

# The eXtended Least Number Heuristic

Gilles Audemard and Laurent Henocque

LSIS equipe INCA - LIM  
Centre de Mathématiques et d'Informatique  
39, Rue Joliot Curie - 13453 Marseille cedex 13 - France  
Tel: 33 4 91 82 85 16 - Fax : 33 4 91 11 36 02  
email: {audemard, henocque}@lim.univ-mrs.fr

**Abstract.** This paper presents an algorithm, XLNH, to generate finite models of first order equational theories. Unlike conventional methods, which focus on using as few individual constants as possible to preserve symmetries, XLNH heuristically selects then fully generates the functions that appear in the problem, using a weighted directed graph of functional dependency. One key issue here is to constructively generate isomorphic partial models then further exploit the resulting symmetries. This algorithm proves very efficient on problems involving a unary bijective function  $f$  (like the additive inverse in a group or ring theory). When such a bijection is fully instantiated, XLNH statically exploits remaining isomorphic subspaces. These ideas are implemented using the public domain SEM software framework, and give order of magnitude improvements on many problems. These results are interesting on their own but potentially generalize to many practical CSP applications.

## 1 Introduction

Equational theories provide a great number of difficult problems. Zhang in [9] defines a set of problems which can form a challenge of finite model search systems. Several open mathematic problems were solved with different approaches: FALCON [10], FINDER [6], MGTP-G [3], LDPP, SATO [8], FMC [5] and MACE [4].

An equational theory is a set of axioms: first order logic formulas involving equality (e.g.  $\forall x, \forall y, \forall z : h(f(x, y)) = f(z, x)$ ). We consider here theories in which all the variables are universally quantified. Finding a finite model for such a theory amounts to finding an interpretation of functional symbols over a finite domain  $D_n$  which satisfies all axioms. The existence of a model demonstrates the consistency of the theory. The existence of a counter model may refute a conjecture.

Finding a model of an equational theory can be viewed as a special kind of constraint program, where the constraints are highly symmetrical. Symmetries arise because all constraints are universally quantified. Known approaches to finite model search have explored ways to tackle those symmetries. MGTP-G [3] uses ad hoc axioms to filter out some symmetries statically. FALCON [10],

and SEM [11] use a dynamic cut and heuristic procedure (LNH: Least Number Heuristic) to avoid exploring symmetrical subspaces during search. On the other hand, many constraint programs of practical or industrial interest exhibit a subproblem having functional semantics.

Our approach generalizes the LNH heuristic to avoid exploring isomorphic subspaces. The new heuristic can be used with many difficult problems, and gives impressive performance improvements in all cases.

The paper is organized as follows: section 2 defines equational theories. Section 3 describes the model equivalence proposition. The basic principles of the enumeration procedure are discussed in section 4. In section 5 we describe the function selection strategy. Experimental results are listed in section 6. Section 7 gives a conclusion.

## 2 Equational theories

### 2.1 Syntax

We use a subset  $\mathcal{L}$  of first order logic, without existential quantifiers, with equality as the only predicate  $\{=\}$ . In  $\mathcal{L}$ , all the variables are universally quantified. The disequality symbol  $\{\neq\}$  denotes the negation of equality. The set of variable names is  $\{x, y, z, x_1, x_2 \dots\}$ . Constants are either integers from the set  $\{0, 1, 2 \dots\}$  or identifiers (most often a letter from the set  $\{a, b, c, k, k_1, k_2 \dots\}$ ). A functional symbol can be any identifier not ambiguous with one of the previous categories, most often a letter from the set  $\{f, g, h, \dots r, s\}$ . A term is recursively built upon functional symbols, variable names and constants.

$$\begin{aligned}
 h(x, 0) &= x \\
 h(0, x) &= x \\
 h(x, g(x)) &= 0 \\
 h(g(x), x) &= 0 \\
 h(h(x, y), z) &= h(x, h(y, z)) \\
 h(x, y) &= h(y, x)
 \end{aligned}$$

**Fig. 1.** Abelian Group Axioms

Since all variables are universally quantified, universal quantifiers are usually omitted in the axioms for simplicity. Figure 1 illustrates the possibilities offered by the language.  $\mathcal{L}$  is rich enough to formulate the axioms of mathematical objects like abelian groups or unit rings. Because  $\mathcal{L}$  has only one predicate, equality, sets of  $\mathcal{L}$  axioms are commonly called "equational theories". It is of considerable interest to mathematicians to prove or refute the existence of finite structures satisfying axioms in  $\mathcal{L}$ . Hence  $\mathcal{L}$  is at the same time an excellent experimentation basis and a field of application. The concepts introduced in this paper can be extended to richer languages like the many sorted first order language used as input to the first order finite model generator SEM [11].

