Investigating the Existence of Large Sets of Idempotent Quasigroups via Satisfiability Testing

Pei Huang[1,3], Feifei Ma[1,2,3](✉), Cunjing Ge[1,3], Jian Zhang[1,3](✉) and Hantao Zhang[4]

[1]State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences, China
[2]Laboratory of Parallel Software and Computational Science
Institute of Software, Chinese Academy of Sciences, China
[3]University of Chinese Academy of Sciences, China
{huangpei, maff, gecj, zj}@ios.ac.cn
[4]Department of Computer Science, University of Iowa City, IA 52242, U.S.A.
hantao-zhang@uiowa.edu

Abstract. In this paper, we describe a method for solving some open problems in design theory based on SAT solvers. Modern SAT solvers are efficient and can produce unsatisfiability proofs. However, the state-of-the-art SAT solvers cannot solve the so-called *large set* problem of idempotent quasigroups. Two idempotent quasigroups over the same set of elements are said to be *disjoint* if at any position other than the main diagonal, the two elements from the two idempotent quasigroups at the same position are different. A collection of $n-2$ idempotent quasigroups of order n is called a *large set* if all idempotent quasigroups are mutually disjoint, denoted by $LIQ(n)$. The existence of $LIQ(n)$ satisfying certain identities has been a challenge for modern SAT solvers even if $n = 9$. We will use a finite-model generator to help the SAT solver avoiding symmetric search spaces, and take advantages of both first order logic and the SAT techniques. Furthermore, we use an incremental search strategy to find a maximum number of disjoint idempotent quasigroups, thus deciding the non-existence of large sets. The experimental results show that our method is highly efficient. The use of symmetry breaking is crucial to allow us to solve some instances in reasonable time.

1 Introduction

In recent decades, automated reasoning tools have been applied to some combinatorial problems which are difficult for conventional mathematical methods. For example, Marijin Heule et al. solved the boolean pythagorean triples problem via a parallelized SAT solver with 800 cores in about 2 days [10]. Generally, these combinatorial problems are hard to solve. The quasigroup problem is among these problems and it has attracted much focus by researchers in the field of combinatorics and automated reasoning. The improvement of automated reasoning techniques made computer search play an important role in the

study of quasigroups [24]. For example, many open problems of the type from QG2 to QG9 have been solved by some finite-model generators such as *MGTP, FINDER, SEM, MACE4* and propositional satisfiability provers *SATO, DDPP*, respectively [6, 19, 27, 23, 25].

The large set problem, which seeks to find a set of combinatorial objects rather than one, is a classic and challenging research topic in combinatorial design theory. Sylvester first proposed the existence of large sets of Kirkman triple systems in the 1850s [3]. In the late last century, the large set of idempotent quasigroups were proposed. Due to its difficulty in construction, the research progress is quite slow in the mathematics field. So, any progress of the large set is something expected[2, 13, 22].

Investigating the existence of large sets of moderate order via computer can provide support for mathematicians to further explore general issues. Sometimes they can use mathematical construction to produce large objects from smaller ones recursively. Besides, many hard combinatorial problems related to quasigroups also have potential value in the field of cryptography [7, 12].

A collection of $n-2$ idempotent quasigroups (IQs) of order n is called a *large set* if any two of them are disjoint, denoted by $LIQ(n)$. Feifei Ma et al. applied *SEM* [27] to some open cases about large sets of idempotent quasigroups with certain identities summarized by L.Zhu [29, 28], and solved $LIQ(n)$ ($n \leq 8$) [14].

In this paper, we attempt to further study the open cases of $LIQ(n)$ for $n \geq 9$. Unlike $LIQ(n)$ ($n \leq 8$), it is difficult to solve these cases via encoding them directly as SAT, SMT, CSP or first order logic formulae, even though $n = 9$ is just one more than $n = 8$. We tested direct encoding ways and used SAT solvers like *MiniSat, Glucose, Treengeling* (1st in SAT 2016 competition in parallel track), SMT solver like *Z3* [5], CSP solver like *Minizinc* [17] and finite-model generators like *SEM, SEMD, MACE4* [15] and they all failed to produce a result in a week for many instances. However, $LIQ(n)$ ($n \leq 8$) can be solved in seconds. These challenging problems, $LIQ(n)$ ($n \geq 9$), promote us to seek for more powerful search strategies.

There have been much progress in SAT solving during the past 20 years; and the state-of-the-art SAT solvers can make very efficient low-level inferences. They have become the core search engine in many tools used for combinational [8, 16] and sequential equivalence checking [1, 11]. Apart from high efficiency, the state-of-the-art SAT solvers can make a claim that the formula is unsatisfiable with a *formal proof*. One can verify the proof emitted by a SAT solver and ensure the result is correct [9, 21]. Yet when other kinds of solvers claim that a formula is unsatisfiable, one has to trust that the solver fully exhausted the search space for the problem. However, SAT solvers are weak in dynamic symmetry breaking, hence may revisit a lot of redundant search space. Usually, when a problem is encoded as Boolean formulae, its structural characteristics may be hidden.

On the other hand, when a problem is encoded as first order logic formulae, the structural information and some properties are well preserved. The finite-model generator can make use of this information to break symmetries. Finite-model generators such as *SEM,SEMD* and *MACE4* are good at dynamic

symmetry breaking and enumerating all solutions. A benefit of using SEM-style finite-model generators is that the search process can exploit high-level structural information in the formulas (e.g., symmetries) to reduce the search space. The core dynamic symmetry breaking method is a heuristic called least number heuristic (LNH) [27]. The basic idea is that many element names, which have not yet been used in the search, are essentially the same. Furthermore, *MACE4* can eliminate isomorphic solutions statically.

