Exploiting Partial Assignments for Efficient Evaluation of Answer Set Programs with External Source Access*

Thomas Eiter, Tobias Kaminski, Christoph Redl, and Antonius Weinzierl
Institut für Informationssysteme, Technische Universität Wien
Favoritenstraße 9-11, A-1040 Vienna, Austria
{eiter,kaminski,redl,weinzierl}@kr.tuwien.ac.at

Abstract

Answer Set Programming (ASP) is a well-known problem solving approach based on nonmonotonic logic programs and efficient solvers. HEX-programs extend ASP with *external atoms* for access to arbitrary external information. In this work, we extend the evaluation principles of external atoms to partial assignments, lift nogood learning to this setting, and introduce a variant of nogood minimization. This enables external sources to guide the search for answer sets akin to theory propagation. Our benchmark experiments demonstrate a clear improvement in efficiency over the state-of-the-art HEX-solver.

1 Introduction

HEX is an extension of Answer Set Programming (ASP) [Gelfond and Lifschitz, 1991] integrating external information sources (e.g. XML/RDF data bases, description logic reasoners, SAT solvers, etc.) using so-called *external atoms*. These pass information from the program, given by predicates and constants, to an external source, which returns output values to the program. For instance, the external atom $\&synonym[car](X)$ might be used to find the synonyms X of car, e.g. $automobile$. HEX is highly expressive supporting e.g., nonmonotonic aggregates [Alviano *et al.*, 2015] and recursive data exchange between programs and external sources.

Current evaluation algorithms for HEX [Eiter *et al.*, 2012] first compute a complete truth-assignment by guessing the truth values of external atoms; only after the assignment is complete, calls to the external sources can check if the guesses were correct. This constitutes a major bottleneck and limits the applicability of HEX-programs because external sources are unable to guide the search algorithm effectively. Intuitively, evaluating external sources earlier, i.e., evaluating them under partial assignments, yields significant performance gains since it avoids many wrong guesses. On the other hand, such an evaluation has non-trivial issues because external sources are largely black boxes and may be nonmonotonic.

In this paper, we overcome the above problems by extending external sources from a Boolean semantics, defined only under complete assignments, to a three-valued semantics which is also defined under partial assignments. This enables novel evaluation techniques towards our main goal of efficiency, namely, (i) evaluating external sources already during the search with partial assignments, and based on this (ii) acquiring additional information about the external sources in the form of learned nogoods [Gebser *et al.*, 2012] and (iii) minimizing the learned nogoods to approximate the semantics of external sources more closely. We note that the semantics of the overall formalism is not changed (answer sets are still two-valued). Moreover, our approach provides full backwards compatibility with existing two-valued sources.

Our techniques are related to *theory propagation* in SMT [Barrett *et al.*, 2009] and minimization techniques in constraint ASP solvers such as CLINGCON [Ostrowski and Schaub, 2012]. These, however, usually rely on a tailored integration of theory solvers crafted by experts, whereas our approach allows a broad range of users, without prior knowledge on solver construction, to harness performance gained by the new learning techniques.

Furthermore, motivated by the formalism at hand, we develop refinements such as the simultaneous minimization of multiple nogoods, from which also related approaches might benefit. Although each additional step of evaluation and minimization is costly, we find in our experiments a good tradeoff and achieve a significant overall performance gain.

After necessary preliminaries in Section 2, we proceed to present our contributions as follows:

- In Section 3, we extend external atoms such that they can be evaluated under partial assignments. The output is then three-valued (true, false, unassigned), while answer sets are still two-valued. We also show how existing external atoms can be captured by our extension.
- Next, we present a novel evaluation algorithm which exploits the extension of external sources for earlier and significantly more effective search space pruning.
- In Section 4, we present nogood learning functions and a suitable minimization algorithm for HEX-programs, which exploit the generalized external atom interface for learning. We further show a close relation between nogood minimization and theory-specific learning.
- In Section 5, we present our prototypical implementation and evaluate our new techniques using a benchmark

*This research has been supported by the Austrian Science Fund (FWF) projects P27730 and W1255-N23.

suite. The results show a speedup of up to two orders of magnitude (an exponential gain is possible in theory).

- In Section 6, we discuss related work and conclude.

2 Preliminaries

We follow Drescher *et al.* [2008] for basic concepts. A (signed) literal is a positive or a negated ground atom $\mathbf{T}a$ or $\mathbf{F}a$, where a is of form $p(c_1, \dots, c_\ell)$ with predicate symbol p and constant symbols $c_1, \dots, c_\ell$ from a finite set $\mathcal{C}$, abbreviated as $p(\mathbf{c})$; we write $c \in \mathbf{c}$ if $c = c_i$ for some $1 \leq i \leq \ell$. For $\sigma \in \{\mathbf{T}, \mathbf{F}\}$ let $\overline{\sigma} = \mathbf{T}$ if $\sigma = \mathbf{F}$ and $\overline{\sigma} = \mathbf{F}$ if $\sigma = \mathbf{T}$, and for a literal $L = \sigma a$ let $\overline{L} = \overline{\sigma} a$. An *assignment* over the (finite) set $\mathcal{A}$ of atoms is a set $\mathbf{A}$ of literals s.t. for all $a \in \mathcal{A}$, $\mathbf{T}a \in \mathbf{A}$ iff $\mathbf{F}a \notin \mathbf{A}$; here $\mathbf{T}a \in \mathbf{A}$ expresses that a is true and $\mathbf{F}a \in \mathbf{A}$ that a is false.

A *nogood* is a set $\{L_1, \dots, L_n\}$ of literals $L_i, 1 \leq i \leq n$ of type $\mathbf{T}a$ or $\mathbf{F}a$. An assignment $\mathbf{A}$ is a *solution* to a nogood δ resp. a set of nogoods Δ, if $\delta \not\subseteq \mathbf{A}$ resp. $\delta \not\subseteq \mathbf{A}$ for all $\delta \in \Delta$.

HEX-Programs. We briefly recall HEX-programs, which generalize (disjunctive) logic programs under the answer set semantics [Gelfond and Lifschitz, 1991]; for more details and background, see Eiter *et al.* [2005, 2014a].

Syntax. HEX-programs extend ordinary ASP programs by *external atoms*, which enable a bidirectional interaction between a program and external sources of computation. A *ground external atom* is of the form $\&g[\mathbf{p}](\mathbf{c})$, where $\mathbf{p} = p_1, \dots, p_k$ is a list of input parameters (predicate names or object constants), called *input list*, and $\mathbf{c} = c_1, \dots, c_l$ are constant output terms. We restrict our theoretical investigation to ground programs as safety conditions allow for applying a grounding procedure.

Definition 1. *A ground* HEX*-program* Π *consists of rules*

$$a_1 \vee \cdots \vee a_k \leftarrow b_1, \dots, b_m, \text{not } b_{m+1}, \dots, \text{not } b_n$$

where each a_i is a ground atom and each b_j is either an ordinary ground atom or a ground external atom.

The *body* of r is $B(r) = \{b_1, \dots, b_m, \text{not } b_{m+1}, \dots, \text{not } b_n\}$.

Semantics. In the following, assignments are over the set $A(\Pi)$ of ordinary atoms that occur in the program Π at hand. The semantics of a ground external atom $\&g[\mathbf{p}](\mathbf{c})$ with k input and l output parameters wrt. an assignment $\mathbf{A}$ is given by the value of a $1{+}k{+}l$-ary *two-valued (Boolean) oracle function* $f_{\&g}$ that is defined for all possible values of $\mathbf{A}$, $\mathbf{p}$ and $\mathbf{c}$. Thus, $\&g[\mathbf{p}](\mathbf{c})$ is true relative to $\mathbf{A}$ iff $f_{\&g}(\mathbf{A}, \mathbf{p}, \mathbf{c}) = \mathbf{T}$. Satisfaction of ordinary rules and ASP programs [Gelfond and Lifschitz, 1991] is then extended to HEX-rules and programs in the obvious way.

