Breaking Local Symmetries in Quasigroup Completion Problems

Ruben Martins and Inês Lynce

IST/INESC-ID, Technical University of Lisbon, Portugal
{ruben,ines}@sat.inesc-id.pt

Abstract. Symmetry breaking is well-known as an important technique for reducing the search space when solving combinatorial problems. Symmetries can be broken either during a preprocessing step or during the search. Local symmetries, in contrast with global symmetries, are applied to a problem instance for which there is a partial assignment. Given such assignment, additional symmetries may hold. We perform an experimental study on breaking local symmetries in quasigroup completion problems, a well-studied problem among combinatorial problems. We break local symmetries by adding additional clauses to a SAT encoding. We conclude that these additional constraints have a puzzling effect on state-of-the-art SAT solvers, mainly due to the heuristics being used by these solvers.

1 Introduction

Quasigroups have been extensively solved in the last decade using Boolean satisfiability (SAT) technology (e.g. see [5, 12, 2]). Most often, the challenge is to solve the quasigroup completion problem that corresponds to a Latin square with some pre-assigned cells. The causes of the popularity of quasigroups seem easy to identify: the problem is simple to explain, some instances are easy to solve, an interesting phase transition can be clearly identified and, of course, SAT solvers are very efficient at tackling its problem instances. Not surprisingly, the popular puzzle *Sudoku*, that has become famous in the last couple of years, is nothing else but a quasigroup completion problem with only one additional constraint for each *sub-grid*.

On the other hand, symmetry breaking is currently a widely used technique in the domain of search problems (e.g. see [4, 6, 1]). Given that breaking symmetries reduces the search space, it seems fairly reasonable trying to break these symmetries either before the search or even during the search. The more symmetries you break the more nodes in the search tree you eliminate. Hence, having a highly symmetric problem becomes no longer a problem but rather a property that makes the problem easier to solve.

These two observations bring us to the focus of this paper: symmetry breaking in quasigroup completion problems. Unfortunately, this task in not straightforward: traditional symmetries that can be identified beforehand, also know as

global symmetries, do not hold for this problem [1]. However, it is possible to easily identify and break local symmetries, i.e. symmetries that hold given a partial assignment. We study the effect of breaking local symmetries in a SAT encoding for the quasigroup completion problem. The experimental results, albeit somehow surprising, shed light to the role played in SAT solvers by additional clauses for breaking local symmetries.

This paper is organized as follows. The next section provides the background for understanding the rest of the paper, namely the definitions for quasigroups and symmetry breaking. Afterwards, we explain how to break local symmetries in the quasigroup completion problem. Section 4 provides experimental results for different problem instances and for different local symmetries. Finally, the paper concludes.

2 Background

In this section we will clarify some important concepts related with quasigroup completion problems and symmetry breaking.

2.1 Quasigroup Completion Problems (QCPs)

Quasigroups have been traditionally presented as a combinatorial problem for which different encodings can be presented. Moreover, quasigroups have the advantage of providing an endless source of problem instances.

Definition 1 (Quasigroup). *A **quasigroup** is an ordered pair $(Q, \cdot)$, where Q is a set and $\cdot$ is a binary operation on Q such that for each a and b in Q, there exist unique elements x and y in Q such that: $a \cdot x = b$ and $y \cdot a = b$. The **order** n of the quasigroup is the cardinality of the set Q* [12].

A particular case of a quasigroup is a $n \times n$ multiplication table which defines a Latin square. Conversely, every Latin square can be taken as the multiplication table of a quasigroup.

Definition 2 (Latin Square). *A **Latin square** is an $n \times n$ table filled with n different symbols, having one symbol in each cell, in such a way that each symbol occurs exactly once in each row and exactly once in each column.*

Figure 1 shows an example of a 5×5 Latin square as the multiplication table of a quasigroup. In what follows we will only consider quasigroups as multiplication tables, i.e. we will refer indistinctly to quasigroups and Latin squares.

In this paper we will study the quasigroup completion problem (QCP), which is the NP-complete problem of filling a partial Latin square [5]: given a Latin square with some symbols pre-assigned, identify a complete assignment such that each symbol occurs exactly once in each row and exactly once in each column, or prove that such an assignment does not exist. Figure 2 shows a QCP of order 5 and a possible solution.

[1] We should note that global symmetries have been studied in the past in the context of the general definition for quasigroups [8].

·	1	2	3	4	5
1	1	2	3	4	5
2	2	3	4	5	1
3	3	5	1	2	4
4	4	1	5	3	2
5	5	4	2	1	3

Fig. 1. Example of a 5×5 Latin square as a multiplication table.

1				**4**
	5			
4			**2**	
	4			
		5		**1**

1	3	2	5	**4**
2	**5**	4	1	3
4	1	3	**2**	5
5	**4**	1	3	2
3	2	**5**	4	**1**

Fig. 2. Example of a QCP of order 5.

