A Powerful Technique to Eliminate Isomorphism in Finite Model Search

Xiangxue Jia[1,2] and Jian Zhang[1]

[1] Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
[2] Graduate University, Chinese Academy of Sciences

Abstract. We propose a general-purpose technique, called DASH (Decision Assignment Scheme Heuristic), to eliminate isomorphic subspaces when generating finite models. Like LNH, DASH is based on inherent isomorphism in first order clauses on finite domains. Unlike other methods, DASH can completely eliminate isomorphism during the search. Therefore, DASH can generate all the models none of which are isomorphic. And DASH is an efficient technique for finite model enumeration. The main idea is to cut the branch of the search tree which is isomorphic to a branch that has been searched. We present a new method to describe the class of isomorphic branches. We implemented this technique by modifying SEM1.7B, and the new tool is called SEMD. This technique proves to be very efficient on typical problems like the generation of finite groups, rings and quasigroups. The experiments show that SEMD is much faster than SEM on many problems, especially when generating all the models and when there is no model. SEMD can generate all the non-isomorphic models with little extra cost, while other tools like MACE4 will spend more time.

Keywords: Isomorphism; scheme; symmetry breaking; LNH; DASH.

1 Introduction

Many problems from various application domains can be regarded as deciding the satisfiability of certain logical formulas. In fact, the satisfiability problem in the propositional logic, known as SAT, is a fundamental problem in computer science and artificial intelligence. Many researchers have been working on SAT, and many algorithms have been proposed to solve the problem. In the 1980's, the algorithms are typically analyzed theoretically. Recently, tool development and empirical evaluation received more attention. Several efficient SAT solvers have been implemented, such as SATO [6], MiniSat [17] and zchaff [3].

However, practical problems are often more naturally described by a set of first-order formulas. In the problem library TPTP [5], fewer than 1% of the problems are described in pure propositional logic. The satisfiability problem in the first-order logic is undecidable in general. But we can try to satisfy first-order formulas in finite domains. This problem is known as finite model generation or finite model searching. Of course, it can be transformed into SAT. But it can

© Springer-Verlag Berlin Heidelberg 2006

also be solved directly, through exhaustive search. During the past 15 years, several efficient finite model searchers have been developed, such as FINDER [11], FALCON [12], SEM [14], Mace4 [22], MACE [21] and Paradox [15]. Although these tools are quite successful in solving some open problems in mathematics, we can still do something to improve their performances. In particular, we can eliminate more symmetric subspaces in the search space. This paper describes such an attempt.

2 Preliminaries and Notations

2.1 Finite Models

A **model** of a set of first-order formulas is an interpretation that gives a value to constant and every entry of function or predicate. A finite model of size n is often defined on the domain $D_n = \{0, 1, \ldots, n-1\}$.

When the arity of a function (predicate) is no more than 2, the function (predicate) can be described by a table. We call an entry of the table a **cell**. Syntactically a cell is given by a term whose arguments are elements of the domain. For example, both $f(0,2)$ and $P(1)$ denote cells, if P is a unary predicate and f is a binary function. A model gives a value to all the cells.

A **partial model** gives values to some cells. We use *Pmod* to denote a partial model.

2.2 Basic Search Procedure

Finite model searchers often work on a set of ground formulas, which are obtained by instantiating the input formulas over the domain D_n. For example, suppose $n = 2$ and the input contains the formula $f(x, y) = x$. We get the following set of ground formulas:

$$f(0,0) = 0; \qquad f(0,1) = 0; \qquad f(1,0) = 1; \qquad f(1,1) = 1.$$

To search for a model, we can use the following backtrack search procedure:

Algorithm 1. The standard search function

```
Srh(S: set of assignments, G: set of ground formulas): Boolean;
{
  for each assignment a in S, Propagate(a,G,S);
  if S contains incompatible assignments then return(FALSE);
  if G has been satisfied then return(TRUE);
  select one unassigned cell c;
  forall v in Dn
    if Srh(S and (c=v), G) then return(TRUE);
  return FALSE;
}
```

We can regard the execution of the search procedure as a search tree. In each node of the tree, we try to choose a value for a cell. Every choice correspomds to a **decision**, and every branch of the search tree correspomds to a **decision sequence**. Each decision sequence leads to a partial model, which also includes the cell assignments derived from the sequence by constraint propagation.

