based on reworking collections of solutions. The main idea is to trade solutions from a collection is allowed as long as the collection grows. Therefore the approach works iteratively, improving the result stepwise and thus can be interrupted at *anytime*. For our approach we introduce the corresponding problem and analyse its complexity by forming a many-one hierarchy. Furthermore we lay out three different elaborations to implement the core functionality. We compare our approach with the methods based on Eiter et al. [9] in an empirical evaluation, showing that the novel Tunas approach is competitive. Also the implementations of the iterative approaches benefit from the multi-shot functionality within the `clingo` solver, by updating existing logic programs instead of re-grounding and solving from scratch. Therefore a re-evaluation of the established methods using state-of-the art solvers and wrappers is of interest as well.

2 Preliminaries

2.1 Multi-Shot Answer Set Programming

A (disjunctive) program $\mathcal{P}$ in ASP is a set of rules r of the form:

$$a_1; \ldots; a_m :- a_{m+1}, \ldots, a_n, \ \mathit{not}\ a_{n+1}, \ldots, \ \mathit{not}\ a_o.$$

where each *atom* a_i is of the form $p(t_1, \ldots, t_k)$, p is a *predicate* symbol of *arity* k and $t_1, \ldots, t_k$ are *terms* built using constants and variables. For $k{=}0$ $p()$ abbreviates to p. A *naf* (negation as failure) *literal* is of the form a or *not* a for an atom a. A rule is called *fact* if $m{=}o{=}1$, *normal* if $m{=}1$ and *integrity constraint* if $m{=}0$. Each rule can be split into a *head* $h(r) = \{a_1, \ldots, a_m\}$

and a *body* $B(r) = \{a_{m+1}, \ldots, not\ a_o\}$, which divides into a positive part $B^+(r) = \{a_{m+1}, \ldots, a_n\}$ and a negative part $B^-(r) = \{a_{n+1}, \ldots, a_o\}$. A term, atom, rule or program is said to be *ground* if it does not contain variables. For a program $\mathcal{P}$ its *ground instance* $Grd(\mathcal{P})$ is the set of all ground rules obtained by substituting all variables in each rule with ground terms. Let M be a set of ground atoms, for a ground rule r we say that $M \models r$ iff $M \cap h(r) \neq \emptyset$ whenever $B^+(r) \subseteq M$ and $B^-(r) \cap M = \emptyset$. M is a *model* of $\mathcal{P}$ if $M \models r$ for each $r \in \mathcal{P}$. M is a *stable model* (also called *answer set*) iff M is a $\subseteq$-minimal model satisfying the Gelfond-Lifschitz *reduct* of $\mathcal{P}$ w.r.t. M. The reduct is defined as $\mathcal{P}^M = \{h(r) \leftarrow B^+(r) | M \cap B^-(r) = \emptyset, r \in \mathcal{P}\}$ [18].

For the solver `clingo` several meta-statements are available, such as the `#show` directive to selectively include (and rename) atoms in the output. The *multi-shot* [14] feature within `clingo` allows altering and rerunning logic programs by defining parametrised subprograms. Subprograms are identified by a name *sp* and arity k and require a list of arguments $(p_1, \ldots, p_k)$. Within the logic program, the subprogram $sp(p_1, \ldots, p_k)$ starts after the directive `#program` $sp(p_1, \ldots, p_k)$`.` and ends before the next subprogram. If not declared the first block is attached to *base/0*. Usually the multi-shot program is handled by a wrapper script, communicating with the grounder and solver through a control object. Subprograms can be altered dynamically, allowing for flexible implementation at runtime. Truth values of atoms can be pre-assigned for each solve call via *assumptions* or *external* atoms. If used, assumptions need to be declared explicitly for each solve call, while external values are set persistently.

2.2 Diverse Solutions

Problems The umbrella term *similar/diverse* covers a bouquet of problems focusing mainly around four decision problems [9]. In our work we target the diverse problem and therefore omit the (symmetric) *similar* co-problem. For convenience, the problems are often referenced by their short notation (Γ). The term "$\mathcal{P}$ and Δ for P" abbreviates "an ASP program $\mathcal{P}$ that formulates a computational problem P and a distance measure Δ that maps a set of solutions for P to a nonnegative integer". The complexity class is added to each problem.

(1) $\mathbf{\Gamma}^{nk}_{\mathcal{P}\Delta}$ **-** n k**-DIVERSE SOLUTIONS:** Given $\mathcal{P}$ and Δ for P and two nonnegative integers n and k, decide whether a set S of n solutions for P exists such that $\Delta(S) \geqslant k$. (Complexity: NP-complete)

(2) $\mathbf{\Gamma}^{+1}_{\mathcal{P}\Delta}$ **-** k**-DISTANT SOLUTION:** Given $\mathcal{P}$ and Δ for P, a set S of solutions for P, and a nonnegative integer k, decide whether some solution s ($s \notin S$) for P exists such that $\Delta(S \cup \{s\}) \geqslant k$. (Complexity: NP-complete)

(3) $\mathbf{\Gamma}^{k\uparrow}_{\mathcal{P}\Delta}$ **-** n**-MOST DIVERSE SOLUTIONS:** Given $\mathcal{P}$ and Δ for P and a nonnegative integer n, find a set S of n solutions for P with the maximum distance $\Delta(S)$. (Complexity: FNP//log-complete)