2.2 QCPs as a SAT Problem

A CNF formula is represented using n Boolean variables $x_1, x_2, \ldots, x_n$, which can be assigned truth values 0 (false) or 1 (true). A *literal* l is either a variable x_i (i.e., a positive literal) or its complement $\neg x_i$ (i.e., a negative literal). A *clause* ω is a disjunction of literals and a *CNF formula* φ is a conjunction of clauses.

Example 1. Consider the following CNF formula: $\varphi = \{\omega_1, \omega_2, \omega_3\}$ where $\omega_1 = (x_1 \vee \neg x_2)$, $\omega_2 = (x_2 \vee x_3)$ and $\omega_3 = (\neg x_2 \vee \neg x_3)$. This formula has 3 variables and 3 clauses, each one with two literals. Clause ω_1 has one positive literal and one negative literal. Clause ω_2 has only positive literals whereas clause ω_3 has only negative literals.

A literal l_j of a clause ω_a that is assigned truth value 1 satisfies the clause, and the clause is said to be *satisfied*. If the literal is assigned truth value 0 then it is removed from the clause. A clause with a single literal is said to be *unit*. Given a unit clause, the unit clause rule may be applied: the unassigned literal has to be assigned value 1 for the clause to be satisfied. The derivation of an *empty* clause indicates that the formula is *unsatisfied* for the given assignment. The formula is satisfied if all its clauses are *satisfied*.

Example 2. For the formula given in example 1 the truth assignment $\{x_1 = 1, x_2 = 1, x_3 = 0\}$ is a satisfying assignment and the truth assignment $\{x_1 = 1, x_2 = 1, x_3 = 1\}$ is not.

The SAT problem consists of deciding whether there exists a truth assignment to the variables such that the formula becomes satisfied.

Different SAT encodings for QCP have been well studied in the past and therefore there exist efficient ways of encoding this problem into SAT [13]. Here we present the two more straightforward ways of encoding this problem: the

minimal encoding and the *extended encoding* [11]. Those encodings use n Boolean variables per cell; each variable represents a number assigned to a cell, and the total number of variables is n^3. Let us use the notation q_{xyz} to refer to variables with $1 \leq x, y, z \leq n$. Variable q_{xyz} is assigned true if and only if the cell in row x and column y is assigned the number z. Hence, q_{253} means that the number 3 appears in row 2, column 5.

The most basic SAT encoding, which is known as the minimal encoding, includes clauses that represent the following constraints:

- At least one number must be assigned to each cell:
  $\bigwedge_{x=1}^{n} \bigwedge_{y=1}^{n} \bigvee_{z=1}^{n} q_{xyz}$
- No number is repeated in the same row:
  $\bigwedge_{y=1}^{n} \bigwedge_{z=1}^{n} \bigwedge_{x=1}^{n-1} \bigwedge_{i=x+1}^{n} (\neg q_{xyz} \vee \neg q_{xiz})$
- No number is repeated in the same column:
  $\bigwedge_{x=1}^{n} \bigwedge_{z=1}^{n} \bigwedge_{y=1}^{n-1} \bigwedge_{i=y+1}^{n} (\neg q_{xyz} \vee \neg q_{iyz})$

The other encoding, which is known as the extended encoding, adds to the minimal encoding redundant clauses that represent the following constraints:

- Each number must appear at least once in each row:
  $\bigwedge_{x=1}^{n} \bigwedge_{z=1}^{n} \bigvee_{y=1}^{n} q_{xyz}$
- Each number must appear at least once in each column:
  $\bigwedge_{y=1}^{n} \bigwedge_{z=1}^{n} \bigvee_{x=1}^{n} q_{xyz}$
- No two numbers are assigned to the same cell:
  $\bigwedge_{x=1}^{n} \bigwedge_{y=1}^{n} \bigwedge_{z=1}^{n-1} \bigwedge_{i=z+1}^{n} (\neg q_{xyz} \vee \neg q_{xyi})$

Naturally, the pre-assigned entries of the QCP will be represented as unit clauses. Both encodings have $\mathcal{O}(n^4)$ clauses. From the several experimental studies comparing these two encodings, the extended encoding has been shown to be clearly more efficient that the minimal encoding [12, 2]. Even though the minimal encoding produces smaller formulas, the extended encoding allows more propagation and has a more realistic representation of the problem structure. Hence, in this paper we will only use the extended encoding.

Definition 3 (Boolean Constraint Propagation). *The process of iterating the unit clause rule is called* ***Boolean Constraint Propagation*** *(BCP). The unit clause rule applies whenever a unit clause $\omega_i = (l_j)$ is identified. In this case, all clauses containing literal l_j are declared satisfied, and the literal $\neg l_j$ is removed from all clauses containing it. This simplification may originate new unit clauses in which case the unit clause rule should be applied again until no unit clauses remain.*

Example 3 (BCP). Consider two clauses $\omega_1 = (x_1 \vee \neg x_2 \vee x_3)$ and $\omega_2 = (x_2 \vee x_4)$. Given the partial assignment $\{x_1 = 0, x_3 = 0\}$, literal $\neg x_2$ in ω_1 must be satisfied and therefore we must assign $x_2 = 0$. Consequently, literal x_4 in clause ω_2 must also be satisfied and therefore we must assign $x_4 = 1$.