## 2.2 Semantics

We use traditional naming conventions in the field of CSP-based finite model generation (FALCON [10], SEM [11]). Without loss of generality, individuals are taken from the set  $N = \{0, 1, 2, \dots\}$  of natural numbers. Since we are only interested in finite models, we interpret a theory  $T$  in  $\mathcal{L}$  on a finite set  $D_n = \{0, 1, 2, \dots, n - 1\}$ . Constants (integers) are interpreted as themselves. We call a *cell* the *ground term*  $f(e_1, \dots, e_k)$  where all  $e_i$  belong to  $D_n$ . Cells map to constraint variables in the associated constraint problem.  $D_n$  is called the *domain* of these variables. The members of  $D_n$  are called *individuals*. An *interpretation*  $I_n$  (or simply  $I$ ) of a theory  $T$  maps each cell to a value from  $D_n$ . The resulting structure defines an operation table for every function that appears in  $T$  (for instance the set:  $\{h(0, 0) = 0, h(0, 1) = 2, h(0, 2) = 3 \dots\}$ ). A *model*  $I$  of order  $n$  of a theory  $T$  is an interpretation on  $D_n$  which satisfies all the theory axioms.

Let  $f$  be a function, and  $I$  an interpretation. We naturally define the interpretation  $I_f$  of  $f$  as the restriction of  $I$  to  $f$  cells. We often use  $f(e_1, \dots, e_k)$  to denote  $I(f(e_1, \dots, e_k))$  when not ambiguous. Let  $g$  be a unary function, we may also use  $g^i(x)$  to denote  $\underbrace{I(g(I(g(\dots(x))))))}_i$

## 2.3 A CSP approach to model generation

An equational theory can be viewed as a special kind of constraint program, a triple  $(V, D, C)$  where  $V$  is the set of constraint variables,  $D$  is the domain (or set of possible values) for these variables, and  $C$  is a set of constraints, i.e. relations listing possible combinations of variables values.

Here, the set  $V$  of variables is the set of function cells and the domain  $D$  is the set  $D_n$ . Different approaches exist to implement the constraints (more efficiently than as extensive lists of compatible tuples). Enumerative model generators usually rely upon constraint propagation algorithms, with a tradeoff between propagation efficiency (i.e. the completeness of the decisions made by the propagation algorithm alone) and the cost of maintaining the associated data structures. FMSET [2] experimented using boolean propagation and a clause flattening technique (to achieve ultimate propagation efficiency), at the expense of additional memory costs. SEM (the public domain tool we based our experiments upon) uses the terminal instances of the axioms and propagates newly known values of function cells upwards in the structure. SEM compensates the loss of most downward propagations by more concise and efficient data structures and indexing. A potentially useful source of (non symmetry aware) improvement of the finite model generation may be achieved using lookahead strategies as shown in [1].

Before the search starts, SEM generates all the terminal instances corresponding to every axiom in the theory. For instance, the axiom  $h(x, g(x)) = 0$  (group inverse) expands to  $h(0, g(0)) = 0, h(1, g(1)) = 0, h(2, g(2)) = 0 \dots$ . These terminal axioms are stored in memory using a pointer based representation that allows for fast upward propagation. Whenever a leaf cell value becomes known

(e.g.  $g(0)$ ), its actual value is substituted in all the constraints where it appears (which may generate a new cell value: here  $h(0, 0) = 0$ ), and the process repeats to fix point or failure as long as new cell values are introduced.

### 3 Model isomorphism

When building an equational theory model, some isomorphic branches in the search tree can be cut by observing that all individuals  $i$  from  $D_n$  that were not used as a cell index or as a cell value in previous choice points are interchangeable. This intuition led to the implementation of the Least Number Heuristic in FALCON ([10]), a program that solved several open problems for the first time.

Because equational theories are highly symmetrical, the LNH alone does not cut all unwanted search branches. This research focuses on that issue, exploring ways to retrieve part of the original symmetries, even after all individuals have been used. Most equational theories involve a unary bijection (as in group or ring axioms), or can be adapted to involve one (like in quasi groups). This section proves a proposition that leads to an improved model generation procedure: the search starts by generating a model of a unary function if it exists. After this model was computed, the search can proceed from a state where remaining symmetries can be deterministically suppressed and thus require no dynamic tests.

**Definition 1.** *Let  $T$  be a theory,  $I$  an interpretation,  $E$  a subset of  $D_n$ , and  $f$  a (unary) functional symbol. We define  $f(E)$  as the set  $\{I(f(e)) | e \in E\}$ . As usual, we also define  $f^{-1}(E)$  as the set  $\{e \in D_n | I(f(e)) \in E\}$ .*

**Definition 2.** *Let  $T$  be a theory,  $g$  a unary function and  $I_g$  an interpretation of  $g$  on  $D_n = \{0, \dots, n-1\}$ . An inclusion minimal subset  $c$  of  $D_n$  such that  $g(c) = c$  is called a cycle. An element  $i \in D_n$  appears in at most one cycle, called  $c_i$ . We define the size  $size(c)$  of a cycle  $c$  as  $|c| - 1$ .*

**Definition 3.** *Let  $g$  be a unary function and  $I_g$  an interpretation of  $g$  on  $D_n$ . The inclusion maximal subset  $D_{I_g}$  of  $D_n$  such that  $g(D_{I_g}) = D_{I_g} = g^{-1}(D_{I_g})$  is the bijective restriction of  $g$ .*

Note that such a bijective restriction is not the union of all the cycles, except when  $I_g$  interprets  $g$  as a bijection. In that case, we even have  $D_{I_g} = D_n$ .

