MXG : A Model Expansion Grounder and Solver

Raheleh Mohebali, Faraz Hach and David G. Mitchell

Computational Logic Laboratory
Simon Fraser University, Burnaby, BC, CANADA
{rmohebal,fhach,mitchell}@cs.sfu.ca

Abstract. We describe MXG, a solver for NP search problems expressed as model expansion (MX). Problems are specified in an extension of first-order logic, and solved by grounding. That is, MXG combines a high-level specification with an instance and produces a propositional formula encoding the solutions. It calls a SAT (or extended SAT) solver to find solutions. MXG is distinguished from other grounding software in its use of a grounding algorithm based on a generalization of the relational algebra.
Keywords: Model Expansion, Grounding.

1 Introduction

The task of model expansion (MX) was proposed as an appropriate formal basis for tools for solving search problems in [8], and further developed in [9]. In the MX framework, an instance is a finite σ-structure $\mathcal{A}$, and a specification is provided as a formula, ϕ, in a suitable logic, over a vocabulary $\sigma \cup \varepsilon \supsetneq \sigma$. A solution is an expansion of $\mathcal{A}$ to $\sigma \cup \varepsilon$ that satisfies ϕ. We call σ the instance vocabulary, and ε the expansion or solution vocabulary.

MXG is a model expansion (MX) solver oriented to solving NP search problems.[1] A specification for MXG is a formula in multi-sorted first-order logic (FO), with order, extended by inductive definitions, cardinality constraints. See [9] for details of the language, excluding the extension with cardinality. As FO model expansion is the same as the task of witnessing the second order (SO) quantifiers in $\exists$SO model checking, the language can express exactly the problems in NP.

MXG operates by grounding. Given a specification and an instance, it generates a propositional CNF formula (possibly extended with cardinality constraints) encoding the solutions for the instance, if any. It then calls a SAT solver (possibly extended to handle cardinality constraints), to find a satisfying assignment, from which it constructs a solution. MXG produces CNF formulas in DIMACS format, so that any standard off-the-shelf SAT solver can be used, albeit after a small amount of programming to translate the output of the SAT solver for interpretation by MXG.

The grounding algorithm of MXG is based on a generalization of the relational algebra [10], whereas all other grounding systems of which we are aware with the exception of Alloy [3], operate by direct substitution. This includes the ASP grounders LPARSE [11] and Gringo [2] and the dlv grounder[5], and those of finite model generators including ModGen[4] and Mace [1].

[1] MXG is available on Model Expansion Project website for both linux and windows systems.

Modern SAT solvers are very effective at solving many search problems, but employing them generally requires designing and implementing a reduction to SAT. MXG may be viewed as a high-level front-end for SAT solvers, allowing them to be much more easily exploited. It implements a uniform, polytime, reduction to SAT for every problem in NP. We have also found MXG to be useful as a generator of benchmark and test instances in tuning and debugging our SAT solver.

For further details on formal aspects of MX and the MXG language, as well as performance on benchmark problems, see [9]. See also [7], for an extended cases study of a problem demonstrating the tailoring of MXG specifications to produce dramatic performance improvements on a challenging benchmark set.

2 Grounding

Let ϕ be a formula over vocabulary $\sigma \cup \varepsilon$. A reduced grounding of ϕ with respect to finite σ-structure $\mathcal{A}$ is a formula ψ over ε only, such, for any structure $\mathcal{B} = (\mathcal{A}; \sigma^{\mathcal{A}}; \epsilon^{\mathcal{B}})$, $\mathcal{B} \models \phi$ iff $\mathcal{B} \models \psi$. A reduced grounding exactly defines the set of solutions for the instance $\mathcal{A}$. One may be obtained by producing a grounding and then "evaluating out" the instance vocabulary. MXG performs the grounding and evaluating out simultaneously.

The algorithm MXG uses is based on "extended relations" and a generalization of the relational algebra. An extended relation $T_{\overline{x}}$ is a table with attributes $\overline{x}$, and a reduced ground formula associated to each entry. Tuples may be represented by pairs $(\overline{a}, \psi_T(\overline{a}))$, where $\psi_T(\overline{a})$ is the ground formula associated with the tuple $\overline{a}$. If $\overline{a}$ does not appear in the table, this is equivalent to $(\overline{a}, false)$ appearing in the table.