BCP is widely used in SAT as a simplification rule that can be performed in polynomial time.

After encoding the problem, for which each pre-assigned cell corresponds to a unit clause, we apply a simple preprocessor based on unit propagation. The same idea is used in the generator `lsencode` [2] by Carla Gomes. This preprocessing greatly reduces the number of variables and clauses in the encoding which increases its efficiency.

2.3 Symmetry Breaking

Recent advances in encodings include identifying and breaking symmetries [6, 4, 17]. There has been a significant effort for studying the effect of symmetry breaking in constraint satisfaction, which has further motivated the study of symmetry breaking in SAT encodings.

Definition 4 (Symmetry). *Given a SAT instance $\mathcal{P}$, with a set of constraints $\mathcal{C}$, a* ***symmetry*** *of $\mathcal{P}$ is a bijective function $f : \mathcal{A} \rightarrow \mathcal{A}$ where $\mathcal{A}$ is the set of partial or full assignments of $\mathcal{P}$, such that the following holds:*

1. *Given $a \in \mathcal{A}$, if a satisfies the constraints in $\mathcal{C}$, then so does $f(a)$.*
2. *Similarly, if a does not satisfy the constraints in $\mathcal{C}$, then neither does $f(a)$.*

Symmetries cause the existence of redundant search paths, which is a clear drawback for backtrack search. Breaking symmetries reduces the search space: this is a clearly advantage for problems having no solution, which implies traversing the whole search space to prove unsatisfiability. For the same reason, breaking symmetries is also an advantage when all the solutions must be found. Moreover, experimental evaluation has shown that (partially) breaking symmetries can also be useful for finding one solution [15]. Observe that with symmetry breaking the freedom of the search is restricted. On the other hand, there is often a trade-off between the cost of eliminating symmetries and the savings derived from having done so.

2	**3**	**4**	**5**	**1**
1	**2**	**3**	**4**	**5**
3	5	1	2	4
4	1	5	3	2
5	4	2	1	3

1	**2**	**3**	**4**	**5**
2	**3**	**4**	**5**	**1**
3	5	1	2	4
4	1	5	3	2
5	4	2	1	3

Fig. 3. Example of a row symmetry in a quasigroup.

We can find a simple example of symmetries in a completed quasigroup. Given a quasigroup we can obtain an equivalent quasigroup by permutation of

[2] Available at `http://www.cs.cornell.edu/gomes/new-demos.htm`.

the n symbols or by transposing the matrix (i.e., exchanging rows and columns). An example of this is shown in Figure 3.

Many methods have been developed for symmetry breaking. Here we present the three main ways of eliminating symmetry:

1. Remodel the problem [17]. A different encoding, e.g. obtained by defining a different set of variables, may create a problem with less symmetries.
2. Add constraints to the model [6, 1]. Such constraints merge symmetries in equivalent classes. Ideally, only one assignment will satisfy these constraints, instead of n assignments, where n is the number of elements in a given equivalent class.
3. Change the search process to avoid symmetrically equivalent states [4, 10, 7]. This can be done by adding constraints to ensure that any assignment symmetric to one assignment already considered will not be explored in the future, or by performing checks that symmetric equivalents have not been visited. This is done for both satisfying and unsatisfying assignments. However, this approach has not found success in SAT. This is unsurprising, because of the reliance of SAT solvers on very small time between branching decisions, limiting the overheads that can be accepted and ruling out these symmetry breaking techniques.

So far we have only mentioned global symmetries. Another interesting type of symmetries is local symmetries [3]. Different names have been given in the past to this same concept, namely conditional symmetries [9]. These symmetries refer to a subproblem $\mathcal{P}' \subset \mathcal{P}$ rather then a problem $\mathcal{P}$ for which a partial assignment to the problem $\mathcal{P}$ is considered. Also, these symmetries arise during search.

1	2	3	**5**	**4**
2	3	1	**4**	**5**
3	4	5	2	1
4	5	2	1	3
5	1	4	3	2

1	2	3	**4**	**5**
2	3	1	**5**	**4**
3	4	5	2	1
4	5	2	1	3
5	1	4	3	2

Fig. 4. Example of a local symmetry in a quasigroup.

Figure 4 shows an example of a local symmetry in a quasigroup: symbols 4 and 5 on the first and second rows can be permuted. Also, if we add constraints to prevent one of the solutions to be found, we can obtain the eliminated solution afterwards.

Even though local symmetries are harder to spot, we believe that dealing with this type of symmetries is an important step in the efficient resolution of hard combinatorial problems.

3 Symmetry Breaking in QCPs

Quasigroups can be represented by a matrix model. Similarly to all matrix models, we may consider the usual row and column symmetries, where rows and columns may be exchanged. Breaking these symmetries has the advantage of reducing the search space without loosing any of the solutions. Once we have found the solutions for the reformulated problem for which symmetries have been eliminated, we may recover the whole set of solutions. However, given that a QCP starts with a subset of cells pre-assigned, breaking these symmetries has no effect.