*Example 1.* Let  $g$  be a unary function under the following interpretation  $I_g$ .

$$\begin{array}{c|cccccccccccc} g & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ \hline & 1 & 2 & 2 & 3 & 4 & 6 & 5 & 8 & 7 & 9 & 9 & 1 \end{array}$$

Under the interpretation  $I_g$ , the elements of  $D_{I_g}$  (here the set  $\{3, \dots, 8\}$ ) belonging to cycles of equal sizes remain interchangeable (e.g. 5 and 7). Obviously however, the individual 2 is not interchangeable with 3.

These intuitions lead to the proposition 1 below.

**Proposition 1.** *Let  $T$  be an axiom system with a unary function  $g$ , and  $I_n$  a model of  $T$ . Let  $h$  be a function in  $T$  ( $h \neq g$ ),  $h(i_1, \dots, i_k)$  a cell, and  $v \in D_{I_g}$  such that  $I_n(h(i_1, \dots, i_k)) = v$  and  $v \notin c_{i_j}$  for all  $i_j$ . Then for every  $w \neq v$  s.t.  $|c_w| = |c_v|$  and  $w$  belongs to none of all  $c_{i_j}$  there exists an isomorphism transforming  $I_n$  to a model  $I'_n$  of  $T$  in which  $I'_n(h(i_1, \dots, i_k)) = w$ .*

*Proof.* Let  $c_v$  and  $c_w$  be the (non empty) cycles of  $v$  and  $w$ . There are two cases:

- $c_v \neq c_w$ : let  $\sigma : D \mapsto D$  be the isomorphism equal to the identity everywhere but on  $c_v$  and  $c_w$  which maps  $g^i(v)$  to  $g^i(w)$  and  $g^i(w)$  to  $g^i(v)$  for all  $i$  in  $[0..|c_v|)$ .
- $c_v = c_w = c$ : let  $k$  be the isomorphism equal to the identity everywhere but on  $c$  which maps  $g^i(v)$  to  $g^i(w)$  for all  $i$  in  $[0..|c|]$ .

By definition,  $\sigma$  is such that  $\sigma(g(i)) = g(\sigma(i))$ .  $\sigma$  naturally extends to a model isomorphism by mapping any ground assignment  $h(i_1, \dots, i_n) = v$  (where  $h \neq g$ ) to  $h(\sigma(i_1), \dots, \sigma(i_n)) = \sigma(v)$  so that  $\sigma(h(i_1, \dots, i_n)) = h(\sigma(i_1), \dots, \sigma(i_n))$  for all  $i_1, \dots, i_n$  in  $D_n$ . These conditions, together with the fact that  $\sigma$  is bijective and that all universally quantified axioms are valid under  $I_n$ , ensure that every terminal instance of any axiom  $t_1 = t_2$  of the theory  $T$  is valid under  $\sigma(I_n)$ .  $\square$

*Example 2.* Assume we want to generate abelian groups (cf. figure 1 axioms) of order 5. Let  $I_n$  be a model of AG that interprets  $g$  as  $I_g$  below:

$$\begin{array}{c|cccc} g & 0 & 1 & 2 & 3 & 4 \\ \hline & 0 & 1 & 2 & 4 & 3 \end{array}$$

If  $I_n$  interprets  $h(1, 2)$  as 3, we know that there exists an isomorphic model  $I'_n$  where  $h(1, 2) = 4$ . This property can be used in the enumeration procedure to avoid exploring symmetrical search spaces.

## 4 Enumeration procedure

Proposition 1 shows that some model isomorphisms remain when a partial interpretation for a unary function  $g$  has been computed. The best situation occurs if the theory axioms ensure that  $g$  is bijective, since in that case  $D_{I_g} = D_n$  and is maximal in size. This suggests to try starting the model generation by completely producing a model for a bijective function  $g$ , if it exists, to exploit the remaining symmetries further. In the rich domain of group and ring theories, the group inverse function  $g$  is not only bijective, but satisfies  $g^2(i) = i$  for all  $i \in D_n$ . Cycles in that case are of size 0 or 1, which even further reduces the number of different  $g$  interpretations.