The answer sets of a HEX-program Π are defined as follows. Let the *FLP-reduct* of Π wrt. an assignment $\mathbf{A}$ be the set $f\Pi^{\mathbf{A}} = \{r \in \Pi \mid \mathbf{A} \models b, \text{ for all } b \in B(r)\}$ of all rules whose body is satisfied by $\mathbf{A}$, and let for assignments $\mathbf{A}_1$, $\mathbf{A}_2$ denote $\mathbf{A}_1 \leq \mathbf{A}_2$ that $\{\mathbf{T}a \in \mathbf{A}_1\} \subseteq \{\mathbf{T}a \in \mathbf{A}_2\}$. Then:

Definition 2. *An assignment $\mathbf{A}$ is an answer set of a* HEX*-program Π, if $\mathbf{A}$ is a $\leq$-minimal model of $f\Pi^{\mathbf{A}}$.*[1]

[1]For ordinary Π, these are Gelfond & Lifschitz's answer sets.

Example 1. *Consider the program $\Pi = \{p \leftarrow \&id[p]()\}$, where $\&id[p]()$ is true iff p is true. Then Π has the answer set $\mathbf{A}_1 = \emptyset$; indeed it is a $\leq$-minimal model of $f\Pi^{\mathbf{A}_1} = \emptyset$.*

Evaluation. For evaluation, HEX-programs Π can be transformed to ordinary ASP programs as follows. Each external atom $\&g[\mathbf{p}](\mathbf{c})$ in Π is replaced by an ordinary *replacement atom* $e_{\&g[\mathbf{p}]}(\mathbf{c})$ and a rule $e_{\&g[\mathbf{p}]}(\mathbf{c}) \vee ne_{\&g[\mathbf{p}]}(\mathbf{c}) \leftarrow$ is added. The answer sets of the resulting *guessing program* $\hat{\Pi}$ are computed by an ASP solver. But the assignment encoded by such an answer set may not satisfy Π, as $f_{\&g}$ may evaluate $\&g[\mathbf{p}](\mathbf{c})$ different from the guess for $e_{\&g[\mathbf{p}]}(\mathbf{c})$. Thus, the answer set is merely a *model candidate*; if a check against the external sources finds no discrepancy, it is a *compatible set*. Formally:

Definition 3. *A* compatible set *of a program Π is an answer set $\hat{\mathbf{A}}$ of the guessing program $\hat{\Pi}$ such that $f_{\&g}(\hat{\mathbf{A}}, \mathbf{p}, \mathbf{c}) = \mathbf{T}$ iff $\mathbf{T}e_{\&g[\mathbf{p}]}(\mathbf{c}) \in \hat{\mathbf{A}}$ for all external atoms $\&g[\mathbf{p}](\mathbf{c})$ in Π.*

Each answer set of Π is the projection of a compatible set to $A(\Pi)$, but not vice versa. To discard the non-answer sets, the evaluation algorithm calls an FLP check which uses unfounded sets to check minimality wrt. $f\Pi^{\mathbf{A}}$ [Eiter *et al.*, 2014a].

Example 2 (cont'd). *Reconsider $\Pi = \{\, p \leftarrow \&id[p]() \,\}$ from above. Then $\hat{\Pi} = \{p \leftarrow e_{\&id[p]}();\ e_{\&id[p]} \vee ne_{\&id[p]} \leftarrow\ \}$ has the answer sets $\mathbf{A}_1 = \emptyset$ and $\mathbf{A}_2 = \{\mathbf{T}p, \mathbf{T}e_{\&id[p]}\}$. Here $\mathbf{A}_1$ is a $\leq$-minimal model of $f\Pi^{\mathbf{A}_1} = \emptyset$, but $\mathbf{A}_2$ not of $f\Pi^{\mathbf{A}_2} = \Pi$.*

3 Extension to Partial Assignments

In this section, we extend assignments and oracle functions to partial assignments, which provide for explicitly representing that some atom is yet unassigned.

Definition 4. *A* partial assignment *over a set $\mathcal{A}$ of atoms is a set $\mathbf{A}$ of signed literals of kind $\mathbf{T}a$, $\mathbf{F}a$ and $\mathbf{U}a$ such that for all $a \in \mathcal{A}$ exactly one of $\mathbf{T}a \in \mathbf{A}$, $\mathbf{F}a \in \mathbf{A}$ or $\mathbf{U}a \in \mathbf{A}$ holds.*

Here, $\mathbf{U}a$ denotes that the atom a is unassigned. A partial assignment not containing any $\mathbf{U}a$ is a *complete assignment*.

For partial assignments $\mathbf{A}, \mathbf{A}'$ we call $\mathbf{A}'$ an *extension* of $\mathbf{A}$, denoted $\mathbf{A}' \succeq \mathbf{A}$, if $\{\mathbf{T}a \in \mathbf{A}\} \cup \{\mathbf{F}a \in \mathbf{A}\} \subseteq \mathbf{A}'$ (i.e., some unassigned atoms in $\mathbf{A}$ are flipped to true resp. false).

Next, we extend oracle functions in order to define the semantics of an external atom $\&g[\mathbf{p}](\mathbf{c})$ wrt. partial assignments.

Definition 5. *A* three-valued oracle function *$f_{\&g}$ for a ground external atom $\&g[\mathbf{p}](\mathbf{c})$ with k input and l output parameters is a $1{+}k{+}l$-ary function such that $f_{\&g}(\mathbf{A}, \mathbf{p}, \mathbf{c}) \in \{\mathbf{T}, \mathbf{F}, \mathbf{U}\}$ for a partial assignment $\mathbf{A}$ and all possible values of $\mathbf{p}$ and $\mathbf{c}$, and $f_{\&g}(\mathbf{A}, \mathbf{p}, \mathbf{c}) \neq \mathbf{U}$ if $\mathbf{A}$ is complete.*

Thus, $\&g[\mathbf{p}](\mathbf{c})$ is true, false or unassigned relative to $\mathbf{A}$, if the value of $f_{\&g}(\mathbf{A}, \mathbf{p}, \mathbf{c})$ is $\mathbf{T}$, $\mathbf{F}$ or $\mathbf{U}$, respectively.

We require that once the output of $f_{\&g}$ is assigned to true or false for some $\mathbf{A}$, the value stays the same for all extensions.

Definition 6. *A three-valued oracle function $f_{\&g}$ is* assignment-monotonic *if $f_{\&g}(\mathbf{A}, \mathbf{p}, \mathbf{c}) = X$, $X \in \{\mathbf{T}, \mathbf{F}\}$, implies $f_{\&g}(\mathbf{A}', \mathbf{p}, \mathbf{c}) = X$ for all assignments $\mathbf{A}' \succeq \mathbf{A}$.*

Assignment-monotonicity guarantees that no compatible set is lost when querying external sources on partial assignments.

Example 3. *Consider the program*

$$a(X,Y) \vee na(X,Y) \leftarrow v(X), v(Y), X \neq Y.$$
$$\leftarrow \&ge[a,2]().$$

It guesses, for a set of vertices $v(X)$, all directed irreflexive subgraphs of the complete graph. Suppose &ge checks if the number of links is greater or equal to a natural number, s.t. the constraint restricts the number of arcs. The according two-valued oracle function $f_{\&ge}(\mathbf{A}, a, 2)$ for a complete assignment $\mathbf{A}$ evaluates to true if $\mathbf{A}$ contains at least two literals $\mathbf{T}a(X,Y)$, and to false otherwise. Extending the oracle to a partial assignment $\mathbf{A}'$ is possible by defining an assignment-monotonic three-valued oracle function $f'_{\&ge}(\mathbf{A}', a, 2)$ yielding true if $|\{a(X,Y) | \mathbf{T}a(X,Y) \in \mathbf{A}'\}| \geq 2$, unassigned if $|\{a(X,Y) | \mathbf{T}a(X,Y) \in \mathbf{A}'$ or $\mathbf{U}a(X,Y) \in \mathbf{A}'\}| \geq 2$, and false otherwise.