The **depth** of a cell assignment is the depth of the search tree at which the cell is assigned its value.

3 Isomorphism Elimination

Two models can be isomorphic to each other. This is due to symmetries on the domain elements.

Definition 1. *A **permutation** of D_n is a one to one mapping (bijection) from D_n onto itself. Such a permutation is called a **symmetry** if it maps models to models.*

Definition 2. *A set G of ground clauses is **symmetric** with respect to a subset of D_n, if G remains the same under any permutation of that subset.*

Because symmetry leads to isomorphism, we use symmetry breaking and isomorphism elimination for the same meaning.

There are two ways of eliminating isomorphism. The first is adding constraints of isomorphism elimination, and the second is eliminating isomorphism during search.

Adding constraints of isomorphism elimination is a kind of preprocessing. It changes the original problem to a new problem by constraining the search path. Paradox [15] uses this static way. Usually it does not get the minimal search space because adding full constraints to eliminate isomorphism will cost too much time to check those constraints.

For the second way, the disadvantage is that if we do not implement a quick symmetry detection method, the time cost by eliminating isomorphism may be more than the time we save. In [9], the authors point out that the computational complexity of the symmetry detection problem is equivalent (within a polynomial factor) to that of the graph isomorphism problem.

When generating finite models of first order logic, some isomorphic branches in the search tree can be cut based on the following observation: all individual values in D_n that are not used as a cell index or as a cell value in previous decisions are interchangeable. This intuition led to the idea of the Least Number Heuristic (LNH) [12]. When implementing this technique, we can use a special variable mdn to denote the maximal designated number. All elements in the subset $\{mdn + 1, \ldots, n - 1\}$ should be interchangeable.

Definition 3. *Let $imdn$ denote the **initial maximal designated number** in the input file. If the input problem does not contain constants, $imdn = -1$.*

However, the LNH alone may not cut all unwanted search branches. And it has been extended in various ways, such as [4]. The current paper also focuses on this issue. We examine each new branch to find out whether it is isomorphic to a branch searched before, and cut all such branches.

4 Decision Assignment Scheme Heuristic

Now we describe a new isomorphism elimination technique: DASH (which stands for Decision Assignment Scheme Heuristic). All that DASH does is to eliminate branches which are isomorphic to some searched branches.

Definition 4. *There are two meanings of* **branch** *in the paper. One is the original meaning for the search tree, and the other is the partial model which is derived from the decision sequence on the branch.*

Let $Pmod_1$ stand for a branch that has been searched, and $Pmod_2$ stand for the current branch which is isomorphic to a sub-branch $Pmod_1'$ of $Pmod_1$. In this situation, we call $Pmod_1$ as **scheme branch**, and $Pmod_2$ as **iso-branch**.

Given a decision sequence, for each domain element in the sequence which is larger than $imdn$, we introduce a distinct auxiliary variable. We call these variables **scheme variables**. Usually we name a scheme variable by one letter, which stands for the sort, and use the integer corresponding to the domain element as its index.

Then we substitute each domain element by the auxiliary variable. The decision (cell assignment) after the substitution is called a **decision scheme**, and the conjunction of all such schemes is called a **scheme**. The number of decision schemes in a scheme is called the **length** of the scheme.

For example, suppose $imdn = 0$, and there is a branch of the search tree which corresponds to the decision sequence: $g(1) = 1$, $g(2) = 4$, $f(1, 2) = 3$, $f(2, 2) = 4$. We can introduce 4 scheme variables: x_1, x_2, x_3, x_4, and get the scheme $g(x_1) = x_1 \wedge g(x_2) = x_4 \wedge f(x_1, x_2) = x_3 \wedge f(x_2, x_2) = x_4$. The length of this scheme is 4.

Definition 5. *For every decision sequence α, there is a scheme denoted by $\hat{\alpha}$ and a partial model denoted by $\bar{\alpha}$ which is derived from α by through constraint propagation.*

Remarks
We should remember that distinct scheme variables can't have the same value and they can't have values smaller than or equal to $imdn$.

In this paper, we assume there is only one sort to simplify the presentation. If there are more than one sorts, we can name the scheme variables as x_1, y_2 and so on. Every different sort has a different letter.