It is easy to generate only non isomorphic (canonic) interpretations of a bijective unary function. The idea simply is to generate the function so that its cycles are increasing, or decreasing in sizes, as suggests the following proposition:

**Proposition 2.** *Let  $g$  be a unary function and  $I_g$  an interpretation of  $g$  on  $D_n$ . There exists an integer  $l$  in  $D_n$  and a permutation  $\sigma$  on  $D_n$  mapping  $I_g$  to an interpretation  $\sigma(I_g)$  such that in  $\sigma(I_g)$ :*

- $D_{I_g} = \{0, \dots, l\}$
- for every cycle  $c$ ,  $c$  elements are consecutive integers and only the highest element  $e$  in  $c$  is such that  $g(e) \leq e$ .
- for every two consecutive cycles  $c_1$  and  $c_2$ ,  $|c_1| \leq |c_2|$

This proposition is easily proved by iteratively building  $\sigma$  as the appropriate renaming on  $D$ . Because an equational theory involves constants (like the neutral element in group axioms), for completeness reasons, proposition 2 cannot be used as such in an enumeration procedure. Constants appearing in equalities must be interpreted first by the algorithm. The chosen values are not interchangeable with any other. In addition, these individuals may belong to a cycle of  $g$ , or not (in that case, the function  $g$  is not bijective). Technically, the individuals selected to interpret  $p$  constants  $k_p$  may without loss of generality satisfy  $I(k_j) < j - 1$  for  $j \in \{0..p - 1\}$ . This leads to the following proposition:

**Proposition 3.** *Let  $k_1, \dots, k_p$  be  $p$  constants,  $I_{k_i}$  their interpretation, and  $g$  a unary function. Let  $I_g$  interpret  $g$  on  $D_n$ . There exist two integers  $m$  and  $l$  in  $D_n$  ( $m \leq l$ ) and a permutation  $\sigma$  on  $D_n$  mapping  $I_g$  to an interpretation  $\sigma(I_g)$  such that in  $\sigma(I_g)$ :*

- $g^n(\{I_{k_i}\}) = \{0, \dots, m - 1\}$
- $D_{I_g} - g^n(\{I_{k_i}\}) = \{m, \dots, l\}$
- for every cycle  $c \subset \{m, \dots, l\}$ ,  $c$  elements are consecutive integers and only the highest element  $e$  in  $c$  is such that  $g(e) \leq e$ .
- for every two consecutive cycles  $c_1 \subset \{m, \dots, l\}$  and  $c_2 \subset D_{I_g}$ ,  $|c_1| \leq |c_2|$

**Definition 4.** *An interpretation  $I_g$  satisfying proposition 3 requirements is called canonic. Let  $c$  be a cycle in  $I_g$ . The smallest element  $s$  in  $c$  is called  $start(c)$ , and the highest element  $e$  in  $c$  is called  $end(c)$ .*

According to the above statements, given  $g$  a unary function and  $I_g$  a canonic interpretation, it is clear that  $size(c) = end(c) - start(c)$  for every cycle  $c$ . In the example 1, if we have one constant interpreted as 0, we have  $m = 3$ ,  $l = 8$ ,  $\{3, \dots, 8\}$  contains 4 cycles of respective lengths 0, 0, 1, 1. Note that two cycles are not in  $\{3, \dots, 8\}$ :  $g(2) = 2$  and  $g(9) = 9$ .

Model generation uses the two propositions 1 and 3 to exclude isomorphic subspaces when building models of a theory  $T$  involving a unary function  $g$ . It operates in three steps.

#### 4.1 step 0

The algorithm interprets the  $p$  constants appearing in equalities so that  $I(k_0) = 0$  and  $I(k_j) \leq I(k_{j-1}) + 1$ . This constructs a set  $\{0, \dots, q\}$  of integers ( $q \leq p$ ).

## 4.2 step 1

It then selects a function  $g$  from the problem statement and iteratively generates all its canonical models, as described in proposition 3. The idea simply is to construct  $I_g$  so that

- $g^n(\{0, \dots, q\}) = \{0, \dots, m-1\}$
- $D_{I_g} - g^n(\{0, \dots, q\}) = \{m, \dots, l\}$  for some  $m, l$  (note that there may exist cycles in  $\{0, \dots, m-1\}$ ).
- $\{m, \dots, l\}$  cycles only contain consecutive elements (note that this property cannot be ensured for the cycles that appear over  $\{0, \dots, m-1\}$ )
- for all  $i$  in  $\{m, \dots, l\}$ , either  $g(i) = i+1$  or  $i$  is the end of a cycle and thus  $g(i) = i - \text{size}(c_i)$
- the cycles in  $\{m, \dots, l\}$  are increasing in size (note again that this property cannot be ensured for the cycles that appear over  $\{0, \dots, m-1\}$ )

This procedure allows to generate only a limited number of interpretations having isomorphic bijective restriction. In the case of a bijection, only non isomorphic models are generated. For example, using this strategy, the generation of bijective functions of order 6 only produces the eleven non isomorphic models whereas using the LNH heuristic produces 32 models.

## 4.3 step 2

For every such generated  $I_g$ , the algorithm enumerates possible models for the theory using an almost standard CSP enumeration procedure, described as algorithm 1. We say that an index  $i$  has been *hit* by the algorithm when either

- the value  $i$  has been assigned to a cell
- the value of a cell  $h(i_1, \dots, i_n)$  has been chosen, or is currently under consideration, and  $i = i_j$  for some  $j$  in  $\{1, \dots, n\}$ .