Note that the definition of answer sets carries immediately over to programs with external atoms that use three-valued oracle functions. This is because answer sets are complete assignments and thus, the oracle function evaluates to $\mathbf{T}$ or $\mathbf{F}$.

A two-valued oracle function, however, cannot handle partial assignments and is thus *not* a special case of a three-valued oracle function that can be passed to an algorithm expecting the latter. However, we can always obtain a three- from a two-valued oracle function such that answer sets remain invariant.

Proposition 1. *For every* HEX*-program Π and external predicate &g with a two-valued oracle function $f_{\&g}(\cdot,\cdot,\cdot)$ there is an external predicate &g′ with assignment-monotonic three-valued oracle function $f_{\&g'}(\cdot,\cdot,\cdot)$ s.t. $\mathcal{AS}(\Pi) = \mathcal{AS}(\Pi')$, where Π' results from Π by replacing every &g by &g′.*

Intuitively, we construct a three-valued oracle function which coincides with the two-valued one for complete assignments, and returns $\mathbf{U}$ otherwise. Hence, Proposition 1 allows us to "wrap" two-valued oracle functions for use by our algorithms below; in the implementation this is the basis for backwards compatibility with existing external sources.

We exploit partial assignments, by extending previous evaluation algorithms. In spirit of *theory propagation* in SMT solvers [Barrett *et al.*, 2009], we use *external theory learning (ETL)*. It is related to *external behavior learning (EBL)*, which encodes observed output of external sources as nogoods [Eiter *et al.*, 2012], but our extension works over partial assignments such that external sources may drive early propagation of truth values implied by the current partial assignment.

As for EBL, we can associate with each external source a *learning-function* Λ that yields a set of nogoods $\Lambda(\&g[\mathbf{p}], \mathbf{A})$ learned from the evaluation of $\&g[\mathbf{p}]$ under an assignment $\mathbf{A}$. Learned nogoods have to be correct, i.e., they must not eliminate compatible sets. Formally, a nogood δ is *correct wrt. a program* Π, if all compatible sets of Π are solutions to δ.

We extend learning functions for partial assignments as follows. Let $\mathcal{E}$ contain all expressions $\&g[\mathbf{p}]$ that occur in Π and $\mathcal{S}$ all signed $(\mathbf{T}, \mathbf{F}, \mathbf{U})$ literals on atoms in $\hat{\Pi}$.

Definition 7. *A* (three-valued) learning function *for a* HEX*-program Π is a mapping $\Lambda : \mathcal{E} \times 2^{\mathcal{S}} \mapsto 2^{2^{\mathcal{S}}}$. It is called* correct *for Π, if all $\delta \in \Lambda(\&g[\mathbf{p}], \mathbf{A})$ are correct for Π, for all $\&g[\mathbf{p}]$ in $\mathcal{E}$ and $\mathbf{A} \in 2^{\mathcal{S}}$.*

Algorithm 1: HEX-CDNL-PA

```
Input: A program $\Pi$
Output: An answer set of $\Pi$ if there exists one and $\bot$ otherwise
Let $\hat{\Pi}$ be the guessing program of $\Pi$
$\hat{\mathbf{A}} \leftarrow \{\mathbf{U}a \mid a \in \mathcal{A}\}$ // all atoms unassigned
$\nabla \leftarrow \emptyset$ // set of dynamic nogoods
$dl \leftarrow 0$ // decision level
while true do
    $(\hat{\mathbf{A}}, \nabla) \leftarrow$ Propagation$(\hat{\Pi}, \nabla, \hat{\mathbf{A}})$                                  (a)
    if some nogood $\delta$ violated by $\hat{\mathbf{A}}$ then                                        (b)
        if $dl = 0$ then return $\bot$
        Analyze conflict, add learned nogood to $\nabla$, set $dl$ to backjump level
    else if $\hat{\mathbf{A}}$ is complete then                                                     (c)
        $\mathbf{A} \leftarrow \hat{\mathbf{A}} \cap \{\mathbf{T}a, \mathbf{F}a \mid a \in A(\hat{\Pi})\}$
        if $\hat{\mathbf{A}}$ is not compatible or $\mathbf{A}$ is not a minimal model of $f\Pi^{\mathbf{A}}$ then
            $\nabla \leftarrow \nabla \cup \{\hat{\mathbf{A}}\}$
        else if there is an unfounded set $U$ of $\hat{\Pi}$ wrt. $\hat{\mathbf{A}}$ then
            Construct violated nogood for $U$ and add it to $\nabla$
            Analyze conflict, add learned nogood to $\nabla$, set $dl$ to backjump level
        else
            return $\mathbf{A}$
    else if Heuristics evaluates $\&g[\mathbf{y}]$ and $\Lambda(\&g[\mathbf{y}], \hat{\mathbf{A}}) \not\subseteq \nabla$ then     (d)
        $\nabla \leftarrow \nabla \cup \Lambda(\&g[\mathbf{y}], \hat{\mathbf{A}})$
    else                                                                                         (e)
        Guess $\sigma a$ with $\sigma \in \{\mathbf{T}, \mathbf{F}\}$ for some variable $a$ with $\mathbf{U}a \in \hat{\mathbf{A}}$
        $dl \leftarrow dl + 1$
        $\hat{\mathbf{A}} \leftarrow (\hat{\mathbf{A}} \setminus \{\mathbf{U}a\}) \cup \{\sigma a\}$
```

Throughout the rest, we assume that learning functions are always correct for the programs at hand.

We now present a procedure for computing an answer set of a HEX-program shown in Algorithm 1. To compute multiple answer sets, we can naively add previous answer sets as constraints and call the algorithm again (cf. Gebser *et al.* [2007] for more elaborated techniques). The basic structure of Algorithm 1 resembles an ordinary ASP solver, but has additional checks in Part (c) and external calls to learn further nogoods in Part (d), which is based on partial assignments. Without the extensions, it computes an answer set $\hat{\mathbf{A}}$ of the guessing program $\hat{\Pi}$ and returns $\hat{\mathbf{A}}$'s projection to the atoms in Π (cf. Drescher *et al.* [2008]). To this end, it starts from a void assignment and does unit propagation in Part (a) to derive further truth values. Part (b) backtracks and learns nogoods from conflicts; Part (c) without the first **if** (i.e., starting at the **elsif**-block) checks minimality wrt. the reduct of $\hat{\Pi}$. If no further truth values are set and the assignment is incomplete, it guesses in Part (e).

The **if**-block in Part (c) checks if $\hat{\mathbf{A}}$ is a compatible set (cf. Definition 3) and if $\hat{\mathbf{A}}$'s projection to $A(\Pi)$, i.e. $\mathbf{A}$, is a minimal model of the FLP-reduct of Π wrt. $\mathbf{A}$. If both conditions are satisfied, $\mathbf{A}$ is an answer set of Π (cf. Definition 2).

The additional calls to external sources and nogood learning in Part (d) are not mandatory but prune the search space; they may eliminate assignments violating known behavior of external sources already early in the search, while correctness of learning functions guarantees that no compatible set (hence no answer set) is eliminated. Notably and in contrast to previous algorithms [Eiter *et al.*, 2012], external atoms are evaluated under partial assignments and use a three-valued oracle function. We can show that this algorithm is sound and complete:

Theorem 1. *If Algorithm 1 returns for* Π *(i) an assignment* $\mathbf{A}$*, then* $\mathbf{A}$ *is an answer set of* Π *(ii)* $\bot$*, then* Π *is inconsistent.*

4 Nogood Learning with Partial Assignments

In this section, we first discuss the generation of nogoods which encode parts of the semantics of external atoms. In contrast to previous work on external behavior learning (EBL), this works for partial assignments. When certain ground instances of an external atom can already be decided, nogoods can be learned early on and incompatible assignments can be identified; thus, they can guide the solver. Afterwards, we present a method for nogood minimization (relying on partial assignments and assignment-monotonicity), and show that this task and theory-specific learning are in fact closely related.

