Encoding The Lexicographic Ordering Constraint in SAT Modulo Theories

Hani A. Elgabou and Alan M. Frisch

Department of Computer Science,
University of York, Heslington, York, YO10 5GH, UK
{he583,alan.frisch}@york.ac.uk
http://www.cs.york.ac.uk/

Abstract. This paper presents eight different SMT encodings of the lexicographic ordering constraint. These constraints are helpful in breaking some kinds of symmetries in combinatorial decision and optimisation problems. The encodings are obtained from the literature then translated into an SMT suitable form. We have done this using two methods, the first is by directly translating the encodings into SMT. The second starts by rewriting the encodings in MiniZinc language, flattening them into FlatZinc instances then using a tool called fzn2smt to translate them to SMT. We evaluate the encodings on a suite of instances of the Social Golfer problem, which is well known for its highly symmetric models. This shows that different encodings of the lexicographic ordering constraint perform best on different problem instances and that no one encoding is dominant.

Keywords: Global Constraints, Lexicographic Ordering, Symmetry, SAT Modulo Theories, MiniZinc, SMTLIB

1 Introduction

Modern Boolean satisfiability (SAT) solvers are essential tools in many applications including scheduling, verification, circuit design and others. The nature of some of today's combinatorial problems makes them difficult to be represented using SAT's Boolean formulas. Satisfiability Modulo Theories (SMT) is introduced to overcome this limitation, where problems can be encoded using logical formulas of combinations of atomic propositions and atomic expressions in one or more Theories (T). The theory part in SMT formulas enables them to naturally describe problems related to any of the SMT supported theories, like arithmetic, arrays and bit-vectors. SMTLIB is the standard modelling language in SMT.

Symmetries arise in many constraint satisfaction problem models. A common form of symmetry is the interchangeability between elements of sets of variables and the corresponding sets of values. An example is the ability to swap any two rows or columns in a Latin Square while preserving satisfiability. Breaking symmetry reduces the search space, which could improve performance.

The Lex ordering constraint enforces lexicographic ordering between two vectors of variables, which makes it useful in breaking symmetry between rows and columns of a matrix of decision variables [1]. For example, enforcing lex on between pairs of rows of a Latin Square eliminates the interchangeability between them and the same applies for the columns.

A recent study demonstrated that SMT solvers have a competitive performance in solving Constraint Satisfaction Problems. In that study, Bofill *et al.* [2] built a tool called fzn2smt to translate CSP instances expressed in flatzinc into the SMT language SMTLIB1.2 standard [3], then solved them using the SMT solver Yices1 [4], which in general performed better than some well known constraint solvers.

Our main aim for this research is to explore ways of further improving this performance by studying some of the instances where fzn2smt-Yices exhibited weaker performance. We found out that some of these instances involve Global Constraints, which are constraints that could have a non-fixed number of variables and encapsulates a set of other constraints [5]. So we decided to study how fzn2smt translate some of these constraints to try to come up with a better translation. Our first study is on encodings of the lexicographic ordering global constraint.

2 Methodology

We present eight alternative decompositions of the lexicographic ordering constraint, each of which is drawn from the constraint solving literature or is a variant of such. We evaluate seven of the decompositions in solving a suite of instances of the Social Golfers Problem (SGP) (Problem 010 in CSPLib [6]) with the Yices2 [4] SMT solver. The eighth decomposition is not evaluated because, as we will see, it is impractical to do so.

Using MiniZinc [7] and fzn2smt we have produced SMT encodings of the SGP instances as follows. MiniZinc provides a default decomposition that can be used for each of its global constraints, including the lexicographic ordering constraint. Indeed, the MiniZinc decomposition of lexicographic ordering is one of the seven we have evaluated. The other six encodings are produced by replacing the MiniZinc default decomposition with one of the alternatives.

We use a MiniZinc model of the SGP in which row and column symmetry is broken by constraining the rows to be in increasing lexicographic order and the columns to be in increasing lexicographic order. With one of the seven decompositions in place, an SGP instance is translated to FlatZinc by the MiniZinc system. The resulting FlatZinc is translated to SMT by fzn2smt. The SMT code generated by this process follows the SMTLIB 1.2 standard and uses linear integer arithmetic logic as provided by the QF_LIA SMT theory.