A question naturally arises: would it be helpful to combine these two paradigms? In 2001, J. Zhang proposed to combine automatic symmetry breaking with SAT [26]. The experimental results in [26] showed the advantage of this method. We employ a similar search strategy in solving the LIQ problem: Use the first order model generator to generate some asymmetric partial potential solutions; with the help of these partial solutions as candidates, a SAT solver can avoid a lot of symmetric search spaces. This simple combination can take advantages of their respective strengths. In addition to that, we also statically add some symmetry breaking constraints. Adding constraints to the basic model has been most used historically by constraint programmers [18]. The experimental results show that this combination can greatly improve the solving efficiency. We found some instances, which cannot be solved with a single solver in a week before, can be solved in minutes now. Due to these strategies, a number of open cases of $LIQ(n)$ have been solved. We not only establish the non-existence of these cases, but also find the maximum number of disjoint $IQ(n)$s, and some other interesting mathematical results.

This paper is organized as follows: in Sect. 2, we introduce some preliminaries about LIQ; In Sect. 3 and 4, we present the encoding method and how to break symmetry and speed up the search process; In Sect. 5 and 6, we present the results about LIQ and the experiments. Furthermore, we evaluate and discuss the experimental results; In the final section, conclusions are drawn.

2 Preliminaries

2.1 Definitions

Let us recall some notations.

A *quasigroup* is denoted as an ordered pair $(Q, \oplus)$, where Q is a set and $\oplus$ is a binary operation on Q. For all constants $a, b \in Q$ and variables $x, y \in Q$ equations $a \oplus x = b$ and $y \oplus a = b$ are uniquely solvable. $|Q|$ is said be the *order* of $(Q, \oplus)$.

It is well-known that the multiplication table of quasigroup $(Q, \oplus)$ is a Latin square. Thus, Latin square and quasigroup are often treated as synonyms. Figure 1 shows the multiplication table of a quasigroup $(Q, \oplus)$ where $Q = \{0, 1, 2, 3\}$.

For all $x \in Q$, if $x \oplus x = x$ (briefly $x^2 = x$), the quasigroup $(Q, \oplus)$ is *idempotent*. We denote an idempotent quasigroup of order n as $IQ(n)$.

Two quasigroups $(Q, \oplus)$ and $(Q, \cdot)$ are said to be *disjoint* if for all $x, y \in Q$, $x \oplus y \neq x \cdot y$ whenever $x \neq y$.

L	0	1	2	3
0	0	2	3	1
1	3	1	0	2
2	1	3	2	0
3	2	0	1	3

Fig. 1. A quasigroup of order 4

Definition 1 (Large Set). *A collection of idempotent quasigroups* $(Q, \oplus_1)$, $(Q, \oplus_2)$, ..., $(Q, \oplus_{n-2})$, *where* $n = |Q|$, *is called a* large set, *if any two of the idempotent quasigroups are disjoint.*

A large set of idempotent quasigroups of order n is denoted by $LIQ(n)$. Figure 2 shows a large set of idempotent quasigroups of order 4, i.e., $LIQ(4)$, which consists of two disjoint $IQ(4)$s.

L1	0	1	2	3
0	0	2	3	1
1	3	1	0	2
2	1	3	2	0
3	2	0	1	3

L2	0	1	2	3
0	0	3	1	2
1	2	1	3	0
2	3	0	2	1
3	1	2	0	3

Fig. 2. Two disjoint $IQ(4)s$ in $LIQ(4)$

Besides the existence of $LIQ(n)$, the maximum number of disjoint $IQ(n)$s for non-existent instances is also concerned.

Definition 2 (Orthogonal). *Two quasigroups* $(Q, \oplus)$ *and* $(Q, \cdot)$ *are said to be* orthogonal, *if for all* $x_1, x_2, y_1, y_2 \in Q$, *the ordered pair* $(x_1 \oplus x_2, y_1 \cdot y_2)$ *is unique.*

$L1$ and $L2$ in Figure 2 are also *orthogonal*. Ordered pairs are shown in Figure 3 and every ordered pair appears only once.

A LIQ is said to have orthogonality, if any two quasigroups in the LIQ are orthogonal. In general, for idempotent quasigroups, orthogonality implies disjointness, but the reverse does not hold.

2.2 The Problems

Teirlinck and Lindner [20], and Chang [4] have already established the existence of $LIQ(n)$. In [4], Chang concluded that there exists an $LIQ(n)$ for any $n \geq 3$ with the exception $n = 6$. The spectrums of some large sets in which

(L1,L2)	0	1	2	3
0	(0,0)	(2,3)	(3,1)	(1,2)
1	(3,2)	(1,1)	(0,3)	(2,0)
2	(1,3)	(3,0)	(2,2)	(0,1)
3	(2,1)	(0,2)	(1,0)	(3,3)

Fig. 3. The ordered pair of L1 and L2

the IQs satisfy certain identities have not been explored extensively up to now. The existence of idempotent quasigroups satisfying the seven "short identities" has been studied systematically. These identities are:

1. $xy \oplus yx = x$ Schröder quasigroup
2. $yx \oplus xy = x$ Stein's third law
3. $(xy \oplus y)y = x$ C_3-quasigroup
4. $x \oplus xy = yx$ Stein's first law; Stein quasigroup
5. $(yx \oplus y)y = x$
6. $yx \oplus y = x \oplus yx$ Stein's second law
7. $xy \oplus y = x \oplus xy$ Schröder's first law

In the above equations, xy is an abbreviation of $(x \oplus y)$. That means xy has higher precedence than $x \oplus y$. Let $LIQ^{(i)}(n)$ denote the large set of idempotent quasigroups of order n satisfying identity (i). The existence of $LIQ^{(i)}(n)$ is still an open problem. In [28], L. Zhu listed several open cases. Since the search spaces of the problem grow exponentially with order n, we pick out some open cases of moderate orders which may be suitable for computer search. In Table 1, we list these open cases where $n \leq 13$.