Three-valued Learning Functions. Let us first assume that we have no further knowledge about external sources and can only observe their (partial) output under a possibly partial input. We introduce a three-valued learning function for the general case, which is in fact a lifting of the according two-valued learning function defined by Eiter *et al.* [2012].

An *input-output* (*io-*)*nogood* is any nogood of form $N = \{\sigma_1 a_1, \dots, \sigma_n a_n\} \cup \{\sigma_{n+1} e_{\&g[\mathbf{p}]}(\mathbf{c})\}$ where $\sigma_1, \dots, \sigma_{n+1} \in \{\mathbf{T}, \mathbf{F}\}$; we let $N_I = \{\sigma_1 a_1, \dots, \sigma_n a_n\}$ be the literals over ordinary atoms (*input part*), $N_O = \{\sigma_{n+1} e_{\&g[\mathbf{p}]}(\mathbf{c})\}$ be the replacement atom (*output part*) of N, and $\sigma(N_O) = \sigma_{n+1}$. We call N *faithful*, if $f_{\&e}(\mathbf{A}, \mathbf{p}, \mathbf{c}) = \overline{\sigma(N_O)}$ for all partial assignments $\mathbf{A} \supseteq N_I$, i.e., it resembles the semantics of the external source. We note the following property:

Proposition 2. *If* N *is a faithful io-nogood, then* N *is correct wrt. all programs.*

As for the converse, correct nogoods (wrt. a given program) may be no faithful io-nogoods (or simply no io-nogoods).

When the oracle of an external atom is evaluated, the solver can create a new nogood for the observed input-output relationship. That is, evaluating $\&g[\mathbf{p}]$ for a partial assignment $\mathbf{A}$, the solver learns, given all true and false literals of input predicates, whether the output contains $\mathbf{c}$, where $f_{\&g}(\mathbf{A}, \mathbf{p}, \mathbf{c}) \neq \mathbf{U}$.

Definition 8. *The learning function for an external predicate with input parameters* $\&g[\mathbf{p}]$ *under partial assignment* $\mathbf{A}$ *is*

$$\Lambda_u(\&g[\mathbf{p}], \mathbf{A}) = \{\mathbf{A}' \cup \{\overline{\sigma e_{\&g[\mathbf{p}]}(\mathbf{c})}\} \mid f_{\&g}(\mathbf{A}, \mathbf{p}, \mathbf{c}) = \sigma \neq \mathbf{U}\}$$

where $\mathbf{A}' = \{\sigma p(\mathbf{c}) \in \mathbf{A} \mid p \in \mathbf{p}, \sigma \neq \mathbf{U}\}$ *is the relevant part of the external atom input.*

Each respective nogood is an io-nogood by construction and as the oracle is assignment-monotonic, also faithful. Hence:

Proposition 3. *Let* $\&g[\mathbf{p}](\cdot)$ *be an external atom in a* HEX*-program* Π. *Then for all assignments* $\mathbf{A}$*, the nogoods* $\Lambda_u(\&g[\mathbf{p}], \mathbf{A})$ *in Definition 8 are correct wrt.* Π.

Example 4 (cont'd). *Assume we add facts* $v(a)$ *and* $v(b)$. *Given the partial assignment* $\mathbf{A} = \{\mathbf{T}v(a), \mathbf{T}v(b), \mathbf{F}a(a,b), \mathbf{U}a(b,a), \mathbf{T}na(a,b), \mathbf{U}na(b,a)\}$*, the learning function* $\Lambda_u(\&ge[a,2], \mathbf{A})$ *yields the single io-nogood* $\{\mathbf{F}a(a,b), \mathbf{T}e_{\&ge[a,2]}()\}$*, which is indeed faithful. For* $\mathbf{A} = \{\mathbf{T}v(a), \mathbf{T}v(b), \mathbf{T}a(a,b), \mathbf{U}a(b,a), \mathbf{F}na(a,b), \mathbf{U}na(b,a)\}$*, no io-nogood can be learned and* $\Lambda_u(\&ge[a,2], \mathbf{A})$ *returns the empty set since* $f'_{\&ge}(\mathbf{A}', a, 2) = \mathbf{U}$.

We can refine a three-valued Λ_u similar to two-valued learning functions, cf. Eiter *et al.* [2012], and tailor it to external sources with specific properties. E.g., if an input predicate p_i is monotonic, we can drop false atoms over p_i from io-nogoods.

The general learning function Λ_u generates nogoods depending on the oracle function given a certain input. However, an external source provider usually knows the source semantics better and can thus provide better nogoods. The latter might include only the necessary atoms in the input; they are thus smaller and prune more of the search space. In such cases, it makes sense to provide custom learning functions $\Lambda_l(\&g[\mathbf{p}], \mathbf{A})$ which generate for $\&g[\mathbf{p}]$ and a (possibly partial) assignment $\mathbf{A}$ a set of nogoods.

Nogood Minimization. By exploiting three-valued semantics, we can also eliminate irrelevant (input) atoms from the nogoods in Λ_u, while faithfulness of io-nogoods is retained.

Definition 9. *Given a faithful io-nogood* N *with* $N_O = \{\sigma e_{\&g[\mathbf{p}]}(\mathbf{c})\}$*, the minimized nogoods of* N *are*

$$\begin{aligned} \mathit{minimize}(N) = \{N' \subseteq N \mid N' \text{ is a faithful io-nogood}, \\ f_{\&g}(N'', \mathbf{p}, \mathbf{c}) = \mathbf{U} \text{ for all } N'' \subsetneq N'_I\}. \end{aligned}$$

This extends to sets S of nogoods by $\mathit{minimize}(S) = \bigcup_{N \in S} \mathit{minimize}(N)$. Note that exponentially many minimal nogoods in the size of N are possible.

Example 5 (cont'd). *For the assignment* $\mathbf{A} = \{\mathbf{T}v(a), \mathbf{T}v(b), \mathbf{F}a(a,b), \mathbf{F}a(b,a), \mathbf{T}na(a,b), \mathbf{T}na(b,a)\}$ *and the faithful io-nogood* $N = \{\mathbf{F}a(a,b), \mathbf{F}a(b,a), \mathbf{T}e_{\&ge[a,2]}()\} \in \Lambda_u(\&ge[a,2], \mathbf{A})$*, we obtain* $\mathit{minimize}(N) = \{\{\mathbf{F}a(a,b), \mathbf{T}e_{\&ge[a,2]}()\}, \{\mathbf{F}a(b,a), \mathbf{T}e_{\&ge[a,2]}()\}\}$.

The minimized nogoods subsume all faithful io-nogoods.

Proposition 4. *Let* $\mathbf{A}$ *be a partial assignment and* N *be a faithful io-nogood for* $\&g[\mathbf{p}]$ *over the atoms in* $\mathbf{A}$. *Then some* $N' \in \mathit{minimize}(\Lambda_u(\&g[\mathbf{p}], \mathbf{A}))$ *exists s.t.* $N' \subseteq N$.

As a subset of each faithful io-nogood occurs among all minimized nogoods, no further faithful io-nogoods prune the search space more effectively. Still there might be further correct nogoods (non-io ones and/or depending on the program).

