A SAT Encoding for the Social Golfer Problem

Ian P. Gent
School of Computer Science
University of St Andrews
Fife, Scotland
ipg@dcs.st-and.ac.uk

Inês Lynce
IST/INESC-ID
Technical University of Lisbon
Lisbon, Portugal
ines@sat.inesc-id.pt

Abstract

When given a combinatorial 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 focus on a particular combinatorial problem: the social golfer problem, and we show how to encode this problem into SAT. An important feature of the social golfer problem is the presence of symmetries, which can be tackled by adding more clauses to the encoding. Our empirical evaluation shows that different encodings can improve or degrade search dramatically depending on the solver. We also show empirically that by choosing the *right* encoding one may exploit the heavy-tail behavior.

1 Introduction

Recent years have seen remarkable progress in propositional satisfiability (SAT), with significant theoretical and practical contributions. Indeed, SAT solvers can currently be used to solve hard benchmark problems. State-of-the-art SAT solvers (e.g. [Moskewicz *et al.*, 2001; Goldberg and Novikov, 2002; Een and Sorensson, 2003; Kautz *et al.*, 2004; Ryan, 2004]), are with no doubt very competitive. And every year a new SAT competition is run with new solvers and new benchmarks. All solvers and benchmarks are classified according to three categories: industrial, handmade and random. Every year, almost all the previous year winners for each category are beaten by a new, more efficient solver. Also, the new solvers are able to solve part of the benchmark problems that were not solved in the previous year in a reasonable amount of time.

The progress in SAT solving has attracted the attention of researchers that usually use other technologies to solve their problems. Encoding problems in CNF format and solving them with SAT solvers is indeed a competitive approach. SAT has the advantage of being very easy in its formulation. Nonetheless, the simplicity of the CNF format makes its use very restrictive. For example, a constraint problem with a few dozen of variables may result in a SAT problem with thousands of variables and millions of clauses. Also, one may argue that a cause of inefficiency is the loss of structure during problem reductions.

Even though the SAT community is extremely motivated for continuously improving SAT solvers performance, 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. Moreover, there is a tight relation between encodings and solvers: different encodings are more or less effective depending on the solvers.

Encodings into SAT are constructed every time a new problem is converted into CNF. In this paper we focus on encoding a particular problem, the social golfer problem, studying the effect that encoding decisions have on performance. This work contributes to better understanding the interplay of satisfiability modeling and solving on combinatorial problems.

The rest of the paper is organized as follows. The next section gives some insights on how to encode a problem into SAT. Section 3 describes a particular combinatorial problem: the social golfer problem. Section 4 explains how to encode the social golfer problem into SAT, including how to break symmetries in this highly symmetric problem. Afterwards, experimental results are given for running both a local search and a backtrack search solver (walksat and siege, respectively) for the two encodings of the social golfer problem: one with no symmetry breaking and other with symmetry breaking. Finally, we conclude the paper and suggest future work.

2 Encoding a Problem into SAT

Encoding combinatorial problems as SAT problems has been mostly motivated by the recent advances in SAT solvers. The new solvers are capable of solving very large, very hard real-world problem instances, which more traditional SAT solvers are totally incapable of.

Nonetheless, only a few problems are naturally encoded as SAT problems. Combinational electronic circuits are the most paradigmatic example. Indeed, more sophisticated logics are frequently more adequate to represent most of the problems. Consequently, encoding such problems as CNF formulas may require a significant effort. Hopefully this effort will be counterbalanced by the performance of SAT solvers.

To encode a combinatorial problem into SAT one must define a set of variables and a set of constraints on the variables. Usually we represent SAT problems as CNF formulas, and therefore a formula is a conjunction of clauses, a clause is a disjunction of literals and a literal is a variable or its negation.

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. Choosing the most adequate variables *is more an art than a science*. 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.

Recent advances in encodings include identifying and breaking symmetries [Crawford *et al.*, 1996; Brown *et al.*, 1988; Smith, 2001]. 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.

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. (Even though symmetrical solutions have to be computed from the solutions found.) Moreover, experimental evaluation has shown that (partially) breaking symmetries can also be useful for finding one solution [Ramani and Markov, 2005]. 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. Complete symmetry breaking make solvers to return a unique solution from each set of symmetrically equivalent ones, which is the one found first by the variable and value ordering heuristics. But usually one is interested in finding any solution as quickly as possible, rather than guaranteeing only distinct solutions are returned.