1. $LIQ^{(1)}(12)$ $LIQ^{(1)}(13)$	2. $LIQ^{(2)}(9)$ $LIQ^{(2)}(12)$
3. $LIQ^{(3)}(10)$ $LIQ^{(3)}(13)$	4. $LIQ^{(4)}(9)$ $LIQ^{(4)}(11)$
5. $LIQ^{(5)}(11)$	6. $LIQ^{(6)}(9)$ $LIQ^{(6)}(13)$
7. $LIQ^{(7)}(9)$ $LIQ^{(7)}(13)$	

Table 1. Open cases for LIQ of moderate sizes

In [14], Feifei Ma et al. studied $LIQ^{(i)}(n)$s with n no more than 8 in table 1. They modeled $LIQ^{(i)}(n)$ via first order formulae and used the finite-model generator *SEM*. However, using the direct method is impracticable for $LIQ^{(i)}(n)$s where n is more than 8. So $LIQ^{(i)}(n)$s, where $n \geq 9$, are still open cases.

The search space grows exponentially with the size of the problem. In practice, we encoded $LIQ^{(i)}(n)$s, where $n \geq 9$, as a SAT instance, using uninterpreted functions and first order formulae in a naive way. However, we could not get any result via SAT solvers like *MiniSat, glucose, Treengeling*, the SMT solver like

Z3, the CSP solver like *Minizinc* and finite-model generators like *SEM, MACE4* in a week for some instances with $n = 9$. However, for $LIQ(n)$s ($n \leq 8$), results could be obtained in a few seconds to minutes.

3 Encoding

In the introduction we have expounded the reason why we choose SAT as core engine. First, a notation $ExactOne(x_1, x_2, ..., x_n)$ will be introduced. $x_1, x_2, ..., x_n$ are Boolean variables and $ExactOne(x_1, x_2, ..., x_n)$ is a formula composed of $x_1, x_2, ..., x_n$. $ExactOne(x_1, x_2, ..., x_n)$ expresses the fact that exactly one of these Boolean variables is true for any satisfying assignment to this formula.

$$ExactOne(x_1, x_2, ..., x_n) = (x_1 \vee x_2 \vee, ..., \vee x_n) \wedge \underbrace{(\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}), ..., \wedge (\overline{x_{n-1}} \vee \overline{x_n})}_{\binom{n}{2}}$$

Without loss of generality, we assume the domain Q to be the set $\{0, 1, \ldots, n-1\}$. $\oplus_f$ is actually a function $L_f : Q \times Q \mapsto Q$ satisfying the constraints of idempotent quasigroup and identity (i). $LIQ(n) = \{L_1, L_2, \ldots, L_{n-2}\}$, where $n = |Q|$, denotes a large set. $L_f(x, y)$ denotes $x \oplus_f y$ in quasigroup L_f. In Section 2, we mentioned that a quasigroup can be seen as a multiplication table (or a matrix called **Latin square**). Then we can use Boolean variable $P_{f,x,y,v}$ to denote that the position(x, y) of f_{th} IQ in the large set is v.

According to the definition of **quasigroup**, it is easy to know that each element in Q occurs exactly once in each row and exactly once in each column in the matrix. So for each row x of f_{th} IQ, we add formula:

$$\bigwedge_{0 \leq v \leq n-1} ExactOne(P_{f,x,0,v}, P_{f,x,1,v}, ..., P_{f,x,n-1,v})$$

For each column y of f_{th} IQ, we add formula:

$$\bigwedge_{0 \leq v \leq n-1} ExactOne(P_{f,0,y,v}, P_{f,1,y,v}, ..., P_{f,n-1,y,v})$$

For each cell (x, y) of f_{th} IQ, we add formula:

$$P_{f,x,y,1} \vee P_{f,x,y,2} \vee, ..., \vee P_{f,x,y,n-1}$$

For f_{th} IQ in the large set, The idempotent property ($x^2 = x$) can be encoded as:

$$\bigwedge_{0 \leq x \leq n-1} P_{f,x,x,x}$$

The disjoint property depicts that for any two latin squares L_j and L_k ($1 \leq j, k \leq n-2$), $L_j(x, y) \neq L_k(x, y)$ except for $x = y$. So the encoding for m disjoint latin squares is:

$$\bigwedge_{1\leq j<k\leq m} \bigwedge_{x\neq y} \bigwedge_{0\leq v\leq n-1} \overline{P_{j,x,y,v}} \vee \overline{P_{k,x,y,v}}$$

The encoding for the seven identities:

1).For $(xy \oplus yx = x)$: $\bigwedge_{0\leq x,y,v_1,v_2\leq n-1} \overline{P_{f,x,y,v_1}} \vee \overline{P_{f,y,x,v_2}} \vee P_{f,v_1,v_2,x}$