In the following, we call a theory-specific learning function $\Lambda_l(\cdot, \cdot)$ *complete* for an external source $\&g$, if for each partial assignments $\mathbf{A}' \subseteq \mathbf{A}$, input lists $\mathbf{p}$, and output lists $\mathbf{c}$ the set $\Lambda_l(\&g[\mathbf{p}], \mathbf{A})$ is the least set such that $f_{\&g}(\mathbf{A}', \mathbf{p}, \mathbf{c}) = \sigma$ with $\sigma \in \{\mathbf{T}, \mathbf{F}\}$ implies $\mathbf{A}' \cup \{\overline{\sigma e_{\&g[\mathbf{p}]}(\mathbf{c})}\}$ is in $\Lambda_l(\&g[\mathbf{p}], \mathbf{A})$; we call it *partial* otherwise. That is, Λ_l learns *all and only* io-nogoods with a premise over the current partial assignment which resemble the semantics of $\&g$.

As it turns out, learning using complete theory-specific learning functions and nogood minimization are closely related. Let $\mathit{min}_{\subseteq}(S) = \{N \in S \mid \nexists N' \in S \text{ s.t. } N' \subsetneq N\}$ be the restriction of S to subset-minimal nogoods.[2] Then:

Proposition 5. *Let* Λ_l *be a complete theory-specific learning function for an external source* $\&g$. *Then for all partial assignments* $\mathbf{A}$ *and input lists* $\mathbf{p}$ *it holds that* $\mathit{minimize}(\Lambda_u(\&g[\mathbf{p}], \mathbf{A})) = \mathit{min}_{\subseteq}(\Lambda_l(\&g[\mathbf{p}], \mathbf{A}))$.

[2]Despite similar names, $\mathit{minimize}$ differs from $\mathit{min}_{\subseteq}$ as it minimizes wrt. oracle results while $\mathit{min}_{\subseteq}$ just selects the minimal sets.

Algorithm 2: Simultaneous Nogood Minimization

```
Input: A set $S$ of faithful io-nogoods $N$ with $N_I = \{\sigma_1 a_1, \dots, \sigma_n a_n\}$
Output: A set of minimal faithful io-nogoods

$ch \leftarrow \emptyset$                                   // cache for oracle calls
for each signed literal $\sigma_i a_i \in N_I$ do
    for each io-nogood $N' \in S$ with $N'_O = \{\sigma_{n+1} e_{\&g[\mathbf{p}]}(\mathbf{c})\}$ do          (a)
        $N^s \leftarrow N'_I \setminus \{\sigma_i a_i\}$                 // smaller oracle input
        if $\langle N^s, \cdot \rangle \notin ch$ then                                              (b)
            $ch \leftarrow ch \cup \{\langle N^s, \{\sigma e_{\&g[\mathbf{p}]}(\mathbf{c}') \mid f_{\&g}(N^s, \mathbf{p}, \mathbf{c}') = \sigma \neq \mathbf{U}\}\rangle\}$
        if $\overline{\sigma_{n+1}} e_{\&g[\mathbf{p}]}(\mathbf{c}) \in output$ for $\langle N^s, output \rangle \in ch$ then      (c)
            Replace $N'$ by $N^s \cup \{\sigma_{n+1} e_{\&g[\mathbf{p}]}(\mathbf{c})\}$ in $S$
return $S$
```

This proposition implies that we have alternative techniques to learn all nogoods that prune the search space in an optimal (cf. Proposition 4) way. As above, it considers only *faithful io-nogoods* while further correct nogoods may exist. Notably, the equality holds only under the premises of *exhaustive* minimization in the first case and a *complete* theory-specific learning function in the second; otherwise different sets of nogoods may be produced. As both operations are expensive and impractical, it makes sense to support both (incomplete) minimization and (incomplete) theory-specific learning functions.

In practice, we use Algorithm 2 to compute only one minimal io-nogood for each learned io-nogood. Instead of minimizing each nogood separately and to avoid redundant queries, we proceed in parallel and use a cache for the external atom output of a set S of io-nogoods with identical input but different outputs. The algorithm works by sequentially removing the same literal simultaneously from the premises of all N in S in Part (a), and checking whether the output for the resulting premises is already in the cache, in Part (b). If not, all outputs $\mathbf{c}'$ s.t. $f_{\&g}(\mathbf{A}, \mathbf{p}, \mathbf{c}') \neq \mathbf{U}$ are computed (this is a single call in the implementation) and stored in the cache. Otherwise, no external source call is needed. It is then checked if the resulting nogood is still faithful in Part (c), and N is replaced by its reduced equivalent in S in this case. Formally:

Proposition 6. *For a set S of faithful io-nogoods with equal input parts and distinct output parts, Algorithm 2 yields exactly one faithful io-nogood $N' \in minimize(N)$ for each $N \in S$.*

5 Implementation and Evaluation

For experimentation, we integrated our techniques into the reasoner DLVHEX with GRINGO 4.4.0 and CLASP 3.1.1 as backends. All benchmarks were run on a Linux machine with two 12-core AMD Opteron 6176 SE CPUs and 128 GB RAM; timeout was 300 secs and memout 8 GB per instance. We used the *HTCondor* load distribution system (http://research.cs.wisc.edu/htcondor) to ensure robust runtimes (i.e., deviations of runs on the same instance are negligible). Average runtimes of 50 instances per problem size for computing all answer sets are reported in seconds,[3] with timeouts in parentheses, and the average number of solutions, where '$\geq$' respects timeouts.

Benchmark Configurations. We used learning function Λ_u and compared 5 configurations. We first tested 3 heuristics

# variables	**never**	**periodic**	**always**	**ngm**	**ngmc**	solutions
4	0.11 (0)	0.12 (0)	0.12 (0)	0.12 (0)	0.12 (0)	2.26
8	0.31 (0)	0.30 (0)	0.21 (0)	0.22 (0)	0.21 (0)	4.34
12	4.99 (0)	3.80 (0)	0.74 (0)	0.50 (0)	0.42 (0)	7.30
16	300.00 (50)	66.01 (0)	3.78 (0)	1.29 (0)	0.95 (0)	18.72
20	300.00 (50)	300.00 (50)	16.64 (0)	2.60 (0)	1.90 (0)	25.60
24	300.00 (50)	300.00 (50)	84.48 (3)	8.04 (0)	3.78 (0)	108.40
28	300.00 (50)	300.00 (50)	223.67 (23)	13.77 (0)	6.19 (0)	135.12
32	300.00 (50)	300.00 (50)	290.68 (47)	24.27 (1)	9.74 (0)	205.14
36	300.00 (50)	300.00 (50)	296.95 (49)	48.66 (2)	15.81 (0)	542.54

Table 1: Random PB Problems with 4 to 36 variables

PBC length	**never**	**periodic**	**always**	**ngm**	**ngmc**	solutions
4	94.62 (0)	27.32 (0)	0.39 (0)	0.30 (0)	0.30 (0)	0.00
6	107.57 (0)	38.67 (0)	10.27 (0)	7.76 (0)	2.03 (0)	317.40
8	103.66 (0)	55.42 (0)	66.49 (0)	156.22 (0)	14.11 (0)	6366.98
10	84.09 (0)	73.37 (0)	141.50 (0)	300.00 (50)	40.73 (0)	18662.86
12	85.52 (0)	92.82 (0)	211.89 (0)	300.00 (50)	68.00 (0)	27067.48
14	90.49 (0)	105.28 (0)	252.78 (0)	300.00 (50)	84.65 (0)	30721.18
16	97.61 (0)	116.34 (0)	286.11 (0)	300.00 (50)	96.74 (0)	32077.92
18	109.20 (0)	130.75 (0)	300.00 (50)	300.00 (50)	109.20 (0)	32554.34
20	116.63 (0)	139.42 (0)	300.00 (50)	300.00 (50)	117.06 (0)	32713.66

Table 2: Random PB Problems with PBC length of 4 to 20

for additional external source calls (cf. Algorithm 1, Part (d)) without nogood minimization, viz. **never** (no additional calls, i.e. DLVHEX without the new techniques), **periodic** (partial evaluation at each 10th heuristics call), and **always** (partial evaluation at every heuristics call). We then tested nogood minimization instead of additional calls (i.e., only for complete assignments): minimizing *all* nogoods (**ngm**) resp. the currently *conflicting* ones (**ngmc**). We omit results for minimization combined with **periodic** or **always**, which was always significantly slower than some other configuration (due to many more external calls with little gain).

