The Impact of Modelling in SAT Problem Solving

Ruben Martins
IST/INESC-ID
Technical University of Lisbon
Lisbon, Portugal
ruben@sat.inesc-id.pt

Abstract

When given a SAT problem, one has two major tasks: to model the problem and to solve the selected model. Whilst much work in SAT algorithms is for building efficient solvers, we argue that many modeling decisions have a direct impact on the solvers performance. We will focus on four distinct problems: Social Golfer, Round Robin, Quasigroup Completation and Towers of Hanoi. Our empirical evaluation on those problems shows that different encodings can improve or degrade search dramatically.

1 Introduction

Recently, we have seen a remarkable progress in propositional satisfiability (SAT), with theoretical and practical contributions. Even though SAT solvers have exponential run time in the worst case, they can currently be used to solve hard problem instances.

The SAT community is extremely motivated for continuously improving SAT solvers performance. However, there is much to be done with respect to SAT encodings. We believe that many applications do not benefit from the efficiency in SAT solving due to inefficiencies introduced while producing SAT encodings.

This paper contributes to a better understanding of the impact of SAT modelling. For that, we will study four distinct problems: Social Golfer, Round Robin, Quasigroup Completation and Towers of Hanoi.

The rest of the paper is organized as follows. The next section gives some insights on how to encode a problem into SAT. Sections 3, 4, 5 and 6, respectly, describe the Social Golfer, Round Robin, Quasigroup Completation and Towers of Hanoi problems. In each of those sections we propose SAT encodings for each problem and compare them to already known encodings. Finally, we conclude the paper and suggest future work.

2 Background

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. The formula is satisfied if all its clauses are satisfied.

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

To encode a problem into SAT one must define a set of variables and a set of constraints on the variables. The set of variables may be defined based on different criteria: the most intuitive variables set, the set with minimum cardinality, the set that will require the smallest number of clauses, etc. Moreover, the definition of the set of constraints may require the definition of additional auxiliary variables. In some cases, these variables are really essential; in other cases, we prefer to have more variables rather than more clauses.

One important step when encoding a problem into SAT is dealing with symmetries. Symmetries cause the existence of redundant search paths, which is a clear drawback for backtrack search. 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.

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

1. Remodel the problem[21, 10]. 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[18]. Such constraints merge symmetries in equivalent classes.

3. Change the search process to avoid symmetrically equivalent states[3, 7].

3 Social Golfer

The Social Golfer problem is derived from a question posted to `sci.op-research` in May 1998:

On a local golf club, there are 32 social golfers, each of whom plays golf once a week, and always in groups of 4. Our goal is to to come up with a schedule of plays for these golfers, to last as many weeks as possible, such that no golfer plays in the same group as any other golfer on more than one occasion. In other words, this problem can be described more explicitly by enumerating four constraints which must be satisfied:

1. The golf club has 32 members.
2. Each member plays golf once a week.
3. Golfers always play in groups of 4.
4. No golfer plays in the same group as any other golfer twice.

For some years, it was not known if a 10 week solution for 32 golfers exists. In 2004, Aguado found a solution using design-theoretic techniques[1]. So far, no constraint programming technique has been able to solve this instance, so it remains a valuable benchmark for the constraint programming community. The best known solution from constraint programming is from Stefano Novello, who found out a 9-week solution by using ECL^{i}PS^e.

Even though the Social Golfer problem was described for 32 golfers playing in groups of 4, it can be easily generalized. An instance to the problem is characterized by a triple *w-p-g*, where w is the number of weeks, p is the number of players per group and g is the number of groups.

For example, figure 1 gives a solution for the social golfer problem 3-2-2, i.e. for scheduling 4 golfers playing in 2 groups of 2 golfers each for 3 weeks.

Week	Group 1	Group 2
1	[1,2]	[3,4]
2	[1,3]	[2,4]
3	[1,4]	[2,3]

Figure 1: A solution for the Social Golfer problem 3-2-2.