The grounding algorithm is recursively operating on the structure of the formula being grounded, and producing an "answer" for each sub-formula. The extended relation $T_{\overline{x}}$ is the answer to formula $\phi(\overline{x})$ wrt structure $\mathcal{A}$, iff for all $\overline{a} : \overline{x} \to A$, $\psi_T(\overline{a})$ be the reduced ground formula for $\phi(\overline{a})$. The answer for an atomic formula is obtained from the given relation when the predicate involved is an instance symbol, and from the universal relation when it is an expansion symbol. The answer for a sentence is an extended relation containing only the empty tuple; the formula associated with that tuple is the reduced grounding of the sentence.

We illustrate the generalization of the relational algebra to extended relations by giving the definition of the extended join operator. The others (union, complement, projection, division) are defined similarly. If there is no entry for a particular tuple with values $\overline{a}$, it is considered to be an entry with the formula part $false$.
Def: $T_{\overline{x}} = R_{\overline{y}} \bowtie S_{\overline{z}}$, where $\overline{x} = \overline{y} \cup \overline{z}$, iff for any $(\overline{a}, \psi_T(\overline{a})) \in T_{\overline{x}}$: $(\overline{a}/\overline{y}, \psi_R(\overline{a}/\overline{y})) \in R_{\overline{y}}$, $(\overline{a}/\overline{z}, \psi_S(\overline{a}/\overline{z})) \in S_{\overline{z}}$, $\psi_T(\overline{a}) = \psi_R(\overline{a}/\overline{y}) \wedge \psi_S(\overline{a}/\overline{z})$.
$T_{\overline{x}}$ is the answer to $\phi_R(\overline{y}) \wedge \phi_S(\overline{z})$ wrt $\mathcal{A}$, where $R_{\overline{y}}, S_{\overline{z}}$ are answers to $\phi_R(\overline{y}), \phi_S(\overline{z})$ consequently with respect to structure $\mathcal{A}$.

2.1 The Basic Algorithm

The basic recursive grounding algorithm $Gnd(\phi, \mathcal{A})$ is defined by:

1. $\phi = P(\overline{x})$, P instance: returns $T_{\overline{x}}$ with tuples $\{(\overline{a}, true) : \overline{a} \in P^{\mathcal{A}}\}$,

2. $\phi = P(\overline{x})$, P expansion: returns $T_{\overline{x}}$ with tuples $\{(\overline{a}, P(\overline{a})) : \overline{a} : \overline{x} \to A\}$,
3. $\phi = \delta \wedge \psi$: $Gnd(\delta, \mathcal{A}) \bowtie Gnd(\psi, \mathcal{A})$,
4. $\phi = \delta \vee \psi$: $Gnd(\delta, \mathcal{A}) \cup Gnd(\psi, \mathcal{A})$,
5. $\phi = \neg\psi$: $\overline{Gnd(\psi, \mathcal{A})}$,
6. $\phi = \exists\overline{y}\psi(\overline{x})$: $\pi_{\overline{x}/\overline{y}} Gnd(\psi(\overline{x}), \mathcal{A})$,
7. $\phi = \forall\overline{y}\psi(\overline{x})$: $Gnd(\psi(\overline{x}), \mathcal{A})/\{\overline{y}\}$.

The complexity of Gnd is $O(n^l)$, where n is the size of the input structure and l is the length of the formula. MXG calls Gnd for each problem axiom, producing for each a ground formula. The union of these formulas is a reduced grounding of the problem axiomatization wrt $\mathcal{A}$. MXG converts this to a propositional CNF formula (which, in general, requires adding extra 'Tseitin' variables). The satisfying assignments for this formula are – after projecting out the Tseitin variables – in one-to-one correspondence with solutions of the problem. MXG records the mapping of ground atoms to propositional atoms (which in DIMACS format are integers), and uses the inverse of this mapping to produce a solution from a satisfying assignment to the propositional formula.

2.2 Grounding with hidden variables

Gnd requires a universal extended relation to be built for each expansion predicate. These relations are very big if the predicate is defined over large domains, and will take a significant computation time during the execution of Gnd. MXG uses a refinement of Gnd, *Gnd-hidden*, based on *extended-hidden* relations in which columns that amount to the inclusion of a universal relation are left implicit. An extended-hidden relation $T_{\overline{x}}^{\overline{y}}$ is an extended relation with explicit attributes $\overline{x}$ and hidden attributes $\overline{y}$. Value of hidden attributes are not assigned explicitly in tuples. $\psi_T(\overline{a})$ for each tuple $(\overline{a}, \psi_T(\overline{a}))$, $\overline{a} : \overline{x} \to A$, is a FO formula with free variables $\overline{y}$. Extended-hidden table $T_{\overline{x}}^{\overline{y}}$ is a compact representation of extended table $T_{\overline{xy}}$ with tuples $((\overline{a}, \overline{b}), \psi_T(\overline{a})(\overline{y}/\overline{b}))$ for all $\overline{b} : \overline{y} \to A$.

