Speed-Up Techniques for Negation in Grounding

Amir Aavani, Shahab Tasharrofi, Gulay Unel, Eugenia Ternovska, and David Mitchell

Simon Fraser University
aaa78@cs.sfu.ca, sta44@cs.sfu.ca, gunel@cs.uwaterloo.ca, ter@cs.sfu.ca, mitchell@cs.sfu.ca

Abstract. Grounding is the task of reducing a first order formula to ground formula that is equivalent on a given universe, and is important in many kinds of problem solving and reasoning systems. One method for grounding is based on an extension of the relational algebra, exploiting the fact that grounding over a given domain is similar to query answering. In this paper, we introduce two methods for speeding up algebraic grounding by reducing the size of tables produced. One method employs rewriting of the formula before grounding, and the other uses a further extension of the algebra that makes negation efficient. We have implemented the methods, and present experimental evidence of their effectiveness.

1 Introduction

Challenging combinatorial search problems are ubiquitous in applications of computer science, including many problems which are NP-hard and challenging in practice. A variety of algorithmic approaches to such problems are used in practice, among these the application of general purpose solvers, such as SAT and ILP solvers. Effective use of these technologies typically requires considerable expertise, both in terms of the application problem at hand and the solving technology. Many users do not have such expertise easily available, and even for experts producing correct and effective reductions to SAT or integer programming is often a time consuming and error-prone exercise.

An approach to improving accessibility and ease of use of combinatorial search methods is to provide a high-level specification or modelling language, in which both expert and non-expert users may specify their problems without direct concern for the underlying solving techniques. The input for a (ground) solver is then generated automatically from a specification together with an instance. Examples of languages for this purpose include ASP languages [7,6], the language of the IDP system [10], SPEC2SAT [2] and ESSENSE [5]. The process of mapping a high-level specification and an instance to a low-level solver input language is grounding.

The input for a combinatorial search problem is a finite structure, and the task is to construct one or more additional relations satisfying a certain property. For example, in the Hamiltonian cycle problem, the input is a graph $G = \langle V; E \rangle$, and the task is to construct a path P of a certain sort in G, thus obtaining an expanded structure $G' = \langle V; E, P \rangle$. When the property is specified by a formula of first order logic (FO), the class of problems which can be specified are those in NP [4]. Examples of systems with specification languages that are natural extensions of FO are described in [8] and [10]. These systems ground to languages that are extensions of SAT. For any fixed FO specification, grounding is polynomial time, but in practice current grounders are often too slow in grounding large instances of some problems.

A method for grounding FO specifications based on an extension of the relational algebra was introduced in [9], and a prototype implementation based on this method is reported in [8]. In this paper, we present refinements to this method to improve the speed of grounding. When naively implemented, the algebraic grounding approach may be slow because very large intermediate tables may be produced. Much work in database query optimization aims to reduce the size of intermediate tables. However, in our observation, the single largest impediment to speed in grounding is due to negation, which is not well studied in query optimization. The problem is that the complement operation, as defined in [8], tends to produce very large tables, which are sometimes universal (containing all possible tuples) or nearly so.

Here, we present two approaches for mitigating the costs of negation. To our knowledge, these have no close analogs in standard database techniques. The first approach, denoted FR for "formula re-writing" constructs a logically equivalent formula which minimizes the cost of negations. The second method, denoted T/F for "True-False Tables", is a further generalization of the relational algebra in which complementation is inexpensive. Both methods are heuristic, meaning there are conditions under which they will produce slow rather than faster grounding, but in practice we generally observe significant speedups.

We have implemented both of the methods in a new grounder. Each method produces order of magnitude speedups over the version of the grounder without these refinements. Moreover, while the naive version of the grounder is slow in comparison with other existing model expansion grounders, the versions with these improvements are highly competitive.

2 Background

We formalize combinatorial search problems in terms of the logical problem of *model expansion (MX)*, defined here for an arbitrary logic $\mathscr{L}$.

Definition 1 (MX). *Given an $\mathscr{L}$-sentence ϕ, over the union of disjoint vocabularies σ and ε, and a finite structure $\mathscr{A}$ for vocabulary σ, find a structure $\mathscr{B}$ that is an expansion of $\mathscr{A}$ to $\sigma \cup \varepsilon$ such that $\mathscr{B} \models \phi$.*

In this paper, ϕ is a problem specification formula, and is fixed for each search problem. $\mathscr{A}$ always denotes a finite σ-structure, called the instance structure, σ is the instance vocabulary, and ε the expansion vocabulary.

Example 1. The following formula ϕ of first order logic constitutes a specification for Graph 3-Colouring:

$$\begin{aligned}
&\forall x \; [\big(R(x) \vee B(x) \vee G(x)\big)] \;\; \wedge \\
&\forall x \; \neg[(R(x) \wedge B(x)) \vee (R(x) \wedge G(x)) \vee (B(x) \wedge G(x))] \;\; \wedge \\
&\forall x \forall y \; [(E(x,y) \vee E(y,x)) \supset (\neg(R(x) \wedge R(y)) \wedge \neg(B(x) \wedge B(y)) \wedge \neg(G(x) \wedge G(y)))]
\end{aligned}$$

An instance is a structure for vocabulary $\sigma = \{E\}$, i.e., a graph $\mathscr{A} = \mathscr{G} = (V;E)$. The task is to find an expansion $\mathscr{B}$ of $\mathscr{A}$ that satisfies ϕ:

$$\underbrace{(\overbrace{V;E^{\mathscr{A}}}^{\mathscr{A}},\ R^{\mathscr{B}},B^{\mathscr{B}},G^{\mathscr{B}})}_{\mathscr{B}} \models \phi.$$

Interpretations of the expansion vocabulary $\varepsilon = \{R, B, G\}$, for structures $\mathscr{B}$ that satisfy ϕ, are proper 3-colourings of $\mathscr{G}$.

