Experimental Evaluation of a State-Of-The-Art Grounder

Joachim Jansen
KU Leuven Department of Computer Science
Celestijnenlaan 200A
3001 Heverlee, België
joachim.jansen@cs.kuleuven.be

Ingmar Dasseville
KU Leuven Department of Computer Science
Celestijnenlaan 200A
3001 Heverlee, België
ingmar.dasseville@cs.kuleuven.be

Jo Devriendt
KU Leuven Department of Computer Science
Celestijnenlaan 200A
3001 Heverlee, België
jo.devriendt@cs.kuleuven.be

Gerda Janssens
KU Leuven Department of Computer Science
Celestijnenlaan 200A
3001 Heverlee, België
gerda.janssens@cs.kuleuven.be

Abstract

Many state-of-the-art declarative systems use a ground-and-solve approach, where the problem statement, expressed in a high-level language, is first grounded into a low-level representation. Next, a solver is used to search a solution for the low-level representation. In order to prevent a combinatorial blowup of the grounding, many intelligent techniques have been developed. In this paper we study in detail three such techniques (Lifted Unit Propagation, Grounding With Bounds, and Reduced Grounding) to get a better insight in their individual merits and their interactions. Our experiments take as benchmarks all the NP problems of the previous Answer Set Programming (ASP) competitions. The experiments are performed with IDP^3 and all tools needed to run them are made publicly available. The first experiment discusses the impact of the three techniques on the "efficiency" of the grounding step, and on each other. In a second set of experiments we show that a reduction in the grounding size as a result of the application of these grounding techniques *does not* reduce the search space. We give an in-depth analysis of our results and discuss what this means for the development of grounding techniques for declarative systems.

1. Introduction

Model generation is a widely used problem-solving paradigm. A problem is specified as a theory in a rich, declarative logic in such a way that models of the theory represent solutions to the problem. In this paper we focus on bounded Model Expansion (MX), where a partial input structure interpreting a finite, known domain is expanded into a total structure that satisfies a given theory. This paradigm is studied in fields such as Constraint Programming (CP), Mixed Integer Programming (MIP), Answer Set Programming (ASP), and Knowledge Representation (KR).

A state-of-the-art approach is to reduce the input theory, formulated in an expressive logic, to a theory in a fragment of the language supported by some search algorithm, while preserving a suitable form of equivalence. Afterwards, the search algorithm is applied to effectively search for models of the theory. We refer to the former reduction process as *grounding* and the latter as *search*. Abusing notation, we use 'grounding' to also refer to the outcome of the reduction process, the ground theory. This two-phase approach is commonly called ground-and-solve. Example systems using this approach are MiniZinc systems [16] which ground MiniZinc to FlatZinc, ASP systems such as Clingo [12], DLV [15], or SMODELS [17] which ground ASP programs, and IDP^3 [3, 5] which grounds FO$(\cdot)^{\text{IDP}}$ to ECNF.

As we will later argue, the main challenge of the grounding phase is to transform away quantified variables. This can be done naively by instantiating universally (existentially) quantified variables for all their possible values and concatenating the results into one large conjunction (disjunction). If done in this manner, the ground size of a formula φ is of the order D^n where D is the size of the domain and n the number of quantifiers in φ. This potential combinatorial blowup is a challenge for ground-and-solve based declarative systems. Most ASP systems approach this challenge by using semi-naive bottom-up evaluation [10, 11], a technique based on semi-naive evaluation in the field of databases [2]. IDP^3, the system studied in the experiments for this paper, uses a top-down approach for grounding.

The contributions of this paper are the following. First, it provides an introduction to three important grounding techniques. These techniques have been introduced in other papers, this paper aims to provide an intuitive explanation of these techniques, to discuss the impact they have on each other, and to help future developers with information about the implementation of these techniques. Further, we confirm claims on the effects of these techniques with an empirical analysis of the three mentioned grounding techniques, and combinations of them. The third and major contribution is the empirical analysis of the effect that a smaller grounding has on the subsequent solving phase of declarative systems. We show, using a rigorous statistical analysis on the performed experiments, that a reduction in the grounding size as a result of the applica-

PPDP '14, September 8–10, 2014, Canterbury, UK.
Copyright © 2014 ACM 978-1-4503-2947-7/14/09…$15.00.
http://dx.doi.org/10.1145/

tion of grounding techniques *does not* reduce the search space for solvers. For this analysis we do not have to take into account *how* the grounding size was reduced. We simply observe the effect of a smaller grounding, broadening this result to other ground-and-solve systems which employ grounding techniques aimed at reducing the grounding size in a similar way.

This paper is organized as follows. Section 2 offers some background of the IDP^3 system and its language FO$(\cdot)^{\text{IDP}}$. Section 3 introduces the three grounding techniques which we will investigate. The first set of experiments determining the effect of these techniques on the efficiency of the grounding step is presented in Section 4. Next, Section 5 shows an analysis of our second set of experiments. These investigate the effect of a smaller grounding on the solving process. We conclude in Section 6.

2. Preliminaries

Several recent proposals for declarative modelling and problem solving use first-order logic (FO) as a base language and support some extension of it. Examples are Enfragmo [1] and the IDP^3, the current version of the IDP Knowledge Base System [6]. IDP^3 supports the language FO$(\cdot)^{\text{IDP}}$, which extends FO with inductive definitions, (partial) functions, types, and aggregates. As argued in [7], the ASP language shows a strong overlap with FO$(\cdot)^{\text{IDP}}$ to the extent that ASP systems could be used as FO$(\cdot)^{\text{IDP}}$ solvers. IDP^3 uses logical operations called *inferences* to solve problems. Amongst others, IDP^3 supports the bounded model expansion inference [5].

IDP^3's inferences take FO$(\cdot)^{\text{IDP}}$ specifications as input. For model expansion, these specifications represent three logical objects: a *vocabulary*, a *theory* and a *structure*. A vocabulary is a set of type symbols, predicate symbols, and function symbols. In this paper, P, Q, R denote predicate symbols, and F, G, H denote function symbols. Without loss of generality, we assume only one domain D. A theory is a conjunction of FO *sentences* and *inductive definitions*. A logical sentence is a logical *formula* where all variables are quantified. We use φ, ψ, π to denote logical formulas. We assume every variable is quantified over D, and if not, we specify that a variable is quantified over D' by stating $\forall x \in D' : \varphi$. A *term* t in a logical formula is either a variable or a function symbol applied to a tuple of terms. We use $\overline{x}$ to denote a tuple of variables, $\overline{t}$ a tuple of terms, $\overline{d}$ a tuple of domain elements, and $\overline{D}$ a cartesian product of D. An *inductive definition* is a set of rules of the form $\forall \overline{x} : P(\overline{t}) \leftarrow \varphi$, where φ is an FO formula and the free variables of φ and $P(\overline{t})$ are among $\overline{x}$. We call $P(\overline{t})$ the *head* of the rule and φ the *body*. The connective $\leftarrow$ is the *definitional implication*, which should not be confused with the material implication $\Rightarrow$. Thus, the expression $\forall \overline{x} : P(\overline{t}) \leftarrow \varphi$ is *not* a shorthand for $\forall \overline{x} : P(\overline{t}) \vee \neg\varphi$. Instead, its meaning is given by the well-founded semantics. The well-founded semantics correctly formalizes definitions that typically occur in mathematical texts [8, 9].