In this section we will study local symmetries in the context of QCPs. These symmetries have already been mention in the previous section: Figure 4 illustrates local symmetries for a QCP of order 5, where symbols 4 and 5 in the first and second rows may be permuted. If we take a closer look at this figure we can make a generalization of this local symmetry.

Fig. 5. Local symmetry $lsym_{22}$ in QCPs.

Consider a quasigroup $\mathcal{Q}$ and two rows (i_1, i_2), two columns (j_1, j_2) and two symbols (a, b), with $1 \leq i_1 < i_2 \leq n$, $1 \leq j_1 < j_2 \leq n$ and $a,\ b \in \{1, \ldots, n\}$. Considering also that $\mathcal{Q}[i_1, j_1]$ refers to the content of the cell in row i_1 and column j_1 of the quasigroup $\mathcal{Q}$ and assume that symbol a occurs in cells $\mathcal{Q}[i_1, j_1]$ and $\mathcal{Q}[i_2, j_2]$ and symbol b occurs in cells $\mathcal{Q}[i_1, j_2]$ and $\mathcal{Q}[i_2, j_1]$. Let us consider the two quasigroups illustrated in Figure 5. For these two quasigroups, for which a partial assignment is given, it is straightforward to identify a function that defines a local symmetry. In what follows we will refer to this local symmetry as $lsym_{22}$.

In order to break $lsym_{22}$ we impose a lexicographically order in the values of $\mathcal{Q}[i_1, j_1]$ and $\mathcal{Q}[i_2, j_1]$ by extending our encoding with additional constraints. For each set of four cells where the pattern shown in Figure 5 may occur for symbols a and b, we add the following constraints to guarantee that $\mathcal{Q}[i_1, j_1] < \mathcal{Q}[i_2, j_1]$:

- If $a < b$: $\neg(q_{i_1 j_1 b} \wedge q_{i_1 j_2 a} \wedge q_{i_2 j_1 a} \wedge q_{i_2 j_2 b})$
- Else If $a > b$: $\neg(q_{i_1 j_1 a} \wedge q_{i_1 j_2 b} \wedge q_{i_2 j_1 b} \wedge q_{i_2 j_2 a})$

This means that only one of the assignments given in Figure 5 may occur. If $a > b$ then the first partial assignment given in Figure 5 (left) cannot occur,

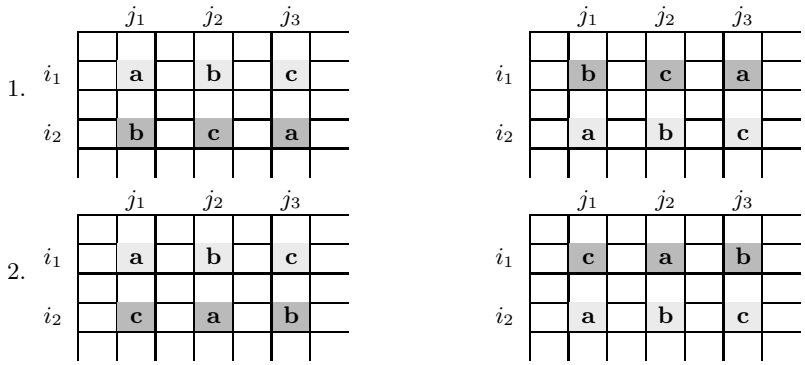

Fig. 6. Local symmetry $lsym_{23}$ in QCPs.

otherwise if $a < b$ then the other partial assignment given in Figure 5 (right) cannot occur. Observe that with those clauses we prevent one of the partial assignments from occurring, although we may not guarantee that one of them will occur in the solution found.

This reasoning may be extended to patterns including more than four cells. For example, in Figure 6 we illustrate another local symmetry denoted as $lsym_{23}$. Basically we take into account three symbols (a, b, c) and six cells distributed by two rows (i_1, i_2) and three columns (j_1, j_2, j_3), with $1 \leq i_1 < i_2 \leq n$, $1 \leq j_1 < j_2 < j_3 \leq n$ and $a,\ b,\ c \in \{1, \ldots, n\}$. In this case, the symbols can be assigned in four distinct ways, but only two local symmetries can be identified, thus grouping the four patterns in two groups. We may obtain one pattern from another by permuting rows i_1 and i_2. We can break these local symmetries similarly to what we have done with $lsym_{22}$, i.e., by imposing a lexicographical order between the values of $\mathcal{Q}[i_1, j_1]$ and $\mathcal{Q}[i_2, j_1]$. To guarantee that $\mathcal{Q}[i_1, j_1] < \mathcal{Q}[i_2, j_1]$ (for case 1.) we add the following constraints:

- If $a < b$: $\neg(q_{i_1 j_1 b} \wedge q_{i_1 j_2 c} \wedge q_{i_1 j_3 a} \wedge q_{i_2 j_1 a} \wedge q_{i_2 j_2 b} \wedge q_{i_2 j_3 c})$
- Else If $a > b$: $\neg(q_{i_1 j_1 a} \wedge q_{i_1 j_2 b} \wedge q_{i_1 j_3 c} \wedge q_{i_2 j_1 b} \wedge q_{i_2 j_2 c} \wedge q_{i_2 j_3 a})$

Similarly, to guarantee that $\mathcal{Q}[i_1, j_1] < \mathcal{Q}[i_2, j_1]$ (for case 2.), the following clauses are added:

- If $a < c$: $\neg(q_{i_1 j_1 c} \wedge q_{i_1 j_2 a} \wedge q_{i_1 j_3 b} \wedge q_{i_2 j_1 a} \wedge q_{i_2 j_2 b} \wedge q_{i_2 j_3 c})$
- Else If $a > c$: $\neg(q_{i_1 j_1 a} \wedge q_{i_1 j_2 b} \wedge q_{i_1 j_3 c} \wedge q_{i_2 j_1 c} \wedge q_{i_2 j_2 a} \wedge q_{i_2 j_3 b})$

We may further consider the same pattern but now within three rows and two columns instead, obtaining the local symmetries $lsym_{32}$ shown in Figure 7. Again, we may group these four pattern in two groups of two patterns, for which one may be obtained from the other by permuting columns j_1 and j_2. Clearly, these symmetries may be broken by adding similar clauses to the ones added for

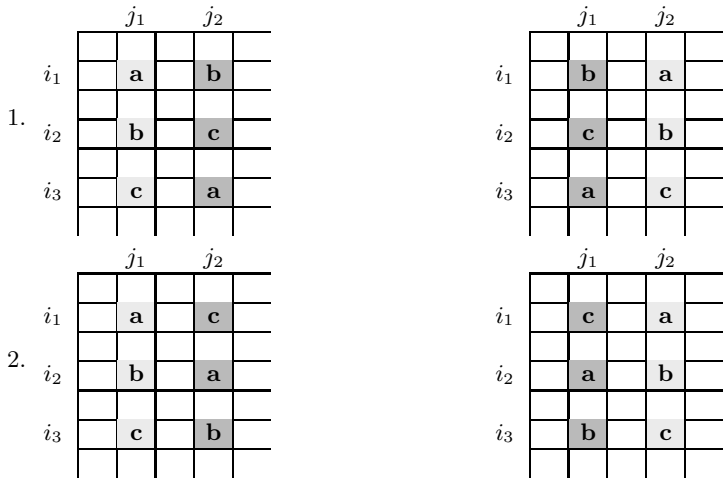

Fig. 7. Local symmetry $lsym_{32}$ in QCPs.

the $lsym_{23}$ case. Those clauses will guarantee that $\mathcal{Q}[i_1, j_1] < \mathcal{Q}[i_1, j_2]$. Finally, observe that many other local symmetries, similar to the ones that we have just mentioned, may arise in QCPs (for example, see Figure 8). Such symmetries involve more rows and columns (and eventually more symbols) and are clearly more complex.

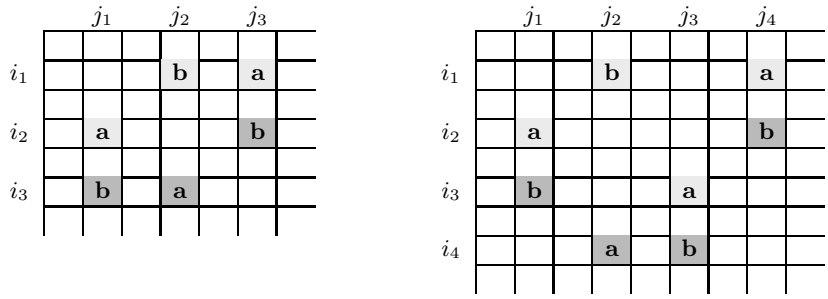

Fig. 8. Example of other (more complex) local symmetries in QCPs.

4 Experimental Results

In this section we compare the efficiency of the encodings presented in the previous section against the encoding that does not break symmetries. On a first

approach, we will study the impact of breaking local symmetries on *satisfiable* QCP instances and afterwards we will study the impact of breaking those symmetries on *unsatisfiable* QCP instances. Observe that for *satisfiable* problem instances the search terminates when one solution is found, regardless the fact that other solutions may exist. On the other hand, for proving *unsatisfiability* the whole search space has to be traversed. For this reason, breaking symmetries is expected to speedup solving *unsatisfiable* instances even more than *satisfiable* instances. We should note, however, that for other search strategies, rather than tree search, symmetry breaking may eventually not speedup the search (e.g. for local search [14]).

For the experiments reported bellow we have used the satisfiable QCP problem instances from [2] and have generated our own unsatisfiable problem instances. All these instances are located near the phase transition. The results were obtained on an Intel Xeon 5160 (3.0GHz with 4GB of RAM) and a timeout of 1000s.