We shall use the term mzn2smt to refer to the pipeline of translating a MiniZinc specification to SMT in two steps, first using MiniZinc to produce FlatZinc and then passing this through fzn2smt to produce SMT. We chose this pipeline at this stage of our research just for convenience and we are aware of some of its possible drawbacks. For instance, some constraints could be naturally represented in SMT, but when they go through the translation process they get broken into smaller ones. That is way we performed two sets of benchmarks, one is for the mzn2smt translation and the other is of the direct translation from each encoding to SMT.

3 Encodings for Lex Ordering Constraints

This section presents eight different encodings for the lex Ordering Constraint. Throughout, we consider a non-strict lex constraint between two vectors A and B of finite-domain variables. Both vectors are considered to be of length n. We write such a constraint as $A \leq_{lex} B$. Each of the following subsections presents a decomposition of the lex constraint followed by the result of passing it through the mzn2smt pipeline. $T_1[i], T_2[i], \ldots$ are auxiliary Boolean arrays introduced by mzn2smt. The index i of these arrays ranges between 1 and n. The generated SMTLIB code does not literally contain arrays; we use the notation as a clean way of naming a set of n distinct SMT variables.

3.1 The AND Decomposition Encoding [8]

This encoding decomposes lex constraint into a conjunction of smaller constraints as shown in the following formula, and because of that it is known as AND Decomposition.

$$\begin{aligned}
&A[1] \leq B[1] \\
&\bigwedge_{i=1}^{n-1} \Big(\bigwedge_{j=1}^{i} (A[j] = B[j]) \Big) \rightarrow (A[i+1] \leq B[i+1])
\end{aligned}$$

$A[1] \leq B[1]$ is there because the first values of the two vectors are the most significant values to compare. The rest of the formula is self explanatory.
A strict ordering can be obtained by changing the formula $(A[i+1] \leq B[i+1])$ to $(A[i+1] < B[i+1])$.

After translation using mzn2smt:

$$\begin{aligned}
& & A[1] \leq B[1] & \quad (1) \\
1 \leq i \leq n-1 & & T_1[i] \Leftrightarrow (A[i] = B[i]) & \quad (2) \\
1 \leq i \leq n-1 & & T_2[i] \Leftrightarrow (A[i+1] \leq B[i+1]) & \quad (3) \\
1 \leq i \leq n-2 & & T_3[i] \Leftrightarrow \bigwedge_{j=1}^{i+1} T_1[j] & \quad (4) \\
& & \neg T_1[1] \vee T_2[1] & \quad (5) \\
1 \leq i \leq n-2 & & \neg T_3[i] \vee T_2[i+1] & \quad (6)
\end{aligned}$$

Number of constraints generated by this encoding is $4n - 4$.

3.2 The AND Decomposition Encoding using Common Sub-expressions Elimination

This encoding, which we call AND CSE, is similar to the previous one and produces a similar formula too. The difference is, in this encoding we use a Boolean array to eliminate common sub-expressions in the formula as presented in line (9). The purpose of this encoding is to compare performance between using the nested loops as in line (4) in the previous encoding and this approach.

The resulting formula after eliminating common sub-expressions using the Boolean array $X[i]$:

$$\begin{array}{ll} & A[1] \leq B[1] \quad (7)\\ & X[1] \Leftrightarrow (A[1] = B[1]) \quad (8)\\ 1 \leq i \leq n-2 & X[i+1] \Leftrightarrow (X[i] \wedge (A[i+1] = B[i+1])) \quad (9)\\ 1 \leq i \leq n-1 & X[n] \rightarrow (A[n+1] \leq B[n+1]) \quad (10)\end{array}$$

A strict ordering can be obtained by changing the formula $(A[i+1] \leq B[i+1])$ in line (3) to $(A[i+1] < B[i+1])$.

After translation using mzn2smt:

$$\begin{array}{ll} & A[1] \leq B[1] \quad (11)\\ & X[1] \Leftrightarrow (A[1] = B[1]) \quad (12)\\ 1 \leq i \leq n-2 & T_1[i] \Leftrightarrow (A[i+1] = B[i+1]) \quad (13)\\ 1 \leq i \leq n-1 & T_2[i] \Leftrightarrow (A[i+1] \leq B[i+1]) \quad (14)\\ 1 \leq i \leq n-2 & X[i+1] \Leftrightarrow (X[i] \wedge T_1[i]) \quad (15)\\ 1 \leq i \leq n-1 & \neg X[i] \vee T_2[i] \quad (16)\end{array}$$

Number of constraints generated by this encoding is $4n-4$.

