Encoding The Lexicographic Ordering Constraint in Satisfiability Modulo Theories

Hani Abdalla Muftah Elgabou

MSc by Research

University of York
Computer Science

February 2015
Dedication

To my Mother, my Wife and my Daughters
Abstract

In this thesis we present nine different SMT encodings for the Lexicographic Ordering Constraint (Lex). These constraints are helpful in breaking some kinds of symmetries in decision problems. The encodings are drawn from the constraint solving literature or is a variant of such. This thesis aims to serve as a single source for all known to date encodings for the lexicographic ordering constraint.

For the purpose of benchmarking, the encodings are translated into an SMT suitable form. We have done this using two methods, the first is by directly translating the encodings into SMT using C# code. 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 evaluated the encodings on a suite of instances of the Social Golfer problem and the Balanced Incomplete Block Design problem, both are well known for their highly symmetric models. We tried to run inference capability tests using unsatisfiable instances of Lex between two long vectors, also by trying to find all solutions for instances of the BIBD problem. We used the SMT solvers Yices2 and Z3 to benchmark the encodings.

Our results show that what we called the Recursive OR encoding performed better than all the other encoding on most of the instances, but it was notably worse on other instances. This behaviour is roughly shared by some of the other encodings, it shows that different encodings perform differently on different problems. The results also show that, in many cases, not having any symmetry braking using either of the nine encodings performed surprisingly better.
# Contents

Abstract iii  
List of figures v  
List of tables vi  
Acknowledgements x  
Declaration xi  

## 1 Introduction  
1.1 Motivation and Aims ................................................. 2  
1.2 Approach and Results ............................................. 2  
1.3 Contributions and Results ........................................ 3  

## 2 Background  
2.1 Boolean Satisfiability .............................................. 5  
2.2 SAT Solvers ......................................................... 7  
2.3 Satisfiability Modulo Theories .................................... 11  
2.4 Symmetry in Satisfiability .......................................... 14  

## 3 Encodings for the *Lex* Ordering Constraint  
3.1 The AND Decomposition Encoding ................................... 19  
3.2 The AND Decomposition Encoding using Common Sub-expression Elimination ................................................. 20  
3.3 The OR Decomposition Encoding ................................... 21  
3.4 The OR Decomposition Encoding using Common Sub-expression Elimination ................................................. 22  
3.5 The Recursive OR Decomposition .................................. 23
3.6 The Arithmetic LEX Encoding ........................................... 24
3.7 Harvey LEX Encoding .................................................... 25
3.8 Alpha LEX Encoding ..................................................... 26
3.9 Alpha M LEX Encoding .................................................. 28

4 Experimental Results ...................................................... 29
  4.1 The Social Golfers Problem ............................................. 30
  4.2 The Balanced Incomplete Block Design ............................ 38
  4.3 Long Vectors Instances ................................................ 43
  4.4 BIBD All Solutions Benchmarks ..................................... 46

5 Evaluation and Conclusions .............................................. 48
  5.1 Future Work ............................................................ 49

Abbreviations ..................................................................... 50

References .......................................................................... 51

Index .................................................................................. 57
List of Figures