Example 1
Consider the following axioms for the Abelian Group:

$$
\begin{gathered}
g(0) = 0 \\
g(x) \neq y \vee g(y) = x \\
f(x, 0) = x \\
f(0, x) = x \\
f(x, g(x)) = 0 \\
f(g(x), x) = 0 \\
f(f(x, y), z) = f(x, f(y, z)) \\
f(x, y) = f(y, x)
\end{gathered}
$$

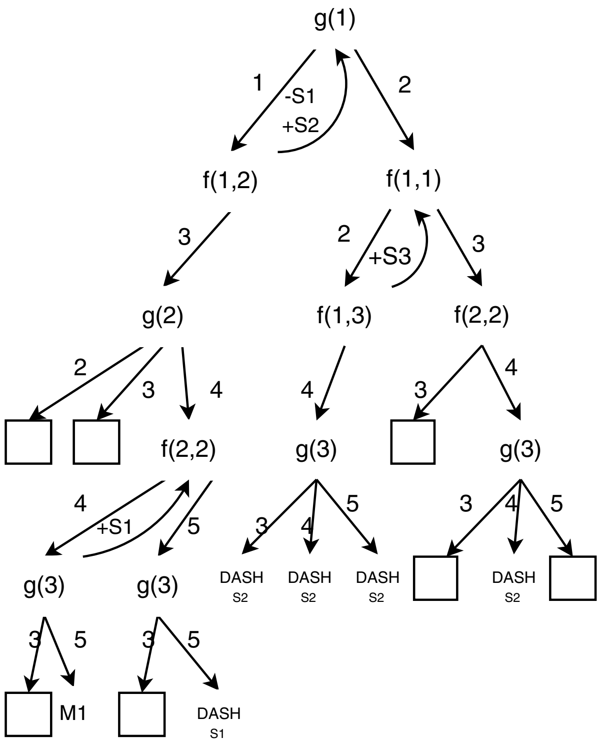

Fig. 1. The search tree for AG6

We can see that $imdn = 0$. Set the order to 6 (i.e., let $n = 6$). If a decision sequence is $g(1) = 1, g(2) = 4, f(1,2) = 3, f(2,2) = 4$, then after the substitution, we get four decision schemes, namely $g(x_1) = x_1, g(x_2) = x_4, f(x_1, x_2) = x_3, f(x_2, x_2) = x_4$. The scheme constructed from them is $g(x_1) = x_1 \wedge g(x_2) = x_4 \wedge f(x_1, x_2) = x_3 \wedge f(x_2, x_2) = x_4$.

The whole search tree using DASH is shown in Fig.1. It is found that there is only one non-isomorphic model of Abelian group of order 6.

In Fig.1, there are three schemes used in the search, namely $S1$, $S2$, $S3$:

$$S1 : g(x_1) = x_1 \wedge g(x_2) = x_4 \wedge f(x_1, x_2) = x_3 \wedge f(x_2, x_2) = x_4$$
$$S2 : g(x_1) = x_1$$
$$S3 : g(x_1) = x_2 \wedge f(x_1, x_1) = x_2$$

Though there are many other schemes that can be added, they are redundant for our method.

The search tree is a depth-first search tree, from left to right. In the figure, the $\square$ means that there exists a contradiction when doing constraint propagation. And the arrows point out the trace of search. M1 means we got a model at that node. DASH S1 means we will backtrack at this node because of the current partial model satisfies S1.

In the rest of this section, we shall discuss how to use the above concepts to describe the branches in the search tree and how to use them to eliminate isomorphism.

First, we realize that we can use some schemes to describe all the branches we have searched. Then when we are exploring a new branch, we would like to know whether it is isomorphic to some previous branch. This can be done by checking whether the partial model derived from the new decision sequence satisfies some existing scheme.

Definition 6. *If there is a series of assignments to all the scheme variables in a scheme such that the scheme becomes a series of assignments in the partial model, we say that "the partial model which is derived from the new decision sequence satisfies the scheme", or in short, "a decision sequence satisfies the scheme".*

Please see Example 2 below.