3.3 The OR Decomposition Encoding [9]

This encoding, known as OR Decomposition, decomposes lex constraint into a formula of smaller constraints dis-joined together, as shown below.

$$\begin{array}{l}(A[1] < B[1]) \vee \\ \bigvee_{i=1}^{n-1} (\bigwedge_{j=1}^{i} (A[j] = B[j])) \wedge (A[i+1] < B[i+1]) \vee \\ (\bigwedge_{i=1}^{n} (A[i] = B[i]))\end{array}$$

A strict ordering can be obtained by removing $\bigwedge_{i=1}^{n}(A[i] = B[i])$ from the above formula.

After translation using mzn2smt:

$$
\begin{aligned}
&1 \le i \le n && T_1[i] \Leftrightarrow (A[i] = B[i]) && (17)\\
&1 \le i \le n && T_2[i] \Leftrightarrow (A[i] < B[i]) && (18)\\
&1 \le i \le n-1 && T_3[i] \Leftrightarrow \bigwedge_{j=1}^{i} T_1[j] \wedge T_2[i+1] && (19)\\
& && T_3[n] \Leftrightarrow \bigwedge_{i=1}^{n} T_1[i] && (20)\\
& && (\bigvee_{i=1}^{n-1} T_3[i]) \vee T_2[1] \vee T_3[n] && (21)
\end{aligned}
$$

Number of constraints generated by this encoding is $3n + 1$

3.4 The OR Decomposition Encoding using Common Sub-expressions Elimination

This version of OR decomposition, called OR CSE, uses a Boolean array to eliminate common sub-expressions from the formula. $X[i]$ is a Boolean array with an index range of $1ton$, this array is used to eliminating common sub-expressions as shown in th following formula.

$$
\begin{aligned}
& && ((A[1] < B[1]) \vee\\
& && (\bigvee_{i=1}^{n-1} X[i] \wedge (A[i+1] < B[i+1]))\ \vee X[n]) \wedge\\
& && X[1] \Leftrightarrow (A[1] = B[1])\\
&1 \le i \le n-1 && X[i+1] \Leftrightarrow (X[i] \wedge (A[i+1] = B[i+1]))
\end{aligned}
$$

A strict ordering can be obtained by removing $X[n]$ from the above formula.

After translation using mzn2smt:

$$
\begin{aligned}
&1 \le i \le n && X[1] \Leftrightarrow (A[1] = B[1]) && (22)\\
&1 \le i \le n-1 && T_2[i] \Leftrightarrow (A[i+1] = B[i+1]) && (23)\\
&1 \le i \le n && T_3[i] \Leftrightarrow (A[i] < B[i]) && (24)\\
&1 \le i \le n-1 && X[i+1] \Leftrightarrow (X[i] \wedge T_2[i]) && (25)\\
&1 \le i \le n-1 && T_4[i] \Leftrightarrow (X[i] \wedge T_3[i+1]) && (26)\\
& && (\bigvee_{i=1}^{n-1} T_4[i]) \vee T_3[1] \vee X[n] && (27)
\end{aligned}
$$

Number of constraints generated by this encoding is $5n - 2$

3.5 Arithmetic Lex Encoding [8]

Another way of encoding lex constraint is using arithmetic constraint. This constraint compares the sum of the values of two vectors with each value multiplied by a factor that represents the significance of the values. We assume all the variables in A and B have a domain of $1tod$.

$$\sum_{i=1}^{n} A[i] \times d^{n-i} \;\leq\; \sum_{i=1}^{n} B[i] \times d^{n-i}$$

mzn2smt translation produces the same formula above.

This encoding is limited by the size of data type used to represent domains of values, for example, if $A[1] \times d^{n-1}$ exceeds the maximum value that can be stored in a 32-bit integer this would cause an arithmetic overflow and a system error in computers.