(4) $\mathbf{\Gamma}^{n\tilde{\uparrow}}_{\mathcal{P}\Delta}$ **-** MAXIMAL n k**-DIVERSE SOLUTIONS:** Given $\mathcal{P}$ and Δ for P and a nonnegative integer k, find a $\subseteq$-maximal set S of at most n solutions for P s.t. $\Delta(S) \geqslant k$. (Complexity: $\mathrm{FP}^{\mathrm{NP}}$-complete; FNP//log-complete if k is bounded)

$\Gamma^{nk}_{\mathcal{P}\Delta}$ addresses the core problem: finding a set S of n solutions for a given program for which the distance measure is larger than k. The distance measure Δ is a function which maps a set of solutions to a non-negative integer. In practice Δ is usually monotone ($\Delta(S) \geqslant \Delta(S \cup S')$ for any sets S, S' of solutions) and can be customised. A default implementation is the minimum of the hamming distance for any pair of solutions within the collection. $\Gamma^{+1}_{\mathcal{P}\Delta}$ asks for *one* solution which is at least k-distant to each element of a provided solution set (*collection*). Therefore $\Gamma^{+1}_{\mathcal{P}\Delta}$ can be utilised to semi-decide $\Gamma^{nk}_{\mathcal{P}\Delta}$. The other two problems target maximising k ($\Gamma^{k\uparrow}_{\mathcal{P}\Delta}$) and n ($\Gamma^{n\hat{\uparrow}1}_{\mathcal{P}\Delta}$) and can be decided by repeatedly solving $\Gamma^{nk}_{\mathcal{P}\Delta}$ for increasing k (resp. n).

Related Work. In [9] the following approaches targeting $\Gamma^{nk}_{\mathcal{P}\Delta}$ were introduced.

Offline solves $\Gamma^{nk}_{\mathcal{P}\Delta}$ directly by collecting some or all answer sets and picking diverse answer sets in a detached process, which in principle solves the NP-complete $k-clique$ problem for a potentially exponential input. Since solutions are generated in vaguely sorted order, restricting the output size will impact the outcome drastically. Therefore Offline faces heavy drawbacks and will not be covered in our evaluation.

Online1 is based on manipulating the original logic program to solve n copies of the problem at the same time while satisfying constraints on the distance measure, targeting $\Gamma^{nk}_{\mathcal{P}\Delta}$ in a single shot. Online1 is a complete approach.

Online2 tries to solve the main problem $\Gamma^{nk}_{\mathcal{P}\Delta}$ by repeatedly solving $\Gamma^{+1}_{\mathcal{P}\Delta}$. The program generates a solution and wrapper script adds it to the program, forcing the next solution to be diverse to the collected solutions. This process repeats until the collection has the desired quantity or the program becomes unsatisfiable. Additional to the original program and a wrapper script, this approach requires a method to add distance constraints. Online2 semi-decides $\Gamma^{nk}_{\mathcal{P}\Delta}$.

Online3 works similar to Online2 but the functionality is embedded into the solver instead of a script. [9] presented clasp-nk, a branch-development of clasp 1.1.3 which included hard-coding a distance calculation for one freely selectable predicate. Since this solver is not competitive with current generation solvers, we did not include Online3 despite showing comparable results to Online2.

Further Related Work. Romero et al. [23] propose a multi-shot framework for computing diverse *preferred* answer sets for *asprin*. The paper presents three advanced diversification techniques for programs with preferences, which are generalised methods based on [9], partly utilising preferences into the solving process. Since Romero et al. are using preferences, the methods are not directly empirical comparable to our work.

3 Iterative Reworking Strategies

So far there exist four problems and two promising approaches. The parallel approach (Online1) lifts the original program to solve $\Gamma^{nk}_{\mathcal{P}\Delta}$, while the iterative

[1] $\Gamma^{n\hat{\uparrow}}_{\mathcal{P}\Delta}$ asks for $\subseteq$-maximal sets and restricts the maximal value to bound output size.

approach (Online2) collects solutions by repeatedly solving and updating the logic program. Once the program becomes unsatisfiable, the diverse collection is subset-maximal and can not grow in any further. In our iterative reworking strategy, we aim to surpass this subset-maximal boundary by trading $m{\geq}0$ solutions from a given collection S for at least $m{+}1$ new solutions. This novel approach does not target any of the introduced problems directly, therefore we define and analyse the corresponding problem before introducing the actual approach.

3.1 Problem Definition and Complexity

Based on the notations from Section 2.2 we describe a new problem, which introduces the possibility of replacing at most m solutions from a given collection S with at least $m{+}1$ solutions to form an improved diverse collection S'.

(5) $\mathbf{\Gamma}^{\leqslant m}_{\mathcal{P}\Delta}$ **-** m**-different** k**-distant set:** Given $\mathcal{P}$ and Δ for P, a set S of solutions for P, two nonnegative integer k and m with $m{\leqslant}|S|$, decide if a set S' of solutions for P exists s.t. $|S'|{>}|S|$, $|S{\setminus}S'|{\leqslant}m$ and $\Delta(S'){\geqslant}k$.