Proposition 4.1. Consider a decision sequence α , the scheme $\hat{\alpha}$ which is derived from α and the partial model $\bar{\alpha}$ which is extended from α by constraint propagation. If we don't consider the symmetry between the values not bigger than *imdn*, the scheme represents all the branches which are isomorphic to some branch $\bar{\alpha}'$ extended from $\bar{\alpha}$.

That is, for any branch $\bar{\beta}$ which is isomorphic to $\bar{\alpha}'$ and the isomorphism preserves the values which are not bigger than *imdn*, the isomorphism mapping θ leads to one assignment σ to all the scheme variables in the scheme $\hat{\alpha}$ such that $\sigma(\hat{\alpha})$ is consistent with $\bar{\beta}$.

For any assignment σ to the scheme variables in $\hat{\alpha}$ which gives distinct values to distinct scheme variables, the scheme becomes a decision sequence which is isomorphic to α.

Proof. By the definition of scheme, obviously the original decision sequence satisfies the scheme. The assignment σ just gives every variable's index as the variable's value.

For any branch $\bar{\beta}$ which is isomorphic to $\bar{\alpha}'$ and the isomorphism preserves the values which are not bigger than *imdn*, denote the isomorphism mapping as θ. Here θ is a mapping: $D_n \rightarrow D_n$, such that $\theta(\bar{\alpha}')$ equals to $\bar{\beta}$.

Then θ can derive an assignment σ such that $\sigma(\hat{\alpha})$ is consistent with $\bar{\beta}$. Here for any scheme variable x_i in $\hat{\alpha}$, $\sigma(x_i) = \theta(i)$.

And for any assignment σ to the scheme variables in $\hat{\alpha}$ which gives distinct values to distinct scheme variables, the scheme becomes a decision sequence β which is isomorphic to α.

The isomorphism mapping $\theta : D_n \rightarrow D_n$ is defined as follows:

- for any scheme variable x_i in $\hat{\alpha}$, $\theta(i) = \sigma(x_i)$.
- for other undefined j in D_n, $\theta(j) = j$.

It is easy to check that $\theta(\alpha) = \beta$.

Proposition 4.2. If the partial model which is derived from the current decision sequence satisfies some scheme, then we can cut the current branch to eliminate isomorphism.

Proof. By Proposition 4.1 and Definition 6, the partial model (also called a branch) is isomorphic to some sub-branch of the branch which the scheme is derived from. Therefore, cutting the current branch can eliminate isomorphism.

Example 2
Consider the Abelian Group of order 6 in Example 1. We have the scheme $g(x_1) = x_1 \wedge g(x_2) = x_4 \wedge f(x_1, x_2) = x_3 \wedge f(x_2, x_2) = x_4$. When we come to a decision sequence which is $g(1) = 1, g(2) = 4, f(1, 2) = 3, f(2, 2) = 5, g(3) = 5$, we will find that the partial model which is derived from this sequence satisfies the scheme by assignments $x_1 = 1, x_2 = 3, x_3 = 2, x_4 = 5, x_5 = 4$, so we get the isomorphic bijection (23)(45) which maps the partial model to some sub-branch of $g(1) = 1, g(2) = 4, f(1, 2) = 3, f(2, 2) = 4$.

After the assignments $x_1 = 1, x_2 = 3, x_3 = 2, x_4 = 5, x_5 = 4$, the scheme becomes a series of assignments, namely, $g(1) = 1, g(3) = 5, f(1, 3) = 2, f(3, 3) = 5$. Here, $f(1, 3) = 2$ and $f(3, 3) = 5$ don't occur in the current decision sequence $g(1) = 1, g(2) = 4, f(1, 2) = 3, f(2, 2) = 5, g(3) = 5$, but these two assignments are obtained by constraint propagation.

Thus the partial model which is derived by $g(1) = 1, g(2) = 4, f(1, 2) = 3, f(2, 2) = 5, g(3) = 5$ satisfies the scheme $g(x_1) = x_1 \wedge g(x_2) = x_4 \wedge f(x_1, x_2) = x_3 \wedge f(x_2, x_2) = x_4$.

So the current branch which is derived from $g(1) = 1, g(2) = 4, f(1, 2) = 3, f(2, 2) = 5, g(3) = 5$ can be eliminated.