The grounding task is to produce a ground formula $\psi = Gnd(\phi, \mathscr{A})$, such that models of ψ correspond to solutions for instance $\mathscr{A}$. Formally, to ground we bring domain elements into the syntax by expanding the vocabulary with a new constant symbol for each element of the domain. For domain A, the domain of structure $\mathscr{A}$, we denote the set of such constants by $\tilde{A}$. In practice, the ground formula should contain no occurrences of the instance vocabulary, in which case we call it reduced.

Definition 2 (Reduced Grounding for MX). *Formula ψ is a* reduced grounding *of formula ϕ over σ-structure $\mathscr{A} = (A; \sigma^{\mathscr{A}})$ if*
1) ψ is a ground formula over $\varepsilon \cup \tilde{A}$, and
2) for every expansion structure $\mathscr{B} = (A; \sigma^{\mathscr{A}}, \varepsilon^{\mathscr{B}})$ over $\sigma \cup \varepsilon$, $\mathscr{B} \models \phi$ iff $(\mathscr{B}, \tilde{A}^{\mathscr{B}}) \models \psi$, where $\tilde{A}^{\mathscr{B}}$ is the standard interpretation of the new constants $\tilde{A}$.

Proposition 1. *Let ψ be a reduced grounding of ϕ over σ-structure $\mathscr{A}$. Then $\mathscr{A}$ can be expanded to a model of ϕ iff ψ is satisfiable.*

Producing a reduced grounding with respect to a given structure $\mathscr{A}$ can be done by an algorithm that, for each fixed FO formula, runs in time polynomial in the size of $\mathscr{A}$. Such a grounding algorithm implements a polytime reduction to SAT for each NP search problem. Simple grounding algorithms, however, do not reliably produce groundings for large instances of interesting problems fast enough in practice.

Grounding for MX is a generalization of query answering. Given a structure (database) $\mathscr{A}$, a boolean query is a formula ϕ over the vocabulary of $\mathscr{A}$, and query answering is equivalent to evaluating whether ϕ is true, i.e., $\mathscr{A} \models \phi$. For model expansion, ϕ has some additional vocabulary beyond that of $\mathscr{A}$, and producing a reduced grounding involves evaluating out the instance vocabulary, and producing a ground formula representing the possible expansions of $\mathscr{A}$ for which ϕ is true.

The grounding algorithms in this paper construct a grounding by a bottom-up process that parallels database query evaluation, based on an extension of the relational algebra. For each sub-formula $\phi(\bar{x})$ with free variables $\bar{x}$, we call the set of reduced groundings for ϕ under all possible ground instantiations of $\bar{x}$ *an answer to* $\phi(\bar{x})$. We represent answers with tables on which an extended algebra operates.

An X-relation is a k-ary relation associated with a k-tuple of variables X, representing a set of instantiations of the variables of X. It is a central notion in databases. In extended X-relations, introduced in [9], each tuple γ is associated with a formula ψ. For convenience, we use $\top$ and $\bot$ as propositional formulas which are always true and, respectively, false.

Definition 3 (extended X-relation; function $\delta_{\mathscr{R}}$). *Let A be a domain, and X a tuple of variables with $|X| = k$. An* extended X-relation *$\mathscr{R}$ over A is a set of pairs (γ, ψ) s.t.*

1) $\gamma : X \rightarrow A$*, and*
2) ψ *is a formula, and*
3) if $(\gamma,\psi) \in \mathscr{R}$ *and* $(\gamma,\psi') \in \mathscr{R}$ *then* $\gamma = \gamma'$*.*
The function $\delta_{\mathscr{R}}$ represented by $\mathscr{R}$ *is a mapping from k-tuples* γ *of elements of the domain A to formulas, defined by:*

$$\delta_{\mathscr{R}}(\gamma) = \begin{cases} \psi & \text{if } (\gamma,\psi) \in \mathscr{R}, \\ \bot & \text{if there is no pair } (\gamma,\psi) \in \mathscr{R}. \end{cases}$$

For brevity, we sometimes write $\gamma \in \mathscr{R}$ to mean that there exists ψ such that $(\gamma,\psi) \in \mathscr{R}$. We also sometimes call extended X-relations simply tables. To refer to X-relations for some concrete set X of variables, rather than in general, we write X-relation.

Definition 4 (answer to ϕ wrt $\mathscr{A}$). *Let* ϕ *be a formula in* $\sigma \cup \varepsilon$ *with free variables X,* $\mathscr{A}$ *a* σ*-structure with domain A, and* $\mathscr{R}$ *an extended X-relation over* $\mathscr{A}$*. We say* $\mathscr{R}$ *is* an answer to ϕ wrt $\mathscr{A}$ *if for any* $\gamma : X \rightarrow A$*, we have that* $\delta_{\mathscr{R}}(\gamma)$ *is a reduced grounding of* $\phi[\gamma]$ *over* $\mathscr{A}$*. Here,* $\phi[\gamma]$ *denotes the result of instantiating free variables in* ϕ *according to* γ*.*

Since a sentence has no free variables, the answer to a sentence ϕ is a zero-ary extended X-relation, containing a single pair $(\langle\rangle,\psi)$, associating the empty tuple with formula ψ, which is a reduced grounding of ϕ.

Example 2. Let $\sigma = \{P\}$ and $\varepsilon = \{E\}$, and let $\mathscr{A}$ be a σ-structure with $P^{\mathscr{A}} = \{(1,2,3),(3,4,5)\}$. The following extended relation $\mathscr{R}$ is an answer to $\phi_1 \equiv P(x,y,z) \wedge E(x,y) \wedge E(y,z)$:

x	y	z	ψ
1	2	3	$E(1,2) \wedge E(2,3)$
3	4	5	$E(3,4) \wedge E(4,5)$

Observe that $\delta_{\mathscr{R}}(1,2,3) = E(1,2) \wedge E(2,3)$ is a reduced grounding of $\phi_1[(1,2,3)] = P(1,2,3) \wedge E(1,2) \wedge E(2,3)$, and $\delta_{\mathscr{R}}(1,1,1) = \bot$ is a reduced grounding of $\phi_1[(1,1,1)]$. The following extended relation is an answer to $\phi_2 \equiv \exists z \phi_1$:

x	y	ψ
1	2	$E(1,2) \wedge E(2,3)$
3	4	$E(3,4) \wedge E(4,5)$

Here, $E(1,2) \wedge E(2,3)$ is a reduced grounding of $\phi_2[(1,2)]$. Finally, the following represents an answer to $\phi_3 \equiv \exists x \exists y \phi_2$, where the single formula is a reduced grounding of ϕ_3.