A *domain atom* is an expression of the form $P(\overline{d})$ or of the form $F(\overline{d}) = d'$. A *structure* I over a vocabulary Σ is a partial interpretation of the symbols in Σ. If P is a symbol in Σ, P^I denotes the interpretation of P in I. As such, structures assign a truth value true (**t**), false (**f**) or unknown (**u**) to every domain atom $P(\overline{d})$, signifying respectively whether $\overline{d} \in P^I$, $\overline{d} \notin P^I$, or it is unknown whether $\overline{d} \in P^I$. A structure which assigns at least one domain atom to **u** is *three-valued*, otherwise it is *two-valued*. A structure I is a *model* for a given theory $\mathcal{T}$ over a vocabulary Σ if I is two-valued and I satisfies $\mathcal{T}$. A structure I satisfies a theory $\mathcal{T}$ if each sentence in the theory evaluates to true under classical first order semantics, and each inductive definition in the theory holds under the well-founded semantics. We denote the satisfaction relation by $I \models \mathcal{T}$. A structure I' *refines* I, denoted as $I \leq_p I'$, iff for each domain atom $P(\overline{d})$, $\overline{d} \in P^I \Rightarrow \overline{d} \in P^{I'}$ and $\overline{d} \notin P^I \Rightarrow \overline{d} \notin P^{I'}$. Given a vocabulary Σ, a theory $\mathcal{T}$ and a structure I, the model expansion inference of IDP^3 outputs a structure I' that both refines I and is a model for $\mathcal{T}$, or it decides that no such structure exists.

As stated before, IDP^3 uses a ground-and-solve approach to implement its model expansion inference. In the grounding step, the input FO$(\cdot)^{\text{IDP}}$ specification is grounded to a propositional theory in Extended Conjunctive Normal Form (ECNF), which is the input language of MINISAT(ID), IDP^3's search engine. At its core, an ECNF theory is a conjunction of propositional clauses, a CNF. Thus, IDP^3's grounding algorithms are required to transform first order logic sentences, occurring in an FO$(\cdot)^{\text{IDP}}$ theory, to clauses, contained in an ECNF theory.

This conversion in turn has two core elements: *instantiating* quantifiers, and *flattening* nested propositional formulas. For example, given a sentence

$$\phi = \exists x : P(x) \wedge Q(x)$$

with x's domain being $\{d_1, d_2\}$, then ϕ's instantiation would be

$$inst(\phi) = (P(d_1) \wedge Q(d_1)) \vee (P(d_2) \wedge Q(d_2))$$

To turn $inst(\phi)$ into clauses, we would need to flatten it. This can efficiently done by introducing helper predicates called *tseitin literals* [18]:

$$\begin{aligned} flat(inst(\phi)) =& (L(d_1) \vee L(d_2)) \\ & \wedge (L(d_1) \Leftrightarrow P(d_1) \wedge Q(d_1)) \\ & \wedge (L(d_2) \Leftrightarrow P(d_2) \wedge Q(d_2)) \end{aligned}$$

Now, turning the $\Leftrightarrow$-formulas into clauses using a standard transformation results in a grounded version of ϕ:

$$\begin{aligned} flat(inst(\phi))' =& (L(d_1) \vee L(d_2)) \\ & \wedge (\neg L(d_1) \vee P(d_1)) \\ & \wedge (\neg L(d_1) \vee Q(d_1)) \\ & \wedge (L(d_1) \vee \neg P(d_1) \vee \neg Q(d_1)) \\ & \wedge (\neg L(d_2) \vee P(d_2)) \\ & \wedge (\neg L(d_2) \vee Q(d_2)) \\ & \wedge (L(d_2) \vee \neg P(d_2) \vee \neg Q(d_2)) \end{aligned}$$

This grounding process assumes a set of normalized first order logic sentences as input. Normalizing a sentence consists of unnesting function symbols by introducing existential quantifiers, transforming function symbols to their graph predicate, pushing quantifiers inwards, and pushing negations through quantifiers.

Inductive definitions occurring in FO$(\cdot)$ specifications are eagerly evaluated, since often, truth values for ground atoms in heads can be derived using a tabled logic programming system such as XSB [13, 14]. Further techniques for grounding definitions are considered out of scope for this paper and the remainder of the paper will focus on grounding formulas.

Finally, IDP^3 and its language FO$(\cdot)^{\text{IDP}}$ support more constraints than only first order sentences and inductive definitions: sum constraints, product constraints and cardinality constraints are the most noteworthy. Recent work [4] on MINISAT(ID) added support for grounded versions of these constraints, and the ECNF language specification has since been extended. For the sake of clarity we also ignore these extra features in the remainder of this paper.

So to summarize, we assume the grounding task consists of transforming a structure and a theory containing only first order sentences, to a conjunction of clauses called the ground theory or *grounding*. The first order sentences are normalized, but not yet instantiated or flattened.

$$\begin{array}{ll|ll}
\neg \mathbf{t} & \rightarrowtail \mathbf{f} & \neg \mathbf{f} & \rightarrowtail \mathbf{t} \\
\varphi \vee \mathbf{t} & \rightarrowtail \mathbf{t} & \varphi \wedge \mathbf{f} & \rightarrowtail \mathbf{f} \\
\forall \overline{x} \in \overline{D} : \mathbf{t} & \rightarrowtail \mathbf{t} & \exists \overline{x} \in \overline{D} : \mathbf{t} & \rightarrowtail \mathbf{t} \\
\forall \overline{x} \in \overline{D} : \mathbf{f} & \rightarrowtail \mathbf{f} & \exists \overline{x} \in \overline{D} : \mathbf{f} & \rightarrowtail \mathbf{f}
\end{array}$$

Figure 1. Some simplification rules

To control the complexity of this task, IDP^3 uses the following three grounding optimization techniques: Reduced Grounding (RED) [4], Lifted Unit Propagation (LUP) [21], and Grounding With Bounds (GWB) [22]. Together, these technique provide three advantages. They refine the input structure I, reduce the size of the resulting ground theory, and reduce the time needed to ground. In the next section, these techniques are studied in more detail.

During this paper, the concept of the *size* of a ground theory is needed. We define the size of a ground theory as the number of ground atoms occurring in the ground theory. Since we assume our ground theory to be a conjunctions of clauses, the size of the theory is the sum of the size of each clause. For instance, the size of $flat(inst(\phi))'$ is 16.

3. Overview of Grounding Techniques

In this section we explain on a high level the three mentioned grounding techniques RED, LUP, and GWB. The aim of this section is to give the reader an intuition as to what these techniques achieve and the impact they have on eachother. Each of these techniques has been described in detail in other publications and references are provided.

3.1 Reduced Grounding

The intuition for the Reduced Grounding (RED) technique is that (sub)formulas of which we know the truth value beforehand don't have to be grounded and can be substituted with their truth value. Whenever the top-down grounder enters a leaf containing an atom whose truth value is known, that value is filled in. We call this an *evaluation*; known domain atoms are substituted with their truth value. Additionally, when a formula has a subformula whose truth value is known, this truth value is propagated, for this formula. E.g. if one of the disjuncts in a disjunction is true, that entire disjunction is true as well. We call this a *simplification* [4]. Some of the rules used in simplifications are shown in Figure 1. Note that RED can only start working after a leaf of the top-down grounding process has been evaluated.

Example Consider the formula

$$\exists x : P(x) \vee \forall y : \psi(x, y) \tag{1}$$

with $D = \{1, 2, 3\}$, and assume I contains the information that $P(1)$ is true. Since there are two quantified variables, the naive grounding has a size of order D^2. Using the above technique we show how the grounding can be simplified to the atom **t**.