One may envision three main different ways of eliminating symmetry:

1. Remodel the problem [Smith, 2001]. 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 [Crawford *et al.*, 1996; Aloul *et al.*, 2003]. Such constraints merge symmetries in equivalent classes. In practice, only one assignment satisfies 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 [Brown *et al.*, 1988; Gent and Smith, 2000; Fahle *et al.*, 2001]. 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.

Another approach that aims *reducing* symmetry was proposed by Pedro Meseguer and Carme Torras [Meseguer and Torras, 2001]. The idea is to use symmetry to guide the search. The authors suggest the use of variable and value selection heuristics to direct the search towards subspaces with high density of non-symmetric states.

3 The Social Golfer Problem

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

> The coordinator of a local golf club has come to you with the following problem. In her club, there are 32 social golfers, each of whom play golf once a week, and always in groups of 4. She would like you to come up with a schedule of play 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.

Since 1998, this problem has become a famous combinatorial problem. It is problem number 10 in CSPLib (`http://www.csplib.org/`). A solution is said to be optimal when *maximum socialisation* is achieved, i.e. when one golfer plays with as many other golfers as possible. Clearly, since a golfer plays with three new golfers each week, the schedule cannot exceed 10 weeks. This follows from the fact that each golfer plays with three other golfers each week. Since there is a total of 31 other golfers, this means that a golfer runs out of opponents after 31/3 weeks.

For some years, it was not known if a 10 week (and therefore optimal) solution for 32 golfers exists. In 2004, Aguado found a solution using design-theoretic techniques [Aguado, 2004]. No constraint programming technique has yet solved 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 posted a 9-week solution, along with the source of the ECL^{i}PS^e program used to find it.

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. The

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.

original question therefore is to find a solution to the w-4-8 problem, with w being the maximum, i.e. to find a solution to 10-4-8 (or prove that none exists). 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.

The social golfer problem is related with other well-known combinatorial problems. Indeed, this problem is a generalisation of the problem of constructing a round-robin tournament schedule, the main difference being that in the social golfer problem the number of players in a group may be greater than two. Also, the social golfer problem of finding a 7 week schedule for 5 groups of 3 players (7-3-5) is the same as Kirkman's Schoolgirl Problem, where the main goal is to arrange fifteen schoolgirls in rows of three so that each schoolgirl walks in the same row with every other schoolgirl exactly once a week.

The social golfer problem is also well-known for being a case study of symmetry for constraint programming (e.g. see [Smith, 2001]). 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.

The exact group of symmetries that arises from this will depend on the encoding chosen. For example, in the model considered by Harvey, Kelsey and Petrie [Harvey *et al.*, 2003], this gives the wreath product of S_8 with S_{10}. This means that the 8 groupings in each week can be permuted in any way, giving S_8, and that the 10 weeks can also be permuted in any way, giving S_{10}.

Eliminating the above symmetries is not expensive and can bring significant enhancements. For example, considering again the solution given in Figure 1, one may assume that symmetries have been eliminated: this explains why golfers are ordered within groups, groups are ordered within weeks with respect to the first player and weeks are ordered with respect to the second player of the first group.

There is also one final symmetry that is not considered above.

- Golfers are interchangeable. That is, the names of the 32 golfers are insignificant.

In the model just mentioned, the additional symmetry would give a semi-direct product of the previous group with S_{32}. This combination of symmetries makes symmetry breaking much more difficult, and to date no efficient method to break *all* symmetries has been presented. From the very beginning, the social golfer problem has been extensively studied as a paradigmatic problem with a significant number of symmetries [Smith, 2001; Puget, 2002]. In this paper, we concentrate only on the initial group of symmetries of the problem, disregarding the more complicated combination for simplicity. It would certainly be interesting to consider approaches to breaking the full group of the problem, following for example [Aloul *et al.*, 2003], but that is outside the scope of this paper.

4 A SAT Encoding for the Social Golfer Problem

To encode the social golfer problem as a SAT problem we must define:

- A set of variables.
- A set of constraints (represented by clauses) on the variables.

The set of constraints must guarantee that each golfer plays golf once a week, golfers always play in groups of a given size and no golfer plays in the same group as any other golfer twice.

4.1 The Model

We have defined SAT variables based on the golfers. Apparently, for a social golfer problem $w-p-g$ it should be enough to have $w \times (p \times g) \times g$ variables. The value of each variable would allow us to conclude whether, in a given week, a certain golfer is scheduled to play in a particular group.