ψ
$[E(1,2) \wedge E(2,3)] \vee [E(3,4) \wedge E(4,5)]$

The relational algebra has operations corresponding to each connective and quantifier in FO, as follows: complement (negation); join (conjunction); union (disjunction), projection (existential quantification); division or quotient (universal quantification). Following [9,8], we generalize each to extended X-relations as follows.

Definition 5 (Extended Relational Algebra). *Let $\mathscr{R}$ be an extended X-relation and $\mathscr{S}$ an extended Y-relation, both over domain A.*

1. *$\neg\mathscr{R}$ is the extended X-relation $\neg\mathscr{R} = \{(\gamma,\psi) \mid \gamma : X \to A, \delta_{\mathscr{R}}(\gamma) \neq \top$, and $\psi = \neg\delta_{\mathscr{R}}(\gamma)\}$*
2. *$\mathscr{R} \bowtie \mathscr{S}$ is the extended $X \cup Y$-relation $\{(\gamma,\psi) \mid \gamma : X \cup Y \to A, \gamma|_X \in \mathscr{R}, \gamma|_Y \in \mathscr{S}$, and $\psi = \delta_{\mathscr{R}}(\gamma|_X) \wedge \delta_{\mathscr{S}}(\gamma|_Y)\}$;*
3. *$\mathscr{R} \cup \mathscr{S}$ is the extended $X \cup Y$-relation $\mathscr{R} \cup \mathscr{S} = \{(\gamma,\psi) \mid \gamma|_X \in \mathscr{R}$ or $\gamma|_Y \in \mathscr{S}$, and $\psi = \delta_{\mathscr{R}}(\gamma|_X) \vee \delta_{\mathscr{S}}(\gamma|_Y)\}$.*
4. *For $Z \subseteq X$, the* Z-projection *of $\mathscr{R}$, denoted by $\pi_Z(\mathscr{R})$, is the extended Z-relation $\{(\gamma',\psi) \mid \gamma' = \gamma|_Z$ for some $\gamma \in \mathscr{R}$ and $\psi = \bigvee_{\{\gamma \in \mathscr{R} \mid \gamma' = \gamma|_Z\}} \delta_{\mathscr{R}}(\gamma)\}$.*
5. *For $Z \subseteq X$, the* Z-quotient *of $\mathscr{R}$, denoted by $d_Z(\mathscr{R})$, is the extended Z-relation $\{(\gamma',\psi) \mid \forall\gamma(\gamma : X \to A \wedge \gamma|_Z = \gamma' \Rightarrow \gamma \in \mathscr{R})$, and $\psi = \bigwedge_{\{\gamma \in \mathscr{R} \mid \gamma' = \gamma|_Z\}} \delta_{\mathscr{R}}(\gamma)\}$.*

To ground using this algebra, we apply the algebra inductively on the structure of the formula, just as the standard relational algebra may be applied for query evaluation. We define the answer to atomic formula $P(\bar{x})$ as follows. If P is an instance predicate, the answer to P is the set of tuples $(\bar{a},\top)$, for $\bar{a} \in P^{\mathscr{A}}$. If P is an expansion predicate, the answer is the set of all pairs $(\bar{a}, P(\bar{a}))$, where $\bar{a}$ is a tuple of elements from the domain A. Correctness of the method then follows, by induction on the structure of the formula, from the following proposition.

Proposition 2. *Suppose that $\mathscr{R}$ is an answer to ϕ_1 and $\mathscr{S}$ is an answer to ϕ_2, both with respect to (wrt) structure $\mathscr{A}$. Then*

1. *$\neg\mathscr{R}$ is an answer to $\neg\phi_1$ wrt $\mathscr{A}$;*
2. *$\mathscr{R} \bowtie \mathscr{S}$ is an answer to $\phi_1 \wedge \phi_2$ wrt $\mathscr{A}$;*
3. *$\mathscr{R} \cup \mathscr{S}$ is an answer to $\phi_1 \vee \phi_2$ wrt $\mathscr{A}$;*
4. *If Y is the set of free variables of $\exists\bar{z}\phi_1$, then $\pi_Y(\mathscr{R})$ is an answer to $\exists\bar{z}\phi_1$ wrt $\mathscr{A}$.*
5. *If Y is the set of free variables of $\forall\bar{z}\phi_1$, then $d_Y(\mathscr{R})$ is an answer to $\forall\bar{z}\phi_1$ wrt $\mathscr{A}$.*

The straightforward proof for cases 1, 2 and 4 is given in [9]; the other cases follow easily.

The answer to an atomic formula $P(\bar{x})$, where P is from the expansion vocabulary, is formally a universal table, but in practice we may represent this table implicitly and avoid explicitly enumerating the tuples. As operations are applied, some subset of columns remain universal, while others do not. Again, those columns which are universal may be represented implicitly. This could be treated as an implementation detail, but the use of such implicit representations dramatically affects the cost of operations, and so it is useful to further generalize our extended X-relations. Following [8], we call the variables which are implicitly universal "hidden" variables, as they are not represented explicitly in the tuples, and the other variables "explicit" variables.

Definition 6 (Extended Hidden X-Relation $\mathscr{R}_Y$; $\delta_{\mathscr{R}_Y}$). *Let X, Y be tuples of variables, with $Y \subseteq X$ (when viewed as sets), and $|X| = k$. An extended hidden X-relation $\mathscr{R}_Y$ is a set of tuples (γ,ψ) s.t.*

1) $\gamma : X \backslash Y \to A$, and
2) ψ is a formula, and
3) if $(\gamma,\psi) \in \mathscr{R}_Y$ and $(\gamma,\psi') \in \mathscr{R}_Y$, then $\psi = \psi'$.