Formula (1) is grounded by iterating over the domain of the quantified variable, instantiating it, and grounding the subformula. In order to improve memory usage, we use a depth-first approach, which means the first instantiation for x has to be ground before more instantiations for x are considered for grounding. Assume we start by instantiating x with 1, which means the grounder will continue by grounding the first (instantiated) disjunct in (2).

$$(P(1) \vee \forall y : \psi(1, y)) \vee \exists x \in D\backslash\{1\} : P(x) \vee \forall y : \psi(x, y) \tag{2}$$

Now the grounder encounters (3) and grounds it by grounding each of the disjuncts, the order of which is not specified.

$$P(1) \vee \forall y : \psi(1, y) \tag{3}$$

$$\left\{ \begin{array}{rcl}
F_{\mathtt{CT}} & \leftarrow & \mathbf{t} \\
F'_{\mathtt{CT}}(x) & \leftarrow & F_{\mathtt{CT}} \\
Q_{\mathtt{CT}}(x) & \leftarrow & F'_{\mathtt{CT}}(x) \wedge P_{\mathtt{CT}}(x) \\
F'_{\mathtt{CT}}(x) & \leftarrow & P_{\mathtt{CF}}(x) \\
F'_{\mathtt{CT}}(x) & \leftarrow & Q_{\mathtt{CT}}(x) \\
 & \cdots &
\end{array} \right\}$$

Figure 2. Example of a symbolic representation for formula (6)

Assume we start by grounding the first disjunct, the atom $P(1)$. We know the truth value of this atom evaluate it to **t**, leading to the following formula.

$$\mathbf{t} \vee \forall y : \psi(1, y) \tag{4}$$

The simplification rule $\varphi \vee \mathbf{t} \rightarrowtail \mathbf{t}$ is applicable and the grounder simplifies the entire disjunction to **t**. This is returned as the result for the first disjunct in formula (2), leading to formula (5). Following the same simplification rule, this entire formula can again be simplified to **t**.

$$\mathbf{t} \vee \exists x \in D\backslash\{1\} : P(x) \vee \forall y : \psi(x, y) \tag{5}$$

Note that we assumed we were "lucky" enough to first instantiate x with 1 in formula (1) and to select the first disjunct when grounding formula (3). There is however no guarantee that this happens and an alternative run of the grounder could end up grounding $\forall y : \psi(x, y)$ first, before finding out simplifications can be made. This shows the unpredictable nature of the benefits of this approach.

A note on implementation Reduced grounding is the most straightforward optimization for the grounding. Given that the implementation of this technique is essentially the application of a variety of substitution rules, implementing it is considered to be no great challenge. Compared to the other two techniques which we will discuss, it is fair to say that this one is the easiest to implement.

3.2 Lifted Unit Propagation

Lifted unit propagation (LUP) is a technique that aims to refine the input structure I without losing any models. This is done by creating a symbolic representation of the theory $\mathcal{T}$ containing the truth dependencies between formulas in $\mathcal{T}$. More specifically, the symbolic representation expresses for each formula when it can be derived to be certainly true (CT) or certainly false (CF), depending on the CT or CF information about its sub- or superformulas. An example of this is that if a disjunction is known to be CF, each of disjuncts has to be CF as well. Using the symbolic representation, we query for all domain atoms which instances can be derived to be CT or CF. This information is then used to created the refined structure I'.

Example Consider the theory containing formula F shown in (6) with $D = \{1, 2, 3\}$, and assume I contains the information that $P(1)$ is true.

$$\forall x \in D : P(x) \implies Q(x) \tag{6}$$

We give part of the symbolic representation as a definition in Figure 2 for this theory and use $F'(x)$, shown in (7) to denote the subformula of F.

$$P(x) \implies Q(x) \tag{7}$$

We show that by using it, we can derive that $Q(1)$ is true.

Since F is a top-level formula it has to be true in order to satisfy the theory. This is expressed in the first rule. The second rule shows that if the universally quantified F is true, it has to be true for each instance in the subformula F' as well. The third rule expresses that when it is known that $F'(x)$ is true and $P(x)$ is true, formula (7) forces that $Q(x)$ is true as well. Because $P(1)$ is known to be true in I, and all $F'(x)$ are true because F is a top-level formula, $Q(1)$ will be returned when the symbolic representation is queried for values for which it is known that $Q(x)$ is true. $Q(1)$ is then inserted into I', leading to a strictly more refined structure than I.

When applying this technique without any of the two other techniques, the resulting grounding will not change, since the information present in I' is not exploited in the grounding process. The benefit of this technique lies in the additional information provided in I' that can be exploited by other grounding technques that use information present in the structure, such as RED.

A note on implementation This technique works for any symbolic representation of the theory. In this paper we use the implementation suggested by Wittocx et al. [20] which is based on Binary Decision Diagrams (BDDs). However, as is mentioned by Wittocx et al., a "complete" symbolic representation of the theory can be very complex and the associated calculation computationally expensive. Because of this, we work with an approximative implementation that places limitations on the complexity of the symbolic representation of the theory. More specifically, the BDDs that are used are limited in the length of their branches and when a BDD becomes too big, it is pruned.

3.3 Grounding With Bounds

Ground With Bounds [22] (GWB) is a technique that detects when formulas are already true or false before grounding them. Recall in our example we showed that the benifits of RED depend greatly on the order in which formulas are grounded, since it needs to evaluate a domain term in a leaf of the grounding tree before being able to propagate (simplify) this information upwards. In contrast to this, GWB tries to detect propagations that RED can do *before* arriving at these leaf atoms. This allows GWB to offer the benefits of RED without depending on the order of grounding.

GWB uses a closely related symbolic representation of the theory used in LUP. The only difference is that it does not contain rules deriving that top-level formulas are true by default, such as the first rule in Figure 2. Using this symbolic representation, we can query each formula for instances that are known to be `CF` or `CT`. Instances of a formula that are known to be false (true) are called the `CF` (`CT`) bound for this formula, hence the name grounding with bounds. Formally, a bound for a formula is a set of assignments to the free variables in that formula that makes its interpretation true (for a `CT` bound) or false (for a `CF` bound). If the set of assignments bound contains is *larger*, this bound is more precise and we denote it as a *tighter* bound on the formula. These bounds are used to limit which subformulas as well as instances for quantified variables are considered for grounding. When GWB is used in the workflow, it will reduce the number of subformulas that need to be grounded. For a conjunction this is the number of conjuncts, for a quantifier this is the instances of the quantified variables for which the subformula is handled.

Example We use again the example theory shown in the LUP section. Consider the theory containing formula F shown in (6) with $D = \{1, 2, 3\}$, and assume I contains the information that $P(1)$ and $Q(1)$ are true. When instantiating the universally quantified variable x in formula F, we are only interested in values for which the subformula F' is not known to be true. Because the grounding for these instantiations would end up being simplified to **t** anyway, these can be dropped from the large conjunction that F will become. We are therefor only interested in values of x for which it is not yet known that $F'(x)$ is true. This corresponds with querying our symbolic representation with $\{x : \neg F'_{\mathtt{CT}}(x)\}$. We refer again to the symbolic representation as a definition in Figure 2 and observe that rule five expresses that the implication is trivially satisfied if its consequent is true. Since $Q(1)$ is known to be true, $F'(1)$ is derived to be true, eliminating it from the answer set of the above query.

A note on implementation Similar to the implementation of LUP above, we use a symbolic representation of the theory that is based on BDDs and is approximative.

3.4 Overview of the workflow

Algorithms 1 and 2 illustrate the high-level structure of our top-down, depth-first grounding algorithm. The three discussed techniques are integrated into the grounding workflow. Lifted Unit Propagation is called before the grounding of the individual formulas (Algorithm 1, line 5), resulting in I' with $I \leq_p I'$. Ground With Bounds is called at the start of every recursive call (Algorithm 2, line 2), eliminating parts of the formula that are unnecessary to ground e.g. reducing the domain of quantified variables). The Reduced Grounding is the combination of the simplify at the end of every recursive call (line 23) and the evaluation of atoms (line 6) in Algorithm 2, this helps reducing the grounding after it is made. More interested readers can find the source code of the grounder discussed here as part of the IDP^3 system at `http://dtai.cs.kuleuven.be/krr/software/idp`

```
  input  : A partial structure $I$ and a theory $T$
  output: A quantifier-free theory equisatisfiable with $T$
1 Function groundTheory($T$,$\mathcal{I}$):
2     $T' \leftarrow$ transform($T$)
3     $S^t \leftarrow$ tseitsinize($T'$)
4     $G \leftarrow \emptyset$
5     $I' \leftarrow$ LUP($I$,$S^t$)
6     for formula $\psi$ in $T'$ do
7         $G \leftarrow G \cup$ groundForm($\psi$, $I'$)
8     return toCNF($G$)
```
Algorithm 1: High level overview of the workflow

```
   input  : Formula $\psi$, structure $I$
   output: A quantifier-free and possibly simpler version of $\psi$
 1 Function groundForm($\psi$,$I$):
 2     $\psi' \leftarrow$ GWB($\psi$)
 3     switch $\psi'$ do
 4         case atom P($\overline{x}$)
 5             if $I$ contains P($\overline{x}$) then
 6                 return $I \rightarrow$ evaluate(P($\overline{x}$))
 7             else
 8                 return P($\overline{x}$)
 9         case $\bigvee_i \psi_i$
10             d $\leftarrow \emptyset$
11             for $\psi_i$ in $\psi'$ do
12                 $d \leftarrow d \cup$ groundForm($\psi_i$,$I$)
13             $out \leftarrow$ disjunction($d$)
14         case $\bigwedge_i \psi_i$
15             $\cdots$ (analog to the disjunctive case)
16         case $\exists x \in D : \psi$
17             d $\leftarrow \emptyset$
18             for $x' \in D$ do
19                 $d \leftarrow d \cup$ groundForm($\psi[x/x']$,$I$)
20             $out \leftarrow$ disjunction($d$)
21         case $\forall x \in D : \psi$
22             $\cdots$ (analog to the existential case)
23     return simplify($out$)
```
Algorithm 2: The grounding of a formula w.r.t. a structure