However, we have chosen a more expressive model. Even though this model has more variables, these variables are quite useful for defining the problem constraints. Instead of $w \times (p \times g) \times g$ variables, this new model has $w \times (p \times g) \times (p \times g)$ variables. When compared with the other model, the difference is that we introduced an additional order relation for golfers within groups. This means that the value of each variable indicates whether golfer i is scheduled to play in group k of week l as the j^{th} player, with $1 \leq i \leq (p \times g)$, $1 \leq j \leq p$, $1 \leq k \leq g$ and $1 \leq l \leq w$. (In what follows we will refer to $x = p \times g$ as the number of golfers.) Although the order of players is irrelevant within groups (as well as the order of groups within weeks and the order of weeks), this model requires most constraints to be at-least-one and at-most-one clauses.

The next step consists in adding clauses to specify that:

- Each golfer plays exactly once per week, i.e.:
  - Each golfer plays at least once per week.
  - Each golfer plays at most once per week.
- Each group in each week has exactly p players, i.e.:
  - At least one golfer must play as the j^{th} golfer, with $1 \leq j \leq p$.
  - At most one golfer can play as the j^{th} golfer, with $1 \leq j \leq p$.

Let us now consider the social golfer problem $w-p-g$, with the number of golfers being given by $x = p \times g$. Consider $\textsc{Golfer}_{ijkl}$ to be a variable equivalent to having golfer i playing as the j^{th} player of group k during week l, with $1 \leq i \leq x$, $1 \leq j \leq p$, $1 \leq k \leq g$ and $1 \leq l \leq w$.

Each of at-least-one clauses referring to golfers has size $x = p \times g$ and is obtained as simply as follows.

$$\bigwedge_{i=1}^{x}\bigwedge_{l=1}^{w}\bigvee_{j=1}^{p}\bigvee_{k=1}^{g} \textsc{Golfer}_{ijkl}$$

The at-most-one clauses referring to golfers are encoded with two sets of binary clauses. The first set of clauses guarantees that each golfer plays at most once in the same group.

$$\bigwedge_{i=1}^{x}\bigwedge_{l=1}^{w}\bigwedge_{j=1}^{p}\bigwedge_{k=1}^{g}\bigwedge_{m=j+1}^{p} \neg\textsc{Golfer}_{ijkl} \vee \neg\textsc{Golfer}_{imkl}$$

The second set of clauses guarantees that each golfer plays at most once per week.

$$\bigwedge_{i=1}^{x}\bigwedge_{l=1}^{w}\bigwedge_{j=1}^{p}\bigwedge_{k=1}^{g}\bigwedge_{m=k+1}^{g}\bigwedge_{n=j+1}^{p} \neg\textsc{Golfer}_{ijkl} \vee \neg\textsc{Golfer}_{inml}$$

Let us now consider the clauses referring to groups of golfers. Each at-least-one clause has size x and is obtained as follows.

$$\bigwedge_{l=1}^{w}\bigwedge_{k=1}^{g}\bigwedge_{j=1}^{p}\bigvee_{i=1}^{x} \textsc{Golfer}_{ijkl}$$

Finally, the at-most-clauses for groups of golfers are encoded by a set of binary clauses.

$$\bigwedge_{l=1}^{w}\bigwedge_{k=1}^{g}\bigwedge_{j=1}^{p}\bigwedge_{i=1}^{x}\bigwedge_{m=i+1}^{x} \neg\textsc{Golfer}_{ijkl} \vee \neg\textsc{Golfer}_{imkl}$$

With the set of variables and clauses described above we have encoded all the constraints of the problem, except the one that mentions that "no golfer plays in the same group as any other golfer twice". To guarantee this condition, we introduce a set of auxiliary variables and a *ladder* matrix.

The set of auxiliary variables allows us to know exactly which golfers are scheduled to play in each match. Hence, we must have $x \times g \times w$ additional variables. Clearly, the value of these new variables depends on the value of the variables GOLFER described above. Consider these new variables to be a set of variables denoted as $\textsc{Golfer'}_{ikl}$, meaning that golfer i is scheduled to play in group k during week l, with $1 \leq i \leq x$, $1 \leq k \leq g$ and $1 \leq l \leq w$. It is easy to establish an equivalence relation between each variable $\textsc{Golfer'}_{ikl}$ and the corresponding GOLFER variables. (Each equivalence may be readily converted into a set of clauses.)

$$\textsc{Golfer'}_{ikl} \leftrightarrow \bigvee_{j=1}^{p} \textsc{Golfer}_{ijkl}$$

These new variables will now be used by the variables in the ladder matrix in such a way that no golfer plays in the same group as any other golfer more than once.