4.1 Satisfiable QCP instances

The local symmetries presented in the previous section occur very often in QCPs. This fact can be confirmed comparing the number of solutions for QCP instances with and without local symmetry breaking clauses. We have run `relsat` [3] to perform this comparison. We were able to count all the solutions for 30 instances of order 30. (Larger instances could not be tested in a reasonable amount of time due to QCPs having in general a significant number of solutions and an increasing difficulty that grows exponentially with its order. Hence, a small n was used so that we could get all solutions.) Table 1 shows the percentage of solutions eliminated by breaking local symmetries. Each value represents the average number of solutions eliminated for the 30 problem instances. Results are given for each one of the local symmetries broken ($lsym_{22}, lsym_{23}, lsym_{32}$) and for the combination of all of them as well ($lsym_{all}$).

$lsym_{22}$	$lsym_{23}$	$lsym_{32}$	$lsym_{all}$
77.191	8.910	10.934	81.668

Table 1. Reduction of the number of solutions when using the different encodings.

Clearly, breaking local symmetries of type $lsym_{22}$ causes a significant reduction in the number of solutions of a given problem instance. Breaking this type of local symmetry is indeed extremely useful. Although it suffices to find only one solution, the significant reduction of the search space may help to find one solution faster.

[3] Available from `http://www.bayardo.org/resources.html`.

Order	w/o $lsym$	$lsym_{22}$	$lsym_{23}$	$lsym_{32}$
35	22,088.3	+2,852.5	+15,255.1	+14,707.1
37	24,805.4	+3,004.1	+15,377.4	+15,418.6
40	28,476	+3,255.2	+17,107.6	+16,723.1
43	31,808.8	+3,230.9	+15,641.9	+15,317.2
45	35,479.9	+3,665.5	+18,206.6	+18,176.5

Table 2. Number of additional literals to encode $lsym_{22}$, $lsym_{23}$ and $lsym_{32}$.

Order	w/o $lsym$	$lsym_{22}$	$lsym_{32}$	$lsym_{23}$	w/o $lsym$	$lsym_{22}$	$lsym_{32}$	$lsym_{23}$
35	100	100	100	100	0.66	0.64	0.825	0.635
37	100	100	100	100	3.44	3.37	3.495	4.015
40	100	100	100	100	18.76	18.63	26.645	19.92
43	90	91	90	89	120.66	134.41	156.11	170.655
45	68	69	68	70	665.22	633.55	802.835	740.22

Table 3. Satisfiable instances using `satz` with a time limit of 6000s.

Table 2 shows the average number of additional literals needed to break the different types of local symmetries. This table includes experiments for 500 satisfiable instances from [2]. These instances can be distinguished into five subsets of 100 instances having different orders: 35, 37, 40, 43 and 45. Remember that each local symmetry is broken by adding one clause. In the case of $lsym_{22}$ all clauses added have 4 literals and for $lsym_{23}$ and $lsym_{32}$ all clauses added have 6 literals. Given that symmetries involving more cells have a lower probability of occurring and are broken at the cost of adding larger clauses, in this section we will focus on the effect of breaking symmetries $lsym_{22}$, $lsym_{23}$ and $lsym_{32}$. Although not many $lsym_{23}$ and $lsym_{32}$ symmetries occur in practice, before performing the search many of these symmetries have to be considered. Hence, a *huge* number of clauses is added. This is not the case for the $lsym_{22}$ symmetries. In theory they cannot occur very often but in practice they occur almost as many times as they occur in theory.

Table 3 shows the percentage of instances solved for each configuration, as well as the CPU time (in seconds) required for finding one solution. The given CPU time refers to the median value obtained from running each subset of 100 problem instances. Although different SAT solvers have been tried, `satz` [4] has came out as the most efficient solver for solving QCPs. (This conclusion has also been reached in previous work [2].) `satz` is a backtrack search SAT solver enhanced with unit propagation and a look-ahead heuristic. This table clearly shows that breaking local symmetries seems not to help solving these problems instances. Although a slightly improvement can be observed for $n = 45$ and $lsym_{22}$, it is not representative. For the remaining cases, `satz` requires in general more time when symmetry breaking clauses are added.

[4] Available from `http://www.laria.u-picardie.fr/~cli/EnglishPage.html`.

These results came as a surprise: we were expecting that symmetry breaking would reduce the CPU time, given that the number of solutions and the search space are dramatically reduced. The only possible explanation for this fact is the heuristic. Clearly, the heuristic is *badly* affected by the new clauses. In order to clarify this fact, we have partially disabled `satz`'s heuristic. The look-ahead heuristic implemented in *satz* chooses the variable that once assigned will imply the highest number of assignments due to unit propagation. We now simply choose the *first unassigned variable* to branch on. This makes the heuristic to choose the variables following a fixed order which is a non-biased approach. This new version of `satz` is called blind`satz`.

$w/o\ lsym$	$lsym_{22}$	$lsym_{32}$	$lsym_{23}$	$lsym_{all}$
88.97	81.57	88.87	88.97	81.095

Table 4. Satisfiable instances using blind`satz` with a time limit of 1000s.