The Social Golfer problem is related with other well-known combinatorial problems, such as, Round Robin[5] and Kirkman's Schoolgirl[4].

The social golfer problem is also well-known for being a case study of symmetry for constraint programming (e.g. see [21, 13]). This problem is highly symmetric, exhibiting the following symmetries:

- Golfers within a group are interchangeable. Order has no significance for groups of golfers.
- Groups within a week are interchangeable. Again, order has no significance when considering groups within a week.
- Weeks are interchangeable. There are no order constraints with respect to weeks.

3.1 A SAT encoding for the Social Golfer problem

We have defined SAT variables based on the possible groups of golfers:

- $GROUP(X)_k$, where X represents any of the possible groups of golfers with size $p \times g$ and with $1 \leq k \leq w$.

Since we have w weeks, we need $\binom{x}{p} \times w$ variables to encode our problem.

We define the set of constraints as following:

- Each golfer plays at least once per week:
  $\bigwedge_{k=1}^{w} \bigwedge_{j=1}^{p \times g} \bigvee_{X} GROUP(X)_k$, with $j \in X$
- Each golfer plays at most once per week:
  $\bigwedge_{k=1}^{w} \bigwedge_{j=1}^{p \times g} \bigwedge_{X} \neg GROUP(X)_k \vee \neg GROUP(X')_k$, with $j \in \{X, X'\}$ and $X \neq X'$
- No golfer plays in the same group as any other golfer more than once:
  $\bigwedge_{k=1}^{w} \bigwedge_{y=1}^{w} \bigwedge_{j=1}^{p \times g} \bigwedge_{z=j+1}^{p \times g} \bigwedge_{X} \bigwedge_{X'} \neg GROUP(X)_k \vee \neg GROUP(X')_y$, with $j, z \in \{X, X'\}$ and $X \neq X'$

In order to break the symmetries between groups and between players we add the following constraints:

- The group of golfers, $GROUP(X)$, with $\{1, j\} \in X$, plays in week j:
  $\bigwedge_{j=1}^{w} GROUP(X)_k$, com $1, j \in X$
- We fix the first week as following:
  $GROUP(1, \ldots, g)_1$
  $\ldots$
  $GROUP(p \times g - g + 1, \ldots, p \times g)_1$

Since our variables represent each group, the symmetries inside each group are automactly deleted.

Our goal with this encoding is to solve the Social Golfer problem instances with the form *w-2-g*. We will

not use this encoding on problems with larger groups since our formula grows exponentially with the size of the group. Although we are only encoding a small subset of problems, we hope to solve them efficiently. We will also use this encoding for the Round Robin problem, which is described in the next section.

3.2 Experimental Results

In this section we will compare our encoding with the one presented by Gent and Lynce[9]. For that, we used an Intel Xeon 5160 (3.0GHz with 4GB of RAM) and the solver `Minisat`[1].

Problem	Gent & Lynce	Martins
15-2-8	1.19	0.1
17-2-9	333.2	0.2
19-2-10	293.38	0.5
21-2-11	3.46	1
23-2-12	97.18	1.57
25-2-13	138.45	14.39
27-2-14	597.43	6.85
29-2-15	946.56	9.81

Figure 2: Comparison between Gent and Lynce encoding and our own encoding.

Figure 2 show us that our encoding is more efficient than the one proposed by Gent and Lynce. We also observe that our encoding is more compact than Gent and Lynce encoding, i.e., uses fewer variables and clauses. Even though our encoding grows exponentially with the size of the group, we also compared both encodings for the Social Golfer problem instance 7-3-5 which is equivalent to the Kirkman's Schoolgirl problem. For this problem, our encoding produces a formula with 3,185 variables and 726,285 clauses, while Gent and Lynce encoding produces a formula with only 5,775 variables and 42,555 clauses. However, even though our encoding produces a larger formula we were able to solve this problem instance within 12 seconds while Gent and Lynce took around 220 seconds to solve this instance.