	1.1	1.2	2.1	2.2	3.1	3.2
3.4	T	**T**	**F**	F	F	F
2.3	T	T	T	T	T	T
2.4	T	T	T	T	F	F
1.2	T	F	F	F	F	F
1.3	T	T	T	F	F	F
1.4	T	T	T	T	T	F

Figure 2: The ladder matrix for the solution given in Figure 1.

The ladder matrix [Gent and Prosser, 2002; Ansótegui and Manyá, 2004; Gent and Nightingale, 2004] is characterized by a set of $\binom{x}{2} \times (g \times w)$ Boolean ladder variables and a set of ladder clauses. Intuitively, one would say that the value of each variable denotes whether two golfers are scheduled to play together in a given group of a given week. But we can do better. We can guarantee that every two golfers play together at most once.

Consider the ladder variables to be denoted as $\textsc{Ladder}_{yz}$, with $1 \leq y \leq \binom{x}{2}$ and $1 \leq z \leq g \times w$. A complete assignment of the ladder variables is said to be *valid* if and only if every row is a sequence of zero or more true assignments followed by false assignments. In other words, after a ladder variable being set to FALSE, no subsequent variables in the same row can be assigned TRUE, i.e.:

$$\forall_y \neg\exists_z \bullet \textsc{Ladder}_{yz} = \textsc{False} \wedge \textsc{Ladder}_{yz+1} = \textsc{True}$$

The behavior of the ladder matrix can be used to guarantee that no two golfers play more than once in the same group. Actually, having an adjacent pair of variables with values TRUE and FALSE identifies precisely in which group of which week two golfers played together.

Whenever a ladder variable is satisfied, there is a set of adjacent variables that must be satisfied. This can be achieved by unit propagation adding the following set of clauses.

$$\bigwedge_{y=1}^{\binom{x}{2}-1}\bigwedge_{z=1}^{g\times w} \neg Ladder_{yz+1} \vee Ladder_{yz}$$

For example, consider the solution for the social golfer problem 3-2-2 given in Figure 1. This solution corresponds to the ladder matrix given in Figure 2. Each line in the matrix corresponds to a pair of golfers. For example, the first line named 3.4 indicates when golfers 3 and 4 play together. Each column in the matrix corresponds to a group of golfers. For example, the second column named 1.2 specifies the second group of golfers playing in the first week. Each pair of adjacent entries within a line with values T/F indicate when two golfers play together. For example, the values of the two entries in bold indicate that golfers 3 and 4 play together in the second group of the first week. Observe that due to the ladder matrix constraints no golfer can play with any other golfer more than once.

Finally, the variables in the ladder matrix must be related with the auxiliary variables described above (denoted as GOLFER'). Having $\textsc{Golfer'}_{ikl}$ and $\textsc{Golfer'}_{mkl}$ satisfied means that both golfers i and m play in the same group k in the same week l. This is equivalent to having $Ladder_{[\binom{x-i}{2}+m-i](l\times k)}$ assigned value TRUE and $Ladder_{[\binom{x-i}{2}+m-i](l\times k+1)}$ assigned value FALSE. Formally:

$$\bigwedge_{l=1}^{w}\bigwedge_{k=1}^{g}\bigwedge_{i=1}^{x-1}\bigwedge_{m=i+1}^{x} \neg\text{GOLFER'}_{ikl} \vee \neg\text{GOLFER'}_{mkl} \vee Ladder_{[\binom{x-i}{2}+m-i](l\times k)}$$

$$\bigwedge_{l=1}^{w}\bigwedge_{k=1}^{g}\bigwedge_{i=1}^{x-1}\bigwedge_{m=i+1}^{x} \neg\text{GOLFER'}_{ikl} \vee \neg\text{GOLFER'}_{mkl} \vee \neg Ladder_{[\binom{x-i}{2}+m-i](l\times k+1)}$$

$$\bigwedge_{l=1}^{w}\bigwedge_{k=1}^{g}\bigwedge_{i=1}^{x-1}\bigwedge_{m=i+1}^{x} Ladder_{[\binom{x-i}{2}+m-i](l\times k+1)} \vee \neg Ladder_{[\binom{x-i}{2}+m-i](l\times k)} \vee \neg\text{GOLFER'}_{ikl}$$

$$\bigwedge_{l=1}^{w}\bigwedge_{k=1}^{g}\bigwedge_{i=1}^{x-1}\bigwedge_{m=i+1}^{x} Ladder_{[\binom{x-i}{2}+m-i](l\times k+1)} \vee \neg Ladder_{[\binom{x-i}{2}+m-i](l\times k)} \vee \neg\text{GOLFER'}_{mkl}$$

4.2 Symmetry Breaking

After establishing the model described above, we considered predicates for breaking symmetries in our SAT encoding for the social golfer problem. Clearly, this problem (and therefore our model) is highly symmetric: golfers within a group are interchangeable, groups within a week are also interchangeable and finally weeks are interchangeable. We suggest to tackle these symmetries by adding more clauses to the encoding.