Our hypothesis was that **periodic** and **always** decrease the runtime over **never** if useful information is obtainable by early evaluation, and increase it otherwise. We expected the tradeoff to be mitigated by just minimizing nogoods on complete assignments using **ngm** instead of evaluating early. We further expected that restricting the minimization to nogoods that directly trigger backjumping, as tested in **ngmc**, is even more effective due to reduced overhead. It was expected that partial evaluation performs better for instances with a small percentage of solutions wrt. the assignments in the search space since a large percentage leaves less room for pruning it.

Pseudo-Boolean Problems. *Pseudo-boolean problems* constitute sets of *pseudo-boolean constraints* (*PBCs*) of the form $C_0 p_0 + \dots + C_{n-1} p_{n-1} \geq C_n$, where all p_i are literals and all C_i are integers [Eén and Sörensson, 2006]. Our implementation guesses an interpretation of the p_i by disjunctive rules and has a constraint $\leftarrow$ not $\&pbCheck[p, pbInst]()$ which is satisfied wrt. a complete assignment to p iff the latter represents a solution for the PBCs. The strict separation of the guess and the check part results in benchmark instances that are well-suited for investigating the effect of a tighter integration of the solving algorithm and the evaluation of external constraints.[4] We extend the semantics of the external atom to

[3]Results for computing the first answer sets are comparable.

[4]Note that for the purpose of solving PBCs as part of a HEX-program (possibly in combination with other external sources), the external source could directly interface a dedicated PB solver.

#	**never**		**periodic**		**always**		**ngm**		**ngmc**		**ngm-sq**		solutions
4	0.19	(0)	0.19	(0)	0.23	(0)	0.41	(0)	0.18	(0)	0.92	(0)	6.68
6	0.35	(0)	0.28	(0)	0.41	(0)	5.47	(0)	0.27	(0)	26.13	(0)	16.64
8	1.50	(0)	0.67	(0)	1.06	(0)	98.66	(10)	0.59	(0)	203.65	(26)	53.28
10	13.05	(0)	1.18	(0)	1.88	(0)	204.46	(24)	1.17	(0)	297.41	(48)	121.28
12	213.17	(13)	6.45	(0)	11.35	(0)	294.01	(49)	6.63	(0)	294.05	(49)	334.72
14	300.00	(50)	19.46	(0)	28.25	(1)	294.02	(49)	20.90	(0)	294.32	(49)	821.76
16	300.00	(50)	25.31	(0)	38.79	(0)	294.03	(49)	28.17	(0)	294.54	(49)	867.84
18	300.00	(50)	74.13	(7)	94.91	(7)	294.05	(49)	86.40	(7)	294.84	(49)	$\geq$2371.26
20	300.00	(50)	142.66	(13)	174.44	(18)	300.00	(50)	180.63	(16)	300.00	(50)	$\geq$3918.82

Table 3: Taxi Assignment Results

#	**never**		**periodic**		**always**		**ngm**		**ngmc**		solutions
10	0.15	(0)	0.16	(0)	0.17	(0)	0.15	(0)	0.15	(0)	3.58
15	0.28	(0)	0.28	(0)	0.30	(0)	0.24	(0)	0.24	(0)	8.44
20	0.88	(0)	0.75	(0)	0.75	(0)	0.52	(0)	0.49	(0)	19.18
25	4.28	(0)	2.60	(0)	2.36	(0)	1.49	(0)	1.36	(0)	55.96
30	24.65	(0)	9.07	(0)	6.77	(0)	3.72	(0)	3.33	(0)	118.94
35	147.77	(0)	32.89	(0)	20.49	(0)	9.78	(0)	8.64	(0)	260.56
40	300.00	(50)	122.03	(0)	50.77	(0)	24.76	(0)	21.57	(0)	553.04
45	300.00	(50)	294.57	(45)	157.35	(8)	78.50	(0)	68.03	(0)	1490.78
50	300.00	(50)	300.00	(50)	261.09	(32)	170.16	(17)	158.21	(12)	$\geq$2850.36

Table 4: Conflicting Strategic Companies Results

$$
\begin{aligned}
\mathit{drives}(X,Y) &\leftarrow \mathit{driver}(X), \mathit{customer}(Y), \mathit{DL}[;\mathit{isIn}](X,A),\\
&\qquad \mathit{DL}[;\mathit{isIn}](Y,A), \mathit{region}(A), \text{not } \mathit{ndrives}(X,Y).\\
\mathit{ndrives}(X,Y) &\leftarrow \mathit{driver}(X), \mathit{customer}(Y), \text{not } \mathit{drives}(X,Y).\\
&\leftarrow \mathit{drives}(X,Y), \mathit{drives}(X1,Y), X \neq X1.\\
\mathit{drivesECust}(X,Y) &\leftarrow \mathit{drives}(X,Y),\\
&\qquad \mathit{DL}[\mathit{eDrives} \uplus \mathit{drivesECust}; \mathit{ECust}](Y).\\
\mathit{driven}(Y) &\leftarrow \mathit{drives}(\_,Y).
\end{aligned}
$$

$$
\begin{aligned}
&\leftarrow \text{not } \mathit{driven}(Y), \mathit{customer}(Y).\\
&\leftarrow \#\mathit{count}\{Y : \mathit{drives}(X,Y)\} > 4, \mathit{driver}(X).\\
&\leftarrow \mathit{drives}(X,Y), \text{not } \mathit{DL}[\mathit{eDrives} \uplus \mathit{drivesECust}; \mathit{ECust}](Y),\\
&\quad \mathit{DL}[\mathit{eDrives} \uplus \mathit{drivesECust}; \mathit{EDriver}](X).\\
&\leftarrow \mathit{drives}(X,Y), \mathit{DL}[\mathit{eDrives} \uplus \mathit{drivesECust}; \mathit{ECust}](Y),\\
&\quad \text{not } \mathit{DL}[\mathit{eDrives} \uplus \mathit{drivesECust}; \mathit{EDriver}](X).
\end{aligned}
$$

Figure 1: Taxi Assignment Rules

partial assignments $\mathbf{A}$ as follows: *&pbCheck*$[p, pbInst]()$ is true wrt. $\mathbf{A}$ if every input PBC fulfills $\sum_{\mathbf{T}p_i \in \mathbf{A}} C_i p_i \geq C_n$; it is false, if some input PBC fulfills $\sum_{\mathbf{F}p_i \notin \mathbf{A}} C_i p_i < C_n$; it is unassigned otherwise.

First, we tested randomly generated problems with $N \in [4, 36]$ variables and $4 \times N$ PBCs with $n = 6$ and $C_i \in [1, 5]$ for $0 \leq i \leq n$ (Table 1). The ratio between N and the number of constraints ensures that only a small fraction of all assignments are answer sets. A clear improvement over **never** is observed whenever partial evaluation is used. Without nogood minimization, **always** shows the best performance, with **periodic** falling in-between **always** and **never**, hence learning the io-behavior of the external source as early as possible outweighs the runtime overhead for querying it additionally. Overall, **ngmc** shows the best performance.

Second, to investigate the behavior when large parts of the search space contain solutions, we fixed the number of variables and PBCs to 15 and 60, resp., and tested different lengths $n \in [4, 20]$ (Table 2). The solution count increases with length, and for $n > 14$ nearly all assignments are answer sets. Expectedly, **periodic** and **always** are slower than **never** if many (more than about half of) the candidates are solutions. Frequent evaluation is detrimental here, as runtime investment has no pay-off in information gain or early search termination. Likewise, minimizing all io-nogoods is worse as identical nogoods are computed for many complete assignments; **ngmc**, however, is very efficient with only a small runtime overhead, because it focuses on valuable (i.e. conflicting) io-nogoods.