The function $\delta_{\mathscr{R}_Y}$ represented by $\mathscr{R}_Y$ is a mapping from k-tuples γ' of elements of the domain A to formulas, defined by

$$\delta_{\mathscr{R}_Y}(\gamma') = \begin{cases} \psi & \textit{if } (\gamma' |_{X \setminus Y}, \psi) \in \mathscr{R}, \\ \bot & \textit{if there is no pair } (\gamma' |_{X \setminus Y}, \psi) \in \mathscr{R}. \end{cases}$$

So, an extended hidden X-relation $\mathscr{R}_Y$ is a compact representation of an extended X-relation by an extended $X\setminus Y$-relation, which may be used whenever the columns for variables of Y are universal. If $X = Y$, we have a compact representation of a universal relation; if $Y = \emptyset$, we have a normal extended X-relation.

All the operations of the algebra generalize easily. The hidden variables technique does not alter the semantics of the operations. Henceforth, the term table may denote either an extended X-relation or a hidden extended X-relation.

Definition 7. ***Basic Grounding Method (B):*** *We ground a sentence ϕ wrt $\mathscr{A}$ using this algebra, proceeding bottom-up according to the structure of the formula, and applying the operation corresponding to each connective or quantifier. At the top, we obtain the answer to ϕ, which is a relation containing only the pair $(\langle\rangle, \psi)$, where ψ is a reduced grounding of ϕ wrt $\mathscr{A}$.*

The Negation Problem. If we naively negate an extended X-relation, we often end up with a universal table. To see this, consider an extended X-relation $\mathscr{R}$. To construct its negation $\neg\mathscr{R}$, we include $(\gamma, \neg\psi)$ for every $(\gamma, \psi) \in \mathscr{R}$ with $\psi \neq \top$, and include $(\gamma, \top)$, for every γ with $\gamma \notin \mathscr{R}$. If there are no tuples $(\gamma, \top) \in \mathscr{R}$ then $\neg\mathscr{R}$ contains a pair for every possible instantiation of X. Once a universal, or nearly universal, table is constructed, all following operations must deal with it. In the following two sections, we describe two methods for mitigating this problem.

3 Formula Rewriting

In this section, we describe a method for pre-processing the specification formula before running the basic grounder. For each sub-formula ψ of ϕ, consider all possible ways to rewrite ψ using De Morgan laws, generalized to include quantifiers. Of all rewritings, we choose the one for which cost of carrying out negations is minimum.

This is a heuristic method. It is often the case that the total grounding time is dominated by the time for joins, and it is possible that by minimizing negation cost we may increase the overall grounding time because our re-writing increases the join cost. However, minimum join order is NP-hard (see, e.g., [3]), so it is unlikely that we can efficiently minimize a cost function which accurately reflects join cost. As the experiments reported in Section 6 indicate, our heuristic tends to significantly reduce grounding time in practice.

We define the cost of constructing the complement of an extended hidden X-relation $\mathscr{R}_Y$ to be $|A|^{|X\setminus Y|}$ where A is the domain. This is a reasonable approximation of the time to construct the complement, because doing so requires that we visit every instantiation of $X\setminus Y$, and do a small constant amount of work for each. We then define the negation cost of a formula, with respect to a structure $\mathscr{A}$, as the sum of the costs of the complement operations carried by the **B** grounder.

The cost of the complement operation corresponding to the negation of a subformula ϕ is primarily determined by the number of variables that are *not* hidden in the answer to ϕ constructed by the **B** grounder. These are the variables that are free in ϕ and also occur as arguments in atomic subformula $P(\bar{x})$ of ϕ, where P is an instance predicate. (Variables that occur only as arguments to predicate symbols of the expansion vocabulary remain hidden.) Let $nhv(\phi)$ denote this set of variables. The cost of complementing the answer to ϕ is essentially the size of a universal relation of arity $|nhv(\phi)|$. Thus, we define the function $\text{Size}_{\mathscr{A}}(\phi)$ to be $|\mathscr{A}|^{|nhv(\phi)|}$.

Now, we define the *negation cost* of formula ϕ with respect to structure $\mathscr{A}$, denoted $\text{Cost}_{\mathscr{A}}(\phi)$, as:

$$\text{Cost}_{\mathscr{A}}(\phi) = \begin{cases} 0 & \text{if } \phi \text{ is atomic,} \\ \text{Size}_{\mathscr{A}}(\phi) + \text{Cost}_{\mathscr{A}}(\alpha) & \text{if } \phi \text{ is } \neg\alpha, \\ \text{Cost}_{\mathscr{A}}(\alpha) + \text{Cost}_{\mathscr{A}}(\beta) & \text{if } \phi \text{ is } \alpha\{\wedge,\vee\}\beta, \\ \text{Cost}_{\mathscr{A}}(\alpha) & \phi \text{ is } \forall\bar{x}\alpha \text{ or } \exists\bar{x}\alpha. \end{cases}$$

The negation cost is the number of tuples that are visited by the B grounder *while performing complement operations*. (Thus, the cost of atoms is zero.)

Our goal is to produce a formula ψ equivalent to ϕ but with minimum negation cost, by transforming ϕ according to standard equivalences. To be precise, we define the set $\text{Rewitings}(\phi)$ to be the set of formulas which is the closure of set $\{\phi\}$ under application of De Morgan's laws and the equivalences $(\forall x\,\alpha) \equiv (\neg\exists x\,\neg\alpha)$, and $(\exists x\,\alpha) \equiv (\neg\forall x\,\neg\alpha)$. To ground ϕ, we will apply a formula-rewriting function, $FR_{\mathscr{A}}(\psi)$, that maps ϕ to the formula in $\text{Rewitings}(\phi)$ with minimum negation cost.