Nr	Problem Name	Origin
1	15 Puzzle	ASP09
2	Blocked NQueens	ASP09
3	Channel Routing	ASP09
4	Connected Dominating Set	ASP09
5	Edge Matching	ASP09
6	Graph Partitioning	ASP09
7	Hamiltonian Path	ASP09
8	Hierarchical Clustering	ASP09
9	Maze Generation	ASP09
10	Schur Numbers	ASP09
11	Travelling Salesperson	ASP09
12	Weight Bounded Dominating Set	ASP09
13	Wire Routing	ASP09
14	Generalized Slitherlink	ASP11
15	Fastfood Optimality Check	ASP11
16	Sokoban Decision	ASP11
17	Knight Tour	ASP11
18	Disjunctive Scheduling	ASP11
19	Packing Problem	ASP11
20	Labyrinth	ASP11
21	Numberlink	ASP11
22	Reverse Folding	ASP11
23	Hanoi Tower	ASP11
24	Magic Square Sets	ASP11
25	Airport Pickup	ASP11
26	Partner Units	ASP11
27	Maze Generation	ASP11
28	Tangram	ASP11
29	Permutation Pattern Matching	ASP13
30	Graceful Graphs	ASP13
31	Bottle Filling Problem	ASP13
32	NoMystery	ASP13
33	Sokoban	ASP13
34	Ricochet Robot	ASP13
35	Solitaire	ASP13
36	Weighted Sequence Problem	ASP13
37	Incremental Scheduling	ASP13
38	Visit all	ASP13
39	Knight Tour With Holes	ASP13
40	Graph Colouring	ASP13
41	Bounded Spanning Tree	-
42	Latin Squares	-
43	Sudoku	-

Table 1. Problems in our benchmark set

4. Grounding Experiments

In this section, we present the detailed results of the experiments we performed on the grounding techniques of the previous section. Table 1 shows the problems used during these experiments and their origins. These are all problems of the three previous ASP competitions that are classified with a "NP" complexity. We also added three problems that were prevously used in grounding experiments performed by another group [19]. For each of these problems the experiment is run with ten instances that were randomly selected. All experiments are performed with a memory threshold of 4GB and a time threshold of 300 seconds.

We aim to investigate the effect of the three grounding techniques presented in the previous section on the "efficiency" of the grounding step. In order to determine this, we compare different combinations of these techniques and measure the effect on the efficiency of the grounding phase. Efficiency is measured using three properties:

ID	LUP	GWB	RED
$\texttt{Run}_{LGR}$	yes	yes	yes
$\texttt{Run}_{LGx}$	yes	yes	no
$\texttt{Run}_{LxR}$	yes	no	yes
$\texttt{Run}_{Lxx}$	yes	no	no
$\texttt{Run}_{xGR}$	no	yes	yes
$\texttt{Run}_{xGx}$	no	yes	no
$\texttt{Run}_{xxR}$	no	no	yes
$\texttt{Run}_{xxx}$	no	no	no

Table 2. Different experiment setups

- The **number** of instances that were successfully grounded within the thresholds.
- The **duration** of the grounding phase, measured in seconds.
- The **size** of the resulting grounding, as defined in Section 2

The three identified grounding optimization techniques can be activated or disabled, so there are eight ways to combine them. The column labeled "ID" of Table 2 accords an identifier to each of the combinations. E.g., $\texttt{Run}_{LGx}$ represents the run where LUP and GWB were activated, and RED was not.

In Section 4.1 we investigate the effect of RED by comparing $\texttt{Run}_{xxx}$ with $\texttt{Run}_{xxR}$. This allows us to determine the benefits of adding the RED technique when no other technique is used. Section 4.2 compares $\texttt{Run}_{xxR}$ with $\texttt{Run}_{LxR}$ to see what added benefit LUP offers when RED is used to take advantage of the extra information derived by LUP. Section 4.3 contains a report of three comparisons to examine the advantages that GWB offers.

Detailed information for all runs is presented in Table 3. Since $\texttt{Run}_{Lxx}$ and $\texttt{Run}_{xGx}$ are not used in any of the comparisons above, no experiments are performed for these combinations. For each run (identified in the leftmost column) we present three statistics.

- $s_{\#}$ is the number of instances that were successfully grounded,
- t_{avg} is the average running time of the successful instances, and
- g_{avg} is the average grounding size of the successful instances

Table 3 shows the runs for which the highest number of successful runs was achieved in **bold**. The last row of the table also shows the total number of successfully ground instances for each run. When not a single instance was successfully grounded, the table shows an "-" for average time and grounding size.

Examining Table 3, we observe the following.

- The three runs that have the most successfully grounded instances are the three runs including GWB.
- There is only one problem for which none of the approaches could successfully ground even a single instance (37). This problem is known to favor the usage of Constraint Programming (CP) techniques. IDP^3 offers an option in which these techniques are used, but we discovered this option was not turned on for this problem in our benchmark set.
- There is one problem for which none of the approaches have an effect on the efficiency of the grounding phase (36). This can also be explained by the fact that this is a problem for which CP does very well. In contrast to problem 37 IDP^3's CP option was turned on for this problem.
- The table shows that there is variety in the difficulty; some problems can be ground by the naive approach and other problems require techniques.
- Consecutive ASP competitions became harder to ground. ASP09 problems can be ground by any run. ASP11 contains a few problems for which the most naive run could not ground a

single instance. ASP13 has problems for which multiple approaches could not ground a single instance.

- Generally, when the resulting grounding size is smaller, the grounding time is also reduced.

All comparisons are presented in Table 4. For the comparison between $\texttt{Run}_i$ and $\texttt{Run}_{i'}$ (identified in the first column) we present two statistics:

- $t^{\%}_{avg}$ is the ratio (in percent) of the average running time of $\texttt{Run}_{i'}$ over the average running time of $\texttt{Run}_i$ when both approaches succeeded. I.e., t_{avg} of $\texttt{Run}_{i'}$ divided by t_{avg} of $\texttt{Run}_i$, only counting instances where both $\texttt{Run}_i$ and $\texttt{Run}_{i'}$ succeed.
- $g^{\%}_{avg}$ is the ratio (in percent) of the average grounding size of $\texttt{Run}_{i'}$ over the average grounding size of $\texttt{Run}_i$ when both approaches succeeded. I.e., g_{avg} of $\texttt{Run}_{i'}$ divided by g_{avg} of $\texttt{Run}_i$, only counting instances where both $\texttt{Run}_i$ and $\texttt{Run}_{i'}$ succeed.

It is important to note that comparisons are only made between instances that both approaches could successfully ground. As a result, the ratios presented in Table 4 can in not be obtained by dividing the corresponding values present in Table 3, unless ofcourse, both approaches were able to solve the exact same set of instances. When not a single instance was successfully grounded by both approaches, the table shows an "-" for ratio of average time and grounding size.