3.1 The mzn2smt Pipeline .................................................. 18
4.1 Generating the Samples ................................................ 30
### List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Average solution time (in seconds) for instances of the SGP using the direct translation and Yices2 SMT solver</td>
<td>33</td>
</tr>
<tr>
<td>4.2</td>
<td>Average number of decisions (divided by 1000) for instances of the SGP using the direct translation and Yices2 SMT solver</td>
<td>33</td>
</tr>
<tr>
<td>4.3</td>
<td>Average number of decisions per millisecond for instances of the SGP using the direct translation and Yices2 SMT solver</td>
<td>34</td>
</tr>
<tr>
<td>4.4</td>
<td>Average solution time (in seconds) for instances of the SGP using mzn2smt translation and Yices2 SMT solver</td>
<td>34</td>
</tr>
<tr>
<td>4.5</td>
<td>Average number of decisions (divided by 1000) for instances of the SGP using mzn2smt translation and Yices2 SMT solver</td>
<td>34</td>
</tr>
<tr>
<td>4.6</td>
<td>Average number of decisions per millisecond for instances of the SGP using mzn2smt translation and Yices2 SMT solver</td>
<td>35</td>
</tr>
<tr>
<td>4.7</td>
<td>Average solution time (in seconds) for instances of the SGP using the direct translation and Z3 SMT solver</td>
<td>35</td>
</tr>
<tr>
<td>4.8</td>
<td>Average number of decisions (divided by 1000) for instances of the SGP using the direct translation and Z3 SMT solver</td>
<td>36</td>
</tr>
<tr>
<td>4.9</td>
<td>Average number of decisions per millisecond for instances of the SGP using the direct translation and Z3 SMT solver</td>
<td>36</td>
</tr>
<tr>
<td>4.10</td>
<td>Average solution time (in seconds) for instances of the SGP using the mzn2smt translation and Z3 SMT solver</td>
<td>36</td>
</tr>
<tr>
<td>4.11</td>
<td>Average number of decisions (divided by 1000) for instances of the SGP using the mzn2smt translation and Z3 SMT solver</td>
<td>37</td>
</tr>
<tr>
<td>4.12</td>
<td>Average number of decisions per millisecond for instances of the SGP using the mzn2smt translation and Z3 SMT solver</td>
<td>37</td>
</tr>
<tr>
<td>4.13</td>
<td>Average solution time (in seconds) for instances of the BIBD using the direct translation and Yices2 SMT solver</td>
<td>40</td>
</tr>
</tbody>
</table>
4.14 Average number of decisions for instances of the BIBD using the direct translation and Yices2 SMT solver .................................................. 40
4.15 Average number of decisions per millisecond for instances of the BIBD using the direct translation and Yices2 SMT solver ................................. 41
4.16 Average solution time (in seconds) for instances of the BIBD using mzn2smt translation and Yices2 SMT solver ........................................... 41
4.17 Average number of decisions for instances of the BIBD using mzn2smt translation and Yices2 SMT solver .................................................. 41
4.18 Average number of decisions per millisecond for instances of the BIBD using mzn2smt translation and Yices2 SMT solver ................................. 41
4.19 Average solution time (in seconds) for instances of the BIBD using the direct translation and Z3 SMT solver .............................................. 42
4.20 Average solution time (in seconds) for instances of The BIBD using the mzn2smt translation and Z3 SMT solver ........................................... 42
4.21 Average solution time (in seconds) for unsat instances of Lex between two vectors using the direct translation and Yices2 SMT solver .................. 44
4.22 Average solution time (in seconds) for unsat instances of Lex between two vectors using mzn2smt translation and Yices2 SMT solver .................. 44
4.23 Average solution time (in seconds) for unsat instances of Lex between two vectors using the direct translation and Z3 SMT solver .................. 44
4.24 Average solution time (in seconds) for unsat instances of Lex between two vectors using mzn2smt translation and Z3 SMT solver .................. 44
4.25 Number of occurrences of atoms (divided by 1000) for the instances of Lex between two vectors using the direct translation ................................. 45
4.26 Number of occurrences of atoms (divided by 1000) for the instances of Lex between two vectors using mzn2smt translation ................................. 46
4.27 Average solution time (in seconds) for finding all solutions for instances of the BIBD using the direct translation and Yices2 SMT solver ................. 47
4.28 Average number of solutions found in 300 seconds for the instance 13-3-1 of the BIBD using the direct translation and Yices2 SMT solver .................. 47
4.29 Number of solutions found in 300 seconds for instances of the BIBD without Lex constraint and using Yices2 SMT solver ...................................... 47
4.30 Average solution time (in seconds) for finding all solutions for instances of
    the BIBD using the mzn2smt translation and Yices2 SMT solver . . . . . . 47
4.31 Average number of solutions found in 300 seconds for the instances 13-3-1
    of the BIBD using the mzn2smt translation and Yices2 SMT solver . . . . . 47
Acknowledgements

All praise to Allah, the lord of the worlds,
the most beneficent and the most merciful.

I wish to express my sincere gratitude to my supervisor Dr. Alan M. Frisch for his patience and all the guidance and encouragement he provided me throughout this research. I would like to thank my assessor Dr. James Cussens for his support and feedback. I wish also to thank all staff and students at the University of York for the great friendly environment and for providing the facilities. With much appreciation, I recognize that this work would not have been possible without the financial assistance of the Libyan people, represented by the Libyan embassy in London.