4.1 When to Add and Delete Schemes

In this subsection, we discuss how to use schemes to detect and eliminate isomorphism. In particular, we will show how to use as few schemes as possible to describe the searched branches and when to add or delete schemes.

If we add all the schemes, the number of schemes is too large. But if we only add the necessary ones, the experiments show that the number is much smaller.

Generally speaking, the number of schemes increases during the search. But in fact, if the original problem has many isomorphic subspaces, the number is small. Because many schemes can be simplified and the branches which satisfy some scheme do not lead to any new scheme.

In many experiments, the maximum number of schemes is not bigger than several hundred. If there is not plenty of isomorphism, using isomorphism elimination techniques is not a good choice.

First of all, every searched branch can lead to a scheme. We should eliminate the redundant ones. Because the schemes are used to eliminate isomorphism, we need not add the scheme derived from the current branch. When we will choose a value for a new cell, we add a scheme derived from the past branch unless that branch is backtracked due to the conflict obtained from constraint propagation.

For instance, consider Example 1. After we have searched the branch derived from the sequence $g(1) = 1, g(2) = 4, f(1, 2) = 3, f(2, 2) = 4$, before we try the case $f(2, 2) = 5$, we add the scheme $g(x_1) = x_1 \wedge g(x_2) = x_4 \wedge f(x_1, x_2) = x_3 \wedge f(x_2, x_2) = x_4$.

Obviously if the past branch already satisfies some scheme, we need not add a new scheme.

When we get a model, we must add a scheme to describe the model.

After all possible values of a cell have been tried, we must backtrack. Before backtracking we will delete all the schemes which are derived from trying values of the current cell.

Consider the above example, after we tried the case $f(2,2) = 5$, we must backtrack. Then we delete the scheme $g(x_1) = x_1 \wedge g(x_2) = x_4 \wedge f(x_1, x_2) = x_3 \wedge f(x_2, x_2) = x_4$. Because after backtracking, we will add a new scheme $g(x_1) = x_1$, and this scheme includes the above one.

Proposition 4.3. Using DASH will not lose any non-isomorphic model, and the generated models are all non-isomorphic ones.

Proof. By using DASH, we only cut the branches which are isomorphic to some searched branches. The models in those branches must be isomorphic to some model which we obtained. So we do not lose any non-isomorphic model.

If there are two generated models which are isomorphic to each other, then the second one must satisfy the scheme derived from the first one. This contradicts with the DASH process.

Therefore, the proposition holds. □

So by the above propositions, we can cut all the iso-branches, and at the same time, without losing any isomorphic class of models.

4.2 The Scheme Checking Process

The time used for adding and deleting schemes is so small that it can be ignored. Almost all the time of the DASH process is spent on checking whether a branch satisfies some scheme. In our implementation, we just use a simple backtracking process for scheme checking. When we check an iso-branch, the depth of the checked branch is not smaller than the length of any scheme. And we check from the earliest scheme to the latest one.

After we select one scheme, we use backtracking search on the scheme variables, trying to find whether there is a set of assignments of the scheme variables such that the scheme becomes a set of assignments which is a subset of the current partial model. Please also see Example 2.

4.3 Adjustable Parameters

Eliminating all the isomorphism during search is not the most efficient way, since as the depth of search becomes bigger, the cost of eliminating isomorphism increases significantly and the benefit of doing so decreases. Thus, we can improve the efficiency by avoiding some improper checking of isomorphism.

We can adjust some parameters to achieve this goal. The most important parameter is the checking depth (i.e., using DASH up to a certain depth). This is easy to understand, since as the checking depth increases, the cost of checking increases greatly.

So in our implementation, we use a variable $maxD$ to record the current maximum depth of the search tree. And we can adjust three parameters, denoted by $factor1$, $factor2$, $factor3$. When the current depth is not larger than $factor1 * maxD$, we check all the schemes to find out whether the current branch satisfies some scheme.

When the current depth is larger than $factor1 * maxD$ and smaller than $factor2 * maxD$, we check all the schemes whose lengths are not bigger than $factor3 * maxD$.

Through our experiments, we found that it is often good to set the parameters $factor1, factor2, factor3$ to the values $0.6, 0.9, 0.3$, respectively.