As we can see in Kirkman's Schoolgirl problem we already have a significant amount of clauses. If we tried to encode problems with a larger group size we would get formulas that are impractical for current solvers. For example, if we would tried to encode the Social Golfer problem instance 5-4-4 we would get a formula with 14,123,280 clauses.

[1]Available from `http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/`.

4 Round Robin

In sports scheduling one of the issues is to find a feasible schedule for a sports league that takes into consideration constraints on how the competing teams can be paired, as well as how each game is distributed in the entire schedule. Here we consider the timetabling problem for the classic round robin schedule.

The n-team round robin problem, that is the round robin problem for n teams, is defined as follows:

1. There are n teams, with n even, and every two teams play each other exactly once.
2. The season lasts n - 1 weeks.
3. Every team plays one match per week.
4. There are $\frac{n}{2}$ fields and, each week, each field is schedule for one game.
5. No team plays more than twice in the same field over the course of the season.

The problem then is to schedule the tournament with respect to all these constraints. Figure 3 below shows an example of a valid schedule for 6-team round robin problem.

	Week 1	Week 2	Week 3	Week 4	Week 5
Field 1	1 2	2 3	1 5	4 6	3 5
Field 2	3 4	1 6	3 6	2 5	1 4
Field 3	5 6	4 5	2 4	1 3	2 6

Figure 3: A 6-team Round Robin timetable.

As shown in 3, a configuration may be represented as a matrix with weeks in columns and fields in rows. Each column satisfies the cardinality constraint: each team appears exactly once, that is all the teams are different. In each row, no team appears more than twice. There is also a global constraint on the matrix: each match appears once, that is all matches are different.

Similiary to the Social Golfer, the Round Robin problem is also highly symmetric, since players, weeks and fields are interchangeable.

4.1 Round Robin as a SAT problem

We considered Manyà and Béjar[5] encoding and tried to improve it by adding additional constraints. We focus on improving their encoding by breaking week and field symmetries and by introducing a *dummy* column[19].

	Week 1	Week 2	Week 3	Week 4	Week 5	Dummy
Field 1	1 2	4 5	3 6	2 3	1 6	4 5
Field 2	3 4	2 6	1 4	1 5	3 5	2 6
Field 3	5 6	1 3	2 5	4 6	2 4	1 3

Figure 4: An example of the *dummy* column.

To deal with week symmetries we force the matches of the form $(1, n)$, with $n \geq 2$, to occur in the week $n-1$. To break field symmetries we impose a fixed first week where the matches of the form $(1, 2)$ to $(n-1, n)$ occur. We will denote the encoding that deals with symmetry breaking by *symmetries*.

We also introduced a *dummy* column. The encoding of this additional column is able to guarantee that each team, plays exactly twice in each field. An example of the *dummy* column can be seen in the figure 4.1.

The encoding that uses the dummy column is called *dummy*. The encoding that has both the dummy column and symmetry breaking is denoted *all*.

Besides improving this encoding we also produced a new encoding for this problem that is based on the encoding for the Social Golfer presented in the previous section.

We have defined SAT variables based on the possible matches in each week and field:

- $\mathcal{V} = \{FIELD_k(X) \cup WEEK_j(X)\}$, where X represents any of the possible matches, with $1 \leq k \leq \frac{n}{2}$ and $1 \leq j \leq n-1$.

$FIELD_k(X)$ indicates that the match X occurs in the field k and $WEEK_j(X)$ indicates that the match X occurs in the week j.

Similarly to what we have done in the previous section, we encoded the following constraints using the variables $FIELD$:

- Each team plays at most twice in each field.
- Each team plays exactly once against each other team.

Using the variables $WEEK$ we encoded the following constraints:

- Each team plays exactly once in each week.
- Each team plays exactly once against each other team.

Then we need to relate both variables, we did that by encoding the following constraint:

- If two different matches occur in the same week then they must occur in different fields. Similarly, if they both occur in the same field then they must occur in different weeks.