The symmetries between players within the same group are eliminated by forcing players in the same group to be in lexicographic order, i.e. in increasing numerical order. In practice, this is done by adding a set of binary clauses as follows.

$$\bigwedge_{i=1}^{x}\bigwedge_{j=1}^{p}\bigwedge_{k=1}^{g}\bigwedge_{l=1}^{w}\bigwedge_{m=1}^{i-1} \neg\text{GOLFER}_{ijkl} \vee \neg\text{GOLFER}_{m(j+1)kl}$$

These clauses guarantee that if a golfer is scheduled to play as the j^{th} golfer, then the $(j+1)^{th}$ golfer has to be in a higher numerical order.

Similarly, we impose the first players of the groups within the same week to be in lexicographic order. Obviously, golfer #1 must be scheduled as the first golfer in the first group within each week. In addition, we use binary clauses to encode symmetry breaking within each week.

$$\bigwedge_{i=1}^{x}\bigwedge_{k=1}^{g}\bigwedge_{l=1}^{w}\bigwedge_{m=1}^{i-1} \neg\text{GOLFER}_{i1kl} \vee \neg\text{GOLFER}_{m1(k+1)l}$$

These binary clauses impose first golfers of subsequent groups to be in lexicographic order.

Finally, additional clauses are used to break symmetries between weeks. This is simply achieved by imposing lexicographic order between the second golfer of the first group of each week. This is encoded as follows.

$$\bigwedge_{i=1}^{x}\bigwedge_{k=1}^{g}\bigwedge_{l=1}^{w}\bigwedge_{m=1}^{i-1} \neg\text{GOLFER}_{i2kl} \vee \neg\text{GOLFER}_{m2k(l+1)}$$

These three sets of binary clauses suffice to break the symmetries that were initially mentioned. Observe that the solution given in Figure 1 satisfies all the constraints we have specified for symmetry breaking. As we mentioned earlier, we leave for future work the interesting question of how best to tackle the combination of these symmetries with the free interchanging of players.

5 Experimental Results

In this section we evaluate empirically our encodings for the social golfer problem[1]. We compare our encoding with and without symmetries. We use two state-of-the-art SAT solvers: walksat and siege.

Experimental results are given for a set of 29 benchmark problems. All these problems are satisfiable. Otherwise, they would not be solved by local search. Moreover, many of the unsatisfiable problem instances of the social golfer problem are trivially found to be unsatisfiable. On the other hand, it is widely accepted that symmetry breaking helps proving unsatisfiability (e.g. see [Ramani and Markov, 2005]), but not much has been said about finding exactly one solution.

Table 1 characterizes each problem instance (named as $w-p-g$) by giving the number of variables and clauses. The larger instances have thousands of variables and around a million of clauses. We have observed that most of the clauses are either binary or ternary, which makes the average clause size (AvgCS) to be between 2 and 3. We have also observed that the additional clauses for breaking symmetries (SBCls), which are all binary clauses, may augment the number of clauses in the initial model for about 30% for the larger instances (for smaller instances this value is smaller).

5.1 Local Search: Walksat

Walksat [Kautz *et al.*, 2004] is a local search solver. The algorithm is quite simple:

- Start with a random truth assignment.
- With probability p:
  - Pick a variable occurring in some unsatisfied clause and flip its truth assignment.
- With probability $1-p$:
  - Make the best possible local move.
- Repeat the last two steps until the assignment satisfies all clauses.

We have tried to run walksat on our benchmark problems of the social golfer problem. Even though we tried many different configurations, walksat was far from being competitive on solving these problems, specially those including clauses for symmetry breaking. (We also tried other local search

[1]For all experimental results a P-IV@1.7 GHz Linux machine with 1 GByte of physical memory was used.