The size of the set $\text{Rewitings}(\phi)$ is easily seen to be exponential in the size of ϕ, but we can compute $FR_{\mathscr{A}}(\phi)$ efficiently by dynamic programming. Algorithm (1) computes the minimum cost of a formula in $\text{Rewitings}(\phi)$. That is, $MinCost_{\mathscr{A}}(\psi) = \text{Cost}_{\mathscr{A}}(FR_{\mathscr{A}}(\psi))$ for each subformula ψ of ϕ.

```
Input: Subformula $\psi$
Output: Minimum costs $P,N$ for $\psi,\neg\psi$ respectively.
/* Throughout, $P_\alpha$ and $N_\alpha$ denote the values returned by
   $MinCost_{\mathscr{A}}(\alpha)$.                                    */
if $\psi \equiv \alpha\wedge\beta$ or $\psi \equiv \alpha\vee\beta$ then
    $P \leftarrow min(P_\alpha + P_\beta, Size_{\mathscr{A}}(\psi) + N_\alpha + N_\beta)$ ;
    $N \leftarrow min(Size_{\mathscr{A}}(\psi) + P_\alpha + P_\beta, N_\alpha + N_\beta)$ ;
else if $\psi \equiv \neg\alpha$ then
    $P \leftarrow min(Size_{\mathscr{A}}(\psi) + P_\alpha, N_\alpha)$ ;
    $N \leftarrow min(P_\alpha, Size_{\mathscr{A}}(\psi) + N_\alpha)$ ;
else if $\psi \equiv \forall\bar{x}\alpha$ or $\psi \equiv \exists\bar{x}\alpha$ then
    $P \leftarrow min(P_\alpha, Size_{\mathscr{A}}(\psi) + N_\alpha)$ ;
    $N \leftarrow min(Size_{\mathscr{A}}(\psi) + P_\alpha, N_\alpha)$ ;
else if $\psi$ is atomic then $\langle P,N\rangle \leftarrow \langle 0, Size_{\mathscr{A}}(\psi)\rangle$;
return $\langle P,N\rangle$;
```

Algorithm 1. The algorithm $MinCost_{\mathscr{A}}(\psi)$ giving minimum costs of computing answers to ψ and $\neg\psi$.

The algorithm to produce the formula $FR_{\mathscr{A}}(\phi)$, is a simple variant of Algorithm (1), in the usual manner for dynamic programming. The algorithm runs in time $O(|\phi|)$.

Proposition 3. *1) Any reduced grounding of $FR_{\mathscr{A}}(\phi)$ wrt $\mathscr{A}$ is a reduced grounding of ϕ wrt $\mathscr{A}$;*
2) The formula in Rewitings(ϕ) *with minimum negation cost can be found in time $O(|\phi|)$.*

Definition 8. ***Formula Re-writing Method (FR):*** *To ground a sentence ϕ wrt $\mathscr{A}$, we compute $FR_{\mathscr{A}}(\phi)$, and then ground $FR_{\mathscr{A}}(\phi)$ wrt $\mathscr{A}$ using the B grounder.*

4 T/F Relational Algebra

Another way to tackle the negation problem is to modify the algebra so that the complement operation can be cheap. Here, we do this by adding a new relation type. We call extended X-relations as defined previously F relations, and the new relation T relations. Absence of a tuple from an F relation is equivalent to being paired with the formula $\perp$, whereas absence from a T relation is equivalent to being paired with $\top$. To construct the complement of an extended X-relation $\mathscr{R}$, we negate each formula occurring in a pair in $\mathscr{R}$, and then change the type of $\mathscr{R}$ (to T if it was F, and vice versa). Thus, the complement operation is linear time, and does not change the number of tuples. All the other operations are adapted to this setting, and in practice their application tends to generate smaller tables than when using the algebra with only F relations.

Definition 9 (T and F relations). *A* default-false extended X-relation *(F relation) is an extended X-relation as defined in Definition 3, extended with a flag that is set to False. A* default-true extended X-relation *(T relation) is an extended X-relation as defined in Definition 3, extended with a flag that is set to True, and with $\delta_{\mathscr{R}}$ defined as:*

$$\delta_{\mathscr{R}}(\gamma) = \begin{cases} \psi & \text{if } \langle \gamma, \psi \rangle \in \mathscr{R}, \\ \top & \text{if there is no pair } \langle \gamma, \psi \rangle \in \mathscr{R}. \end{cases}$$

Rather than explicitly identifying the type of every extended X-relation, we often simply write $\mathscr{R}^F$ if $\mathscr{R}$ is an F relation, and $\mathscr{R}^T$ if $\mathscr{R}$ is a T relation. When the type is clear from the context, or does not matter, we may omit the superscript. The definition of an answer to formula ϕ wrt $\mathscr{A}$ of Definition 4 applies to both T and F extended X-relations.

The operations on F tables to produce F tables are just those for regular extended X-relations, as defined in Definition 5. There are many possible additional operations when we consider both T and F relations. For example, a join may be applied to two F relations, two T relations, or one of each, and in each case we may choose to produce either an F table or a T table. We have adopted a simple heuristic strategy: For each choice of operation and the types of its argument(s), we make a fixed choice to produce either a T or F table. The full set of operations used is as follows.

Definition 10 (Algebra for T and F relations). *For each operation of Definition 5, except complement, we have an equivalently defined operation mapping F relations to F relations. In addition, we have the following. Let $\mathscr{R}^F$ and $\mathscr{R}^T$ be extended X-relations and $\mathscr{S}^F$ and $\mathscr{S}^T$ extended Y-relations, both over domain A. Then*

1. *$\neg\mathscr{R}^F$ is the extended X-relation*

$$\{(\gamma,\psi) \mid \gamma : X \to A, \gamma \in \mathscr{R}^F \text{ and } \psi = \neg\delta_{\mathscr{R}}(\gamma)\}.$$

$\neg\mathscr{R}^T$ is defined dually.