We say that a cycle has been *hit* when one of its members has been *hit*. The algorithm keeps track of a high water mark called  $mdn_s$  for every cycle size  $s$ . The value  $mdn_s$  represents the end index of the highest cycle of size  $s$  that has been hit. Remember that cycles with equal sizes are consecutive. Let  $v \in D_{I_g}$ , we note  $mdn(v)$  the value  $mdn_{|c_v|}$ . By the proposition 1, when selecting a value  $v$  for a cell  $h(i_1, \dots, i_k)$ , only values smaller than  $mdn(v) + 1$  need to be tried.

After an interpretation of  $g$  has been computed, the index  $m$  identifying the start of the bijective restriction section is known, and thus all  $mdn$  values are set equal to  $\max(0, m-1)$ .

Traditional implementations of the LNH heuristic (as in [10]) use only one  $mdn$  value. They attempt to favor the heuristic application by selecting new cells so that the  $mdn$  does not change, as long as possible. To achieve this, it is enough to select cells  $h(i_1, \dots, i_k)$  with indexes smaller than  $mdn$ . If impossible, a new cell is selected that yields the smallest possible change to  $mdn$  values. In practice, it suffices to choose  $h(i_1, \dots, i_k)$  so that  $\max(i_1, \dots, i_k)$  is the smallest.

---

**Algorithm 1** The enumeration procedure

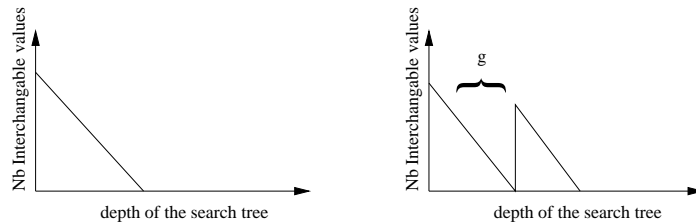
---

```
function XLNH( $S$ : set of assign.,  $F$ : set of terminal axiom):boolean;  
begin  
  forall non propagated assignments  $a$  in  $S$ , Propagate( $a, F, S$ )  
  if  $S$  contains incompatible assignments then return(false)  
  if  $F$  is empty then return(true)  
  select unbound cell  $b(i_1, \dots, i_n)$   
    (so that  $\max(i_1, \dots, i_n)$  is the smallest)  
  update  $mdn(i_j)$  for all  $i_j$   
  forall  $v \in D$  s.t.  $v \leq mdn(v) + 1$   
    if XLNH( $S \cup b = v, F$ ) then return(true)  
  return false  
end
```

---

This strategy however picks candidate cells in all existing functions (including functions that depend upon the value of others, which thus should not be taken in consideration here).

Our approach focuses on the generation of an interpretation for all functions, one after another. The algorithm thus evaluates all of a function cells before changing the function. This prevents spreading the model generation over all function symbols. During the generation of a function, we use an extended version of the LNH, that treats as equivalent the individuals belonging to the same cycle, or to cycles of equal sizes, as long as these individuals have not been "hit".

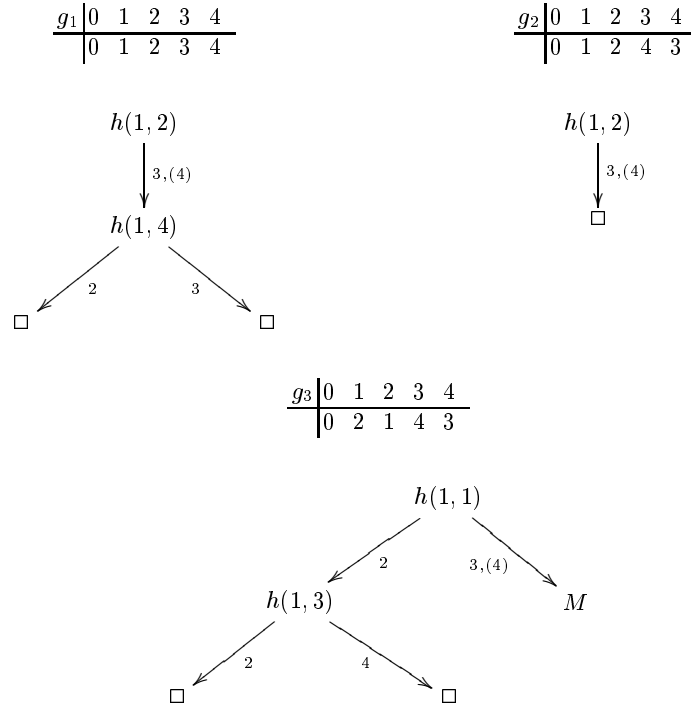


**Fig. 2.** LNH versus XLNH

The figure 2 illustrates the difference between the LNH and the XLNH heuristics. By using LNH, after a certain search tree depth is reached, all  $D_n$  values are used, and yet are no more interchangeable (as illustrated in the leftmost diagram in figure 2). By using XLNH, after a canonical interpretation of the unary function  $g$  was computed, existing cycles in the bijective restriction let some individuals become interchangeable again (as in the rightmost diagram in figure 2). Hence the XLNH heuristic allows to first compute in a deterministic way the canonical models of a unary function, then statically exploit remaining isomorphisms to prune the search. The implementation of the heuristic is thus

entirely static, and requires no complex run time tests. The only tests performed are integer comparisons, at a null cost.