Finally and most importantly, I would like to express my love and gratitude to Najwa, my wife, for her continued encouragement and patience.
Declaration

I declare that the research described in this thesis is original work, which I undertook at the University of York during 2013 - 2015. Except where stated, all of the work contained within this thesis represents the original contribution of the author.
Chapter 1

Introduction

Modern SAT solvers are essential tools in many applications today, including solving problems related to scheduling, verification, circuit design and more. Before to be solved, a problem need to be modelled as a Boolean Satisfiability Problem, in which the problem is translated to a group of Boolean constraints. These constraints are represented by groups of Boolean formulas called Conjunctive Normal Form or CNF. The increasing complexity of some of today’s combinatorial problems makes them difficult to be represented by only using SAT’s Boolean formulas. Satisfiability Modulo Theories (SMT) has been introduced to overcome this limitation [4] [21], 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.

The phenomenon of symmetry arises 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 still preserving validity of its rules. Breaking symmetry reduces the search space, which could improve performance [49] [15].

The $Lex$ ordering constraint enforces lexicographic ordering between two vectors of variables, which makes it useful in breaking symmetry between rows and between columns of a matrix of decision variables [53] [23] [3]. For example, enforcing $Lex$ between each row and its next in a Latin Square eliminates the interchangeability between the rows, the same applies for the columns.
1.1 Motivation and Aims

A recent study demonstrated that SMT solvers have a competitive performance in solving Constraint Satisfaction Problems (CSP). In that study, Bofill et al. [12] built a tool called fzn2smt to translate CSP instances expressed in Flatzinc into the SMT language SMTLIB1.2 standard. Then they solved these instances using the Yices1 SMT solver [22]. Yices1, in general, performed better than some well known constraint solvers. This performance makes encoding CSP problems in SMT a promising research line. Bofill’s results show that fzn2smt and Yices1 underperformed on instances that involve global constraints, some of them have the lexicographic ordering constraint. The aim of this thesis is to find whether there are better ways to encode the lexicographic constraint in SMT. We will try this by studying and benchmarking different SMT encodings of the lexicographic ordering constraint and compare their performance. We will also study the effect that Bofill’s translator has on these encodings when it is used to produce them.

1.2 Approach and Results

This thesis presents nine different encodings of the lexicographic ordering constraint.

1. The AND encoding [27]
2. The AND CSE encoding
3. The OR encoding [27]
4. The OR CSE encoding
5. The Recursive OR encoding [30]
6. The Arithmetic encoding [27]
7. Harvey’s encoding [27]
8. The Alpha encoding [30]
9. The AlphaM encoding [1]

Seven of the encodings are obtained from the literature then for the purpose of benchmarking 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 follows Bofill’s method
of translating MiniZinc instances to SMT using MiniZinc’s mzn2fzn and the fzn2smt tool. It starts by rewriting the encodings in MiniZinc language, flattening them into FlatZinc instances using MiniZinc’s mzn2fzn tool, then translating them to SMT using fzn2smt. We evaluate the encodings on a suite of instances of the Social Golfer problem and Balanced Incomplete Block Design. Both are well known for their highly symmetric models. We also tried to run inference capability tests using unsatisfiable instances of Lex between two long vectors and by trying to find all solutions for some instances of the Balanced Incomplete Block Design problem. We used Yices2 [22] and Z3 [17] SMT solvers to benchmark the encodings.

The results in Chapter 4 are for the benchmarking suite problems for each of the SMT solvers. We evaluated the performance of each encoding by recording the time it takes the solver to solve its instances. To find the best possible typical timing for each instance, we ran 30 different samples per instance. These samples were generated by randomly arranging the lines of code of the instances.

To test the effect of each encoding on the benchmarking problems, we also ran tests on all the instances without including any symmetry braking constraints (No Lex). Although many results show that the No Lex has better timings than all of the encodings, the picture would be completely different in cases of unsatisfiable instances or when searching for all possible solutions, as we will see in the all solutions benchmarks section. In the cases of unsatisfiable instances and all solutions, a solver needs to explore all possible paths of the search tree to find all solutions or to prove none is exists. Symmetry breaking aims to reduce number of these paths and makes the solving process runs faster. An example to this effect is clear in the results of the unsatisfiable instance of 8-4-4 in the result tables for the BIBD problem.