The algorithm computes extended-hidden relations $T_{\emptyset}^{\overline{x}}$ with one tuple $(\emptyset, P(\overline{x}))$ for each expansion $P(\overline{x})$. $Gnd - Hidden$ procedure is the same as Gnd except that it applies the relational algebra operations adapted to extended-hidden relations. The adapted definition of join is:
Def: $T_{\overline{x_t}}^{\overline{y_t}} = R_{\overline{x_r}}^{\overline{y_r}} \bowtie S_{\overline{x_s}}^{\overline{y_s}}$, where $\overline{x_t} = \overline{x_r} \cup \overline{x_s}$, $\overline{y_t} = (\overline{y_r} \cup \overline{y_s})/\overline{x_t}$, iff for any tuple $(\overline{a}, \psi_T(\overline{a})) \in T_{\overline{x_t}}^{\overline{y_t}} : (\overline{a}/\overline{x_r}, \psi_R(\overline{a}/\overline{x_r})) \in R_{\overline{x_r}}^{\overline{y_r}}$, $(\overline{a}/\overline{x_s}, \psi_S(\overline{s}/\overline{x_s})) \in S_{\overline{x_s}}^{\overline{y_s}}$, $\psi_T(\overline{a}) = \psi_R(\overline{a}/\overline{x_r}) \wedge \psi_S(\overline{a}/\overline{x_s})$.

2.3 Grounding refinements

MXG implements a number of refinements to improve performance, such as use of hash tables in join computation, minor formula re-writing to reduce number of Tseitin variables, and pushing outer negations inside.

3 Inductive Definitions (IDs)

The MX language is FO(ID), the extension of FO with inductive definitions under a two-valued well-founded semantics. Reductions of inductive definitions to SAT are not trivial, and the question of how to obtain good performance in a ground solver with inductive definitions is not resolved (but see [6]). In the current version of MXG, we correctly implement two fragments of the inductive definitions of FO(ID), so we can use a standard SAT solver.

– A defined predicate is computed at grounding time, if it is positive and all predicate symbols in its body are instance predicates or are effectively instance predicates (as they have already been computed during grounding). This is done by rewriting the rules as FO implication, grounding to propositional Horn clauses, and computing the minimum model in polynomial time. This model is the well-founded model of ID. The defined predicate is then treated as an instance predicate for the remainder of the grounding of this specification. **Example :** To find the distance of vertices in a graph $G = (V; Edge)$ from a particular vertex $Start \in V$ we can use the following ID:

$$\begin{gathered}\{Dist(a,b) \leftarrow a = Start \wedge b = MIN \\ Dist(a,b) \leftarrow Dist(a',b') \wedge Edge(a',a) \wedge SUCC(b',b)\}\end{gathered}$$

$Dist(a,b)$ is true iff the distance of $a, Start$ be b. MXG rewrites the rules as the following FOL formulas and calls $Gnd - Hidden$ procedure for each of them. Then the $ComputeMinModel$ is called which finds the interpretation of $Dist$.

$$\begin{gathered}\forall ab : ((a = Start \wedge b = MIN) \supset Dist(a,b)) \\ \forall ab : ((\exists a'b' : (Dist(a',b') \wedge Edge(a',a) \wedge SUCC(b',b))) \supset Dist(a,b))\end{gathered}$$

– MXG replaces other definitions with their *Completion*. The substitution is correct if the induction defined is over a well-founded order, but not in general. **Example :** To find a Hamiltonian Cycle in a graph $G = (V; Edge)$ one might use the following ID, where $HamCycle$ and $Reached$ are expansion predicates indicating subsequently which edges belong to the Hamiltonian cycle, and vertices are reachable through $HamCycle$ edges from a fixed vertex MIN:

$$\begin{gathered}\{Reached(v) \leftarrow v = MIN \\ Reached(v) \leftarrow Reached(v') \wedge HamCycle(v',v)\}\end{gathered}$$

As this ID is not of the first form, MXG computes the completion of ID and adds it to the problem axioms:

$$\forall v : (Reached(v) \Leftrightarrow (v = MIN \vee [\exists v' : (Reached(v') \wedge HamCycle(v',v))]))$$

The completion in this case, may have models that are not models of the definition.