2).For $(yx \oplus xy = x)$: $\bigwedge_{0\leq x,y,v_1,v_2\leq n-1} \overline{P_{f,y,x,v_1}} \vee \overline{P_{f,x,y,v_2}} \vee P_{f,v_1,v_2,x}$

3).For $(xy \oplus y)y = x$: $\bigwedge_{0\leq x,y,v_1,v_2\leq n-1} \overline{P_{f,x,y,v_1}} \vee \overline{P_{f,v_1,y,v_2}} \vee P_{f,v_2,y,x}$

4).For $(x \oplus xy = yx)$: $\bigwedge_{0\leq x,y,v_1,v_2\leq n-1} \overline{P_{f,x,y,v_1}} \vee \overline{P_{f,y,x,v_2}} \vee P_{f,x,v_1,v_2}$

5).For $(yx \oplus y)y = x$: $\bigwedge_{0\leq x,y,v_1,v_2\leq n-1} \overline{P_{f,y,x,v_1}} \vee \overline{P_{f,v_1,y,v_2}} \vee P_{f,v_2,y,x}$

6).For $yx \oplus y = x \oplus yx$: $\bigwedge_{0\leq x,y,v_1,v_2\leq n-1} \overline{P_{f,y,x,v_1}} \vee \overline{P_{f,v_1,y,v_2}} \vee P_{f,x,v_1,v_2}$

7).For $xy \oplus y = x \oplus xy$: $\bigwedge_{0\leq x,y,v_1,v_2\leq n-1} \overline{P_{f,x,y,v_1}} \vee \overline{P_{f,v_1,y,v_2}} \vee P_{f,x,v_1,v_2}$

Actually, there is a lot of redundancy in the encoding. For example, $P_{f,x,x,x}$ must be assigned true and $P_{f,x,x,v}(v \neq x)$ must be assigned false. So there are many redundant clauses like $\overline{P_{f,1,1,2}} \vee \overline{P_{f,1,1,3}} \vee P_{f,2,3,1}$. These clauses can be eliminated in the encoding phase. Besides redundant clauses, many clauses can be simplified. For instance, known $P_{f,0,1,2}$(we will explain it in the next section), $\overline{P_{f,0,1,2}} \vee \overline{P_{f,1,0,v_2}} \vee P_{f,2,v_2,0}$ can be simplified as $\overline{P_{f,1,0,v_2}} \vee P_{f,2,v_2,0}$. Although the state-of-the-art SAT solvers can simplify these during the preprocessing phase, it is better to remove them in the encoding phase.

4 Search Strategies

Arguably, many hard combinatorial problems allow isomorphic solutions, and we say these problems have symmetries. The search may revisit equivalent states over and over again. Exploiting symmetry can reduce the search time to solve the problem. It is common to identify three main approaches to break symmetry. The first method is to reformulate the problem so it has a reduced number of symmetries. The second is to add symmetry breaking constraints before search starts, thereby making some symmetric solutions unacceptable while leaving at least one solution in each symmetric equivalence class. The final approach is to break symmetry dynamically during search, adapting the search

procedure appropriately. Although symmetry breaking technique and automatic identification of the symmetry have been concerned by researchers in the past, state-of-the-art SAT solvers do not have the ability to identify and break symmetry automatically. So, it is vital for us to handle the symmetries of the problem at hand.

4.1 Symmetry Breaking via Adding Constraints

First, we examine the structure of the problem to identify symmetries.

Lemma 1. *If there is a large set* $LIQ(n) = \{L_1, L_2, \ldots, L_{n-2}\}$*, then there exists a large set such that* $L_j(0,1) = j+1$*.*

Proof. According to $x^2 = x$, we know that $L_j(0,0) = 0$ and $L_j(1,1) = 1$. Since $a \oplus_j x = b$ and $y \oplus_j a = b$ are uniquely solvable, $L_i(0,1)$ can not be 0 or 1. All candidates for it include $2, 3, ..., n-1 (n-2$ elements). Due to the property of *disjoint*, for any $1 \leq j_1, j_2 \leq n-2$, $L_{j_1}(0,1) \neq L_{j_2}(0,1)$ but the cardinality of collection of $L_j(0,1)$ is $n-2$. So they must be exactly $\{2, 3, ...n-1\}$. □

Obviously, if $\{L_1, L_2, \ldots, L_{n-2}\}$ is a large set then any permutation of L_1, L_2, ..., L_{n-2} is also a large set. The permutation may not affect search time when a large set exists. However, when the large set does not exist, the solver will nearly enumerate all permutations and conclude that it is unsatisfiable. In other words, $\forall f_1, f_2 \in \{1, 2, ...n-2\}$ there is no essential difference between Boolean variable $P_{f_1,x,y,v}$ and $P_{f_2,x,y,v}$.

We can take advantage of *lemma 1* to fix the order of $\{L_1, L_2, \ldots, L_{n-2}\}$. Since $L_j(0,1)$ must be exactly $\{2, 3, ...n-1\}$, we specify its order by assigning

$$L_1(0,1) = 2 < L_2(0,1) = 3 < ... < L_{n-2}(0,1) = n-1$$

So, by *lemma 1*, we can add the following unit clauses:

$$\bigwedge_{1 \leq f \leq n-2} P_{f,0,1,f+1}$$

then Boolean variable $P_{f_1,x,y,v}$ and $P_{f_2,x,y,v}$ are different in a way and $(n-3)!-1$ isomorphic cases eliminated.