A strict ordering can be achieved by changing $\leq$ to $<$.Number of constraints generated by this encoding is 1.

3.6 Harvey Lex Encoding [8]

This encoding is presented by [8] who attribute it to Warwick Harvey. The general formula as presented by the source is

$$(A[1] < (B[1] + (A[2] < (B[2] + (... + (A[n] < (B[n]) + 1)...)))) = 1$$

To remove the ellipsis and encode the decomposition in Minizinc, we introduce $X[i]$, a Boolean array used to eliminate common sub-expressions, where i is an index with possible values from 1 to $n - 1$.

$$\begin{array}{ll} & X[1] \\ & X[n] = (A[n] < (B[n] + 1)) \\ 0 \leq i \leq n-2 & X[n-i-1] = A[n-i-1] < (B[n-i-1] + Bool2Int(X[n-i])) \end{array}$$

We get a strict version by changing $B[n]+1$ to $B[n]+0$ in the above formula.

Translation from MiniZinc to SMT using mzn2smt produces the following. $int[i]$ is an integer array introduced by fzn2smt to encode the Bool2Int function of MiniZinc. $int[i]$ has a domain of 0 to 1 and a size of 1 to n

$$\begin{array}{lll} & X[1] & (28) \\ & X[n] \Leftrightarrow ((A[n] - B[n]) \leq 0) & (29) \\ 1 \leq i \leq n & int[i] \leq 1 & (30) \\ 1 \leq i \leq n & int[i] \geq 0 & (31) \\ 1 \leq i \leq n-1 & X[i+1] \rightarrow (int[i] = 1) & (32) \\ 1 \leq i \leq n-1 & \neg X[i+1] \rightarrow (int[i] = 0) & (33) \\ 1 \leq i \leq n-1 & X[n-i] \Leftrightarrow ((A[n-i] - B[n-i] - int[i]) \leq -1) & (34) \end{array}$$

The translated Harvey encoding generates $5n - 1$ constraints.

3.7 Alpha Lex Encoding [9]

We call this encoding Alpha, because it uses a Boolean array as an index to track the relations between values. This Boolean array is called $\alpha[i]$ and behaves as follows:

if $\alpha[i] = 1$ then $A[j] = B[j]$ for all $1 <= j <= i <= n$

and

if $\alpha[i] = 1$ and $\alpha[i+1] = 0$ then $A[i+1] < B[i+1]$

This makes all values from $\alpha[1]$ to $\alpha[i]$ equal to 1 while $A[i] = B[i]$ holds, and equal to 0 from the first occurrence of $A[i] < B[i]$ till the end of vectors. This encoding could be changed to a strict lex by adding the constraint $\alpha[n+1] = 0$. α is a Boolean matrix so $\alpha[i] = 1$ is equivalent to $\alpha[i] = true$ and $\alpha[i] = 0$ is equivalent to $\alpha[i] = false$.

$$\alpha[0] = 1 \tag{35}$$
$$0 \leq i \leq n-1 \qquad \alpha[i] = 0 \rightarrow (\alpha[i+1] = 0) \tag{36}$$
$$1 \leq i \leq n \qquad \alpha[i] = 1 \rightarrow (A[i] = B[i]) \tag{37}$$
$$0 \leq i \leq n-1 \qquad ((\alpha[i] = 1) \wedge (\alpha[i+1] = 0)) \rightarrow (A[i+1] < B[i+1]) \tag{38}$$
$$0 \leq i \leq n-1 \qquad \alpha[i] = 1 \rightarrow (A[i+1] \leq B[i+1]) \tag{39}$$

After translation using mzn2smt:

Here we use $\alpha^{'}$ instead of α because the mzn2smt translator changed the range of α from $0 \leq i \leq n$ to $1 \leq i \leq n+1$