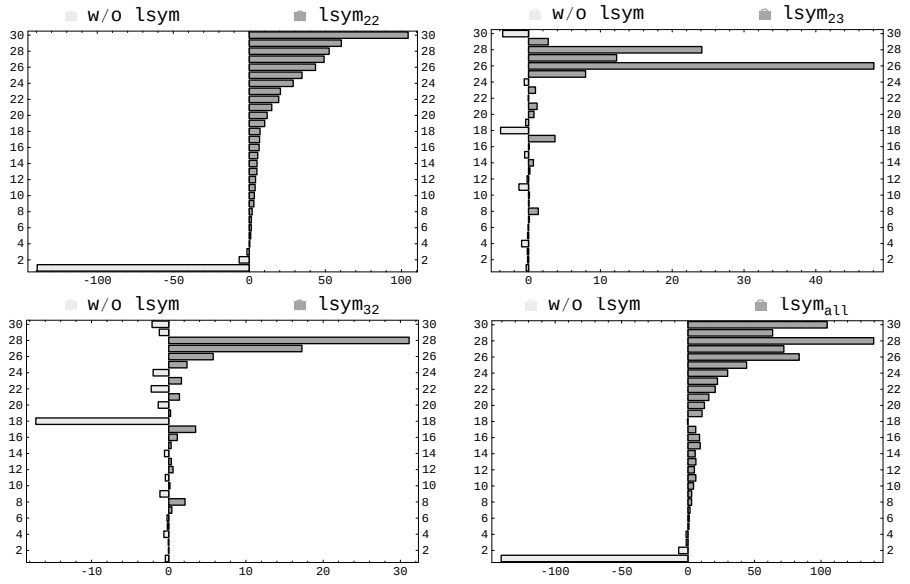

Fig. 9. Satisfiable instances against no symmetry breaking ($w/o\ lsym$).

Table 4 shows the median CPU time required to find a solution using blind`satz`. Results are reported for only 30 instances with $n = 37$. (No larger instances could be tried due to blind`satz` being much slower that `satz`.) For blind`satz`, breaking symmetries of type $lsym_{22}$ improves the performance. But breaking symmetries of types $lsym_{23}$ and $lsym_{32}$ has almost no impact in the CPU time,

most probably because these symmetries rarely occur in practice and there is a significant overhead on dealing with additional clauses.

Figure 9 compares directly the performance of using each local symmetry breaking technique with no symmetry breaking. Each bar represents one of the 30 instances. The bar size corresponds to the difference (in seconds) between using no symmetry breaking and using a specific type of symmetry breaking (either $lsym_{22}$, $lsym_{23}$, $lsym_{32}$ or $lsym_{all}$). From the figure, it is clear that $lsym_{22}$ is able to reduce the required CPU time for all but 3 problem instances. This reduction can be up to 105 seconds. The advantages of using only $lsym_{23}$ or $lsym_{32}$ are not so clear, although a slight improvement may be observed. Finally, $lsym_{all}$ also reduces the CPU time, even though the reduction is not as large in general as for the $lsym_{22}$ approach. However, the $lsym_{all}$ approach seems to be more robust.

4.2 Unsatisfiable QCP instances

$w/o\ lsym$	$lsym_{22}$	$lsym_{32}$	$lsym_{23}$	$lsym_{all}$
376.075	360.655	378.52	377.955	358.47

Table 5. Unsatisfiable instances using blind`satz` with a time limit of 1000s.

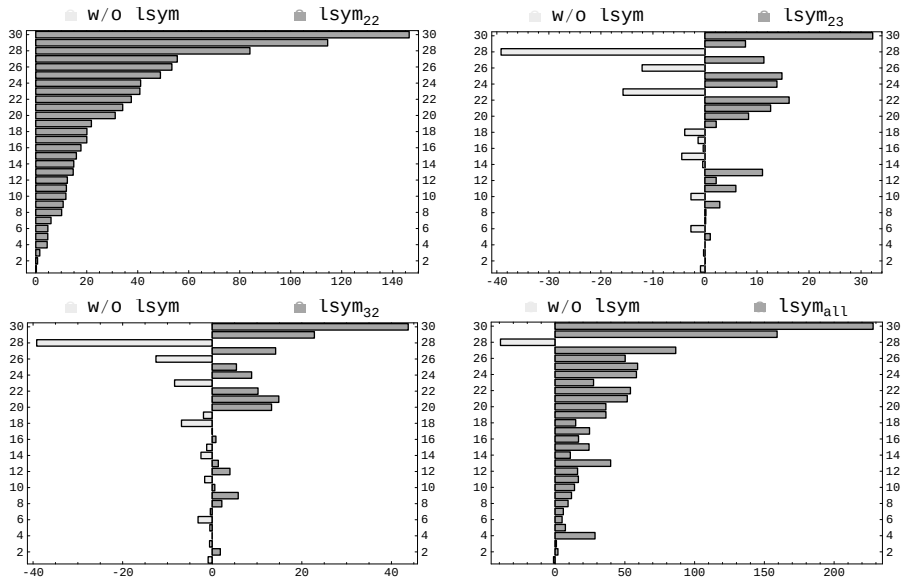

Fig. 10. Unsatisfiable instances against no symmetry breaking ($w/o\ lsym$).