Because the cost of adding and deleting schemes is very small, we still do these things. If we only want to get the non-isomorphic models, we can use DASH to check whether the new model is isomorphic to some existing model which has already been generated.

4.4 The New Algorithm

In this subsection, we describe an abstract procedure about DASH. We shall use the notations introduced earlier. The new search procedure is shown in Fig 2. It is extended from the basic search procedure of SEM [14]. We use upper-case letters to denote the new statements.

```
Void srh() {
    forward = TRUE;
    while (TRUE)  {
      if (forward)    {
        choose one cell whose value has not been fixed;
        if (all cells are assigned values)  {
            Number_Mod++;
            if (Number_Mod == Max_N_Mod) return;
            else forward = FALSE;
        }
      }
      if (forward) push stack;
      else {
        if (stack is empty)  return;
        ADD a scheme for the current decision sequence;
        restore stack;
      }
      forward = FALSE;
      for each of current cell's possible values do  {
        assign the value to the cell;
        propagate the effect of this assignment;
        if (contradiction does not occur )   {
           CHECK all schemes;
           if (the current partial model satisfies some scheme)
              { restore stack; continue; }
            else {
                forward = TRUE;
                break;
            }
        }
        else  restore stack;
      }
      if (forward == FALSE) {
         DELETE schemes of current depth;
         pop stack;
      }
    }
}
```

Fig. 2. The Search Procedure with DASH

5 Experimental Results

Using the above ideas, we have implemented an automatic tool, called SEMD, based on SEM [14]. We have tested the tool on a number of well-known problems and compared it with SEM, newSEM [4], Mace4 [22] and Paradox [15]. The problems include:

– Logic and abstract algebra: the problems "Finite Abelian Groups". We try to find all the finite Abelian Groups of every order. In Table 1, we set $factor1 = factor2 = factor3 = 1$. Table 2 shows that adjusting factors can improve the efficiency. We denote $factor1$, $factor2$, $factor3$ by $t1, t2, t3$ respectively, in Table 2. From the fundamental theorem on finite Abelian groups, we

Table 1. Abelian Groups

	SEM + LNH			SEM + DASH				newSEM + XLNH			Mace 4	
n	m	t	r	m	t	r	Cuts	m	t	r	m	t
6	6	0	30	1	0	23	5	2	0	40	6	0
16	135	0.68	2347	5	0.21	479	71	44	0.41	696	135	1.79
26	16	8.90	10480	1	2.25	1291	178	2	7.87	3308	16	98.28
32	2295	133.72	76483	7	12.55	2623	404	529	63.92	11313	2295	802.24
33	15	71.85	48659	1	8.95	2527	351	15	11.57	3347	+	+
34	20	78.96	51306	1	9.76	2611	356	2	66.75	12786	+	+
35	13	185.86	95009	1	16.81	3097	457	13	14.55	3711	+	+
36	2142	309.79	146386	4	22.26	3524	553	321	145.07	21601	+	+
37	1	239.30	112003	1	20.01	3407	478	1	23.35	4779	+	+
38	22	240.40	115528	1	22.51	3533	511	2	175.14	23366	+	+
39	17	397.98	174819	1	29.20	4017	600	17	34.82	5811	+	+
40	2220	589.14	233198	3	35.61	4407	671	282	303.37	39124	+	+
49	+	+	+	2	78.77	6420	874	8	188.26	12858	+	+
50	+	+	+	2	87.58	6570	934	+	+	+	+	+

Table 2. Abelian Groups with Different DASH Factors

	$t1 = t2 = t3 = 1$				$t1 = 0.6, t2 = 0.9, t3 = 0.3$				newSEM + XLNH		
n	m	t	r	Cuts	m	t	r	Cuts	m	t	r
43	1	43.62	4929	708	1	40.01	7443	669	1	61.35	7781
44	2	50.83	5188	750	2	45.98	7858	699	31	662.61	66025
45	2	56.34	5494	801	2	52.78	8930	749	180	93.24	9024
46	1	57.41	5498	786	1	53.31	8602	733	+	+	+
47	1	61.88	5795	803	1	58.49	8103	784	1	86.70	9329
48	5	106.08	6913	1058	5	91.65	14082	912	+	+	+
49	2	78.77	6420	874	2	73.29	9673	829	8	188.26	12858
50	2	87.58	6570	934	2	83.30	10547	867	+	+	+

know that the number of nonisomorphic Abelian groups of order n, where $n = p_1^{e_1} \dots p_n^{e_n}$, is $f(e_1) \dots f(e_n)$, where $f(x)$ denotes the number of out-of-order partitions of x. Thus we can verify that the number of models obtained by SEMD is just the number of nonisomorphic Abelian groups.