Problem	# Vars	# Cls	AvgCS	% SBCls
3-2-2	108	446	2.43	9%
5-3-2	495	2547	2.46	10%
4-3-3	864	5598	2.41	17%
7-4-2	1456	8556	2.46	12%
9-5-2	3375	21665	2.45	13%
5-4-4	4000	35032	2.35	24%
11-6-2	6732	46026	2.45	14%
7-6-3	9450	79965	2.38	21%
13-7-2	12103	86751	2.44	15%
6-5-5	13500	147950	2.30	28%
7-7-3	14406	127302	2.38	21%
5-8-3	14880	135780	2.37	22%
3-6-6	15876	207054	2.26	31%
15-8-2	20160	149912	2.44	15%
6-7-4	21756	227402	2.33	26%
3-8-5	24480	303260	2.28	29%
5-7-5	28175	339185	2.29	29%
17-9-2	31671	242541	2.44	16%
4-7-6	32340	440013	2.26	31%
3-9-5	34020	432360	2.28	29%
10-9-3	41310	388341	2.37	22%
6-9-4	43740	484614	2.32	26%
8-10-3	44400	426435	2.37	22%
19-10-2	47500	372630	2.43	16%
3-9-6	48843	701811	2.25	32%
5-10-4	49000	554140	2.32	27%
4-8-7	63616	995876	2.23	34%
5-10-5	76250	991925	2.28	30%
4-9-7	88452	1419075	2.23	34%

Table 1: Social golfer problems: number of variables and clauses.

solvers without success.) This is as suggested by Prestwich, that symmetry breaking constraints reduce the number of solutions and therefore make it harder for local search to find solutions [Prestwich, 2001].

Nonetheless, we have run a problem for a significant number of seeds. Figure 3 compares the average number of flips per second and the total CPU time for including or not including symmetry breaking clauses on the encodings (SymBreak and NoSymBreak, respectively). Results were obtaining running walksat with 1500 seeds for problem 7-4-2. From these results, which we believe to be representative of our SAT benchmark problems of the social golfer problem, we may conclude that:

- Walksat performs more flips per second (in average) without clauses for symmetry breaking. This may be explained by the overhead produced by the additional symmetry breaking clauses.
- Walksat requires more CPU time to solve instances with symmetry breaking clauses.
- Adding clauses to break symmetries affects negatively both the number of flips and the CPU time, although the consequences are more negative for the CPU time. Indeed, for the encoding with symmetry breaking clauses we may observe an extremely fluctuation on the expected time to find a solution, which is probably associated with a heavy-tail distribution [Gomes *et al.*, 2000].

5.2 Backtrack Search: Siege

Siege [Ryan, 2004] is a randomized backtrack search SAT solver enhanced with clause recording. The data structures are carefully implemented and the decision heuristic is very efficient, specially for structured problems.

Siege has been shown to be quite competitive on solving our benchmark problems. We have run siege on each problem for 1500 seeds. Figure 4 compares the number of nodes (median and mean, using a logarithmic scale) for including or not including symmetry breaking clauses. Figure 5 makes the same comparison for the CPU time. Apparently, including symmetry breaking clauses often does not compensate. Furthermore, results for including symmetry breaking clauses are more negative for the number of nodes rather than for the CPU time. The same holds for the median values when compared with the mean values.

With the aim of clarifying the differences between median and mean values, we have run one of the problems where those differences could be observed (problem 6-7-4) for 10000 seeds. Figure 6 gives the number of nodes and the CPU time. From these plots we may conclude that adding symmetry breaking clauses seems to avoid a heavy-tail behavior exhibited by the encoding with no symmetry breaking. Hence, we claim that adding symmetry breaking clauses may avoid the heavy-tail behavior, in particular for the most difficult instances.

6 Conclusions and Future Work

Recent advances in SAT solving motivate an increasing number of combinatorial problems to be encoded into SAT. We argue that modeling decisions have an impact on the solver's performance. We have encoded the social golfer problem into SAT. Two different encodings - with and without symmetry breaking - have been empirically evaluated with local search and backtrack search solvers. A somewhat surprising observation is that some of the encodings, depending on the solvers, may exhibit a heavy-tail distribution. In such circumstances, choosing the *right* encoding can make the difference between heavy-tail behavior or not. In a near future, we plan to do a more comprehensive evaluation, which includes evaluating more instances, trying different encodings and also encoding new problems.

References

[Aguado, 2004] Alejandro Aguado. A 10 days solution to the social golfer problem, 2004. Manuscript.

[Aloul *et al.*, 2003] F. Aloul, K. A. Sakallah, and I. Markov. Efficient symmetry breaking for boolean satisfiability. In *Proc. IJCAI*, pages 271–276, August 2003.