Field and week symmetries were dealt in the same way as described in the Manyà and Béjar improved encoding. This is encoding is called Martins.

4.2 Experimental Results

In this section we compare the encodings presented in the previous section with the one presented by Manyà and Béjar. For that, we used an Intel Xeon 5160 (3.0GHz with 4GB of RAM) and the solver. A "-" on the following results means that the solver reached the time limit.

	n		
Encoding	10	12	14
Manyà & Béjar	4.53	25.43	-
symmetries	1.1	4.55	660.31
dummy	2.8	92	-
all	0.76	5	4493.65
Martins	0.22	8.76	4.4

Figure 5: Comparison between the different encodings with a time limit of 6000 seconds.

As we can see in figure 5, our changes into the original encoding of Manyà and Béjar produced better results in comparison with the original encoding. The *symmetries* encoding was the most significant improvement. This supports the idea that symmetry breaking is essential when encoding a problem into SAT. The *dummy* encoding while being superior to the original encoding did not match the remaining encodings.

Moreover, the encoding with the best performance was the one denoted by Martins.

5 Quasigroup Completation Problem

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.

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].

In this paper we will study the quasigroup completion problem (QCP), which is the NP-complete problem of filling a partial Latin square [6]: 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 6 shows a QCP of order 5 and a possible solution.

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**

Figure 6: Example of a QCP of order 5.

5.1 QCP as a SAT problem

Different SAT encodings for QCP have been well studied in the past and therefore there exist efficient ways of encoding this problem into SAT [15]. The two most straightforward ways of encoding this problem are the *minimal encoding* and the *extended encoding* [11]. We used the extended encoding with additional constraints that deal with symmetry breaking.

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. However, given that a QCP starts with a subset of cells pre-assigned, breaking these symmetries has no effect in practice.

Although we cannot study global symmetries in the context of QCPs, we can study another type of symmetries. In this section we will identify and encode local symmetries that occur in QCPs.

Figure 7 shows a type of local symmetry that is present in QCPs.

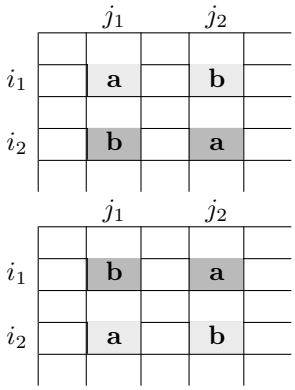

Figure 7: Local symmetry $lsym_{22}$ in QCPs.

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 $a,\ b \in \{1, \ldots, n\}$. Consider 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 7. 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 lexicographical 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 7 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 7 may occur. If $a > b$ then the first partial assignment given in Figure 7 (left) cannot occur, otherwise if $a < b$ then the other partial assignment given in Figure 7 (right) cannot occur. Observe that with these 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. We can use the same idea

present in $lsym_{22}$ to identify local symmetries with 2 rows and 3 columns and with 3 columns and 2 rows. These local symmetries will be referred to as $lsym_{23}$ and $lsym_{32}$.

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.

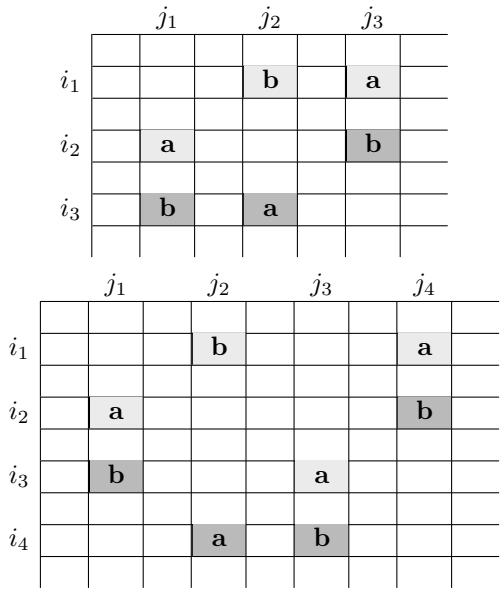

Figure 8: Example of other (more complex) local symmetries in QCPs.