– Combinatorics: Quasigroup existence problems described in [16]. For this problem, we only compare SEMD with SEM. Since there is no unary function in this problem and newSEM is based on unary functions, newSEM can not

Table 3. Quasigroup Existence Problem

	SEM +LNH		SEM + DASH		
Order	Time	Rounds	Time	Rounds	Cuts
10	62.44	18423609	33.02	6717481	35
11	190.46	69742027	113.97	25427909	33

speed up in this problem. In Table 3, we only search for one quasigroup for each different order. For an order which is less than 10, DASH can not cut any branch for the first model. In Table 4, we search for one quasigroup which satisfies the QG5 identity. Though we can cut some branches, the scheme checking process costs too much time and the total time used by SEMD is a little longer than that for SEM.

Table 4. QG5 Existence Problem

	SEM +LNH			SEM + DASH			
Order	Models	Time	Rounds	Models	Time	Rounds	Cuts
9	0	0	66	0	0	52	1
10	0	0	166	0	0	134	2
11	1	0.01	421	1	0.01	297	4
12	0	0.05	2878	0	0.05	1770	79
13	0	0.51	27764	0	0.68	14894	834
14	0	8.66	430040	0	11.91	182304	10060

Table 5. HSI Problem

	SEM +LNH			SEM + DASH			
Order	Models	Time	Rounds	Models	Time	Rounds	Cuts
2	5	0	15	5	0	15	0
3	68	0	228	44	0	182	15
4	2199	0.05	7867	657	0.23	3259	375

– HSI problem [13]: This problem also comes from Algebra, and there are some results about order 2 and order 3 in [19].

In these tables, all running times ("t") are given in seconds. We use "+" to indicate that the running time is more than 600 seconds. "Rounds" ("r") means the number of rounds of the search loop, and "Cuts" means the number of branches which are cut by DASH, while "n" means the size of the model, "m" means the number of models. The experimental results are obtained on a personal computer with Pentium 4 2.6G, 512M memory.

We can see that, when using SEMD (SEM+DASH), the numbers of models are quite small, and the running times are acceptable. It should be noted that SEMD gets all the non-isormorphic models, while the other tools just find out all models without eliminating the redundant ones.

Paradox is quite slow on this type of problems, because it is weak in symmetry breaking and when the order increases, the SAT instance generated by Paradox becomes too large. So we do not give the experimental data for Paradox.

6 Related Works and Conclusions

During the past several years, symmetry breaking and isomorphism elimination have attracted much attention from researchers. It was shown that they can play a key role in the solution of many problems. Sophisticated symmetry breaking methods have been developed, such as the addition of symmetry breaking constraints (see for example [10]), symmetry breaking during search (SBDS) [7], or symmetry breaking by dominance detection (SBDD) [20]. Especially the latter is quite interesting.

SBDD works by checking whether the current choice point under investigation represents a symmetric variant of a part of the search space that has been investigated completely before. The core of an SBDD symmetry breaking code is dominance detection which was automated in [8] by using the generic computational group theory tool, yielding a method named GAP-SBDD.

The basic idea of this paper is similar to that of SBDS and SBDD. But the latter deals with constraint satisfaction problems (CSPs) and they mainly consider the symmetries among variables in a CSP. We try to solve the finite model generation problem, in which the symmetries are different.

For finite model enumeration, there are some isomorphism elimination methods which are similar to the Least Number Heuristic (LNH) or are extensions to the LNH, see for example, [1,18,4]. It is worth mentioning that the tool newSEM

[4] is quite close to ours. But it requires that the problem has a unary function, so it is useless for some problems. For instance, in QG problems, SEMD can work well, but newSEM does not work. From Table 1, we can see that SEMD is almost always better than newSEM on the Abelian group problem. More recently, Boy de la Tour and Countcham [2] investigated the integration of SEM-style search procedure and McKay's general method of isomorph-free exhaustive enumeration. But their tool SEMK is not available yet.