*Example 3.* Figure 3 illustrates the generation of all models of order 5 abelian groups (cf. figure 1). Only three canonic interpretations of  $g$  are generated. In figure 3, bracket surrounded values are the ones suppressed from the search tree by the XLNH heuristic (by proposition 1). Missing branches are suppressed by constraint propagation. The  $\square$  symbol represents inconsistency. The symbol  $M$  represents the obtention of a model.



**Fig. 3.** The search tree for order 5 abelian groups

By looking at this diagram, we can observe that the first generated interpretation of  $g$  is the identity, all cycles having the same size zero. In that case, the rest of the search proceeds as with a LNH heuristic starting with  $mdn = 0$ . This results in suppressing many isomorphic interpretations.

## 5 Function Selection Strategy

Experimental results show that the sequence of selected functional symbols impacts on computation times. It can be clearly understood why when we observe

that some functional symbols never appear as top-level term labels in the axioms (these functions are called "pure output") while others never appear as sub-terms (they are "pure input" functions). It is obvious that pure input functions are uniquely determined when all other functions are known, and thus should not be explicitly enumerated. On the other hand, pure output functions should be generated in priority, because SEM's propagation is essentially of a bottom up kind.

The other functions appearing both as top-level terms and as sub-terms in the problem axioms are generated in a statically computed intuitive order, based on the number of times a function has a previously generated function as its input. This heuristic is computed statically (before search starts) by building a weighted directed graph from the problem axioms where nodes are labelled with functional symbols, and weighted arrows represent the number of recursive function invocation in axioms ( $f(g(\dots)\dots)$  eventually introduces the arrow from  $g$  to  $f$  or increments its weight by one). The existence of a static heuristic clearly improves program performance.

Constants deserve a distinct treatment depending whether they appear in equations or disequations. In the former case (like of the additive inverse "zero"), the constants are valuated before program starts, and result in introducing non interchangeable individuals (step zero of the algorithm). In the latter case, however, (like in an axiom introducing a counter example for associativity), the constants are better valuated once the functions where they appear are entirely generated.

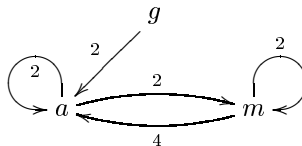
**Definition 5.** Let  $T$  be an equational theory,  $F$  its set of functional symbols (without constants) and  $C$  its set of axioms. Let  $G = (X, W)$  be the weighted and oriented dependency graph defined as follows:

- the set of vertices is isomorphic to  $F$ .
- the edge  $(f_i \rightarrow f_j)$  belongs to  $W$  if there exists an axiom in  $C$  where  $f_i$  appears as a sub-term of  $f_j$ .
- the weight of the edge  $(f_i \rightarrow f_j)$  equals the number of times there exists an axiom in  $C$  where  $f_i$  appears as a sub-term of  $f_j$ .

*Example 4.* The following theory defines the axioms of an unit ring:  $g, a$  is the group and  $m$  is the multiplicative law.

$$\begin{array}{ll}
 a(0, x) = x & a(x, 0) = x \\
 a(g(x), x) = 0 & a(x, g(x)) = 0 \\
 m(x, 1) = x & m(1, x) = x \\
 a(x, a(y, z)) = a(a(x, y), z) & m(x, m(y, z)) = m(m(x, y), z) \\
 a(m(x, y), m(x, z)) = m(x, a(y, z)) & a(m(x, z), m(y, z)) = m(a(x, y), z)
 \end{array}$$

This set of axioms yields the following dependency graph:



Note: the weighted graph construction is not semantical, and depends upon the theory syntax. Hence, two different formulations of the same problem could have different graphs. This issue is field of future research.

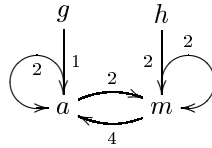
### 5.1 Function selection algorithm

The best experimental results are achieved by selecting the next function to instantiate according to the following preference order:

1. select a constant appearing in an equation (step 0)
2. select a pure output bijective unary function (step 1)
3. select a pure output unary function (step 1)
4. select a pure output n-ary function (step 2)
5. select a function that with maximal sum of weights of arrows coming from already generated functions (step 2)
6. select a constant appearing in already fully generated functions (step 2)

Note that all vertices having only input edges functionally depend upon the other functional symbols, and are uniquely determined as soon as the other functions have been entirely generated. Pure input functions are thus never generated explicitly.