$$\alpha^{'}[1] \tag{40}$$
$$1 \leq i \leq n+1 \qquad T_1[i] \Leftrightarrow \neg\alpha^{'}[i+1] \tag{41}$$
$$1 \leq i \leq n \qquad T_2[i] \Leftrightarrow (A[i] = B[i]) \tag{42}$$
$$1 \leq i \leq n \qquad T_3[i] \Leftrightarrow (A[i] < B[i]) \tag{43}$$
$$1 \leq i \leq n \qquad T_4[i] \Leftrightarrow (A[i] \leq B[i]) \tag{44}$$
$$1 \leq i \leq n \qquad T_5[i] \Leftrightarrow (\alpha[i] \wedge T_1[i+1]) \tag{45}$$
$$1 \leq i \leq n \qquad \neg T_1[i] \vee T_1[i+1] \tag{46}$$
$$1 \leq i \leq n \qquad \neg T_5[i] \vee T_3[i] \tag{47}$$
$$1 \leq i \leq n \qquad \neg\alpha^{'}[i] \vee T_4[i] \tag{48}$$
$$1 \leq i \leq n \qquad \neg\alpha^{'}[i+1] \vee T_2[i] \tag{49}$$

Number of constraints generated by this encoding is $9n + 2$

3.8 Alpha M Encoding [7]

This decomposition, which we call Alpha M, is the default decomposition used by Minizinc [7]. Like the previous Alpha encoding it uses a binary array as a bookkeeping mechanism for relations between the corresponding values in both vectors. The index of the Alpha array ranges from 1 to $n + 1$.

$$\alpha[1] = 1 \tag{50}$$
$$1 \leq i \leq n \qquad \alpha[i] = ((A[i] < B[i]) \vee \alpha[i+1]) \wedge (A[i] \leq B[i]) \tag{51}$$

$\alpha[i] = 1$ makes sure that $A[i] \leq B[i]$ is true. We obtain a strict version by adding $\alpha[n+1] = 0$ to the constraints.

After translation using mzn2smt:

$$\alpha[1] \tag{52}$$
$$1 \leq i \leq n \qquad T_1[i] \Leftrightarrow (A[i] \leq B[i]) \tag{53}$$
$$1 \leq i \leq n \qquad T_2[i] \Leftrightarrow (A[i] < B[i]) \tag{54}$$
$$1 \leq i \leq n \qquad T_3[i] \Leftrightarrow (T_2[i] \vee \alpha[i+1]) \tag{55}$$
$$1 \leq i \leq n \qquad \alpha[i] \Leftrightarrow (T_3[i] \wedge T_1[i]) \tag{56}$$

Number of constraints generated by this encoding is $4n + 1$

4 The Social Golfers Problem

The Social Golfers Problem is a computational problem of partitioning a set of golfers into g groups of size s in each of w weeks such that no two players meet more that once in the same group. An instance of the Social Golfer problem is usually denoted $g - s - w$, which stand for number of groups, the group size and number of weeks. We use $m = g * s$ to denote the number of players

The table below shows one possible solution to the instance 3-2-3, where rows and columns represent players and weeks respectively, and each value in the table denotes a group number. So as an example column 2 can be interpreted as follows; In $Week_2$, $Player_1$ and $Player_3$ meet in the first group, $Player_2$ and $Player_5$ meet in the second group and $Player_4$ and $Player_6$ meet in the third.

$Week_1$	$Week_2$	$Week_3$	
1	1	1	$Player_1$
1	2	2	$Player_2$
2	1	2	$Player_3$
2	3	3	$Player_4$
3	2	3	$Player_5$
3	3	1	$Player_6$

The Social Golfer is known for its highly symmetric models. We use the lex constraint to break two groups of symmetries in the problem; symmetries in weeks and symmetries in players. The MiniZinc model that we used for the problem maps Players and Weeks to groups in an array as above. Symmetry among the players is broken by constraining the rows to be in lex increasing order and symmetry among the weeks is broken by constraining the columns to be in lex increasing order.

The model that we used for the SGP is a modified model created by H. Kjellerstrand [10] and it has two constraints: The first is to make all groups contain s players, while the second is to make sure that each two players play together at most once in each week.

$Schedule[,]$ is a two dimensional integer array that holds the weekly assignment of players to groups. Each group has exactly s players:

$$1 \leq group \leq g \quad 1 \leq week \leq w \quad (\sum_{player=1}^{m} Bool2Int(Schedule[player, week] = group)) = s$$

Where Bool2Int() is Boolean to integer converter function and m = number of players, which equals to $g \times s$.