4.1 Experimental Evaluation of RED

The comparison between $\texttt{Run}_{xxx}$ and $\texttt{Run}_{xxR}$ is shown in the second row of Table 4. This comparison shows that adding RED is very benificial to the efficiency of the grounding step; on average the grounding is done in 71.66% of the time and the resulting grounding is only 37.62% as big. Moreover, for some problems, the decrease in ground size is several orders of magnitude (e.g. problems 2, 3, and 21). On the other hand, there are problems for which this technique has no effect (e.g. problems 10, 19, and 36). Table 3 shows that the addition of RED also has a positive influence on the number of successfully grounded instances, going from 226 to 313. From this we can conclude that the addition of RED without any additional techniques only has positive effects on the efficiency of the grounding and has no drawbacks. Combined with the fact that the implementation of this technique is rather straightforward, as mentioned in Section 3.1, it is highly advisable to implement it.

4.2 Experimental Evaluation of LUP

This section compares $\texttt{Run}_{xxR}$ with $\texttt{Run}_{LxR}$ to see what added benefit LUP offers when RED is used to take advantage of the extra information derived by LUP. The experimental data for the comparison can be found in Table 4 in the third row. This comparison shows that the additional information that LUP derives can improve the usage of the RED technique even further; on average the grounding is done in 87.62% of the time and the resulting grounding is only 69.03% as big. Additionally, the number of successfully grounded instances increases from 313 to 326. For some problems no additional information could be derived, so the grounding size remains the same, whilst increasing the average running time (e.g. problems 3, 5, and 18). From this we can derive that the computational cost of the LUP execution is acceptable; even for problems in which it derives nothing and thus offers no advantages, the average grounding time increases at most by 5%. For LUP we can thus conclude that the implementation of this technique is definitely worthwhile. Whilst it offers no improvements in the efficiency of the grounding step as a standalone option when no other options are activated, the extra elements it derives are able to be successfully exploited by other grounding techniques. We also note that the implementation of this technique is more challenging than the implementation of RED, since special data structures and reasoning techniques need to be implemented to create and query the symbolic representation. Even with our approximative implemention of LUP (as mentioned in Section 3.2), we observe an increased average grounding time for some of the problems. This indicates special care needs to be taken to not make the implementation too costly.

4.3 Experimental Evaluation of GWB

In order to determine the effect of the GWB technique we perform three comparisons. First we compare $\texttt{Run}_{xxR}$ with $\texttt{Run}_{xGR}$ to determine the benefit of using GWB in combination with RED as opposed to when only RED is used. Next we compare $\texttt{Run}_{LxR}$ with $\texttt{Run}_{LGR}$ to see analyse how much GWB benefits from the extra information derived by LUP. To illustrate the approximative nature of our GWB implementation, we compare $\texttt{Run}_{LGx}$ with $\texttt{Run}_{LGR}$. This will give an idea of how much derivations were "missed" because of approximative method.

The comparison between $\texttt{Run}_{xxR}$ and $\texttt{Run}_{xGR}$ is shown in the fourth row of Table 4 and it shows that on average the grounding is done in 62.58% of the time. As argued when introducing the GWB technique (see Section 3.3), results show that the grounding size remains the same. This shows that although GWB and RED achieve the same purpose (i.e., reduce the grounding size), the advantage that the GWB technique does this *beforehand* leads to a substantial decrease in grounding time.

The fifth row of Table 4 shows a similar comparison of $\texttt{Run}_{LxR}$ with $\texttt{Run}_{LGR}$. The effect of adding GWB when both LUP and RED are already activated leads to a grounding time that is on average 54.67% that of before. This is the largest decrease in grounding time for the addition of a single technique. This additionally shows that GWB is able to use the extra information derived by LUP effectively; it reduces the grounding time to 54.67% of the former time, as opposed to the 62.58% reduction that was witnessed in the previous comparison in which LUP was absent. The grounding size is reduced as well (98.68%). This is counter-intuitive because we argued in the previous section that adding GWB when RED is already present the grounding size should remain the same. This is also confirmed by the by the fourth row in the table. We examined this and discovered that this reduction in grounding size is caused by three problems where there is a certain lack of optimization in the implementation of our technique. More specifically, some Tseitin symbols are introduced twice where only one was necessary for $\texttt{Run}_{LxR}$. On the other hand, $\texttt{Run}_{LGR}$ did not contain these duplicates.

The comparison between $\texttt{Run}_{LGx}$ and $\texttt{Run}_{LGR}$ is shown in the last row of Table 4. The grounding size difference here shows the approximative nature of our GWB technique. If GWB was complete, the addition of the RED technique would not lead to a smaller average grounding size. This is constrasted by the observation that in this comparison, the average grounding size is reduced to 47.24% of the original size. This means that the simplifications that are not done by GWB due to its approximative nature account for over half of the remaining grounding size. The significant speedup offered by GWB, combined with the fact that the implementation is highly approximative (on average half of the remaining grounding size could be prevented by GWB but had to be simplified by RED) serves as a good motivation towards future work investigating the possibility to reduce the approximative nature of the GWB method without incurring too much overhead.

As was the case with LUP, the implementation of the GWB technique is challenging. Nonetheless, we observe that this technique is essential when building a state-of-the-art grounder. Despite its current approximative implementation in IDP^3, GWB ap-