*Example 5.* As an example, let us consider the theory *RNG041-1* from the TPTP problem collection [7]. We have the following dependency graph:



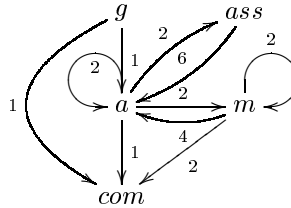
According to the previous selection strategy, we first select the constants which appear in equality equation, after  $g$  (pure input unary bijective), then  $h$  (pure input unary). Then, the choice of  $a$  before  $m$  is guided by the dependencies. Hence the instantiation order is:  $g, h, a, m$ .

**Table 1.** Comparing different orders for the *RNG041-1* problem

	$g, a, h, m$	$g, h, a, m$	$g, m, a, h$
Time	0.13	0.06	0.15
Nodes	89	879	634
Models	0	0	0

The table 1 illustrates those concerns by comparing the results obtained with the TPTP problem *RNG041-1* at order 6 using different function orderings. This example suggests a few comments: the best node complexity is achieved by selecting function  $a$  (the additive law) just after its inverse  $g$ . However, the best execution times are obtained with our function selection strategy, because many nodes (cell value choices for  $h$  in that case) trigger very immediate fails.

*Example 6.* Let us consider the theory *RNG025-8* from the TPTP problem collection [7]. We have the following dependency graph:



In this case, the statically computed ordering is the following one:  $g, a, ass, m, com$ . Three constants appear in the set of clauses as disequations literals, and are thus interpreted last.

Again, table 2 list the results obtained with different function orderings. The size of models is equal to 5.

**Table 2.** Comparing different orders for the *RNG025-8* problem

	$g, a, ass, m, com$	$g, a, m, ass, com,$	$g, ass, a, m, com$	$g, m, a, ass, com$
Time	0.14	0.33	$\geq 10$ minutes	0.78
Nodes	3 149	1 280	-	6 398
Models	3 072	1 280	-	2 048

The choice of  $g$  first is obvious, because it is pure input, unary bijective.  $com$  needs not be generated because it is pure output. Choosing  $a$  after  $g$  is natural because it has  $g$  as input (and  $com$  is discarded). Then  $ass$  should be preferred over  $m$  because the axioms involve six occurrences of  $a$  as a sub-term of  $ass$ , instead of only four in  $m$ . Again, you may observe the existence of a node count per execution time trade off. As before, the best execution times are obtained at the expense of more choice nodes, which suggests that many choices lead to very quick failures in the best option.

## 6 Experimentation

We have implemented the heuristic XLNH on the SEM software, as a heuristic variant. We thus use exactly the same data structures and propagation algorithm. Our implementation currently only handles problems involving a unary bijective function, which are numerous. SEM's source code is available at the web address [www.cs.uiowa.edu/~hzhang/sem.html](http://www.cs.uiowa.edu/~hzhang/sem.html). All results are obtained on a K6II 400Mhz with 128 Mb of RAM. We limit to two hours the maximum time to solve a problem.

We compare SEM + XLNH and classical SEM (+ LNH) on different mathematical problems: abelian groups first, then several ring problems from the

TPTP collection [7]. We have tested XLNH on a large number of these problem instances and obtained very good results, and list them for three problems *RNG041-1* (rating 0.22 -very easy-), *RNG025-8* (rating 0.67 -medium-) and *RNG030-6* (rating 1 -difficult-).

**Table 3.** Abelian Groups

Order	SEM + XLNH			SEM + LNH		
	Models	Time	Nodes	Models	Time	Nodes
32	529	168	9 769	2 295	956	421 178
33	15	28	2 769	15	1 151	466 883
34	2	239	26 077	20	1 402	481 249
35	13	41	3 477	13	1 700	490 606
36	321	375	27 975	2 142	2 345	872 374
37	1	65	4 107	1	2 848	921 379
38	2	532	39 789	22	3 525	935 527
39	17	86	4 350	17	4 263	946 669
40	282	816	39 130	2 220	5 632	1 393 433
41	1	116	5 163	+	+	+
42	42	1 247	54 822	+	+	+
43	1	154	5 396	+	+	+
44	31	1 788	58 137	+	+	+
45	180	226	6 122	+	+	+
46	2	2 481	60 281	+	+	+
47	1	361	14 096	+	+	+
48	3 345	4 446	75 905	+	+	+
49	8	492	22 976	+	+	+
50	22	6 375	232 718	+	+	+
51	21	636	25 053	+	+	+
52	+	+	+	+	+	+

We generate all models of the abelian group and only solve the satisfiability problem on the TPTP's instances. The table 3 shows the comparison between LNH and XLNH for abelian groups. Our approach is 7 times faster for even orders and 70 times faster for odd orders. Our method shows that odd order abelian group generation is much easier than that of even order abelian groups. This agrees with known results about the number of non isomorphic finite abelian group instances (it is known that every subgroup order divides the order of the group). This result shows the efficiency of XLNH. The results table 3 stops at order 52. However we can generate all odd order abelian groups up to order 63 (278 models obtained in 2470 seconds and 40 764 nodes). We can observe that XLNH always produces fewer models than LNH.