2. *(a) $\mathscr{R}^F \bowtie \mathscr{S}^T$ is the extended $X \cup Y$-relation*

$$\{(\gamma,\psi) \mid \gamma : X \cup Y \to A,\ \gamma|_X \in \mathscr{R}^F \text{ and } \psi = \delta_{\mathscr{R}}(\gamma|_X) \wedge \delta_{\mathscr{S}}(\gamma|_Y)\}$$

(b) $\mathscr{R}^T \bowtie \mathscr{S}^T$ is the extended $X \cup Y$-relation

$$\{(\gamma,\psi) \mid \gamma : X \cup Y \to A,\ (\gamma|_X \in \mathscr{R}^T \text{ or } \gamma|_Y \in \mathscr{S}^T) \text{ and } \psi = \delta_{\mathscr{R}}(\gamma|_X) \wedge \delta_{\mathscr{S}}(\gamma|_Y)\}$$

3. *(a) $\mathscr{R}^T \cup \mathscr{S}^T$ is the extended $X \cup Y$-relation*

$$\{(\gamma,\psi) \mid \gamma : X \cup Y \to A,\ \gamma|_X \in \mathscr{R}^T, \gamma|_Y \in \mathscr{S}^T \text{ and } \psi = \delta_{\mathscr{R}}(\gamma|_X) \vee \delta_{\mathscr{S}}(\gamma|_Y)\}$$

(b) $\mathscr{R}^T \cup \mathscr{S}^F$ is the extended $X \cup Y$-relation

$$\{(\gamma,\psi) \mid \gamma : X \cup Y \to A,\ \gamma|_X \in \mathscr{R}^T \text{ and } \psi = \delta_{\mathscr{R}}(\gamma|_X) \vee \delta_{\mathscr{S}}(\gamma|_Y)\}$$

4. *The Y-*projection *of $\mathscr{R}^T$, denoted $\pi_Y(\mathscr{R}^T)$, is the extended Y-relation:*

$$\{(\gamma,\psi) \mid \gamma' \in \mathscr{R} \text{ for every } \gamma' \text{ with } \gamma'|_Y = \gamma \text{ and } \psi = \bigvee_{\{\gamma' \in \mathscr{R} \mid \gamma'|_Y = \gamma\}} \delta_{\mathscr{R}}(\gamma')\}$$

5. *The Y-*quotient *of $\mathscr{R}^T$, denoted by $d_Y(\mathscr{R}^T)$, is the extended Y-relation*

$$\{(\gamma,\psi) \mid \gamma = \gamma'|_Y \text{ for some } \gamma' \in \mathscr{R}^T \text{ and } \psi = \bigwedge_{\{\gamma' \in \mathscr{R} \mid \gamma'|_Y = \gamma\}} \delta_{\mathscr{R}}(\gamma')\}.$$

Proposition 4. *Suppose that $\mathscr{R}_1^F, \mathscr{R}_2^T, \mathscr{R}_3^T$ are answers to ϕ_1, ϕ_2, ϕ_3, respectively, all wrt structure $\mathscr{A}$. Then*

1. *$\neg\mathscr{R}_1^F$ is an answer to $\neg\phi_1$ wrt $\mathscr{A}$, and $\neg\mathscr{R}_2^T$ is an answer to $\neg\phi_2$ wrt $\mathscr{A}$.*
2. *(a) $\mathscr{R}_1^F \bowtie \mathscr{R}_2^T$ is an answer to $\phi_1 \wedge \phi_2$ wrt $\mathscr{A}$.*
   (b) $\mathscr{R}_2^T \bowtie \mathscr{R}_3^T$ is an answer to $\phi_2 \wedge \phi_3$ wrt $\mathscr{A}$.
3. *If Y is the set of free variables of $\exists\bar{z}\phi_2$, then $\pi_Y(\mathscr{R}_2^T)$ is an answer to $\exists\bar{z}\phi_2$ wrt $\mathscr{A}$.*
4. *If Y is the set of free variables of $\forall\bar{z}\phi_2$, then $d_Y(\mathscr{R}_2^T)$ is an answer to $\forall\bar{z}\phi_2$ wrt $\mathscr{A}$.*
5. *(a) $\mathscr{R}_2^T \cup \mathscr{R}_3^T$ is an answer to $\phi_2 \vee \phi_3$ wrt $\mathscr{A}$.*
   (b) $\mathscr{R}_1^F \cup \mathscr{R}_2^T$ is an answer to $\phi_1 \vee \phi_2$ wrt $\mathscr{A}$.

The proofs are straightforward, and generalize those for the standard version.

Definition 11. ***True/False Table Method (T/F):*** *To ground a sentence ϕ wrt $\mathscr{A}$, we construct an F table as the answer to each atomic formula, and then apply the algebra bottom-up according to the structure of the formula.*

It is, perhaps, interesting to observe that we can use a T/F algebra with all possible operations, and in polynomial time compute the optimum choice for which operation to apply at each connective. However, we are not able to use this fact to speed up grounding in practice, as any method we see to carry out this computation requires as much work as constructing a grounding. Hence our heuristic application of T and F tables.

Indeed, it can be shown both that there are cases where the B method performs better than our T/F method, and also where algebras with additional operations perform better. Example 3 illustrates this latter case. Nonetheless, as our empirical results in Section 6 illustrate, the method tends to work well in practice.