	$\texttt{Run}_{LGR}$			$\texttt{Run}_{LGx}$			$\texttt{Run}_{LxR}$			$\texttt{Run}_{xGR}$			$\texttt{Run}_{xxR}$			$\texttt{Run}_{xxx}$		
ID	$s_{\#}$	t_{avg}	g_{avg}	$s_{\#}$	t_{avg}	g_{avg}	$s_{\#}$	t_{avg}	g_{avg}	$s_{\#}$	t_{avg}	g_{avg}	$s_{\#}$	t_{avg}	g_{avg}	$s_{\#}$	t_{avg}	g_{avg}
1	**10**	0	215558	**10**	0	223240	**10**	0	215558	**10**	0	222583	**10**	0	222583	**10**	0	296408
2	**10**	1	3190	**10**	1	13312	**10**	14	3190	**10**	1	4992	**10**	14	4992	9	45	17904694
3	**10**	3	74836	**10**	3	126229	8	86	49209	10	3	74836	8	85	49209	4	131	117658700
4	**10**	0	6952	**10**	0	59696	**10**	0	6952	**10**	0	6952	**10**	0	6952	**10**	0	161406
5	**10**	4	4116943	**10**	4	6478315	**10**	13	4116943	**10**	3	4116943	**10**	12	4116943	7	16	42750706
6	**10**	0	34301	**10**	0	194975	**10**	0	34301	**10**	0	34301	**10**	0	34301	**10**	0	194975
7	**10**	0	2600	**10**	0	4549	**10**	0	2600	**10**	0	78234	**10**	0	78234	**10**	0	141799
8	**10**	1	48132	**10**	1	68761	9	0	47722	9	1	776837	9	1	776837	9	1	1225968
9	**10**	0	9692	**10**	0	45014	**10**	1	9692	**10**	0	11934	**10**	1	11934	**10**	3	3186601
10	**10**	0	71502	**10**	0	71537	**10**	0	73406	**10**	0	73144	**10**	0	73144	**10**	0	75076
11	**10**	0	16880	**10**	0	47877	**10**	0	30782	**10**	0	16604	**10**	0	16604	**10**	1	1366231
12	**10**	0	1100	**10**	0	29281	**10**	1	1100	**10**	0	1100	**10**	1	1100	**10**	8	10176834
13	**10**	0	112314	**10**	0	309399	**10**	25	112314	**10**	0	125686	**10**	25	125686	6	31	48805667
14	**10**	9	1063733	7	6	9049236	5	100	154492	5	47	33537369	5	139	33537369	0	-	-
15	**10**	1	15903	**10**	2	1013460	**10**	20	15903	**10**	16	14349222	**10**	39	14349222	0	-	-
16	**10**	1	1507466	**10**	4	10195036	**10**	12	1507466	4	37	73869847	4	38	73869847	0	-	-
17	**10**	2	30806	**10**	2	50625	**10**	15	30806	8	7	2293888	8	9	2293888	7	4	1586793
18	**3**	30	1904374	**3**	30	2089858	1	23	1737497	**3**	30	1904374	1	22	1737497	0	-	-
19	**3**	11	6635640	2	6	3969862	**3**	11	6635640	2	6	3969748	2	6	3969748	2	6	3970001
20	**8**	72	632396	3	33	45063936	**8**	90	632396	6	37	6059844	6	40	6059844	3	37	47697810
21	**10**	0	150285	**10**	0	406347	7	37	14247	**10**	0	151076	7	37	14615	2	10	26014222
22	**1**	10	1487006	**1**	15	5509897	**1**	31	1487006	**1**	10	1492710	**1**	31	1492710	**1**	63	22641079
23	**10**	1	884547	**10**	3	5667748	**10**	1	884547	**10**	9	16027486	**10**	9	16027486	**10**	12	19663568
24	**10**	0	2804	**10**	0	83751	**10**	0	2804	**10**	0	6146	**10**	0	6146	7	3	102586
25	**8**	25	25194237	**8**	25	32407583	2	66	3348329	**8**	24	25328233	2	65	3368745	0	-	-
26	**10**	1	3813188	**10**	1	5240087	**10**	20	3813188	**10**	1	3813388	**10**	21	3813388	0	-	-
27	**10**	65	50060	**10**	66	815970	4	87	27996	**10**	65	202036	2	60	91321	0	-	-
28	**10**	16	367124	0	-	-	**10**	34	367124	**10**	16	367124	**10**	34	367124	0	-	-
29	**10**	28	4458517	**10**	29	4481669	5	34	529380	**10**	29	4458517	5	33	529380	4	30	1601321
30	**10**	0	31059	**10**	0	33536	**10**	0	31059	**10**	0	31059	**10**	0	31059	**10**	0	36893
31	**10**	5	1045989	**10**	6	4029795	0	-	-	**10**	4	1063372	0	-	-	0	-	-
32	**5**	13	6735660	4	17	18976800	0	-	-	2	27	16920003	0	-	-	0	-	-
33	**10**	5	2113681	**10**	5	3082032	3	77	133749	**10**	5	2113834	3	77	133801	0	-	-
34	**10**	26	7705908	**10**	27	10077373	**10**	41	7705908	**10**	35	16966275	**10**	49	16966275	**10**	51	19420756
35	**10**	0	35550	**10**	0	882915	**10**	4	35550	**10**	0	43156	**10**	10	43156	0	-	-
36	**10**	0	1702	**10**	0	1702	**10**	0	1702	**10**	0	1702	**10**	0	1702	**10**	0	1702
37	0	-	-	0	-	-	0	-	-	0	-	-	0	-	-	0	-	-
38	**10**	0	25715	**10**	0	485210	**10**	0	25715	**10**	0	314532	**10**	0	314532	**10**	0	957380
39	**10**	13	2400376	**10**	14	3189129	0	-	-	0	-	-	0	-	-	0	-	-
40	**10**	0	15535	**10**	0	32013	**10**	0	15535	**10**	0	15535	**10**	0	15535	**10**	0	162686
41	**10**	0	46834	**10**	0	50334	**10**	0	46834	**10**	0	277584	**10**	0	277584	**10**	0	281084
42	**10**	1	1888556	**10**	1	2775305	**10**	2	1888556	**10**	3	4779401	**10**	3	4779401	**10**	3	4860001
43	**10**	5	1109217	**10**	5	1547641	**10**	80	1194156	**10**	5	1183534	**10**	76	1183534	5	4	2181169
Sum	**388**			368			326			358			313			226		

Table 3. Detailed experiment results for all benchmarks, for each discussed combination of grounding techniques

Comparison	$t_{avg}^{\%}$	$g_{avg}^{\%}$
$\texttt{Run}_{xxx}$ vs $\texttt{Run}_{xxR}$	71.66	37.62
$\texttt{Run}_{xxR}$ vs $\texttt{Run}_{LxR}$	87.62	69.03
$\texttt{Run}_{xxR}$ vs $\texttt{Run}_{xGR}$	62.58	100
$\texttt{Run}_{LxR}$ vs $\texttt{Run}_{LGR}$	54.67	98.68
$\texttt{Run}_{LGx}$ vs $\texttt{Run}_{LGR}$	85.94	47.24

Table 4. Ratios of average grounding time and size between runs

pears to be the best of the three discussed techniques to decrease the grounding time.

5. Solver Behaviour on Optimized Ground Theories

As mentioned in our introduction, the IDP^3 system uses the ground-and-solve approach. The previous section was concerned with investigating the effects of RED, LUP, and GWB on the grounding step. This section is dedicated to examining the effect that a smaller grounding has on the search process. We state this question more clearly:

Let $\mathcal{T}$ be some FO($\cdot$) theory, and $\mathcal{T}_1$, $\mathcal{T}_2$ two ground theories which are equisatisfiable to $\mathcal{T}$. Also, let $\mathcal{T}_1$ be obtained by a more optimized grounding algorithm than $\mathcal{T}_2$, so that the size of $\mathcal{T}_1$ is smaller than the size of $\mathcal{T}_2$. How difficult is it for a state-of-the-art CDCL solver to find a model starting from $\mathcal{T}_1$ resp. $\mathcal{T}_2$?

In general, the larger a ground theory, the harder it is to find a model for it, so we expect to take less time to find a model M_1 for $\mathcal{T}_1$ than to find a model M_2 for $\mathcal{T}_2$. This is experimentally confirmed by [19] on the Bounded Spanning Tree, Latin Squares and Sudoku problem set. In this section, we will investigate in more detail what aspect of finding a model becomes harder: is the actual search tree for finding M_2 bigger than that for finding M_1? Or is the cause of the slowdown simply due to the overhead of keeping more constraints consistent?

We first investigate a simple example to get a grasp of what the difference between $\mathcal{T}_1$ and $\mathcal{T}_2$ might look like. Assume:

$$\mathcal{T} = \varphi \land P \land (P \lor (\psi \land \pi))$$

A simple grounding mechanism would introduce a Tseitin literal [18] L_T to unnest the rightmost conjunct:

$$\mathcal{T}_2 = \varphi \land P \land (P \lor L_T) \land (L_T \Leftrightarrow \psi \land \pi)$$

While a smart grounding algorithm using LUP could derive that P must be true, and hence could use RED to obtain a simpler, but equisatisfiable, flattened theory:

$$\mathcal{T}_1 = \varphi \land P$$

So the difference between $\mathcal{T}_1$ and $\mathcal{T}_2$ is the fact that the constraint $(P \lor (\psi \land \pi))$ is no longer present in $\mathcal{T}_1$, since it is a logical consequence of $\mathcal{T}_1$. As observed by [19], the unit propagation (UP) capability of a CDCL-solver does not eliminate the difference between optimized groundings. For example, UP should also derive P to be true, but the resulting theories are still not equal in size:

$$UP(\mathcal{T}_1) = \varphi$$

and

$$UP(\mathcal{T}_2) = \varphi \wedge (L_T \Leftrightarrow \psi \wedge \pi)$$

So solving $\mathcal{T}_1$ will be easier than solving $\mathcal{T}_2$, even with unit propagation of P taken into account. However, it is theoretically easy to find some assignment satisfying $(L_T \Leftrightarrow \psi \wedge \pi)$: simply assign L_T the truth value of $\psi \wedge \pi$, which is always possible since L_T is a Tseitin variable not occurring in φ. On the other hand, a solver might choose a value for L_T before the truth value of $(\psi \wedge \pi)$ is known, potentially incurring an exponential blowup of the search space as compared to solving $\mathcal{T}_1$. In this section, we investigate how much trouble a modern solver has with the extra constraints in an unoptimized ground theory.

In theory using an optimized small grounding instead of a naive larger one can speed up the search process in two aspects:

1. The solver has to keep track of less and smaller formulas and assign values to fewer variables, typically reducing the solve time by a factor proportional to the reduction of the size of the ground theory.
2. The omitted ground formulas represented hard constraints for the solver, so the optimized grounding represents a less complex problem. As a result, the search tree needed to find a model is smaller, potentially leading to an exponential speedup.