Now that the sequence of $IQ(n)$s is fixed, we use $DIQ^{(i)}_{(n)}(l)$ to denote the first l disjoint $IQ(s)$s satisfying identity (i).

$$DIQ^{(i)}_{(n)}(l) = \{L_1, L_2, ..., L_l \mid L_j(0,1) = j+1\}(1 \leq l \leq n-2)$$

Proposition 1. *If* $DIQ^{(i)}_{(n)}(l)$ *does not exist and* $DIQ^{(i)}_{(n)}(l-1)$ *exists, then the maximum number of disjoint* $IQ^{(i)}(n)s$ *is* $l-1$*.*

Proof. Suppose there exist m ($m \geq l$) disjoint IQs when $DIQ^{(i)}_{(n)}(l)$ does not exist and $DIQ^{(i)}_{(n)}(l-1)$ exists. Then we know that l disjoint IQs also exist. We assume they are $\{L_{j_1}, L_{j_2}, ..., L_{j_l}\}$ where $j_1 < j_2 <, ..., < j_l$. According to lemma 1 we know $L_{j_1}(0,1) = j_1 + 1$, $L_{j_2}(0,1) = j_2 + 1$,...,$L_{j_l}(0,1) = j_l + 1$. We can define a permutation in Cauchy form:

$$\sigma : \begin{pmatrix} 0\,1 & 2 & 3 & \dots & l & l+1 \dots n-1 \\ 0\,1\; j_1+1 & j_2+1 & \dots & j_l+1 & * & \dots \quad * \end{pmatrix}$$

The '*' can be any legitimate number. We can perform σ^{-1} on $\{L_{j_1}, L_{j_2}, ..., L_{j_l}\}$. It is easy to know that all constraints still hold under permutation σ^{-1} and $\{L_{j_1}, L_{j_2}, ..., L_{j_l}\}^{\sigma^{-1}} = \{L_1, L_2, ..., L_l\} = DIQ^{(i)}_{(n)}(l)$. Thus $DIQ^{(i)}_{(n)}(l)$ must exist however it contradicts with the known conditions. So the assumptions can not be true. □

Proposition 1 reveals that any l disjoint $IQ^{(i)}(n)$s must be isomorphic to some $DIQ^{(i)}_{(n)}(l)$. When $l = n-2$, $DIQ^{(i)}_{(n)}(l)$ is exactly $LIQ^{(i)}(n)$. So we can search for $DIQ^{(i)}_{(n)}(l)$s successively, increasing l step by step. Once for some l $DIQ^{(i)}_{(n)}(l)$ is unsatisfiable we can conclude that $LIQ^{(i)}(n)$ doesn't exist and the maximum number of disjoint IQs is $l-1$. The reason why $DIQ^{(i)}_{(n)}(l)$s are searched by increasing l instead of decreasing l is that it usually takes much more time to solve an unsatisfiable case than a satisfiable one.

The framework of incremental search for $LIQ^{(i)}(n)$ is shown in Algorithm 1. At each step, $DIQ^{(i)}(n)(l)$ is encoded as a set of Boolean formulas, denoted by $Encode(DIQ^{(i)}_{(n)}(l))$, and solved by a SAT solver.

Algorithm 1: Incremental search for $LIQ^{(i)}(n)$

Input: n is order, i is identity (i)
Output: The existence of LIQ and the maximum number of disjoint IQs

```
for l ← 2 to n − 2 do
    result ← Solve Encode(DIQ^(i)_(n)(l)) ;
    if result is UNSAT then
        return NONEXISTENT AND max = l − 1;
    end
end
return EXISTENT
```

4.2 Combine Solvers to Break Symmetry

Due to adding constraints, some apparent symmetries have been eliminated but there still remains a lot. Although the state-of-the-art SAT solvers are very

efficient for constraint propagation and conflict analysis, they are not good at breaking symmetry dynamically during search. Thus, we can use other solvers, which can break symmetry dynamically during search, to enumerate a small subspace and then add these candidate partial solutions to the clauses set to help a SAT solver to eliminate a lot of symmetric states.

It is easy to know that if there is a large set $\{L'_1, L'_2, ..., L'_{n-2}\}$ then there exists an isomorphic $DIQ^{(i)}_{(n)}(n-2) = \{L_1, L_2, ..., L_{n-2}\}$ where $L_1(0,1) = L'_1(0,1) = 2$. In section 4.1, the search process can be seen as expanding L_1 to L_{n-2} step by step. So, if we find all non-isomorphic L_1s as a candidate set and test whether they can be expanded to a large set, then a lot of isomorphic search spaces can be avoided. In this case, all non-isomorphic L_1s can be seen as candidate partial solutions. So, we use the first order logic solver to generate all non-isomorphic L_1s. $C^{(i)}(n)$ is used to denote the candidate set formed by all non-isomorphic L_1s which satisfy identity (i) of order n. We use $\Sigma_{C^{(i)}(n)}$ to denote the first order logic formula encoding.

$$\begin{aligned}\Sigma_{C^{(i)}(n)} = \{&\forall x\forall y\forall z(y = z \bigvee P(x,y) \neq P(x,z)),\\ &\forall x\forall y\forall z(x = z \bigvee P(x,y) \neq P(z,y)),\\ &\forall x P(x,x) = x,\\ &Identity(i)\\ &P(0,1) = 2,\\ &\forall x\forall y(x = y \bigvee P(x,y) \neq P(x,y))\}\end{aligned}$$

The (un)satisfiability of the problem is preserved. We summarize this process as Algorithm 2.