5.2 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.

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) using `satz`[2].

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

5.2.1 Satisfiable 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. Figure 9 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

Figure 9: 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.

Figure 10 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 hundred times each subset of 100 problem instances.

This figure 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.

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 branching 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`.

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

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

Figure 10: Satisfiable instances using `satz` with a time limit of 6000s.

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

Figure 11: Satisfiable instances using blind`satz` with a time limit of 1000s.

Figure 11 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.

5.2.2 Unsatisfiable Instances

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

Figure 12: Unsatisfiable instances using blind`satz` with a time limit of 1000s.

We finally evaluate the impact of local symmetry breaking in unsatisfiable problem instances. The first step for this evaluation was to build a generator of unsatisfiable instances. This was done based on [20]. 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.

Figure 12 shows the median CPU time needed by blind`satz` to prove unsatisfiability. 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.

6 Towers of Hanoi

The Towers of Hanoi are a mathematical puzzle invented by the French mathematician, Edouard Lucas, in 1883. It consists of three towers, and a number of disks of different sizes which can slide onto any tower. The puzzle starts with the disks neatly stacked in order of size on one tower, the smallest at the top, thus making a conical shape.

The objective of the puzzle is to move the entire stack to another tower, obeying the following rules:

- Only one disk may be moved at a time.
- Each move consists of taking the upper disk from one of the towers and sliding it onto another tower, on top of the other disks that may already be present on that tower.
- No disk may be placed on top of a smaller disk.

For an easier understanding we will consider that the disks are numbered in an increasing order, from the smallest to the largest. Given n disks there is always a solution with $2^n - 1$ steps. For each problem, we also know which disk will be moved at each step. This is given by the following recursive function (seq): $S_1 = \{1\};\ S_n = \{S_{n-1}, n, S_{n-1}\}$.

The Towers of Hanoi have some interesting properties[22] that will help us to solve this problem in a more efficient way. On the next section, we will use the following properties of this problem:

Property 1. Given the first $2^{n-1} - 1$ steps we can determine the remaining steps by doing the same steps while considering different initial and final towers.

Property 2. While moving disks, we can never have disks of the same parity in direct contact.

Property 3. At the top of all towers we can have at most one even disk.

Property 4. If we consider one additional fixed disk on the bottom of each tower, respectly numbered with $n + 1$, $n + 2$ and $n + 3$, we can guarantee that there is always one even disk at the top of all towers.

n	S&K	Prestwich	(seq)	(pp1)	(pp2)	(pp3)	(pp4)
4	0.16	0.01	0.01	0.01	0.01	0.02	0.02
5	8.31	0.14	0.01	0.01	0.08	0.10	0.15
6	54.70	1.07	0.05	0.13	1.10	1.94	1.36
7	5252.27	10.97	0.32	0.99	14.38	19.44	12.15
8	-	332.32	1.44	16.08	154.45	187.28	220.30
9	-	-	3.47	391.12	2616.45	-	3003.58
10	-	-	71.66	-	-	-	-
11	-	-	1345.33	-	-	-	-
12	-	-	6950.09	-	-	-	-

Figure 13: Comparison between the different encodings.

6.1 Towers of Hanoi as a SAT problem

We took the well known encoding of Prestwich[17], that is based on the STRIPS[8] language, and improved it by encoding the properties described in the previous section.

Prestwich encoding is based on the following set of variables:

- $obj(d_j, i)$, with $1 \leq j \leq n$ and $1 \leq i \leq 2^n - 1$;
- $origin(t_j, i)$, with $1 \leq j \leq 3$ and $1 \leq i \leq 2^n - 1$;
- $dest(t_j, i)$, with $1 \leq j \leq 3$ and $1 \leq i \leq 2^n - 1$;
- $on(d_j, t_k, i)$, with $1 \leq j \leq n$, $1 \leq k \leq 3$ and $1 \leq i \leq 2^n$;

The variable *obj* describes which disk is moved in each step; the variable *origin/dest* describes the initial/final tower of each step and the variable *on* describes which tower each disk is on.