In other words: we try to expand the given, potentially empty collection S of solutions by exchanging up to m solutions for a greater number of solutions. It is valid to exchange less than m solutions as long as the collection grows. While sharing similarities with $\Gamma^{nk}_{\mathcal{P}\Delta}$ and $\Gamma^{+1}_{\mathcal{P}\Delta}$, $\Gamma^{\leqslant m}_{\mathcal{P}\Delta}$ introduces a new parameter m, creating $|S|{+}1$ possible sub-problems for a given set S of solutions. For convenience we will write "$\Gamma^{\leqslant m}_{\mathcal{P}\Delta}$ with $m{=}t$" as $\Gamma^{\leqslant t}_{\mathcal{P}\Delta}$ for any non-negative integer t. We will proceed to show equality of $\Gamma^{+1}_{\mathcal{P}\Delta}$ and $\Gamma^{\leqslant 0}_{\mathcal{P}\Delta}$ (Lemma 1) as well as interchangeability of $\Gamma^{nk}_{\mathcal{P}\Delta}$ and $\Gamma^{\leqslant n-1}_{\mathcal{P}\Delta}$ (Lemma 2). We will furthermore establish a hierarchy within $\Gamma^{\leqslant m}_{\mathcal{P}\Delta}$ (Lemma 3) to show NP-completeness (Theorem 1). For the proofs we assume Δ to be monotone ($\Delta(S){\geqslant}\Delta(S{\cup}S')$ for any sets S, S' of solutions).

Lemma 1. *$\Gamma^{+1}_{\mathcal{P}\Delta}(S,k)$ iff $\Gamma^{\leqslant 0}_{\mathcal{P}\Delta}(S,k)$.*

Proof. $\Gamma^{+1}_{\mathcal{P}\Delta}(S,k) \Rightarrow \Gamma^{\leqslant 0}_{\mathcal{P}\Delta}(S,k)$: if there exists a solution s s.t. $\Delta(S{\cup}\{s\}) \geqslant k$ then the set $S'{=}S{\cup}\{s\}$ satisfies $|S'|{>}|S|$, $|S{\setminus}S'|{\leqslant}0$ and $\Delta(S'){\geqslant}k$.
$\Gamma^{\leqslant 0}_{\mathcal{P}\Delta}(S,k) \Rightarrow \Gamma^{+1}_{\mathcal{P}\Delta}(S,k)$: if there exists a set of solutions S' s.t. $|S'|{>}|S|$, $|S{\setminus}S'|{\leqslant}0$ and $\Delta(S'){\geqslant}k$ then $S{\subset}S'$ and for all $s{\in}(S'{\setminus}S)$: $\Delta(S \cup \{s\}) \geqslant k$. □

Lemma 2. *$\Gamma^{nk}_{\mathcal{P}\Delta}(n,k)$ and $\Gamma^{\leqslant n-1}_{\mathcal{P}\Delta}(S,k)$ are interchangeable for $|S| = n{-}1$.*

Proof. $\Gamma^{nk}_{\mathcal{P}\Delta}(n,k) \Rightarrow \Gamma^{\leqslant n-1}_{\mathcal{P}\Delta}(S,k)$ with $|S|{=}n{-}1$: if there exists a set of solutions S' s.t. $|S'|{=}n$ and $\Delta(S'){\geqslant}k$ then for any possible set S of solutions with $|S|{=}n{-}1$: $|S'|{>}|S|$, $|S{\setminus}S'|{\leqslant}n{-}1$ and $\Delta(S'){\geqslant}k$. [2]
$\Gamma^{\leqslant n-1}_{\mathcal{P}\Delta}(S,k) \Rightarrow \Gamma^{nk}_{\mathcal{P}\Delta}(n,k)$: given a set S of at least $n{-}1$ solutions, if there exists a set S' of solutions s.t. $|S'|{>}|S|{\geqslant}n{-}1$ and $\Delta(S'){\geqslant}k$ then there exists $S'' \subseteq S'$ where $|S''|{=}n$ and $\Delta(S'') \geqslant k$. □

We covered complexity for $\Gamma^{\leqslant m}_{\mathcal{P}\Delta}(S,k)$ for $m{=}0$ and $m{=}|S|$, for the remaining $0{<}m{<}|S|$ we can build a hierarchy using a poly-time many-one-reduction.

[2] This implication even holds if S and S' share elements ($S \cap S' \neq \emptyset$) or if $\Delta(S){<}k$.

Lemma 3. *$\Gamma^{\leqslant m}_{\mathcal{P}\Delta}(S,k) \leq_p \Gamma^{\leqslant m+1}_{\mathcal{P}\Delta}(S,k)$ (poly-time many-one-reduction)*

Proof. $\Gamma^{\leqslant m}_{\mathcal{P}\Delta}(S,k)$ has the problem instance $(\mathcal{P},\Delta,S,k)$ which we will convert to $(\mathcal{P}^{\pm},\Delta^{\pm},S^{\pm},k^{\pm})$ for $\Gamma^{\leqslant m+1}_{\mathcal{P}^{\pm},\Delta^{\pm}}(S^{\pm},k^{\pm})$ by introducing two artificial, unique solutions ϵ^{+} and ϵ^{-}. The program $\mathcal{P}$ is altered to accept ϵ^{+} and ϵ^{-}, forming $\mathcal{P}^{\pm}$. $\Delta^{\pm}$ is derived from Δ by adding distance information for ϵ^{-} and ϵ^{+}: ϵ^{-} has distance 0 to any solution and ϵ^{+} has distance $k{+}1$ to any original solution. To avoid loopholes regarding $k{=}0$, the distance value for each original solution pair is increased by 1 for $\Delta^{\pm}$, therefore k has to be updated as well ($k^{\pm} = k{+}1$). The "negative" solution ϵ^{-} is added to the initial solution set $S^{\pm} = S \cup \{\epsilon^{-}\}$, whereas the "positive" solution ϵ^{+} remains as potential solution to compensate for the forced selection of ϵ^{-}. Both instances return the same value for their problem:

$\Gamma^{\leqslant m}_{\mathcal{P}\Delta}(S,k) \Rightarrow \Gamma^{\leqslant m+1}_{\mathcal{P}^{\pm}\Delta^{\pm}}(S^{\pm},k^{\pm})$: if there is a solution set S' for s.t. $|S'|{>}|S|$, $|S\backslash S'|{\leqslant}m$ and $\Delta(S'){\geqslant}k$ with $\epsilon^{-},\epsilon^{+} \notin (S{\cup}S')$ then the solution set $S^{\pm\prime} = S'{\cup}\{\epsilon^{+}\}$ satisfies all criteria for $\Gamma^{\leqslant m+1}_{\mathcal{P}^{\pm}\Delta^{\pm}}(S{\cup}\{\epsilon^{-}\},k{+}1)$: $|S^{\pm\prime}| > |S{\cup}\{\epsilon^{-}\}|$, $|(S{\cup}\{\epsilon^{-}\})\backslash S^{\pm\prime}| \leqslant m{+}1$ and $\Delta^{\pm}(S^{\pm\prime}) \geqslant k{+}1$.

$\Gamma^{\leqslant m+1}_{\mathcal{P}^{\pm}\Delta^{\pm}}(S^{\pm},k^{\pm}) \Rightarrow \Gamma^{\leqslant m}_{\mathcal{P}\Delta}(S,k)$: if there is a solution set $S^{\pm\prime}$ such that $|S^{\pm\prime}|{>}|S{\cup}\{\epsilon^{-}\}|$, $|(S{\cup}\{\epsilon^{-}\})\setminus S^{\pm\prime}| \leqslant m{+}1$ and $\Delta^{\pm}(S^{\pm\prime}) \geqslant k{+}1$ (implying $\epsilon^{-},\epsilon^{+}{\notin}S$ and $\epsilon^{-}{\notin}S^{\pm\prime}$) then $S' = S^{\pm\prime}\backslash\{\epsilon^{+}\}$ satisfies $|S'|{>}|S|$, $|S \setminus S'| \leqslant m$ and $\Delta(S'){\geqslant}k$. This implication holds for both cases $\epsilon^{+}{\in}S^{\pm\prime}$ and $\epsilon^{+}{\notin}S^{\pm\prime}$.

□

Theorem 1. *Problem $\Gamma^{\leqslant m}_{\mathcal{P}\Delta}$ - m-DIFFERENT k-DISTANT SET is NP-complete.*

Proof. As proven in Lemma 3 there exists a hierarchy within $\Gamma^{\leqslant m}_{\mathcal{P}\Delta}$ where $\Gamma^{\leqslant m}_{\mathcal{P}\Delta} \leq_p \Gamma^{\leqslant m+1}_{\mathcal{P}\Delta}$. Since the problem for the lowest m ($m{=}0$) corresponds to the NP-complete problem $\Gamma^{+1}_{\mathcal{P}\Delta}$ (Lemma 1), all problems are NP-hard. And since the top-most problem ($m{=}|S|$) corresponds to the NP-complete problem $\Gamma^{nk}_{\mathcal{P}\Delta}$(Lemma 2) all problems in the hierarchy lie within NP. Since lower and upper bound are NP-complete, all problems in $\Gamma^{\leqslant m}_{\mathcal{P}\Delta}$ are NP-complete as well. □

3.2 Reworking Methods

The *Trade Up Navigation for Answer Sets* (short *Tunas*) approach is a framework that allows to solve $\Gamma^{nk}_{\mathcal{P}\Delta}$ by iteratively solving $\Gamma^{\leqslant m}_{\mathcal{P}\Delta}$. The core idea is to remove up to m solutions from a collection to gain a larger collection, therefore improving the result. Since $\Gamma^{\leqslant m}_{\mathcal{P}\Delta}$ is easier to compute than $\Gamma^{\leqslant m+1}_{\mathcal{P}\Delta}$, the approach starts for $m{=}0$ and ends at a given maximal value M. Like this, building a collection starts just like Online2, only when the program becomes unsatisfiable, the actual reworking process starts. An outline of the general implementation can be seen in Algorithm 1. The different implementations of the core function *replaceM*(*ctl*, m, *pool*, *idx*), which provides deletion candidates and new solutions, are explained afterwards.