To decide which aspect is the dominant one, we extract from the benchmark set introduced in the previous section those instances that could be grounded by both the most naive grounding approach ($\texttt{Run}_{xxx}$) and the most advanced one ($\texttt{Run}_{LGR}$) in under 300 seconds. This resulted in a set of 226 instances, for which a distribution of the grounding sizes is given in Figure 3. It is clear that, on average, the ground sizes of the optimized grounding are more than ten times smaller than those of the naive grounding.

On these two ground theories we run MINISAT(ID), IDP^3's state-of-the-art CDCL-based solver, for each instance and compare the results. For this ground-and-solve workflow IDP^3 was given a 900 second time limit, as well as a 4GB memory limit. The time needed for MINISAT(ID) to solve the grounded instances is given in Figure 4. A first conclusion is that of the 226 instances, 32 more could be solved when using the optimized grounding, and that, on average, it takes more time to solve $\texttt{Run}_{xxx}$ instances than $\texttt{Run}_{LGR}$. These findings are consistent with [19] and with the general intuition that smaller groundings lead to reduced solve time.

Note that CDCL-solvers such as MINISAT(ID) follow a depth-first search strategy: at every search step, the solver *decides* a new atom to assign a truth value to, *propagates* implied truth values of other literals, and checks whether the assignment of truth values is still consistent. If not, a *conflict* occurs, and the CDCL-solver backjumps to a consistent state. To verify the cause of the solve time difference for optimized and unoptimized groundings, we plot the number of decisions, propagations and conflicts of our solve runs in Figures 5, 6 and 7 respectively. From these three measures, the number of conflicts is the best representative for search tree width, since every conflict results in a backjump step, triggering the exploration of a new branch in the search tree. The number of decisions often also is an indicator of search tree width, but a high number of decisions does not necessarily entail a high number of branches in a search tree. For instance, an unconstrained problem with n variables will require n decisions to be made, even though the search tree never branches. The number of propagations on the other hand is a measure of overhead: if two solving runs incur the same number of conflicts, but a different number of propagations, then the solving run with the most propagations will have done more work in each branch of the search tree, and will typically take more time.

So from a theoretical point of view, problems for which a solver obtained many conflicts are hard for that solver, while problems with many propagations in each search branch simply state that many variables had to be assigned a value to keep the solver state consistent, implying either a lot of variables, or a lot of constraints to keep track of.

Given these thoughts, we see in Figure 7 that the amount of instances solved with relatively few conflicts (less than ten thousand) is equal between both grounding approaches. It is only when the number of conflicts gets high that optimizing the grounding leads to more solved problem instances. A similar observation can be made in Figure 5 for the number of decisions of each solving run. Figure 6 on the other hand shows is different because even for problems with relatively few propagations the optimized grounding instance requires significantly less propagations than its unoptimized counterpart.

These observations are consistent with the hypothesis that the most significant cause for solve time speedup with optimized groundings is simply the reduction of overhead incurred by e.g. performing more propagations during solving. The above observations are *not* consistent with the hypothesis that the most significant cause for solve time speedup with optimized groundings is the reduction of the search tree inferred by the removal of hard constraints. Using this hypothesis, we would expect the number of conflicts for the solver runs on the unoptimized instances to be significantly larger than the number of conflicts for the solver runs on the optimized instances, over the whole range of instances. The increase observed between runs with more than ten thousand conflicts can be explained by the fact that faster solving times for the optimized grounding (see Figure 4) lead to more plot points. These extra plot points will generally be associated with a large number of conflicts, since they represent difficult search problems that resulted in a solving timeout for the naive grounding.

To further investigate this issue, we conducted a statistical analysis of our data. We are trying to support or debunk the claim that optimizing the grounding leads to a less search problem, or alternatively, leads to less overhead during search. We made the reasonable assumption that the number of conflicts during search is a good indicator of problem complexity, and the number of propagations is a good indicator of problem overhead. All that is left to check is whether a large increase in ground size due to disabling grounding optimizations leads to a large increase of conflicts or propagations when solving the problem instance. We quantify an increase in ground size of a problem instance as the ratio of the size of the unoptimized grounding to the size of the optimized grounding (i.e., grounding size of $\texttt{Run}_{xxx}$ divided by grounding size of $\texttt{Run}_{LGR}$). Similarly, an increase in conflicts resp. propagations is quantified as the ratio of conflicts resp. propagations observed when solving the unoptimized grounding to the ratio of conflicts resp. propagations observed when solving the optimized grounding. This second quantification is only meaningful when the solving run did not hit the timeout, so we restrict our benchmark set to the 160 instances for which both the optimized and unoptimized grounding were solved.

Given these formal notions, we now correlate increases in ground size to increases in conflicts or increases in propagations. This is done by calculating Spearman's rank correlation coefficient for both of these measures. The results, shown in Table 5, indicate that an increase in ground size does only very marginally lead to an increase in the number of conflicts, while an increase in ground

correlated variables	Spearman ρ	95% confidence interval
increase in ground size, increase in conflicts	0.046	$[-0.109, 0.198]$
increase in ground size, increase in propagations	0.617	$[0.511, 0.704]$

Table 5. Correlating increases in ground size

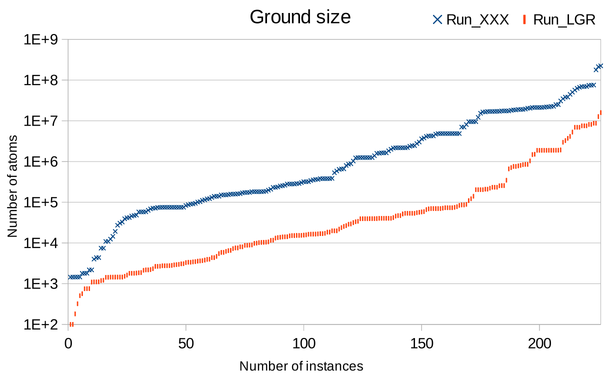

Figure 3. Cactus plot of ground sizes for instances grounded by xxx and LGR

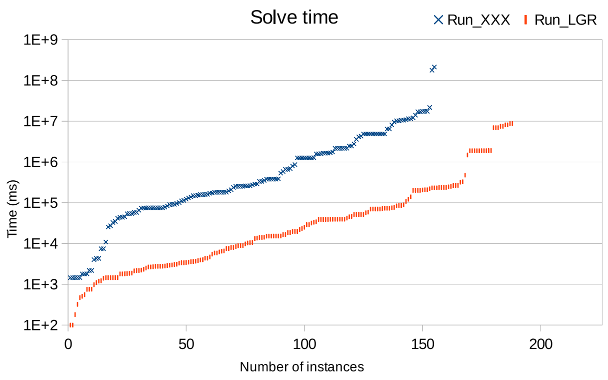

Figure 4. Cactus plot of solve times for instances grounded by xxx and LGR

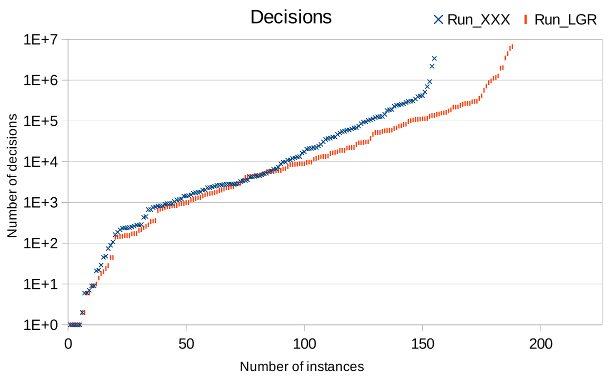

Figure 5. Cactus plot of the number of decisions made by MiniSAT(ID) while finding a model for instances grounded by xxx and LGR

size is strongly correlated to an increase in propagations. Also, the obtained correlations are well within appropriate error margins.

Given this statistical data, and given the preceding plot-based observations, we find that optimizing the grounding process does not significantly reduce the search tree of a subsequent solving step, but it does lead to less overhead for the solver during search. This can intuitively be explained by the fact that it is relatively simple to derive that these omissible constraints are logical consequences of the original theory, since we were able to detect during grounding that the constraints could be omitted. Nonetheless, it goes to the credit of modern CDCL-solvers that they are not distracted by these extra constraints, but instead seem to largely ignore them in their search trees. We suspect that activity-based heuristics, which prioritize assignments to variables occurring in difficult constraints, play a key role in the observed behaviour.