We only implemented the sequential version of Algorithm 2; it is easy to implement a parallel version. This also can be seen as an example of the divide and conquer method with symmetry breaking.

The search framework of Algorithm 2 can help us get the result about a $LIQ(n)$ quickly but it may fail in getting the maximum number of disjoint IQs. A small modification can make it capable of getting the maximum number of disjoint IQs. One just needs to delete the symmetry breaking constraints which are introduced in section 4.1. However, this will be slower than the original version. In addition, $C^{(i)}(n)$ can also be extended to denote more non-isomorphic disjoint IQs. We use *Hybrid search* to denote the original version, *Hybrid search1* to denote the version without symmetry breaking constraints and *Hybrid search2* to denote the version extending the concept of $C^{(i)}(n)$.

5 New Results

Table 2 lists the results about investigating the existence of large sets of idempotent quasigroups. The column of '*Maximum*' presents the maximum number of

Algorithm 2: Hybrid search for $LIQ^{(i)}(n)$

Input: n is order, i is identity (i)
Output: The existence of LIQ and the maximum number of disjoint IQs

$max \leftarrow 1$;
Generate $C^{(i)}(n)$ by $\Sigma_{C^{(i)}(n)}$ with a finite model generator which can break symmetry.;
for *all* $L_1 \in C^{(i)}(n)$ **do**
  for $l \leftarrow 2$ **to** $n-2$ **do**
    $result \leftarrow$ Solve $\{Encode(DIQ^{(i)}_{(n)}(l)) + Encode(L_1)\}$ with SAT solver ;
    if $result$ *is* $UNSAT$ **then** Break ;
    else if $l > max$ **then** $max \leftarrow l$;
  end
  if $max == n-2$ **then return** $EXISTENT$;
end
if $max < n-3$ **then** $max = unknown$;
return $NONEXISTENT$ AND max

disjoint $IQ(n)$s for a non-existent instance. And the fifth column marks whether the second strategy, hybrid search, was used. The maximum number of disjoint $IQ(n)$s for $LIQ^{(2)}(9)$, $LIQ^{(4)}(11)$ and $LIQ^{(5)}(11)$ are obtained by *Hybrid search1* which is a modified version of Algorithm 2 without symmetry breaking constraints.

Order n	Identity i	Existence of LIQ	Maximum	Hybrid
9	2	NO	6	✓
	4	NO	6	-
	6	NO	6	-
	7	NO	6	-
11	4	NO	4	✓
	5	NO	2	✓

Table 2. The result about LIQ

The current generation of SAT solvers support emission of unsatisfiability proofs. And standards for such proofs exist, as well as checkers. When the hybrid method was applied we should check the proof file and $C^{(i)}(n)$, although $C^{(i)}(n)$ may be hard to be verified by a formal method. $C^{(i)}(n)$ is a small fraction of the whole problem which can be double checked by different solvers. If one wants to verify the result, one just need to check $DIQ^{(i)}_{(n)}(max+1)$, where max denotes the maximum number of disjoint IQs.

In addition to investigating the existence of large sets, we also check the orthogonality of some LIQs. The definition of the orthogonality (Definition 2)

of LIQs has been introduced in section 2.1. While LIQ cannot imply orthogonality, all small order LIQs we found so far do have the orthogonality. This is also true for all $LIQ^{(i)}(n)$, where $n \leq 8$, found by Feifei Ma et al. Only $LIQ^{(1)}(8)$, $LIQ^{(3)}(4)$, $LIQ^{(4)}(4)$, $LIQ^{(7)}(8)$ exist according to [14]. $LIQ^{(3)}(4)$ and $LIQ^{(4)}(4)$ can be constructed simply by hand, and they have been checked by mathematicians. Our work is to check $LIQ^{(1)}(8)$ and $LIQ^{(7)}(8)$. We enumerate all large sets of $LIQ^{(1)}(8)$ and $LIQ^{(7)}(8)$. The number of $LIQ^{(1)}(8)$s and $LIQ^{(7)}(8)$s are 240 (some isomorphic solutions are eliminated). We examined all the $LIQ^{(1)}(8)$s and $LIQ^{(7)}(8)$s and concluded that all large sets of $LIQ^{(1)}(8)$s and $LIQ^{(7)}(8)$s have orthogonality.

6 Experimental Evaluation

In this section, we will use some experimental data to show the efficiency of our method. The experiment is performed on a Dell laptop with Intel(R) Core(TM) i7-6500U CPU(2.50GHz), operating system Ubuntu 16.04 and 16G memory.

The first strategy of adding symmetry breaking constraints will not be discussed in this section, because it is a universal method to save search time. The search framework also can help us avoid some isomorphic search spaces when getting the maximum number of disjoint IQs for nonexistent instance according to *Proposition 1*. It may prove that the large set does not exist when l is small. Due to the first search strategy, we can use the SAT solver *Glucose 4.1* to prove that $LIQ^{(4)}(9)$, $LIQ^{(6)}(9)$ and $LIQ^{(7)}(9)$ do not exist.

However, $LIQ^{(2)}(9)$, $LIQ^{(3)}(10)$, $LIQ^{(4)}(11)$ and $LIQ^{(5)}(11)$ are still hard for the first search strategy. We used *Treengeling* and parallel version of *Glucose 4.1* to solve these hard instances on a computer server with 100 cores (Intel Xeon CPU E7-8870 @ 2.40GHz 32M Cache). However, these instances exhausted a week without any results. Owing to the hybrid method we prove that they do not exist within several minutes except for $LIQ^{(3)}(10)$.