1.3 Contributions and Results

This thesis is the first collection of all known to date encodings for the lexicographic ordering constraints. We translated and evaluated these encodings in SMT. Our benchmarks results show that, typically, formula size could greatly affect the solving time. Our results also show that different encoding perform differently on different instances, this implies that using specialized algorithms to propagate Lex might be better than decomposing it into many smaller constraints. The results also proved that Bofill et al would have generally got better performances if they had used a direct translation to SMT where it is
possible instead of using the fn2smt. One surprising finding in this research, is that in the long vectors test, which are large instances of pure \textit{Lex}, the DNF OR encoding performed better than the CNF AND encoding. CNF is the modelling language for SAT solvers, which are at the heart of the SMT solving process.
Chapter 2

Background

2.1 Boolean Satisfiability

The Boolean Satisfiability Problem (SAT) is the problem of finding one or more assignments for a propositional formula which evaluate it to true, or proving no such assignment exists.

The kind of propositional formulas that Boolean satisfiability deals with are formulas composed of Boolean variables connected with relations of Boolean algebra. The basic forms of these relations are AND, OR and NOT ($\land, \lor, \neg$).

Let $A$, $B$ and $C$ be Boolean variables. The following three formulas are examples of propositional formulas.

$$A \lor B \quad (2.1)$$

$$(A \lor B) \land \neg C \quad (2.2)$$

$$\neg C \quad (2.3)$$

A single Boolean variable is a formula and a formula with no variables is called an empty formula.

Let $\alpha$, $\beta$ be propositional formulas, from the above examples we can note that all the following are also propositional formulas:
Throughout this thesis we will use formula to refer to a propositional formula and variable for Boolean variable.

Finding an assignment for a Boolean formula or proving none exists is the main goal of the Boolean SAT. A valid assignment for a formula is the set of true and false values that can be assigned to all of its variables which evaluates the formula to true. Formulas that have one or more valid assignment are denoted satisfiable (Sat), while the ones with no such assignment are called unsatisfiable (Unsat). Here are some examples of formulas and their satisfying assignments:

\[(A \lor \neg B) \land (\neg A \lor \neg B)\]  
\[A = true and B = false\]

\[(A \lor \neg B) \land (\neg A \lor \neg B)\]  
\[A = false and B = false\]

\[(\neg A \lor B) \land (A \lor B) \land \neg B\]  
Unsat (has no satisfying assignment)

2.1.1 Conjunctive Normal Form

Conjunctive Normal Form or CNF can be regarded as an input language to almost all current SAT solvers. Before solving a combinatorial problem using SAT, the problem needs to be translated or encoded in Conjunctive Normal Form.

Conjunctive Normal Form is a conjunction of disjunctions. It is conjunctions of clauses in which each clause is formed of disjunction of literals. A literal is a variable in one logical state. For any variable \(A\), \(A\) is a literal and \(\neg A\) is regarded as a different literal.

The following is an example of a formula in CNF:

\[(A \lor B) \land (A \lor \neg B \lor \neg C) \land D\]

In the above formula, \(A, B, C\) and \(D\) are literals, so are their negations. \((A \lor B)\) is a clause and also \((A \lor \neg B \lor \neg C)\). \(D\) at the end of the formula is called a unit clause, which is
a clause that contains only one literal. We will see later how unit clauses are very helpful in solving CNF formulas.

Using CNF has its advantages, among them are, transforming any propositional formula to CNF is relatively easy, can be done in a linear time and produces a formula of a linear size compared to the original but with more variables [16]. Also, the structure of the encoding facilitated relatively small but efficient algorithm to solve SAT problems, namely algorithms based on the DPLL algorithm [16].

2.2 SAT Solvers

SAT problem was the first problem to be proven to be NP-Complete [14]. A consequence of SAT’s NP-Completeness is that there are no known algorithms that can solve worst-case instances of SAT in a feasible time. However, the importance of SAT solving to a wide range of applications made it an active research field during the last decade. The advances in modern SAT solvers made them efficient in solving many difficult real world problems on different domains, this success drove the development of SAT solvers which in turn inspired more applications [44].

2.2.1 DPLL algorithm

most modern SAT solvers are based on the DPLL (Davis, Putnam, Logemann and Loveland) algorithm, which is a Search-Backtracking algorithm presented by Davis et al in 1962 [16].

DPLL algorithm works only on formulas in CNF. To solve a CNF formula, DPLL first tries to simplify the input CNF, then selects one of its literals and assign it a value, either true or false, and checks whether there is a conflict. A conflicting value is a value that evaluates a formula to false. When a conflict arose, DPLL backtracks and flips the value of the literal. In case of no conflicts DPLL picks another literal, assigns it a value and repeats the same operation. DPLL works recursively though all the literals in the formula, at the same time, trying to resolve the CNF on each recursion. DPLL terminates with two possible outcomes, the first, it manages to assign values to all of the atoms in the CNF, which proves the satisfiability of the formula, or it hits a conflict which cannot be resolved by backtracking, in this case the formula is proven to be unsat.