We encode property 1 (pp1) by reducing our search space to the first $2^{n-1} - 1$ steps and by changing the goal state so that we have the larger disk on the first tower and the remaining disks on the second tower.

Property 2 (pp2) is encoded in a straightforward way.

To encode property 3 (pp3) we used an auxiliary variable denoted by $top(d_j, t_k, i)$, with $1 \leq j \leq n$, $1 \leq k \leq 3$ e $1 \leq i \leq 2^n$. This variable describes which disk is on top of each tower.

To encode the additional fixed disks in property 4 (pp4) we increased the number of disks in the problem but imposed that those disks should never move their places. To guarantee that there is always one even disk at the top of all towers we used the variable *top* defined in the previous property.

6.2 Experimental Results

In this section we will compare our encoding with the Selman and Kautz[16, 14] encoding and the Prestwich encoding. For that, we used an AMD Sempron 2400+ (1666 Mhz with 1 GB of RAM) and the solver `PicoSAT`[4] with a time limit of 10,000 seconds.

In figure 13 we can see that Prestwich encoding is more efficient than Selman and Kautz encoding, being able to solve instances of size $n = 8$. However, our encodings showed a better performance than Prestwich encoding. (seq) is able to solve instances until 12 disks, which is significantly better than both encodings presented by Selman and Kautz (7 disks), and Prestwich (8 disks).

n	# vars	#claus	{(seq),(pp1)}
4	166	1,106	0.01
5	405	3,140	0.02
6	948	8,319	0.07
7	2,163	21,091	0.15
8	4,850	51,872	0.39
9	10,737	124,758	0.85
10	23,536	294,917	2.32
11	51,183	687,533	5.04
12	110,574	1,584,462	12.37

Figure 14: Combined encoding of (seq) with (pp1).

Due to our good results we tried to combine several properties with the goal of producing a better encoding. Combining (seq) with (pp1) we produced an encoding that can be solved without search. Figure 14 shows that with this encoding we can solve 12 disks in just a few seconds. We can also solve larger problem instances, however the size of the formula grows exponencially. Taking this in mind we developed a new

[4]Available from `http://fmv.jku.at/picosat/`.

encoding that can be solved without search and that is smaller than (seq) combined with (pp1). This new encoding will be presented in the next section.

6.3 A Simplified Encoding

If we consider properties (seq), (pp1), (pp2) and (pp3) we are able to produce an encoding based only on the *on* variables. By doing this we are drastically reducing the size of our encoding while mantaining the ability to solve this encoding without search. This encoding is called (simp).

n	Encoding	#vars	#claus
10	{(seq),(pp1)}	23,536	294,917
	(simp)	15,330	67,846
11	{(seq),(pp1)}	51,183	687,533
	(simp)	33,759	158,427
12	{(seq),(pp1)}	110,574	1,584,462
	(simp)	73,692	362,160

Figure 15: Comparation between the simplified encoding (simp) and {(seq),(pp1)}.

As we can see in figure 15 our simplified encoding is much smaller than the previous encoding. With the simplified encoding we were able to solve instances with 18 disks, which is significantly better than the previous encodings.

7 Conclusions and Future Work

Taking into account the experimental results for the problems described above, we can conclude that, even though an efficient SAT solver is needed, SAT modelling is also very important in order to efficiently solve benchmark problems.

In the Social Golfer problem we showed that with a simple change of variables we can produce an encoding that has better performance on a subset of problem instances. We then applied this encoding to the Round Robin problem where we also achieved good results.

In the Quasigroup Completation problem we have identified and broken different types of local symmetries. 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.

In the Tower of Hanoi problem we have presented a new encoding that is based on additional properties of this problem. With this encoding we were able to solve instances with 18 disks. This is significantly better than any of the current encodings for this problem.