Some notes to the general framework: The logic program itself is handled by the control structure *ctl* which allows for grounding and solving as well as for additional functionality such as managing external atoms. The wrapper script

Algorithm 1: Tunas Framework

Input: A fitting multi-shot logic program P, minimum distance measure K, number of solutions N, maximum Number M of deletion candidates, set *Init* of subprograms to ground initially, implementation of *replaceM*

```
1  ctl.load(P)
2  ctl.ground(Init)                                   // ground original program
3  idx ← 1
4  pool ← {}
5  do:
6  |  for m ← 0 to M:
7  |  |  (delS, newS, idx) ← replaceM(ctl, m, pool, idx)        // core function
8  |  |  if newS ≠ ∅:
9  |  |  |  for del ∈ delS:
10 |  |  |  |  ctl.release_external(visible(del))      // remove from collection
11 |  |  |  pool ← ( pool \ delS ) ∪ newS
12 |  |  |  break
13 while (|pool| ≤ N) and (newS ≠ ∅)
```

uses indices to reference solutions, the set *pool* holds all indices from the current collection. The basic framework consists mainly of two loops: one to enlarge the collection and one to escalate the current m up to the maximum M. A core function is called (line **7**) to solve the current problem which returns indices of deletion candidates (*delS*) and indices of new solutions (*newS*) to be added to the collection. If a solution for the current m was found, the deletion is made permanent by releasing corresponding atoms (line **10**) and updating the set *pool* which represents the collection. Also the current escalation of m is disrupted to start over from 0.

Since the basic framework is explained, let us have a detailed look into the possible implementations of *replaceM*. Each implemtation requires a base program in the form of subprogram calls (*Init*, line **2**).

TunasMND implements generating multiple (M) solutions and nondeterministic (ND) choosing of deletion candidates. In other words: the logic program guesses the deletion candidates and generates all new solutions in a single call. An outline of the wrapper functionality can be seen in Algorithm 2 implementing *replaceM* from the framework in Algorithm 1. The initial program (see *Init* from Algorithm 2) contains the subprogram names of the modified original program. For TunasMND this is comparable to Online1 to compute up to $M+1$ diverse solutions.

The amount m of deletion candidates (and consequently $m+1$ new solutions) is predetermined by the wrapper and passed onto the logic program via an assumption over a corresponding predicate (*line* **1**). An internal mechanism induces the temporary concealment of exactly m solutions. If successful, their indices can be extracted from the model (*line* **8**). Also the model contains $m+1$ new solutions. The `#show` directive (*line* **2**) within the logic program is used to

Algorithm 2: TunasMND - *replaceM*(*ctl, m, pool, idx*) : (*delS, newS, idx*)

Input: A control structure *ctl* which handles the current logic program, a number $m \geqslant 0$ of deletion candidates, a set *pool* of indices representing established solutions, the next solution index *idx*

Output: A set of indices referencing to deletion candidates, a set of indices referencing to new solutions, the next solution index

```
1 if mdl ← ctl.solve(assumption = {(chooseNum(m), true)}):
2     grnd ← {p(idx+i, t_1, ..., t_k) : p(i, t_1, ..., t_k) ∈ mdl ∧ #show(p(i, t_1, ..., t_k)) ∈ Init}
3     registerPredicates(ctl, grnd )
4     ctl.ground( grnd )
5     for i ← 0 to m:
6         ctl.ground( {itDistM(idx+i))} )
7         ctl.assign_external(visible(idx+i), true)
8     return({i : choose(i) ∈ mdl}, {idx, ..., idx+m}, idx+m+1)
9 return(∅, ∅, idx)
```

distinguish between solution defining atoms (*goal atoms*) and auxiliary atoms. Since generated solutions proceed to become collected solutions, their index is updated before grounding them back into the logic program (*line* **2**).

For multi-shot programs, adding atoms to a program is possible by defining sub-programs which contain the atoms. We implemented the function *registerPredicates* to construct and register those subprograms automatically, allowing flexible and dynamic handling of heterogenous goal atoms. Afterwards the distance constraints for the new solutions are grounded (*line* **6**), which become active, when setting the corresponding external atom to *true* (*line* **7**).

If successful the set of indices for the deletion candidates and the newly generated solutions are returned (*line* **8**). Otherwise empty sets are returned.

TunasMIT implements generating multiple (M) solutions and iterative (IT) choosing of deletion candidates: all new solutions are generated at once, while the deletion candidates are provided by the wrapper. In comparison to TunasMND, two mayor changes are required: disabling the choosing mechanism and handling the deletion candidates before solving. The candidates are selected by iterating over all combinations of size m from the set *pool*. The current set of deletion candidates is temporarily excluded from the collection by setting the corresponding external key predicate (*visible/1*) to *false* before solving the current program. If no solution could be found, the deletion candidates are made visible again and the wrapper continues with the next combination of m deletion candidates.

We will use *TunasM* to address TunasMIT and TunasMND at once.

TunasS implements the single (S) generating, iterative choosing (IT) approach. The underlying logic program is an extension of the logic program for Online2 - in fact Online2 and TunasS are equivalent for M=0. In comparison to TunasMIT the solve call is embedded into another loop because each solve call generates

only one instead of m+1 solutions. The currently generated solution streak is held in limbo until either all m+1 solutions could be computed or the program becomes unsatisfiable, resulting in the deletion of the current streak.

We left out the single solving (S), nondeterministic choosing (ND) approach due to contradiction in progression: The (ongoing) suggestion of deletion candidates requires the program to stay unaltered in case the generating part fails and different deletion candidates need to be derived. Yet at the same time the program needs to be updated and solved to generate solutions. Splitting the program into two separate programs boils down to TunasS.