**Table 4.** some TPTP problems

Problem	Size	XLNH			LNH		
		Model	Time	Nodes	Model	Time	Nodes
<i>RNG041-1</i>	8	0	0.05	1 236	0	24	175 672
	9	0	0.06	1 673	0	123	736 625
	10	0	0.09	2 049	0	632	3 061 678
	11	0	0.09	2 497	0	2 928	12 221 898
	12	0	0.16	2 880	-	-	-
	14	0	0.22	3 725	-	-	-
	16	0	0.4	4 584	-	-	-
<i>RNG025-8</i>	8	1	19	872 609	1	135	6 965 608
	9	1	2.8	86 629	1	12	92 396
	10	1	1.6	10 435	1	41	19 276
	11	1	2.5	14 889	1	221	67 835
	12	1	8.3	148 203	1	529	244 126
	13	1	5.4	28 901	1	6 462	985 767
	14	1	11	39 383	-	-	-
	15	1	9.5	51 683	-	-	-
16	-	-	-	-	-	-	
<i>RNG030-6</i>	11	0	0.76	1 465	0	216	576 977
	12	0	28.7	83 007	-	-	-
	13	0	1.5	2 379	-	-	-
	14	0	3.5	6 250	-	-	-
	15	0	17	32 987	-	-	-
	16	0	6 826	4 039 087	-	-	-
	17	0	5	5 338	-	-	-
	18	0	278	400 055	-	-	-
19	0	8	7 375	-	-	-	

Table 4 lists the results obtained for several TPTP problems, sorted by increasing difficulty, according to the TPTP collection difficulty ratings. The XLNH heuristic proves very efficient for the RNG class instance (rings with added axioms). The *RNG041-1* problem, an easy one, is solved in less than 1 second at order 16. The table stops here because of paper size limitations. However XLNH solves *RNG041-1* at order 30 in 3.5 seconds and 10 989 nodes. The program is limited by memory. The *RNG025-8* problem shows a difficulty peak at orders being a power of two (order 8). This is expected, as in the case of abelian groups, because there exist many finite groups of order a power of two. XLNH fails at order 16 but solves the problem at order 17 easily (25 seconds). The *RNG030-6*, a currently open instance of the TPTP collection (rating 1), exhibits comparable behavior at power of two orders. Here again, because the problem involves many functions, the limitation comes from processor memory rather than combinatorial complexity.

## 7 Conclusion

We demonstrate the efficiency of exploiting the underlying structure of equational theories to generate their finite models. The existence of a pure output unary function, specially if it is bijective, allows to avoid exploring many isomorphic subspaces. Our results generalize the least number heuristic in situations when a unary function exists in the theory. This generalization proves very efficient in reducing the number of search nodes explored by the enumerator, and thus produces a lot fewer isomorphic solutions.

Themes of future work include: the extension to non bijective unary functions, integration of dynamic symmetry tests, changes in the data structures to consume less memory and obtain solutions at higher orders, exhaustive generation of canonical solutions for finite abelian groups or other theories up to unprecedented orders, the adaptation of these results to practical CSP problems involving a functional subpart (and eventually a bijection like in the Travelling Sales Person problem).

## References

1. Gilles Audemard, Belaid Benhamou, and Laurent Henocque. Two techniques to improve finite model search. In D. McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction (CADE-17)*, volume 1831 of *LNCS*, pages 302–308. Springer, June 2000.
2. Belaid Benhamou and Laurent Henocque. A hybrid method for finite model search in equational theories. *Fundamenta Informaticae*, 39(1-2):21–38, 1999.
3. Masayuki Fujita, John Slaney, and Franck Bennett. Automatic generation of some results in finite algebra. In *Proceedings of International Joint Conference on Artificial Intelligence*, pages 52–57. Morgan Kaufmann, 1993.
4. William McCune. A Davis-Putnam program and its application to finite first-order model search: quasigroup existence problems. Technical Memorandum ANL/MCS-TM-194, Argonne National Laboratories, IL/USA, 1994.
5. Nicolas Peltier. A new method for automated finite model building exploiting failures and symmetries. *Journal of Logic and Computation*, 8(4):511–543, 1998.
6. John Slaney. Finder : Finite domain enumerator. version 3 notes and guides. Technical report, Austrian National University, 1993.
7. Christian B. Suttner and Geoff Sutcliffe. The TPTP problem library - v2.1.0. Technical Report JCU-CS-97/8, Department of Computer Science, James Cook University, 15 December 1997.
8. Hantao Zhang and Mark Stickel. Implementing the Davis-Putnam method. *Journal of Automated Reasoning*, 24:277–296, 2000.
9. Jian Zhang. Problems on the Generation of Finite Models. In Alan Bundy, editor, *12th International Conference on Automated Deduction*, LNAI 814, pages 753–757, Nancy, France, June 26–July 1, 1994. Springer-Verlag.
10. Jian Zhang. Constructing finite algebras with FALCON. *Journal of Automated Reasoning*, 17(1):1–22, August 1996.
11. Jian Zhang and Hantao Zhang. SEM: a system for enumerating models. In Chris S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 298–303, 1995.