[Ansótegui and Manyá, 2004] Carlos Ansótegui and Felip Manyá. Mapping problems with finite-domain variables into problems with boolean variables. In *Proceedings of*

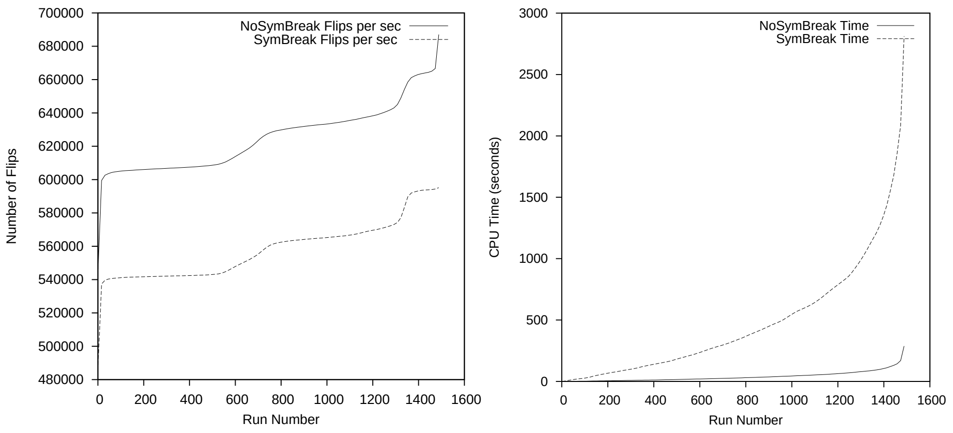

Figure 3: Walksat: average number of flips per second and total CPU time for problem 7-4-2.

the International Conference on Theory and Applications of Satisfiability Testing, May 2004.

[Brown *et al.*, 1988] C. A. Brown, L. Finkelstein, and P. W. Purdom Jr. Backtrack searching in the presence of symmetry. In *6th 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.

[Crawford *et al.*, 1996] 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.

[Een and Sorensson, 2003] N. Een and N. Sorensson. An extensible SAT solver. In *Sixth International Conference on Theory and Applications of Satisfiability Testing*, May 2003.

[Fahle *et al.*, 2001] T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In *Proc. CP 2001*, pages 93–107, 2001.

[Gent and Nightingale, 2004] Ian P. Gent and Peter Nightingale. A new encoding of alldifferent into sat. In AM Frisch and I Miguel, editors, *Proc. 3rd International Workshop on Modelling and Reformulating Constraint Satisfaction Problems, CP2004*, pages 95–110, 2004.

[Gent and Prosser, 2002] Ian P. Gent and Patrick Prosser. Sat encodings of the stable marriage problem with ties and incomplete lists. In *Fifth International Symposium on Theory and Applications of Satisfiability Testing*, May 2002.

[Gent and Smith, 2000] Ian P. Gent and Barbara M. Smith. Symmetry breaking during search in constraint programming. In *Proc. ECAI*, pages 599–603, 2000.

[Goldberg and Novikov, 2002] E. Goldberg and Y. Novikov. BerkMin: a fast and robust SAT-solver. In *Proceedings of the Design and Test in Europe Conference*, pages 142–149, March 2002.

[Gomes *et al.*, 2000] Carla P. Gomes, Bart Selman, Nuno Crato, and Henry A. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24(1/2):67–100, 2000.

[Harvey *et al.*, 2003] Warwick Harvey, Tom Kelsey, and Karen Petrie. Symmetry group generation for CSPs. Technical Report APES-60-2003, APES Research Group, July 2003. Available from http://www.dcs.st-and.ac.uk/˜apes/apesreports.html.

[Kautz *et al.*, 2004] Henry Kautz, Bart Selman, and David McAllester. Walksat in the 2004 sat competition. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, 2004.

[Meseguer and Torras, 2001] P. Meseguer and C. Torras. Exploiting symmetries within constraint satisfaction search. *Artificial Intelligence*, 129(1-2):133–163, 2001.

[Moskewicz *et al.*, 2001] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Engineering an efficient SAT solver. In *Design Automation Conference*, pages 530–535, June 2001.

[Prestwich, 2001] Steven Prestwich. First-solution search with symmetry breaking and implied constraints. In *Workshop on Symmetry in Constraint Satisfaction Problems*, 2001.

[Puget, 2002] Jean-François Puget. Symmetry breaking revisited. In Pascal Van Hentenryck, editor, *Proc. CP 2002*, pages 446–461. Springer-Verlag, 2002.

[Ramani and Markov, 2005] 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. Springer-Verlag, 2005.

[Ryan, 2004] L. Ryan. Efficient algorithms for clause-learning SAT solvers. Master's thesis, Simon Fraser University, February 2004.