4 Experimental Evaluation

To show feasibility of our approach, we compare it with the established methods Online1 and Online2. The evaluation focuses around solving $\Gamma^{k\uparrow}_{\mathcal{PD}}$- n-MOST DIVERSE SOLUTIONS, which combines highest complexity along with highest relevance for applications. We analyse the behaviour of the approaches regarding the following questions: **Q1**: Is the Tunas approach competitive with existing methods in terms of solving the problem $\Gamma^{k\uparrow}_{\mathcal{PD}}$? **Q2**: How do the approaches perform in time? **Q3**: How reliable are the methods?

Software: We implemented our approaches along with Online1 and a multi-shot version of Online2. Online1 was extended to maximise for k to solve $\Gamma^{k\uparrow}_{\mathcal{PD}}$. The other approaches required repetitive solving $\Gamma^{nk}_{\mathcal{PD}}(n,k)$ for increasing k since they only semi-decide $\Gamma^{nk}_{\mathcal{PD}}(n,k)$: if $\Gamma^{nk}_{\mathcal{PD}}(n,k)$ was satisfied, k is incremented. This repeats infinitely but can be interrupted at any time (timeout).

Setup and Hardware: Our environment is a virtual machine (Debian 5.10.46, 64Bit, Intel Xeon Gold, 3GHz, clingo 5.5, python 3.8.8). We tested for $n \in \{3, 5, 10, 15, 25\}$, timeout at 300s sharp. Each configuration was repeated 20 times. We use random generated seeds (random frequency: 10%). The test-setup including all data is available for download [7].

Test instances: We used 5 instances for our evaluation: (**I1**) Phylogenesis (ancestry tress for languages) and (**I2**) blocks-world (planing problem) from [9], (**I3**) PC configurator (configuration problem) from [20] (with 2 instead of the default 10 hardware instances), (**I4**) an encoding of stable extensions and (**I5**) preferred extensions, both for the same argumentation framework (AF) instance [13]. **I1** uses a custom distance (nodal), all other use hamming distance. The first three instances have a vast search space ($>10^9$ answer sets). The AF instances **I4** and **I5** share an identical and comparatively small solution space (7696 answer sets), but the problem classes differ in complexity.

The average maximal k for **I3**, **I4** and **I5** are listed in Table 1. For **I1** and **I2** all approaches reached the maximal k, and thus they will not be discussed further here. We ordered the methods from non-deterministic to iterative, where each Tunas method is evaluated for $M \in \{1, 2\}$. The entries can be ranked for each instance and n, for example for **I4** and n=5, Online1 has the highest value (63.0), whereas the Online2 has the lowest value (58.6), implying better results from Online1 than Online2 for this setting.

PC I3	$n=3$	$n=5$	$n=10$	$n=15$	$n=25$
Online1	**46.5**	37.7	29.3	23.7	9.3
TunasMND	42.5 (43.2)	39.5 (**40.0**)	36.2 (36.3)	33.8 (33.9)	23.4 (22.3)
TunasMIT	42.0 (43.0)	39.8 (39.6)	36.4 (36.1)	**34.0** (**34.0**)	30.6 (30.9)
TunasS	41.9 (42.0)	39.8 (39.7)	**36.5** (36.4)	**34.0** (**34.0**)	**31.1** (30.9)
Online2	42.0	39.8	36.2	33.9	30.9
AF stable I4	$n=3$	$n=5$	$n=10$	$n=15$	$n=25$
Online1	**81.0***	**63.0**	47.9	40.9	34.3
TunasMND	**81.0** (**81.0**)	61.8 (61.7)	48.8 (48.3)	42.0 (42.0)	**36.0** (35.5)
TunasMIT	**81.0** (**81.0**)	61.7 (62.2)	48.4 (**48.9**)	42.2 (42.3)	**36.0** (35.9)
TunasS	**81.0** (**81.0**)	61.6 (62.1)	48.4 (48.8)	42.3 (**43.0**)	**36.0** (35.9)
Online2	75.1	58.6	46.4	41.0	35.0
AF pref. I5	$n=3$	$n=5$	$n=10$	$n=15$	$n=25$
Online1	54.9	35.1	23.2	15.1	5.1
TunasMND	77.6 (73.7)	54.0 (54.1)	43.5 (43.9)	37.9 (37.7)	31.4 (31.1)
TunasMIT	**77.8** (73.2)	54.5 (54.4)	43.2 (43.9)	37.9 (37.6)	31.2 (30.9)
TunasS	70.7 (72.2)	55.4 (55.9)	44.3 (44.0)	38.1 (38.1)	31.9 (31.4)
Online2	71.9	**56.7**	**45.0**	**39.1**	**33.1**

Table 1: maximal k for $\Gamma^{k\uparrow}_{\mathcal{PA}}(n)$ for 3 instances, average over 20 runs. For Tunas numbers outside (resp. inside) of brackets reference to M=1 (resp. M=2). * marks a proven maximal value by Online1, best results are bold.

In general we observe that Online1 performs best for smaller n, which is not surprising since the size of the problem encoding is related to n. For **I3** all iterative methods perform on the same level (except TunasMND for n=25). Notably is the lead for Online1 for n=3, which results from low chances to guess a solution which can form a diverse collection of size 3. A closer look at the data reveals that **I3** has no repeating attempts, implying high time cost for returning unsatisfiable. This explains the equal performance of the iterative methods, since all Tunas methods start with m=0 and are unable to progress to m=1.