4 Cardinality Constraints

Cardinality constraints are very useful in practice, but not conveniently represented in FO model expansion. MXG together with a SAT+Cardinality solver provides some simple cardinality options. A cardinality formula $\forall \overline{x} : \odot(n; \overline{y}; \phi(\overline{x}, \overline{y}))$ in MXG where $\odot$ is one of the $UB, LB, CARD$ states that for any choice of $\overline{x} : |\overline{y} : \phi(\overline{x}, \overline{y})|$ is less than,

greater than, or equal to n respectively for $UB, LB, CARD$. There is no standard format for propositional cardinality clauses. MXG uses the MXC [2] , the SAT+Cardinality solver for solving the cardinality constraints. $\#\ lb\ ub\ l_1\ \ldots\ l_n$ is a cardinality clause in MXC. It bounds the number of true literals from the set $\{l_1, \ldots, l_n\}$ to be at least lb and at most ub.
To ground the formula $\forall\overline{x} : UB(n; \overline{y}; \phi(\overline{x}, \overline{y}))$, a propositional cardinality clause:

$$\#\ 0\ n\ [\phi(\overline{a}, \overline{b_1})]\ \ldots\ [\phi(\overline{a}, \overline{b_k})]$$

is generated for each $\overline{a} : \overline{x} \rightarrow A$, where $[\phi(\overline{a}, \overline{b_i})]$ is the propositional variable assigned to $\phi(\overline{a}, b_i : \overline{y} \rightarrow A)$. The lower bound, upper bound values in the cardinality clause are set accordingly for $LB, CARD$. Then $Gnd - Hidden$ is called for each each FOL sentence $\phi(\overline{a}, \overline{b_i})$ with the additional constraint that a propositional variable $[\phi(\overline{a}, \overline{b_i})]$ is assigned to the whole sentence.

5 Future work

MXG is in its early stages of development, but already performs quite well. Future work includes:

- Handling general inductive definitions,
- Improved methods for implementing the relational algebra operations,
- Adding arithmetic, and more general use of cardinality and other aggregates.

References

1. K. Claessen and N. orensson. New techniques that improve mace-style finite model finding. CADE-19, Workshop W4. Model Computation Principles, Algorithms, Applications, 2003.
2. Martin Gebser, Torsten Schaub, and Sven Thiele. Gringo : A new grounder for answer set programming. In *LPNMR*, pages 266–271, 2007.
3. Daniel Jackson. Automating first-order relational logic. In *Proc. ACM SIGSOFT Conf. Foundations of Software Engineering*, pages 130–139, 2000.
4. Sun Kim and Hantao Zhang. Modgen: Theorem proving by model generation. In *National Conference on Artificial Intelligence*, pages 162–167, 1994.
5. Nicola Leone, Simona Perri, and Francesco Scarcello. Backjumping techniques for rules instantiation in the dlv system. In *NMR*, pages 258–266, 2004.
6. Maarten Mariën, Rudradeb Mitra, Marc Denecker, and Maurice Bruynooghe. Satisfiability checking for pc(id). In *LPAR*, pages 565–579, 2005.
7. David Mitchell, Faraz Hach, and Raheleh Mohebali. Faster phylogenetic inference with mxg. In *Proc. of LPAR*, 2007.
8. David Mitchell and Eugenia Ternovska. A framework for representing and solving NP search problems. In *Proc. of the 20th National Conf. on Artif. Intell. (AAAI)*, pages 430–435, 2005.
9. David Mitchell, Eugenia Ternovska, Faraz Hach, and Raheleh Mohebali. Model expansion as a framework for modelling and solving search problems. Technical Report TR 2006-24, School of Computing Science, Simon Fraser University, December 2006.
10. Murray Patterson, Yongmei Liu, Eugenia Ternovska, and Arvind Gupta. Grounding for model expansion in k-guarded formulas. In *Search and Logic: Answer Set Programming and SAT*, 2006.
11. Tommi Syrjnen. Lparse 1.0 user's manual, 1998. http://www.tcs.hut.fi/Software/smodels/.

[2] MXC is available on Model Expansion project website.