Example 3. This example shows that by using a larger set of operations for T and F tables we may improve on the performance of our T/F method. Let ϕ be the formula $\neg R(x,y) \wedge \neg S(x,y)$, and $\mathscr{A}$ be the structure with domain $A = \{1,2,3\}$ over vocabulary $\{R,S\}$, where

$$R^{\mathscr{A}} = \{(1,1),(1,2),(2,1),(2,2),(3,1),(3,2)\} \text{ and } S^{\mathscr{A}} = \{(1,3),(2,3),(3,3)\}.$$

The T/F method takes the following steps:

1. $T_1^T(\mathscr{A}) = \neg R^F(\mathscr{A})$, hence $T_1^T(\mathscr{A})$ is: $\{\ (1,1,\perp)\ ,(1,2,\perp),(2,1,\perp),(2,2,\perp), (3,1,\perp),(3,2,\perp)\ \}$
2. $T_2^T(\mathscr{A}) = \neg S^F(\mathscr{A})$, hence $T_2^F(\mathscr{A})$ is: $\{(1,3,\perp),(2,3,\perp),(3,3,\perp)\}$
3. $T_3^T(\mathscr{A}) = T_1^T(\mathscr{A}) \bowtie T_2^T(\mathscr{A})$, hence $T_3^T(\mathscr{A})$ is: $\{(1,1,\perp),(1,2,\perp),(1,3,\perp), (2,1,\perp),(2,2,\perp),(2,3,\perp),(3,1,\perp),(3,2,\perp),(3,3,\perp)\}$

However, an alternative evaluation could be:

1. $T_1^F(\mathscr{A}) = \neg R^F(\mathscr{A})$, hence $T_1^F(\mathscr{A})$ is: $\{(1,3,\top),(2,3,\top),(3,3,\top)\}$
2. $T_2^T(\mathscr{A}) = \neg S^F(\mathscr{A})$, hence $T_2^F(\mathscr{A})$ is: $\{(1,3,\perp),(2,3,\perp),(3,3,\perp)\}$
3. $T_3^F(\mathscr{A}) = T_1^F(\mathscr{A}) \bowtie T_2^T(\mathscr{A})$, hence $T_3^F(\mathscr{A})$ is: $\{\}$

5 Comparing the Methods: An Example

To illustrate the differences between the basic grounder and the methods FR and T/F, we use the following axiom from Example 1, a specification of 3-Colouring, (re-writing the implication as a disjunction)

$$\forall x \forall y\ (\neg E(x,y) \vee \neg(R(x) \wedge R(y))). \tag{1}$$

Here, $E(x,y)$ is an instance predicate and $R(x)$, for red, is an expansion predicate. We consider the steps taken by each of the three algorithms on the graph $(V;E)$ with

$$V = \{1,\ldots,5\},\ \ E = \{(1,2),(1,3),(1,4),(1,5),(2,1),(3,1),(4,1),(5,1)\}.$$

The Basic Grounder first negates $E(x,y)$ producing a relation $T_1(x,y)$ of size 17 with all the tuples not occurring in $E(x,y)$. Then, from the subformula$\neg(R(x) \wedge R(y))$ produces a relation $T_2(x,y)$ with both variables x and y hidden (because they occur only in expansion predicates). T_2 contains a single pair consisting of the empty tuple and formula $\neg(R(x) \wedge R(y))$. The relations T_1 and T_2 are unioned to produce $T_3(x,y)$. T_3 contains any tuple occurring in either T_1 or T_2. Since x and y are hidden in T_2, it implicitly contains all possible combinations of x and y. Thus, T_3 contains 25 tuples where 17 of them (the ones present in T_1) are paired with formula $\top$ and the others are paired with formula $\neg(R(m) \wedge R(n))$ (with appropriate constant symbols m and n). Then, quotient operator is applied to T_3 to give us a formula (which is then converted to CNF).

The Formula Rewriting method first converts formula (1) to the following formula which minimizes the negation cost:

$$\neg\exists x \exists y\ (E(x,y) \wedge (R(x) \wedge R(y))) \tag{2}$$

The basic grounder is run on the rewritten formula. The first operation is a join to compute an answer to $R(x) \wedge R(y)$, which produces the extended relation $T_1(x,y)$ with both x and y hidden. It has a single pair consisting of the empty tuple and formula $R(x) \wedge R(y)$. Then, E and T_1 are joined to form T_2. As T_2 includes only the tuples present in both E and T_1, it only has the 8 tuples in E. Then, projections are applied producing a zero-ary relation T_3 with only one tuple. Finally, the complement of T_3 is constructed. Since there are no free variables, this operation involves only negating the formula paired with the single tuple in T_3.

The T/F tables method first complements the table for E, producing a T relation $T_1^T(x,y)$ having the same 8 tuples as in E, but the formula paired with each tuple negated (i.e., $\top$ replaced by $\bot$). Then the answer to $\neg(R(x) \wedge R(y))$ is constructed. This is the T relation $T_2^T(x,y)$ with both x and y hidden and formula $\neg(R(x) \wedge R(y))$ paired with the single empty tuple. Then, the union operation is applied on two T relations T_1^T and T_2^T, and a T relation T_3^T is generated. It contains only those tuples that appear in both T_1^T and T_2^T, and therefore contains only the 8 tuples of T_1^T. Finally the quotient is applied to T_3^T to generate the final result.

This example illustrates that the use of either FR or T/F methods can significantly reduce the size of intermediate tables produced during grounding. Such a large benefit is not obtained for all formulas, however, formulas with an instance relation guarding the universal quantifiers, as in the examle, are common.

For this example, the choice of instance structure does not change the result. The basic method takes $O(|V|^2)$ steps for grounding formula 1 while the two other methods take $O(|E|)$ steps for the same task, which is typically much smaller. However, in general the benefits obtained by the new methods are not always independent of the instance structure.

6 Experimental Evaluation

In this section we report an experimental evaluation of the methods. We have implemented a grounder based on the extended relational algebra which can be run in three modes: mode B; mode FR, and mode T/F, corresponding to the three methods of Definitions 7, 8 and 11. We present the results for four problems, Graph Colouring, Hamiltonian Cycle, Blocked Queens, and Latin Square Completion. These represent a variety of combinatorial structures, are easily modelled in a pure FO language, and illustrate the variation in the relative performance of the present grounder relative to similar grounders.

Table 1. Comparision of grounding times for the three modes of our grounder, basic (B), formula re-writing (FR) and True-False approach (T/F). We give mean times for all instances in each benchmark set, in units of $1/10$ seconds. Numbers in parentheses are the speedup factor. The best time for each problem is presented in bold.