The upper bounds of k for **I4** and **I5** are identical since they share an identical solution space, but belong to different complexity classes. Online1 shows good results for **I4**, even leading for $n\leq 5$ but performs worst for **I5**. Online2 performs unexpected sub-optimal for **I4** but leads for **I5**. For **I4** all Tunas methods perform at similar levels. For **I5** TunasS performs slightly better than TunasM with exception for n=3 where TunasM outperforms all other methods. To better understand the results, the median[3] run time can be seen in Figure 1 for n=5. The maximal value on the x-axis for each curve corresponds to the data in Table 1 while the y-axis reflects the median time to reach the corresponding k. The optimal curves are wide and flat: large k values for low execution times. The plot visualises also different results for different timeouts. For Example with $timeout$=10s TunasS has a larger k for both AF instances as Online2,

[3] Not all timelines cover the same k due to the time limit, distorting the mean value. The median is robust against those outliers by setting missing data to the time limit.

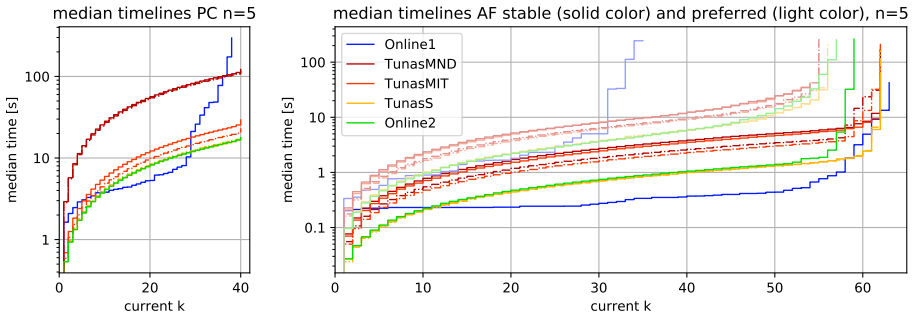

Fig. 1: Median timelines of current k for 20 runs, n=5. Tunas methods: solid lines for M=2, striped lines for M=1. The right figure includes AF stable (solid colors) and AF preferred (light colors).

implying better performance for this timeout. Timelines for TunasS and Online2 progress similar, while TunasM requires more time for larger M. In contrast to the iterative methods, Online1 has unsteady escalation times. One interesting curve is Online2 for **I4**: at around k=56 Online2 struggles to keep the pace, implying lower probability of satisfying $\Gamma^{nk}_{\mathcal{P}\Delta}(n,k)$. This effect can be seen for the Tunas methods as well but for higher k. The curves for **I3** (PC, n=5) hint an explanation to the performance drop of TunasMND for n=25: since TunasMND progresses significantly slower it is expected to be the first to suffer a performance decline for increased difficulty or lower timeout.

Now, let us answer the question. **Q1**: None of the evaluated methods is superior, all methods lead at least once for maximal k. The outcome depends highly on the instance, n and the timeout. **Q2**: Online1 is fastest for lower n. Also it is the only terminating approach, since the other approaches are allowed to start over. Online2 is the fastest method, providing good results in a short amount of time. TunasS behaves similar as Online2: in most cases they are comparable in speed and performance. Depending on instance, n and timeout one performs better than the other. The TunasM methods proceed comparatively slow but perform surprisingly good in some cases, such as **I5** for n=3. The value of M seems to have no major impact for TunasS, but for TunasM; the performance differs most for n=3. Also a higher M is often related to higher computation time, which is important for low timeouts. **Q3**: Online1 is suited for smaller n but can not guarantee providing any result, since grounding and solving may require excessive time in comparison to the original problem. Online2 generates good results but not for all instances: for **I4** Online2 shows the weakest results, for n=3 it is the only method not reaching the maximum. The performance for the Tunas methods in comparison with Online2 seems to be tied to the performance of Online1: The more a program is suited for Online1, the better perform the Tunas methods, often surpassing both, Online1 and Online2. This implies the combination of traits of both methods, which makes sense when interpreting

Tunas as the progression from Online2 to Online1. Therefore for an unknown program, TunasS states a good choice, since it inherits a similar time behaviour as Online2 but can surpass Online2 if the program would be suited for Online1.

5 Conclusion and Future Work

We presented a novel and competitive approach to compute diverse answer sets of logic programs based on reworking methods. To characterise the base mechanism, we introduced a new problem related to the approach and prove NP-completeness. Basic analysis of the approach leads to three elaborations, which we implemented along with two methods from Eiter et al. [9] (as multi-shot variant). Our empirical evaluation to find n-MOST DIVERSE SOLUTIONS reveals no superior method, since problem instance, number n of solutions and timeout highly influence the performance. However the Tunas methods show promising results, especially for large solution spaces and typical NP problems.

For future work, we believe investigating the connection of facet counting weights [13] and diverse answer sets is a promising direction. Furthermore, the extension to preferred logic programs would be of interest [23].

Acknowledgements

This work was partly supported by Deutsche Forschungsgemeinschaft (DFG) in project 389792660 (TRR 248, Center for Perspicuous Systems), and by the Bundesministerium für Bildung und Forschung (BMBF) in project 01IS20056_NAVAS.