6. Conclusion

This paper presents three grounding techniques in detail and gives an intuitive description for them. For these techniques two sets of experiments are run. The first provides an empirical analysis of the three mentioned grounding techniques. These experiments show that the RED technique is easy to implement and has no downsides, whilst offering a great reduction in the grounding size and time. The LUP technique is harder to implement and care needs to be taken that it does not have an unacceptable computational cost. Combined with RED, the LUP technique offers a reduction in the grounding size and time. Finally, the GWB technique is also hard to implement but provides the largest reduction in grounding time. We show that when RED is used (and there is no reason not to since it has no downsides), GWB does not offer any reductions in the grounding size. We also show that the addition of the LUP technique also benefits GWB, leading to an even larger reduction in grounding time. It is hard to generalize these results to systems other than IDP^3, since there is no clear mapping to the techniques that are present in, for example, ASP grounders, which are based around semi-naive bottom-up evaluation.

The second set of experiments results in an empirical analysis of the effect that a smaller grounding has on the subsequent solving phase of declarative systems. We observe that a smaller grounding results in a significant speedup, but this speedup is not due to large reductions in the search tree, but rather due to less overhead in satisfying easy-to-satisfy constraints. This result can be applied to other ground-and-solve systems which employ grounding techniques aimed at reducing the grounding size in a similar way.

References

[1] Aavani, A., Wu, X.N., Tasharrofi, S., Ternovska, E., Mitchell, D.G.: Enfragmo: A system for modelling and solving search problems with logic. In Bjørner, N., Voronkov, A., eds.: LPAR. Volume 7180 of LNCS., Springer (2012) 15–22

[2] Bancilhon, F.: Naive evaluation of recursively defined relations. In Brodie, M., Mylopoulos, J., eds.: On Knowledge Base Management Systems. Topics in Information Systems. Springer New York (1986) 165–178

[3] De Cat, B.: Separating Knowledge from Computation: An FO(.) Knowledge Base System and its Model Expansion Inference. PhD thesis, KU Leuven, Leuven, Belgium (May 2014)

[4] De Cat, B., Bogaerts, B., Devriendt, J., Denecker, M.: Model expansion in the presence of function symbols using constraint programming. In: ICTAI, IEEE (2013) 1068–1075

[5] De Cat, B., Jansen, J., Janssens, G.: IDP3: Combining symbolic and ground reasoning for model generation. (2013)

[6] De Pooter, S., Wittocx, J., Denecker, M.: A prototype of a knowledge-based programming environment. In Tompits, H., Abreu, S., Oetsch, J., Pührer, J., Seipel, D., Umeda, M., Wolf, A., eds.: INAP/WLP. Volume 7773 of Lecture Notes in Computer Science., Springer (2011) 279–286

[7] Denecker, M., Lierler, Y., Truszczynski, M., Vennekens, J.: A Tarskian informal semantics for answer set programming. In Dovier, A., Costa, V.S., eds.: ICLP (Technical Communications). Volume 17 of LIPIcs., Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2012) 277–289

[8] Denecker, M., Ternovska, E.: A logic of nonmonotone inductive definitions. ACM Trans. Comput. Log. **9**(2) (April 2008) 14:1–14:52

[9] Denecker, M., Vennekens, J.: The well-founded semantics is the principle of inductive definition, revisited. In Baral, C., De Giacomo, G., Eiter, T., eds.: KR, AAAI Press (2014)

[10] Faber, W., Leone, N., Perri, S.: The intelligent grounder of DLV. In Erdem, E., Lee, J., Lierler, Y., Pearce, D., eds.: Correct Reasoning. Volume 7265 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2012) 247–264

[11] Gebser, M., Kaminski, R., König, A., Schaub, T.: Advances in Gringo series 3. In Delgrande, J.P., Faber, W., eds.: LPNMR. Volume 6645 of LNCS., Springer (2011) 345–351

[12] Gebser, M., Schaub, T., Thiele, S.: GrinGo : A new grounder for Answer Set Programming. In Baral, C., Brewka, G., Schlipf, J.S., eds.: LPNMR. Volume 4483 of LNCS., Springer (2007) 266–271

[13] Jansen, J., Janssens, G.: Refining definitions with unknown opens using XSB for IDP^3. Number AIB-2014-09 in Aachener Informatik Berichte, RWTH Aachen University (jun 2014) 15–29

[14] Jansen, J., Jorissen, A., Janssens, G.: Compiling input$*$ FO$(\cdot)$ inductive definitions into tabled Prolog rules for IDP^3. TPLP **13**(4-5) (2013) 691–704

[15] Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM Trans. Comput. Log. **7**(3) (2006) 499–562

[16] Nethercote, N., Stuckey, P., Becket, R., Brand, S., Duck, G., Tack, G.: Minizinc: Towards a standard CP modelling language. In Bessiere, C., ed.: CP'07. Volume 4741 of LNCS., Springer (2007) 529–543

[17] Niemelä, I., Simons, P., Syrjänen, T.: Smodels: A system for answer set programming. In: Proceedings of the 8th International Workshop on Non-Monotonic Reasoning, Breckenridge, Colorado, USA (2000) CoRR, cs.AI/0003033.

[18] Tseitin, G.S.: On the complexity of derivation in the propositional calculus, *Zapiski nauchnykh seminarov*. LOMI **8** (1968) 234–259 English translation of this volume: Studies in Constructive Mathematics and Mathematical Logic, Part 2, A. O. Slisenko, eds. Consultants Bureau, N.Y., 1970, pp. 115-125.

[19] Vaezipoor, P., Mitchell, D., Mariën, M.: Lifted unit propagation for effective grounding. CoRR **abs/1109.1317** (2011)

[20] Wittocx, J., Denecker, M., Bruynooghe, M.: Constraint propagation for extended first-order logic. CoRR **abs/1008.2121** (2010)

[21] Wittocx, J., Denecker, M., Bruynooghe, M.: Constraint propagation for first-order logic and inductive definitions. ACM Trans. Comput. Logic **14**(3) (August 2013) 17:1–17:45

[22] Wittocx, J., Mariën, M., Denecker, M.: Grounding FO and FO(ID) with bounds. J. Artif. Intell. Res. (JAIR) **38** (2010) 223–269

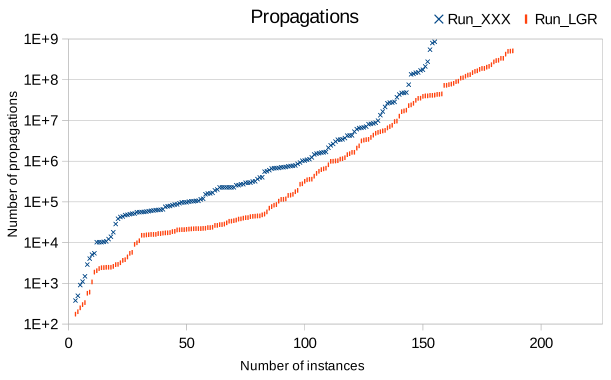

Figure 6. Cactus plot of the number of literals propagated while finding a model for instances grounded by xxx and LGR

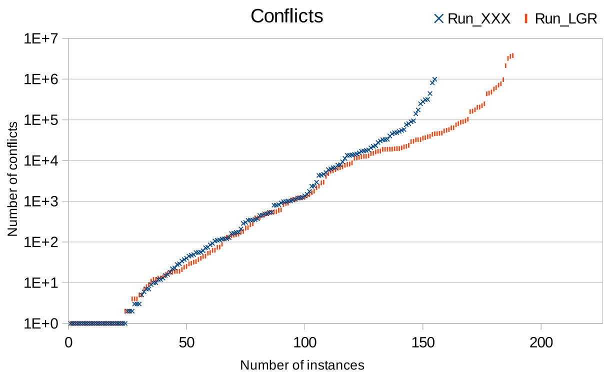

Figure 7. Cactus plot of encountered conflicts while finding a model for instances grounded by xxx and LGR