[Smith, 2001] Barbara M. Smith. Reducing symmetry in a combinatorial design problem. In *Proceedings of the Third International Workshop on Integration of AI and OR Techniques*, pages 351–359, 2001.

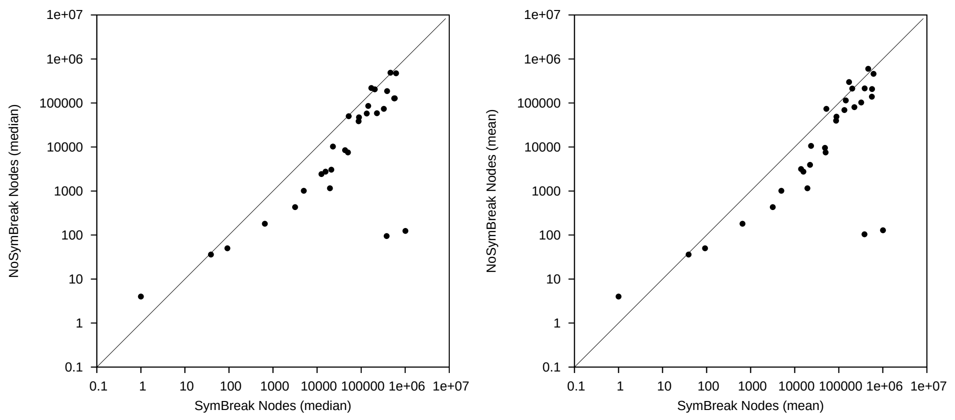

Figure 4: Siege: comparison of the number of nodes (median and mean) for a set of problems.

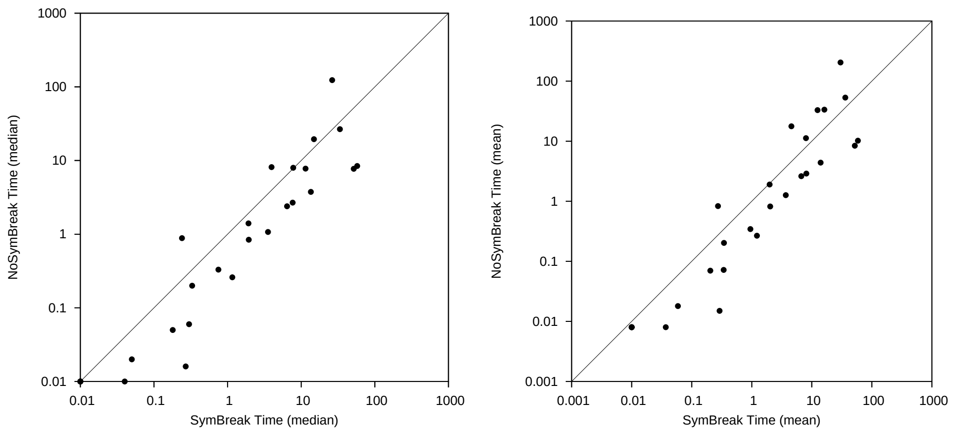

Figure 5: Siege: comparison of the CPU time (median and mean) for a set of problems.

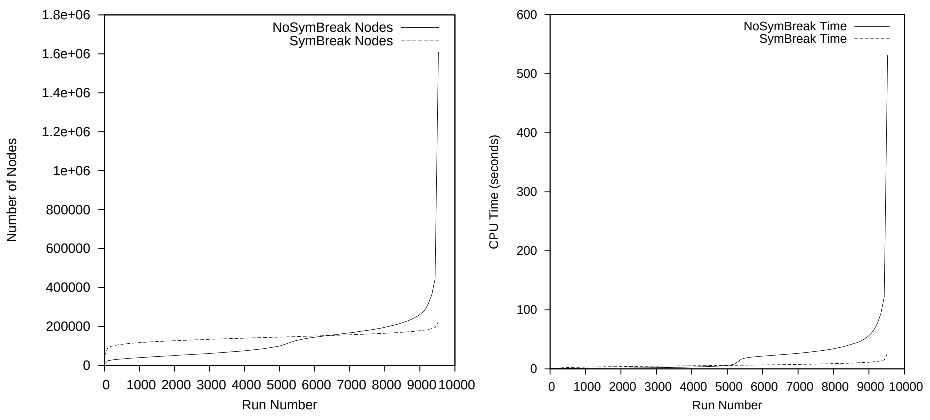

Figure 6: Siege: number of nodes and CPU time for problem 6-7-4.