References

1. Abels, D., Jordi, J., Ostrowski, M., Schaub, T., Toletti, A., Wanko, P.: Train scheduling with hybrid answer set programming. TPLP **21**(3), 317–347 (2021)
2. Alviano, M., Calimeri, F., Dodaro, C., Fuscà, D., Leone, N., Perri, S., Ricca, F., Veltri, P., Zangari, J.: The ASP system DLV2. In: Balduccini, M., Janhunen, T. (eds.) Proc of LPNMR 2017. LNCS, vol. 10377, pp. 215–221. Springer (2017)
3. Alviano, M., Dodaro, C.: Anytime answer set optimization via unsatisfiable core shrinking. TPLP **16**(5-6), 533–551 (2016)
4. Alviano, M., Dodaro, C., Faber, W., Leone, N., Ricca, F.: WASP: A native ASP solver based on constraint learning. In: Cabalar, P., Son, T.C. (eds.) Proc. of LPNMR 2013. LNCS, vol. 8148, pp. 54–66. Springer (2013)
5. Alviano, M., Romero, J., Schaub, T.: Preference relations by approximation. In: Thielscher, M., Toni, F., Wolter, F. (eds.) Proc. of KR 2018. pp. 2–11. AAAI Press (2018)
6. Brewka, G., Delgrande, J.P., Romero, J., Schaub, T.: asprin: Customizing answer set preferences without a headache. In: Bonet, B., Koenig, S. (eds.) Proc. of AAAI 2015. pp. 1467–1474. AAAI Press (2015)
7. Böhl, E., Gaggl, S.A.: Tunas - Fishing for diverse Answer Sets: a Multi- Shot Trade up Strategy (Tool & Experiments) (Jun 2022). https://doi.org/10.5281/zenodo.6762554

8. Dimopoulos, Y., Nebel, B., Koehler, J.: Encoding planning problems in nonmonotonic logic programs. In: Steel, S., Alami, R. (eds.) Proc. of ECP'97. LNCS, vol. 1348, pp. 169–181. Springer (1997)
9. Eiter, T., Erdem, E., Erdogan, H., Fink, M.: Finding similar/diverse solutions in answer set programming. Theory Pract. Log. Program. **13**(3), 303–359 (2013)
10. Eiter, T., Falkner, A.A., Schneider, P., Schüller, P.: Asp-based signal plan adjustments for traffic flow optimization. In: Giacomo, G.D., Catalá, A., Dilkina, B., Milano, M., Barro, S., Bugarín, A., Lang, J. (eds.) Proc. of ECAI 2020. FAIA, vol. 325, pp. 3026–3033. IOS Press (2020)
11. Erdem, E., Lifschitz, V.: Transformations of logic programs related to causality and planning. In: Gelfond, M., Leone, N., Pfeifer, G. (eds.) Proc. of LPNMR'99. LNCS, vol. 1730, pp. 107–116. Springer (1999)
12. Faber, W.: An introduction to answer set programming and some of its extensions. In: Manna, M., Pieris, A. (eds.) Reasoning Web. Declarative Artificial Intelligence Summer School 2020. LNCS, vol. 12258, pp. 149–185. Springer (2020)
13. Fichte, J.K., Gaggl, S.A., Rusovac, D.: Rushing and strolling among answer sets - navigation made easy. In: Proc. of AAAI 2022 (2022)
14. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Multi-shot asp solving with clingo. TPLP **19**(1), 27–82 (2019)
15. Gebser, M., Kaminski, R., Schaub, T.: Complex optimization in answer set programming. TPLP **11**(4-5), 821–839 (2011)
16. Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., Schneider, M.: Potassco: The potsdam answer set solving collection. AI Commun. **24**(2), 107–124 (2011)
17. Gebser, M., Maratea, M., Ricca, F.: The seventh answer set programming competition: Design and results. TPLP **20**(2), 176–204 (2020)
18. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R., Bowen, Kenneth (eds.) Proc. of International Logic Programming Conference and Symposium. pp. 1070–1080. MIT Press (1988)
19. Gençay, E., Schüller, P., Erdem, E.: Applications of non-monotonic reasoning to automotive product configuration using answer set programming. J. Intell. Manuf. **30**(3), 1407–1422 (2019)
20. Gorczyca, P.: Configuration Problem ASP Encoding Generator (Nov 2020). https://doi.org/10.5281/zenodo.5777217
21. Janhunen, T., Kaminski, R., Ostrowski, M., Schellhorn, S., Wanko, P., Schaub, T.: Clingo goes linear constraints over reals and integers. TPLP **17**(5-6), 872–888 (2017)
22. Kahraman, M.K., Erdem, E.: Personalized course schedule planning using answer set programming. In: Alferes, J.J., Johansson, M. (eds.) Proc. of PADL 2019. LNCS, vol. 11372, pp. 37–45. Springer (2019)
23. Romero, J., Schaub, T., Wanko, P.: Computing Diverse Optimal Stable Models. In: Carro, M., King, A., Saeedloei, N., Vos, M.D. (eds.) TC of ICLP 2016. OASIcs, vol. 52, pp. 3:1–3:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2016)
24. Soininen, T., Niemelä, I.: Developing a declarative rule language for applications in product configuration. In: Gupta, G. (ed.) Proc. of PADL '99. LNCS, vol. 1551, pp. 305–319. Springer (1999)
25. Soininen, T., Niemelä, I., Tiihonen, J., Sulonen, R.: Representing configuration knowledge with weight constraint rules. In: Provetti, A., Son, T.C. (eds.) Proc. of ASP'01 (2001)