Each pair of players only meets at most once

$$\begin{aligned}
&1 \leq pa \leq m \quad 1 \leq pb \leq m \\
&1 \leq wa \leq w \quad 1 \leq wb \leq w \\
&where\ pa \neq pb \wedge wa \neq wb \\
&(Schedule[pa, wa] \neq Schedule[pb, wa]) \vee \\
&(Schedule[pa, wb] \neq Schedule[pb, wb])
\end{aligned}$$

For any two distinct players, pa and pb, and any two distinct weeks, wa and wb, players pa and pb cannot play in the same group in both week wa and wb.

From the assignment array $Schedule[,]$ it is clear that symmetries can happen between weeks and between players. To break symmetry between weeks we put lex constraint ordering between each two neighbouring columns and the same is done for players.

Lex constraint on weeks:

$$\begin{aligned}
1 \leq week \leq w-1 \quad & [Schedule[player, week] \mid player \in 1..m] \leq_{lex} \\
& [Schedule[player, week+1] \mid 1player \in 1..m]
\end{aligned}$$

Lex constraint on players:

$$\begin{aligned}
1 \leq player \leq m-1 \quad & [Schedule[player, week] \mid week \in 1..w] \leq_{lex} \\
& [Schedule[player+1, week] \mid week \in 1..w]
\end{aligned}$$

5 Benchmarks

The MiniZinc implementation includes a set of libraries to decompose global constraints and made to be called from MiniZinc models. We created a MiniZinc global library for each of the seven lex decompositions, then we called them from the Social Golfer MiniZinc code. We modified a MiniZinc model created by

H. Kjellerstrand [10] for the problem by removing his implementation of symmetry breaking code and adding a symmetry breaking based on lexicographical orderings constraint. All benchmarks were run on a Windows PC with Intel i7 1.8Ghz processor and 8GB of RAM and using Yices 2.2.1 as an SMT solver. We ran 30 samples for each instance, each sample is created by choosing a random ordering of the constraints from a uniform distribution over all orderings of the sample SMT file. Each figure in the following two tables represents an average of 30 timings for each encoding on each instance. We only used satisfiable instances obtained from [11]. We ran two sets of benchmarks, the first is for samples translated to SMT using the mzn2smt pipeline, while the second is for the same samples but translated directly from the each of the 7 presented encodings in this paper. Both sets of benchmarks share the same code for the SGP, the difference is only in the code related to different the lex constraint encodings.

Table 5 shows the timings in seconds for each of the encodings on instances of the SGP produced by mzn2smt. As it can be seen from these results the OR encoding takes the lead by a very narrow margin, it also seems that using CSE did not made any improvement to the timings of AND and OR encodings.

Instances G-S-W	AND	AND CSE	OR	OR CSE	Alpha	Alpha M	Harvey
5-3-5	0.27	**0.25**	0.28	0.26	**0.25**	0.27	0.33
5-3-6	1.40	1.79	1.70	2.02	1.60	**1.59**	1.94
5-3-7	12.53	13.08	**11.07**	13.94	13.47	14.77	16.71
6-3-5	0.27	0.27	**0.25**	0.26	0.26	0.29	0.32
6-3-6	1.63	1.64	1.70	1.79	1.69	**1.55**	1.87
6-3-7	6.77	6.72	7.14	7.32	**6.10**	6.84	6.81
6-4-4	0.63	0.63	**0.62**	0.63	0.66	0.64	0.68
6-4-5	5.32	4.82	5.01	**4.53**	4.97	4.88	4.89
8-4-4	**1.08**	0.98	1.11	1.17	1.09	1.14	1.20
8-4-5	10.88	10.80	10.42	9.73	10.76	**9.63**	10.80
8-4-6	85.45	87.43	**79.19**	90.63	84.18	93.65	83.29
Arithmetic mean	11.48	11.67	**10.77**	12.03	11.37	12.29	11.71
Geometric mean	2.66	2.67	**2.65**	2.77	2.66	2.73	2.97

Table 1. Solution timings (in Seconds) for instances of The SGP using fzn2smt translation

Table 5 is for the directly translated samples. Here Alpha encodings takes a marginal lead while Harvey encoding came last by a relatively wide gap. Compared to the mzn2smt translation results, the only improvement is in AND encoding results.