We finally evaluate the impact of local symmetry breaking in unsatisfiable problem instances. The first step for this evaluation was the creation of a generator of unsatisfiable instances. This was done based on [16]. The new generator generated 30 problem instances of order 35 with 67% pre-assigned values (this value corresponds to the phase transition). Again, we observed that `satz` is more efficient when no symmetry breaking clauses are added. For this reason, blind`satz` was tried as an alternative.

Table 5 shows the median CPU time needed by blind`satz` to prove unsatisfiability. Moreover, Figure 10 gives four bar charts comparing the results for the four local symmetry breaking configurations ($lsym_{22}, lsym_{22}, lsym_{32}, lsym_{all}$) against no symmetry breaking. With no surprise, $lsym_{22}$ and $lsym_{all}$ have the best performance. Again, $lsym_{23}$ and $lsym_{32}$ do not improve much the performance of the basic encoding in terms of efficiency. In addition, we also observed that breaking symmetries seems to have more impact on solving harder problem instances.

5 Conclusions and Future Work

In this paper we have evaluated the impact of breaking local symmetries on SAT encodings for the quasigroup completion problems. We have identified different types of local symmetries and observed that some of them can be quite effective on reducing the number of solutions. However, the addition of new clauses for breaking symmetries has a negative impact on the performance of the SAT solver. This is due not only to the overhead of dealing with additional clauses but also to the heuristics being used by SAT solvers. These heuristics have been designed not having these clauses into account. As future work we envision developing new heuristics for coping with symmetry breaking clauses.

Acknowledgments This work is partially supported by Fundação para a Ciência e Tecnologia under research projects POSI/SRI/41926/01 and POSC/EIA/-/61852/2004.

References

1. F. Aloul, K. A. Sakallah, and I. Markov. Efficient symmetry breaking for boolean satisfiability. In *International Joint Conference on Artificial Intelligence*, pages 271–276, August 2003.
2. C. Ansótegui, A. del Val, C. F. Iván Dotú, and F. Manyá. Modeling choices in quasigroup completion: SAT vs CSP. In *Proceedings of the National Conference on Artificial Intelligence*, 2004.
3. B. Benhamou and M. R. Saidi. Eliminating local symmetry in CSP. In *International Symmetry Conference, ISC'07*, 2007.
4. C. A. Brown, L. Finkelstein, and P. W. Purdom. Backtrack searching in the presence of symmetry. In *6^{th} International Conference, on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, volume 357 of *Lecture Notes in Computer Science*, pages 99–110. Springer-Verlag, 1988.

5. C. Colbourn. The complexity of completing partial latin squares. *Discrete Applied Mathematics*, pages 25–30, 1984.
6. J. M. Crawford, M. L. Ginsberg, E. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *Proceedings of the International Conference on Principles of Knowledge and Reasoning*, pages 148–159, 1996.
7. T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In *International Conference on Principles and Practice of Constraint Programming*, pages 93–107, 2001.
8. M. Fujita, J. Slaney, and F. Bennett. Automatic generation of some results in finite algebra. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 52–57, 1993.
9. I. P. Gent, T. Kelsey, S. A. Linton, I. McDonald, I. Miguel, and B. M. Smith. Conditional Symmetry Breaking. In *Principles and Practice of Constraint Programming - CP 2005*, pages 256–270, 2005.
10. I. P. Gent and B. M. Smith. Symmetry breaking during search in constraint programming. In *Proceedings of the European Conference on Artificial Intelligence*, pages 599–603, 2000.
11. C. P. Gomes and D. Shmoys. Completing quasigroups or latin squares: A structured graph coloring problem. In *Proceedings for Computational Symposium on Graph Coloring and Generalizations*, Lecture Notes in Computer Science, pages 22–39, 2002.
12. C. P. Gomes and D. Shmoys. The promise of LP to boost CSP techniques for combinatioral problems. In *CP-AI-OR'02*, pages 291–305, 2002.
13. H. A. Kautz, Y. Ruan, D. Achlioptas, C. P. Gomes, B. Selman, and M. Stickel. Balance and filtering in structured satisfiable problems. In *IJCAI'01*, pages 351–358, 2001.
14. S. Prestwich. First-solution search with symmetry breaking and implied constraints. In *CP Workshop on Modelling and Problem Formulation*, 2001.
15. A. Ramani and I. L. Markov. Automatically exploiting symmetries in constraint programming. In *Recent Advances in Constraints*, volume 3419 of *Lecture Notes in Computer Science*, pages 98–112, 2005.
16. P. Shaw, K. Stergiou, and T. Walsh. Arc consistency and quasigroup completion. In *Proceedings of the ECAI-98 workshop on non-binary constraints*, 1998.
17. B. M. Smith. Reducing symmetry in a combinatorial design problem. In *Third International Workshop on Integration of AI and OR Techniques*, pages 351–359, 2001.