SAT modelling can be done in several different ways and it is currently a field that needs deeper research. New techniques of modelling should be exploited in order to take full advantage of the SAT solver.

Acknowledgments

I would like to thank Inês Lynce for all her help and support. Without her this paper could not have been done.

This work is partially supported by Fundação para a Ciência e Tecnologia under research project SATPot (reference POSC/EIA/61852/2004).

References

[1] A. Aguado. A 10 days solution to the Social Golfer Problem (Personal Communication), 2004.

[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 (AAAI)*, 2004.

[3] R. Backofen and S. Will. Excluding Symmetries in Constraint-Based Search. *Constraints*, 7(3-4):333–349, 2002.

[4] N. Barnier and P. Brisset. Solving the Kirkman's schoolgirl problem in a few seconds. In *International Conference on Principles and Practice of Constraint Programming*, 2002.

[5] R. Béjar and F. Manyà. Solving the Round Robin Problem Using Propositional Logic. In *Proceedings of the National Conference on Artificial Intelligence*, pages 262–266, 2000.

[6] C. Colbourn. The Complexity of Completing Partial Latin Squares. *Discrete Applied Mathematics*, 8:25–30, 1984.

[7] T. Fahle, S. Schamberger, and M. Sellmann. Symmetry Breaking. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, pages 93–107, 2001.

[8] R. Fikes and N. J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2(3/4):189–208, 1971.

[9] I. Gent and I. Lynce. A SAT Encoding for the Social Golfer Problem. In *IJCAI'05 Workshop on Modelling and Solving Problems with Constraints*, 2005.

[10] C. Gervet. Interval Propagation to Reason about Sets: Definition and Implementation of a Pratical Language. *Constraints*, 1:191–244, 1997.

[11] C. P. Gomes and D. Shmoys. Completing Quasigroups or Latin Squares: A Structured Graph Coloring Problem. In *Proceedings of the Computational Symposium on Graph Coloring and Generalizations*, pages 22–39, 2002.

[12] C. P. Gomes and D. Shmoys. The Promise of LP to Boost CSP Techniques for Combinatioral Problems. In *International Conference on Integration of AI and OR Techniques*, pages 291–305, 2002.

[13] W. Harvey. Symmetry Breaking and the Social Golfer Problem. In *CP'01 Workshop on Symmetry in Constraint Satisfaction Problems*, pages 9–16, 2001.

[14] H. A. Kautz, D. McAllester, and B. Selman. Encoding Plans in Propositional Logic. In *Proceedings of the International Conference on the Principle of Knowledge Representation and Reasoning*, pages 374–384, 1996.

[15] H. A. Kautz, Y. Ruan, D. Achlioptas, C. P. Gomes, B. Selman, and M. Stickel. Balance and Filtering in Structured Satisfiable Problems. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 351–358, 2001.

[16] H. A. Kautz and B. Selman. Planning as Satisfiability. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 359–363, 1992.

[17] S. D. Prestwich. Variable Dependency in Local Search: Prevention Is Better Than Cure. In *Proceedings of the International Symposium on Theory and Applications of Satisfiability Testing (SAT)*, pages 107–120, 2007.

[18] J. F. Puget. On the Satisfiability of Symmetrical Constrained Satisfaction Problems. In *Proceedings of the International Symposium on Methodologies for Intelligent Systems*, pages 350–361, 1993.

[19] J.-C. Regin. Global Constraints. In *International Summer School on Contraint Programming*, pages 96–101, 2005.

[20] P. Shaw, K. Stergiou, and T. Walsh. Arc Consistency and Quasigroup Completion. In *ECAI'98 Workshop on Non-binary Constraints*, 1998.

[21] B. M. Smith. Reducing Symmetry in a Combinatorial Design Problem. In *International Workshop on Integration of AI and OR Techniques*, pages 351–359, 2001.

[22] T. R. Walsh. The Towers of Hanoi Revisited: Moving the Rings by Counting the Moves. *Information Processing Letters*, 15(2):64–67, 1982.