Problem	B	FR	T/F
Graph Col.	33.7	7.5 (4.5)	**6.3** (5.3)
Ham. Cycle	992	**129** (7.7)	150 (6.6)
Blocked Queens	36.9	**3.7** (10)	4.1 (9.0)
Latin Square Comp.	273	**25.7** (10)	31.8 (8.6)

Table 1 gives the mean grounding times for each mode of operation on each of the problems. The total number of instances represented is 187, as follows: Graph colouring (17 instances); Hamiltonian Cycle (30 instances); Blocked Queens (40 instances); Latin Square Completion (100 instances). The instances are from the Asparagus repository [1]; all instances and axiomatizations are available online at http://www.cs.sfu.ca/research/groups/mxp/examples/. The times are for running on a Sun Java workstation with an Opteron 250 cpu, with 2GB of RAM, running Suse Enterprise Linux 2.6.11.4. The code is written in C++ and compiled with g++ version 3.3.5, on the same machine.

Both FR and T/F are substantially faster than B on all problems we looked at, including others not reported here. In general, we expect the T/F approach to be more powerful than FR and we attribute the slightly better performance of FR over T/F here to the fact that the implementation of FR has been somewhat optimized, while that of T/F has not been.

We also compared the speed of our grounder with other grounders for FO MX, namely GidL [10] and MXG [8], on the same problem instances. The grounding time comparison is shown in Table 2. GidL was run in two modes: with approximation on (G+A), and off (G-A). The approximation method attempts to reduce the size of grounding, and sometimes also affects the speed.

MXG and both versions of GidL failed to ground any instance of Hamiltonian Cycle using the (very standard) axiomatization we used for our grounder. To report running times for these, we modified the axioms by pushing leading quantifiers in as far as possible. (Our grounder does this implicitly.)

Table 2. Comparision of grounding times for our grounder with GidL and MXG. The columns are: GidL with approximation on (G+A), GidL with approximation off (G-A), MXG, and our new grounder with formula re-writing (FR) and T/F tables (T/F). Values are mean times for all instances of each problem, in $1/10$-ths of seconds. The best time for each problem is presented in bold.

Problem	G-A	G+A	MXG	FR	T/F
Graph-Col.	10.1	10.2	16.7	7.5	**6.3**
Ham. Cyc.	$>$1200	603	578.7	**129**	150
Bl. Queens	97.3	**1.3**	25.4	3.7	4.1
Latin Sq. C.	**16.9**	30.2	764	25.7	31.8

Remark 1. The languages of GidL and MXG allow (to different degrees) aggregate operations and inductive definitions. We used pure FO axiomatizations without these here, as the handling of these special forms is a distinct issue.

No grounder dominates in this comparison: Each of G+A, G-A, FR and T/F have minimum mean times for one problem. We also observe that FR appears relatively robust: it does not rank worse than second on any of the problems, whereas G+A ranks third once and each of the others ranks third or worse at least twice.

Our grounder with method B, in comparison with the other relational algebra based grounder MXG, is somewhat slower on three problems and somewhat faster on one. MXG includes some techniques that improve on the method B, and GidL includes a number of techniques to improve speed over naive substitution.

7 Conclusions and Future Work

We have described and implemented two speed-up techniques for grounding based on extended relational algebra. We presented an experimental evaluation, in which the methods exhibited speedup factors of 4 to 10 times. Speedups on other problems we have tried are similar, so we conclude that the methods are effective and probably useful in practice.

Future work on the methods reported here includes:

- A more sophisticated T/F table method. Under most conditions, T/F tables are strictly more powerful than B and FR, but this power is not fully exploited in our current T/F method. We plan to develop a stronger method for T/F tables, probably involving additional operations, and possibly in combination with FR.
- Strategies that effectively combine the techniques reported here with the adaptations of standard query optimization methods such as join ordering and pipelining.

We believe that extended relational algebra has great potential for developing efficient grounders. Since control is distinct from the basic operations, many algorithmic techniques may be applied, possibly in surprising combinations, and advanced database query optimization techniques can be adapted to grounding. We have described the grounding method, and the specific methods introduce in this paper, in terms of FO and reduction to propositional logic, but the ideas can be used for many other choices of languages.

Acknowledgements

The authors are grateful to the Natural Sciences and Engineering Research Council of Canada (NSERC), MITACS and D-Wave Systems for financial support. we also appreciate the anonymous reviewers' insightful comments.

References

1. The Asparagus Library of Examples for ASP Programs, `http://asparagus.cs.uni-potsdam.de/`
2. Cadoli, M., Schaerf, A.: Compiling problem specifications into SAT. Artificial Intelligence 162, 89–120 (2005)

3. Cluet, S., Moerkotte, G.: On the complexity of generating optimal left-deep processing trees with cross products. In: Y. Vardi, M., Gottlob, G. (eds.) ICDT 1995. LNCS, vol. 893, pp. 54–67. Springer, Heidelberg (1995)
4. Fagin, R.: Generalized first-order spectra and polynomial-time recognizable sets. In: SIAM-AMC Proceedings of Complexity of Computation, vol. 7, pp. 43–73 (1974)
5. Frisch, A.M., Harvey, W., Jefferson, C., Hernández, B.M., Miguel, I.: Essence: A constraint language for specifying combinatorial problems. Constraints 13(3), 268–306 (2008)
6. Gebser, M., Schaub, T., Thiele, S.: Gringo: A new grounder for answer set programming. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS (LNAI), vol. 4483, pp. 266–271. Springer, Heidelberg (2007)
7. 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), 499–562 (2006)
8. Mohebali, R.: A method for solving np search based on model expansion and grounding. Master's thesis, Simon Fraser University (2006)
9. Patterson, M., Liu, Y., Ternovska, E., Gupta, A.: Grounding for model expansion in k-guarded formulas with inductive definitions. In: Proc. IJCAI 2007, pp. 161–166 (2007)
10. Wittocx, J., Mariën, M., Denecker, M.: Grounding with bounds. In: Proc. AAAI 2008, pp. 572–577 (2008)