In order to evaluate the efficiency of the hybrid method, we compared the running times of using the hybrid method against just using a single SAT solver or a finite-model generator. The results are shown in Table 3. The first three columns show the search time of only using *Algorithm 1* and the fourth column shows search time of using the hybrid method. *SEM* and *MACE4* did almost the same in our experiment. We used *Glucose 4.1* and *MACE4* in *Hybrid search*. All the implementations are in github [1].

From Table 3, we know that combining SAT and first order logic can significantly improve the efficiency of search for LIQ. In particular, for some hard instances that finite-model generators and SAT solvers cannot solve in a week, the hybrid method can solve it in minutes.

Table 4 shows the results of different versions of the hybrid method which have been introduced in section 4.2. *Hybrid search* is the original version and

[1] https://github.com/huangdiudiu/LIQ-search

Instance	Glucose 4.1		Glucose(Parallel)		MACE 4		Hybrid search	
	Time	Result	Time	Result	Time	Result	Time	Result
$LIQ^{(2)}(9)$	>1 week	-	>1 week	-	>1 week	-	51.88s	UNSAT
$LIQ^{(4)}(9)$	14.372s	UNSAT	10.60s	UNSAT	>24h	-	0.25s	UNSAT
$LIQ^{(6)}(9)$	889.23s	UNSAT	616.735s	UNSAT	>24h	-	0.94s	UNSAT
$LIQ^{(7)}(9)$	1020.74s	UNSAT	560.07s	UNSAT	>24h	-	0.32s	UNSAT
$LIQ^{(4)}(11)$	>1 week	-	>1 week	-	>1 week	-	8.19s	UNSAT
$LIQ^{(5)}(11)$	>1 week	-	>1 week	-	>1 week	-	12.88s	UNSAT

Table 3. The running times of different methods in solving $LIQ^{(i)}(n)$

Hybrid search1 is the version that can find the maximum number of disjoint $IQ(n)$s. *Hybrid search2* extends the concept of $C^{(i)}(n)$, which is trying to find all non-isomorphic $\{L_1, L_2\}$ by the finite-model generator.

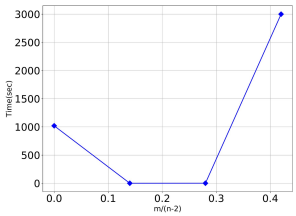
Instance	Hybrid search			Hybrid search1			Hybrid search2		
	MACE	SAT	Total(s)	MACE	SAT	Total(s)	MACE	SAT	Total(s)
$LIQ^{(2)}(9)$	4.79	12.18	16.97	4.79	282.52	241.46	921.33	21.00	942.33
$LIQ^{(4)}(9)$	0.01	0.28	0.29	0.01	1.06	0.39	1.45	1.19	2.03
$LIQ^{(6)}(9)$	0.08	0.55	0.63	0.08	20.78	20.86	>3000	-	>3000
$LIQ^{(7)}(9)$	0.03	0.50	0.53	0.03	2.15	2.18	32.84	1.97	34.81
$LIQ^{(4)}(11)$	0.01	10.83	10.84	0.01	211.17	211.27	731.11	13.62	744.73
$LIQ^{(5)}(11)$	1.01	23.06	24.07	1.01	95.84	96.85	>3000	-	>3000

Table 4. Comparison of different versions of the hybrid method

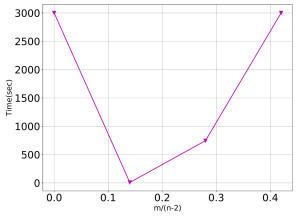
It is easy to know that fixing the sequence of IQs can improve the search process from the comparison between *Hybrid search* and *Hybrid search1*. However, it will sacrifice the ability of getting the maximum number of disjoint $IQ(n)$s. From the comparison of *Hybrid search* and *Hybrid search2*, we know that finding all non-isomorphic L_1s as candidate set $C^{(i)}(n)$ is more efficient than finding two disjoint IQs as candidate set.

The hybrid method divides the problem into two parts. One part, $C^{(i)}(n)$, is solved by a finite-model generator (first order logic) and the other part is solved by a SAT solver. So the hybrid ratio of first order logic formulae and SAT encoding will affect the performance. Actually, we observe that finding all non-isomorphic L_1s as candidate set $C^{(i)}(n)$ is the most efficient in our experiment.

We take $LIQ^{(7)}(9)$ and $LIQ^{(4)}(11)$ as examples to show that how the hybrid ratio affects the performance. If m denotes the number of $IQ(n)$s that are encoded as first order logic formulae, then we use $m/(n-2)$ to denote the hybrid ratio of first order logic formulae. Fig 4 shows the relationship between the hybrid ratio $m/(n-2)$ and search time. We find that $m/(n-2) = 1/(n-2)$ is the best hybrid ratio for almost all instances. When $m/(n-2) = 0$, that means all of the problems are solved by a SAT solver. When $m/(n-2) = 1$, that means all of the problems are solved by a finite-model generator.



(a) $LIQ^{(7)}(9)$



(b) $LIQ^{(4)}(11)$

Fig. 4. The relationship between the hybrid ratio $m/(n-2)$ and run time. The abscissa axis displays the value of $m/(n-2)$.