To simplify a formula, DPLL uses two formula resolution techniques, Unit Propaga-
tion or Boolean constraints Propagation (BCP) and the Pure Literal Rule. In both the
algorithm tries to reduce the input formula as much as possible before starting the search.
Unit propagation relies on two implications of a unit clause in a formula, for example,
let literal $C$ be a unit clause in a formula, removing all other clauses that contain $C$ will
not change the satisfiability state of the formula, also removing any occurrence of the
complement of $C$ from the formula has no effect on its satisfiability. To demonstrate this
here is an example:

\[
\begin{align*}
(A \lor B \lor C) \land (\neg A \lor B) \land (\neg B \lor \neg C) \land A \\
B \land (\neg B \lor \neg C) \land A \\
B \land \neg C \land A
\end{align*}
\]

In (2.4), $A$ forms a unit clause, so $(A \lor B \lor C)$ and $\neg A$ were eliminated from the
formula. This resulted in $B$ being the new unit clause as shown in formula (2.5), so we
propagate $B$. The technique is to keep applying unit propagation till no chances of further
simplifying the formula. (2.6) can not be simplified again, because all clauses are unit,
so from (2.6), the satisfying assignment for the formula is $B = true$, $C = False$ and
$A = true$. If the propagation resulted in an empty clause then the formula is proven to
be unsatisfiable as shown by the following.

\[
\begin{align*}
(A \lor \neg B) \land (A \lor B) \land \neg A \\
Propagate \neg A, (\neg B) \land (B) \land \neg A \\
Propagate B, (Empty \ Clause) \land (B) \land \neg A
\end{align*}
\]

As for the Pure Literal Rule, a pure literal is a Boolean variable that only appears in a
single Boolean state throughout its occurrences in a CNF formula. The Pure Literal Rule
states that a CNF formula remains equisatisfiable when setting all of its pure literals to
$True$. This property implies that removing all clauses containing pure literals also retains
the equisatisfiability. For example, consider the following formula.
\[(A \lor B) \land (A \lor \neg B) \land \neg B\]  
\[2.10\]

\[A \text{ is Pure, } A = \text{True}\]  
\[2.11\]

\[A \land \neg B\]  
\[2.12\]

Both (2.10) and (2.12) are logically equivalent and their only satisfiable assignment is \(A = \text{True}\) and \(B = \text{False}\).

In the next pseudo code for the DPLL algorithm, \textit{PureLiteral}(\(\phi\)) and \textit{UnitPropagate}(\(\phi\)) will apply the Pure Literal rule and Unit Propagation on the input formula \(\phi\). \textit{PickLiteral}(\(\phi\)) picks a different literal \(l\) from \(\phi\) on each run. On a successful pick, \(l\) gets assigned a value, a failure means that there are no more literals to choose, i.e. all literals in \(\phi\) have been successfully assigned a value and a satisfiable assignment has been found.

\begin{algorithm}
\textbf{Algorithm 1 : DPLL (\(\phi\))}
\begin{algorithmic}
\Require CNF Formula: \(\phi\)
\Ensure Satisfiability of \(\phi\) (True, False)
\State \(\phi = \text{PureLiteral}(\phi)\)
\State \(\phi = \text{UnitPropagate}(\phi)\)
\If {\(\Box \not\in \phi\) (\(\Box\) = An empty clause)}
\State \textit{if \textit{PickLiteral}(\(\phi\)) then}
\State \textit{return True}
\EndIf
\EndIf
\Return False
\end{algorithmic}
\end{algorithm}

\subsection{SAT Solvers Enhancements}

The importance of SAT solving to many applications motivated an intensive research efforts to improve the DPLL algorithm [45] [46]. The performance of Modern SAT solvers has been greatly increased by fine tuning DPLL and introducing new techniques, such as Optimized BCP, Conflict Analysis, Clause Learning, Heuristic Strategies and Restarts, in addition to efforts to increase performance by using SAT solvers in parallel to solve problems [61] [34]. Here we will briefly discuss some of the important enhancements.

- Optimized BCP
When solving SAT problems most of the SAT Solvers’ running time is spent on BCP, that is the case because whenever the DPLL algorithm assigns a value to a literal the propagation algorithm is continuously revisiting all the clauses checking whether any of them have become unit clauses. To ease this overhead Moskewicz et al introduced the Watched 2-Literals method which greatly improved the performance of SAT solving [46].