Instances G-S-W	AND	AND CSE	OR	OR CSE	Alpha	Alpha M	Harvey
5-3-5	0.	27 0.28	0.27	0.29	0.27	**0.25**	0.42
5-3-6	1.93	1.88	2.07	**1.53**	1.91	1.78	3.13
5-3-7	12.36	13.59	13.65	16.18	**10.73**	17.84	32.51
6-3-5	**0.26**	0.27	0.29	0.34	0.27	**0.26**	0.48
6-3-6	**1.54**	1.76	1.94	1.88	1.69	1.83	3.18
6-3-7	7.27	7.26	7.24	6.98	6.94	**6.93**	10.56
6-4-4	0.72	0.69	**0.65**	0.66	**0.65**	0.68	1.11
6-4-5	4.81	5.06	5.04	4.72	4.74	**4.64**	6.34
8-4-4	1.12	1.03	1.21	1.25	1.16	**1.00**	1.71
8-4-5	10.41	10.57	**9.89**	10.78	10.08	10.56	21.26
8-4-6	82.84	82.16	81.60	**78.15**	90.06	82.68	127.09
Arithmetic mean	11.23	11.32	11.26	**11.16**	11.68	11.68	18.89
Geometric mean	2.73	2.78	2.84	2.86	**2.71**	2.76	4.66

Table 2. Solution timings (in Seconds) for instances of The SGP using a direct translation

6 Conclusion and Future Work

Although the averages of the results provide no clear judgement about which encoding is better, from individual results some conclusions still can be made. For example, in 5 on the instance 5-3-7 Alpha encoding performed better than OR CSE, while on the instance 8-4-6 it was the opposite. So conjoining both encodings into a single one might perform better on both instances. This variability also could be a good property for building a portfolio of encodings, where an instance tackled using multiple encodings in parallel and get solved first by its most efficient encoding.

This research is still a work in progress and for the next stages we plan to answer some of the questions that have arisen. For instance, the question of the effect of conjoining two or more encodings and examining which SMT encodings obtain Generalized Arc Consistency and its impact on performance. Also the main question of our study, which is; Can we do better by directly translating from MiniZinc instead of going through the mzn2smt?. We are also planning to study other global constraints and our next nominee is the value precedence constraint, which is useful in breaking value symmetries.

References

1. Flener, Pierre, Alan M. Frisch, Brahim Hnich, Zeynep Kiziltan, Ian Miguel, Justin Pearson, and Toby Walsh. Breaking row and column symmetries in matrix models, *Principles and Practice of Constraint Programming-CP 2002*, pp. 462-477. Springer Berlin Heidelberg, 2002.

2. Bofill, Miquel, Miquel Palah, Josep Suy, and Mateu Villaret. Solving constraint satisfaction problems with SAT modulo theories, *Constraints* 17, no. 3 (2012): 273-303.
3. SMTLIB The Satisfiability Modulo Theories Library, `http://smt-lib.org/`
4. The Yices SMT Solver, `http://yices.csl.sri.com/`
5. van Hoeve W. and Katriel I. , Global Constraints, *Handbook of Constraint Programming*. Amsterdam, The Netherlands. Elsevier. 2006
6. CSPLib: A problem library for constraints, `http://www.csplib.org/`
7. G12 MiniZinc Distribution, `http://www.MiniZinc.org/software.html`
8. Frisch, Alan M., Brahim Hnich, Zeynep Kiziltan, Ian Miguel, and Toby Walsh. Propagation algorithms for lexicographic ordering constraints, *Artificial Intelligence* 170, no. 10 (2006): 803-834.
9. Gent, Ian P., Patrick Prosser, and Barbara M. Smith. A 0/1 encoding of the gaclex constraint for pairs of vectors, *Notes of the ECAI-02 Workshop W9 Modelling and Solving Problems with Constraints*. 2002.
10. My Constraint Programming Blog, `http://www.hakank.org/constraint_programming_blog/`
11. Warwick's Results Page for the Social Golfer Problem, `http://web.archive.org/web/20050308115423/http://www.icparc.ic.ac.uk/~wh/golf/`
12. Crawford, James, Matthew Ginsberg, Eugene Luks, and Amitabha Roy. Symmetry-breaking predicates for search problems, *5th Int. Conf. on Knowledge Representation and Reasoning KR 96* (1996): 148-159.