7 Conclusion

This paper describes an application of automated reasoning techniques and tools to an interesting problem in combinatorics: the large set of idempotent quasigroups (LIQs) satisfying the short identities. The LIQs of moderate orders which are difficult for mathematical methods can also be quite challenging for computer search. We present some effective search strategies for this problem, and the core idea is symmetry breaking. We find that combining the power of SAT solving and finite model generation is far more efficient than using a single solver. As a result, a number of open cases have been solved. Besides, we find that all $LIQ^{(1)}(8)$s and $LIQ^{(7)}(8)$s have orthogonality.

Acknowledgements. This work has been supported by the National 973 Program of China under Grant 2014CB340701, the National Natural Science Foundation of China under Grant 61100064, and the CAS/SAFEA International Partnership Program for Creative Research Teams. Feifei Ma is also supported by the Youth Innovation Promotion Association, CAS. We thank Lie Zhu and Yanxun Chang for suggesting these open problems and help.

References

1. Jason Baumgartner, Hari Mony, Viresh Paruthi, Robert Kanzelman, and Geert Janssen. Scalable sequential equivalence checking across arbitrary design transformations. In *International Conference on Computer Design*, pages 259–266, 2006.
2. H. Cao, L. Ji, and L. Zhu. Large sets of disjoint packings on 6k + 5 points. *Journal of Combinatorial Theory Series A*, 108(2):169–183, 2004.
3. A. Cayley. On the triadic arrangements of seven and fifteen things. *Philosophical Magazine*, pages 50–53, 1850.
4. Yanxun Chang. The spectrum for large sets of idempotent quasigroups. *Journal of Combinatorial Designs*, 8(2):79–82, 2015.
5. Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.
6. Masayuki Fujita, John K. Slaney, and Frank Bennett. Automatic generation of some results in finite algebra. In *International Joint Conference on Artifical Intelligence*, pages 52–57. Morgan Kaufmann, 1993.
7. Danilo Gligoroski, Smile Markovski, and Svein J. Knapskog. A public key block cipher based on multivariate quadratic quasigroups. *CoRR*, abs/0808.0247, 2008.
8. Evguenii I. Goldberg, Mukul R. Prasad, and Robert K. Brayton. Using sat for combinational equivalence checking. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 114–121, 2001.
9. Marijn Heule, Warren A. Hunt Jr., and Nathan Wetzler. Verifying refutations with extended resolution. In *International Conference on Automated Deduction*, volume 7898, pages 345–359, 2013.
10. Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 228–245, 2016.
11. Daher Kaiss, Marcelo Skaba, Ziyad Hanna, and Zurab Khasidashvili. Industrial strength sat-based alignability algorithm for hardware equivalence verification. In *Formal Methods in Computer-Aided Design*, pages 20–26, 2007.
12. C. Koscielny. Generating quasigroups for cryptographic applications. *International Journal of Applied Mathematics and Computer Science*, 12:559–569, 2002.
13. Jiaxi Lu. On large sets of disjoint steiner triple systems ii. *Journal of Combinatorial Theory*, 37(2):147–155, 1983.
14. Feifei Ma and Jian Zhang. Computer search for large sets of idempotent quasigroups. In *Asian Symposium on Computer Mathematics*, volume 5081, pages 349–358, 2008.
15. William McCune. Mace4 reference manual and guide. *CoRR*, cs.SC/0310055, 2003.
16. Alan Mishchenko, Satrajit Chatterjee, Robert K. Brayton, and Niklas Eén. Improvements to combinational equivalence checking. In *International Conference on Computer-Aided Design*, pages 836–843, 2006.

17. Ralph Becket Nicholas Nethercote, Peter J. Stuckey, Sebastian Brand, Gregory J. Duck, and Guido Tack. MiniZinc: towards a standard cp modelling language. In *International Conference on Principles and Practice of Constraint Programming*, volume 4741, pages 529–543. Springer, 2007.
18. Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming*. Elsevier, 2006.
19. John K. Slaney, Masayuki Fujita, and M. Stickel. Automated reasoning and exhaustive search: Quasigroup existence problems. *Computers & Mathematics with Applications*, 29(2):115–132, 1995.
20. Luc Teirlinck and C. C. Lindner. The construction of large sets of idempotent quasigroups. *European Journal of Combinatorics*, 9(1):83–89, 1988.
21. Nathan Wetzler, Marijn Heule, and Warren A. Hunt Jr. Drat-trim: Efficient checking and trimming using expressive clausal proofs. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 422–429, 2014.
22. Landang Yuan and Qingde Kang. Some infinite families of large sets of kirkman triple systems. *Journal of Combinatorial Designs*, 16(3):202–212, 2008.
23. Hantao Zhang. SATO: an efficient propositional prover. In *International Conference on Automated Deduction*, pages 272–275, 1997.
24. Hantao Zhang. Combinatorial designs by SAT solvers. In *Handbook of Satisfiability*, pages 533–568. IOS Press, 2009.
25. Hantao Zhang and Mark Stickel. Implementing the davis–putnam method. *Journal of Automated Reasoning*, 24(1-2):277–296, 2000.
26. Jian Zhang. Automatic symmetry breaking method combined with SAT. In *ACM Symposium on Applied Computing*, pages 17–21, 2001.
27. Jian Zhang and Hantao Zhang. SEM: a system for enumerating models. In *International Joint Conference on Artificial Intelligence*, volume 2, pages 298–303, 1995.
28. Lie Zhu. Personal communication, September 2007.
29. Lie Zhu. Large set problems for various idempotent quasigroups, 2014.