One of the advantages of DASH is that we can get all the nonisomorphic models with just a little overhead. We can tell the program to find all the models and use DASH to eliminate all the models which are isomorphic to some nonisomorphic model. We have tried to do this by using other tools such as Mace4. They check isomorphism after all the models are generated. And much more time is needed. Therefore, DASH becomes an efficient technique for finite model enumeration.

The most important contribution of our work is presenting a new method of describing the isomorphism class. The experiments show that it is efficient. But we still have something to do in the future. Most importantly, we need to find a more efficient scheme checking method.

Acknowledgements

We are grateful to the anonymous reviewers for their comments and suggestions. This work was partially supported by NSFC grant No. 60125207.

References

1. Jackson D., Jha S., and Damon C.A. Isomorph-free model enumeration: A new method for checking relational specifications. *ACM Transactions on Programming Languages and Systems*, 20(2):302–343, 1998.
2. Boy de la Tour T. and Countcham P. An isomorph-free SEM-like enumeration of models. *Electr. Notes Theor. Comput. Sci.*, 125(2):91–113, 2005.
3. Moskewicz M et al. Chaff: Engineering an efficient SAT solver. In *Proc. 39th Design Automation Conference*, pages 530–535, 2001.
4. Audemard G. and Henocque L. The extended least number heuristic. In *Proc. of the 1st Int'l Joint Conference on Automated Reasoning*, pages 427–442, 2001.
5. Sutcliffe G. and Suttner C.B. The TPTP problem library – CNF release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.
6. Zhang H. An efficient propositional prover. In *Proc. 14th Int'l Conf. on Automated Deduction (CADE-14)*, pages 272–275, 1997.
7. Gent I. and Smith B. Symmetry breaking in constraint programming. In *Proc. ECAI'00*, pages 599–603, 2000.
8. Gent I., Harvey W., Kelsey T., and Linton S. Generic SBDD using computational group theory. In *Proc. CP'03*, pages 333–347, 2003.
9. Crawford J. A theoretical analysis of reasoning by symmetry in first order logic. Technical report, AT&T Bell Laboratories, 1996.
10. Crawford J., Ginsberg M., Luks E., and Roy A. Symmetry-breaking predicates for search problems. In *Proc. KR'96*, pages 149–159, 1996.

11. Slaney J. Finite domain enumerator. system description. In *Proc. 12th Int'l Conf. on Automated Deduction (CADE-12)*, pages 798–801, 1994.
12. Zhang J. Constructing finite algebras with FALCON. *Journal of Automated Reasoning*, 17(1):1–22, 1996.
13. Zhang J. Computer search for counterexamples to Wilkie's identity. In *Proc. 20th International Conference on Automated Deduction (CADE-20)*, pages 441–451, 2005.
14. Zhang J. and Zhang H. SEM: a system for enumerating models. In *Proc. 14th Int'l Joint Conf. on Artificial Intelligence (IJCAI)*, pages 298–303, 1995.
15. Claessen K. and Sörensson N. New techniques that improve mace-style finite model finding. In *Proceedings of the CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications (Miami, USA)*, 2003.
16. Fujita M., Slaney J., and Bennett F. Automatic generation of some results in finite algebra. In *Proc. 13th Int'l Joint Conf. on Artificial Intelligence (IJCAI)*, pages 52–57, 1993.
17. Eén N. and Sörensson N. The MiniSat page. Webpage, Chalmers University, http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/, 2005.
18. Peltier N. A new method for automated finite model building exploiting failures and symmetries. *J. of Logic and Computation*, 8(4):511–543, 1998.
19. Burris S. and Lee S. Small models of the high school identities. *Intl J. of Algebra and Computatio*, 2:139–178, 1992.
20. Fahle T., Schamberger S., and Sellmann M. Symmetry breaking. In *Proc. CP'01*, pages 93–107, 2001.
21. McCune W. MACE 2.0 reference manual and guide. Technical Report No. 249, Argonne National Laboratory, Argonne, IL, USA, 2001.
22. McCune W. Mace4 reference manual and guide. Technical Report No. 264, Argonne National Laboratory, Argonne, IL, USA, 2003.