Taxi Assignment. We use a HEX-program to assign taxi drivers to customers under constraints. Naturally, a customer and her driver must be in the same region; customers may share the driver, and a taxi fits at most 4 customers. A description logic (DL) ontology has information such as locations of individuals, e-customers (customers demanding electric cars), and e-drivers (drivers of electric cars); e-customers must be assigned to e-drivers, and normal customers to normal drivers.

The answer sets of the program with facts `driver(d)`, `customer(c)` and `region(r)` for drivers *d*, customers *c* and regions *r*, and the rules in Figure 1 encode legal assignments. The ontology is queried using *DL-atoms* [Eiter *et al.*, 2008], which offer a more tailored syntax for specific external atoms; e.g. $\mathit{DL}[\mathit{eDrives} \uplus \mathit{drivesECust}; \mathit{EDriver}](X)$ retrieves all individuals in the class $\mathit{EDriver}$, where "$\mathit{eDrives} \uplus \mathit{drivesECust}$" enhances the role $\mathit{eDrives}$ with the true atoms of $\mathit{drivesECust}$ before evaluation. DL-atoms are implemented in the *DL-Lite* plug-in for DLVHEX [Eiter *et al.*, 2014b]. Exploiting monotonicity of DLs, the three-valued evaluation under a partial assignment $\mathbf{A}$ is as follows: it returns true, if the query is true with the minimal addition of assertions (in the example $\{\mathit{eDriver}(d, d') \mid \mathbf{T}\,\mathit{drivesECust}(d, d') \in \mathbf{A}\}$); false, if the query is not true with the maximal possible addition ($\{\mathit{eDriver}(d, d') \mid \mathbf{F}\,\mathit{drivesECust}(d, d') \notin \mathbf{A}\}$); and unassigned otherwise.

In our tests, we increased the number N of drivers and customers from 4 to 20, which were put in $N/2$ regions randomly, with the drivers balanced among regions; half of the customers were e-customers (Table 3). As DL-atoms have output constants, we could also compare simultaneous and sequential nogood minimization (**ngm-sq**). The results for partial evaluation using **periodic**, **always** and **ngmc** are mixed and there is no clear winner; however, all of them are significantly faster than **never**. Indeed, the external DL calls are costly, and waiting a bit until the next one can pay off. As the premise of an io-nogood can be large but the output often depends only on a small part, minimization can drastically shrink io-nogoods. However, the cost is many external calls, which can be reduced by minimizing nogoods with same premise simultaneously; in total, a significant speedup results.

Conflicting Strategic Companies. *Strategic Companies* is a popular problem in ASP. We use the encoding schema of Leone *et al.* [2006], with an additional constraint that checks for *conflicts* on strategic sets via the external atom *&conflict*$[s]()$, e.g. imposing legislative restrictions. On complete assignments, it is true if companies c_i, c_j in s are related by an external conflict relation $R \subseteq C \times C$, and false otherwise. We use a three-valued oracle function that returns true for a partial assignment $\mathbf{A}$ if $\mathbf{T}s(c_i), \mathbf{T}s(c_j) \in \mathbf{A}$ holds for some $(c_i, c_j) \in R$, false if $\mathbf{F}s(c_i) \in \mathbf{A}$ or $\mathbf{F}s(c_j) \in \mathbf{A}$ for every $(c_i, c_j) \in R$, and unassigned otherwise.

Since finding a strategic set is computationally hard, excluding candidate strategic sets with a conflict early in the search by partial evaluations should noticeably decrease the runtime.

We ran tests on instances with $N \in [10, 50]$ companies, at most N randomly assigned control relations, $5 \times N$ products with randomly assigned producers, and $N/2$ randomly created conflicts (Table 4); the latter cut more than 90% of the strategic sets (i.e., solution candidates). Partial evaluation always decreases the runtime, where $\mathbf{ngmc}$ shows the best results. Notably, for computing strategic sets containing a specific company (which is $\mathbf{\Sigma_2^P}$-hard) we obtain similar results.

Instances and details on the experiments can be found at http://www.kr.tuwien.ac.at/research/projects/inthex/partialevaluation.

6 Discussion and Conclusion

Related Work. Our work is most closely related to constraint ASP solving [Gebser *et al.*, 2009] and SMT (in particular theory propagation) [Nieuwenhuis *et al.*, 2006]. However, our approach is more general as it supports arbitrary external sources, even if they may be semantically black boxes; thus, the context for learning also differs. While theory literals and constraint atoms are directly related via a theory and shared constraint variables, respectively, external atoms are only indirectly related via their input. Hence, ETL is realized by learning io-relations in our case, while values of theory literals and constraint atoms can be propagated within the solver. Also Ostrowski and Schaub [2012] considered nogood minimization, using different algorithms that avoid expensive resets of the constraint solver. In our more general setting, this is inapplicable, while other issues arise; e.g., that external atoms can have multiple output values motivates our algorithm for simultaneous io-nogood minimization. Overall, the mentioned approaches do not lend themselves to a direct comparison.

Antic *et al.* [2013] considered partial HEX-semantics before, employing *Approximation Fixpoint Theory* (*AFT*). Although, we only consider two-valued answer sets and do not apply a fixpoint construction, our partial oracle functions coincide with their three-valued ones. Similarly, Pelov *et al.* [2004] have defined a family of partial stable model semantics for logic programs with aggregates using AFT. Assignment-monotonic oracle functions are also related to their approximating aggregate relations which must be *precision-monotone* and generalize ordinary aggregate relations to a three-valued semantics.

Typical integration schemas for SMT have been identified [Balduccini and Lierler, 2013b], which are also applicable to ASP modulo theories; a comparison is given in [Balduccini and Lierler, 2013a]. In *black-box* integration, the SAT solver blindly generates a model and passes it for checking to the theory solver. If it passes the check, it is returned, otherwise it is added as a constraint to the instance and the solver restarts. This allows for easy coupling with arbitrary theories but does not enable search space pruning. In *grey-box* integration, the theory solver is only called for complete models of the SAT instance, but the SAT solver is merely suspended during checking and can continue its search afterwards; integration is still relatively simple. Only in *clear-box* integration, the SAT solver is interleaved with the theory solver, which is called already for partial assignments and in turn may propagate further truth values or detect inconsistencies. However, the integration is much more challenging as the theory solver must identify atoms implied by the given partial assignment, or inconsistency reasons, respectively. Applied to HEX, the grey-box schema corresponds to the algorithms before EBL [Eiter *et al.*, 2012] was introduced (black-box integration, i.e., complete restarts, was never used). With EBL, the algorithms followed an intermediate schema between grey- and clear-box: external sources were still only evaluated under complete assignments but the learned nogoods possibly pruned the search space.

Conclusion. The techniques introduced in this paper yield full-fledged clear-box integration. Moreover, due to automatic nogood minimization, developers of external sources do not need to manually describe implied truth values or inconsistency reasons, but only need to implement a three-valued oracle function, which keeps the integration of sources simple.

In our experiments, the new techniques yield a speedup of up to two orders of magnitude; unsurprisingly, their ranking depends on the instances. This is similar to observations by Ostrowski and Schaub [2012], who reported mixed results for different propagation delays. Our results are also in line with results in SMT, where theory propagation, if doable with small overhead, is crucial for performance [Dutertre and de Moura, 2006; Lahiri *et al.*, 2006; Nieuwenhuis and Oliveras, 2005]. We observed that in most cases learning from complete assignments plus minimization of conflicting nogoods (based on partial assignments) outperforms learning during search; hence, this setting is suggestive as a default. This is explained by the fact that in this case, learning focuses on nogoods that are useful for conflict resolution, thus the information gain is similar and the overhead much smaller. This matches the observation by Nieuwenhuis *et al.* [2006] that conflict analysis uses only a small fraction of the lemmas learned by theory propagation, which can be addressed with *lazy explanations* [Gent *et al.*, 2010]. The speedup can be up to exponential, as evidenced by an external atom whose truth value is definite after assigning a single input atom, e.g. $\&empty[p]()$ to check if an atom over p is true. Each naive nogood eliminates one of exponentially many assignments, but a linear number of minimized ones eliminates all wrong guesses.

Although our minimization algorithm cannot be improved in the worst case (the nogood is already minimal), more sophisticated algorithms for computing minimal conflicts have been considered in the literature, which are likely to improve the performance when nogoods comprise many irrelevant literals. E.g., the QUICKXPLAIN algorithm [Junker, 2004] employs a divide-and-conquer strategy to find relaxations of overconstrained problems, which allows pruning parts of a binary search tree not containing relevant constraints. Based on this, Shchekotykhin *et al.* [2015] developed the MERGEXPLAIN algorithm, capable of efficiently determining multiple minimal conflicts during one problem analysis run, which could be integrated into our approach for obtaining all minimal io-nogoods.

Topics for ongoing and future work include further heuristics for external evaluation (e.g., evaluating or skipping based on the past information gain), further nogood-minimization strategies using knowledge about external sources (e.g., functionality or convexity), and exploiting partial assignments also in the minimality check of answer set candidates.

References

[Alviano *et al.*, 2015] Mario Alviano, Wolfgang Faber, and Martin Gebser. Rewriting recursive aggregates in answer set programming: back to monotonicity. *Theory and Practice of Logic Programming (TPLP)*, 15(4-5):559–573, 2015.

[Antic *et al.*, 2013] Christian Antic, Thomas Eiter, and Michael Fink. Hex semantics via approximation fixpoint theory. In Pedro Cabalar and Tran Cao Son, editors, *Logic Programming and Nonmonotonic Reasoning, LPNMR 2013*, volume 8148 of *LNCS*, pages 102–115. Springer, 2013.

[Balduccini and Lierler, 2013a] Marcello Balduccini and Yuliya Lierler. Hybrid automated reasoning tools: from black-box to clear-box integration. *CoRR*, abs/1312.6105, 2013.

[Balduccini and Lierler, 2013b] Marcello Balduccini and Yuliya Lierler. Integration schemas for constraint answer set programming: a case study. *Theory and Practice of Logic Programming (TPLP)*, 13(4-5-Online-Supplement), 2013.

[Barrett *et al.*, 2009] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 825–885. IOS Press, 2009.

[Drescher *et al.*, 2008] Christian Drescher, Martin Gebser, Torsten Grote, Benjamin Kaufmann, Arne König, Max Ostrowski, and Torsten Schaub. Conflict-driven disjunctive answer set solving. In Gerhard Brewka and Jérôme Lang, editors, *Principles of Knowledge Representation and Reasoning, KR 2008*, pages 422–432. AAAI Press, 2008.

[Dutertre and de Moura, 2006] Bruno Dutertre and Leonardo Mendonça de Moura. A fast linear-arithmetic solver for DPLL(T). In Thomas Ball and Robert B. Jones, editors, *Computer Aided Verification, CAV 2006*, volume 4144 of *LNCS*, pages 81–94. Springer, 2006.

[Eén and Sörensson, 2006] Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):1–26, 2006.

[Eiter *et al.*, 2005] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *International Joint Conference on Artificial Intelligence, IJCAI 2005*, pages 90–96. Professional Book Center, 2005.

[Eiter *et al.*, 2008] Thomas Eiter, Giovambattista Ianni, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Combining answer set programming with description logics for the semantic web. *Artificial Intelligence*, 172(12-13):1495–1539, 2008.

[Eiter *et al.*, 2012] Thomas Eiter, Michael Fink, Thomas Krennwallner, and Christoph Redl. Conflict-driven ASP solving with external sources. *Theory and Practice of Logic Programming (TPLP)*, 12(4-5):659–679, 2012.

[Eiter *et al.*, 2014a] Thomas Eiter, Michael Fink, Thomas Krennwallner, Christoph Redl, and Peter Schüller. Efficient HEX-program evaluation based on unfounded sets. *Journal of Artificial Intelligence Research*, 49:269–321, February 2014.

[Eiter *et al.*, 2014b] Thomas Eiter, Michael Fink, Christoph Redl, and Daria Stepanova. Exploiting support sets for answer set programs with external evaluations. In Carla E. Brodley and Peter Stone, editors, *Conference on Artificial Intelligence, AAAI 2014*, pages 1041–1048. AAAI Press, 2014.

[Gebser *et al.*, 2007] Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. Conflict-driven answer set enumeration. In Chitta Baral, Gerhard Brewka, and John S. Schlipf, editors, *Logic Programming and Nonmonotonic Reasoning, LPNMR 2007*, volume 4483 of *LNCS*, pages 136–148. Springer, 2007.

[Gebser *et al.*, 2009] Martin Gebser, Max Ostrowski, and Torsten Schaub. Constraint answer set solving. In Patricia M. Hill and David S. Warren, editors, *International Conference on Logic Programming, ICLP 2009*, volume 5649 of *LNCS*, pages 235–249. Springer, 2009.

[Gebser *et al.*, 2012] Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, 187-188:52–89, August 2012.

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

[Gent *et al.*, 2010] Ian P. Gent, Ian Miguel, and Neil C. A. Moore. Lazy explanations for constraint propagators. In Manuel Carro and Ricardo Peña, editors, *Practical Aspects of Declarative Languages, PADL 2010*, volume 5937 of *LNCS*, pages 217–233. Springer, 2010.

[Junker, 2004] Ulrich Junker. QUICKXPLAIN: preferred explanations and relaxations for over-constrained problems. In Deborah L. McGuinness and George Ferguson, editors, *Innovative Applications of Artificial Intelligence, IAAI 2004*, pages 167–172. AAAI Press / The MIT Press, 2004.

[Lahiri *et al.*, 2006] Shuvendu K. Lahiri, Robert Nieuwenhuis, and Albert Oliveras. SMT techniques for fast predicate abstraction. In Thomas Ball and Robert B. Jones, editors, *Computer Aided Verification, CAV 2006*, volume 4144 of *LNCS*, pages 424–437. Springer, 2006.

[Leone *et al.*, 2006] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic (TOCL)*, 7(3):499–562, July 2006.

[Nieuwenhuis and Oliveras, 2005] Robert Nieuwenhuis and Albert Oliveras. DPLL(T) with exhaustive theory propagation and its application to difference logic. In Kousha Etessami and Sriram K. Rajamani, editors, *Computer Aided Verification, CAV 2005*, volume 3576 of *LNCS*, pages 321–334. Springer, 2005.

[Nieuwenhuis *et al.*, 2006] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, November 2006.

[Ostrowski and Schaub, 2012] Max Ostrowski and Torsten Schaub. ASP modulo CSP: The clingcon system. *Theory and Practice of Logic Programming (TPLP)*, 12(4-5):485–503, 2012.

[Pelov *et al.*, 2004] Nikolay Pelov, Marc Denecker, and Maurice Bruynooghe. Partial stable models for logic programs with aggregates. In Vladimir Lifschitz and Ilkka Niemelä, editors, *Logic Programming and Nonmonotonic Reasoning, LPNMR 2004*, volume 2923 of *LNCS*, pages 207–219. Springer, 2004.

[Shchekotykhin *et al.*, 2015] Kostyantyn M. Shchekotykhin, Dietmar Jannach, and Thomas Schmitz. MergeXplain: Fast computation of multiple conflicts for diagnosis. In Qiang Yang and Michael Wooldridge, editors, *International Joint Conference on Artificial Intelligence, IJCAI 2015*, pages 3221–3228. AAAI Press, 2015.