- Conflict Analysis and Clause Learning

Most of state-of-the-art SAT solvers are Conflict Driven Clause Learning solvers (CDCL), where the search is guided by the analysis of conflicts to produce learned clauses and strategies to decide where to Back-Jump from a conflict (Non-Chronological-Backtracking) [62]. Conflict analysis is also used to calculate when it is a good time to decide that the current solving efforts are unfruitful then restart. Restarts throw away all current assignments but preserve some of the useful learned clauses. These methods are used to prune the search space and guide the solver to more promising paths to find a SAT assignment.

- Heuristic Strategies

Heuristic Strategies concerned with the question; After propagation which literal to pick next to assign a value?. This choice could greatly affect the solving process, making bad ones could lead the solver to endless paths of unsat assignments. There are different heuristic strategies around but all of them depend on conflict analysis [62].

2.2.3 Applications of SAT Solvers

SAT solving has provided solutions for many applications, these include; software and hardware testing [39] [43], model checking and design verification and debugging [11] [55], in fact SAT solvers works as a backbone for some of today’s digital circuits design applications [52] [42].

Take Model Checking application as an example, most of the SAT based model checking applications rely on the concept of a Safety Properties [44] [11] [37], which is a set of Boolean constraints that must be satisfied on all states of the model. To check a model SAT solvers are used to try to find a complement of its Safety Properties, which is basically trying to find any assignment that makes the model fail.
Although Verification and Model Checking are the main application areas of SAT solvers, they also show promising performances in other real world domains, such as scheduling, planning [60] and optimisation using Max-SAT [28].

2.3 Satisfiability Modulo Theories

As SAT solvers started to gain popularity in more applications, the need for more expressive modelling language beyond the SAT’s CNF became more apparent. Some problems could be naturally represented by means of one or more background theories. For example, the arithmetic elements in a code, the physical properties of a system or the states of a model could be represented by the theory of linear arithmetic [10]. Surely some of these problems could still be modelled using CNF, but doing this could result in very large and tedious formulas and with a different abstraction level from the original model [51]. Consider the formula $A + B = 3$ and $A - B = 1$, where $1 \leq A \leq 3$ and $1 \leq B \leq 3$ for integers $A$ and $B$. A possible CNF encoding for the formula $(A + B = 3) \land (A - B = 1)$ is:

\[
\begin{align*}
A_1 &= \text{true for } A = 1, \quad A_2 = \text{true for } A = 2 \text{ and } A_3 = \text{true for } A = 3 \\
B_1 &= \text{true for } B = 1, \quad B_2 = \text{true for } B = 2 \text{ and } B_3 = \text{true for } B = 3 \\
\text{Possible values are } \{1, 2, 3\}. \quad \text{Encoded by the constraints in (2.13)}
\end{align*}
\]

Conflicting values for $(A, B)$ are $(1, 1), (2, 2), (3, 3), (3, 1), (1, 3), (2, 3), (3, 2)$

The conflicting values are encoded by the constraints in (2.14) and (2.15)

The CNF:

\[
\begin{align*}
(A_1 \lor A_2 \lor A_3) \land (B_1 \lor B_2 \lor B_3) \land \\
(\neg A_1 \lor \neg B_1) \land (\neg A_2 \lor \neg B_2) \land (\neg A_3 \lor \neg B_3) \land (\neg A_3 \lor \neg B_1) \land \\
(\neg A_1 \lor \neg B_3) \land (\neg A_2 \lor \neg B_3) \land (\neg A_3 \lor \neg B_2)
\end{align*}
\]

There are other ways to encode the previous formula in CNF [29] [58] [56], but introducing more variables other than $A$ and $B$ is unavoidable in all of them. If we repeat the same process but with domains of $-10 \leq A \leq 10$ and $-10 \leq B \leq 10$, we would need 21 auxiliary variables in addition to a special algorithm to figure out all the conflicting pairs of values. The answer to this problem was extending the available SAT solvers to deal directly with atoms in one or more background theories $\mathcal{T}$ instead of encoding them into
Procedures for reasoning over background theories ($T - Solvers$) have been gaining interest since the 1970s and already employed in some domains [47] [54] [20], such as Knowledge Representation and Reasoning, Constraint Satisfaction Problems and AI. However, these procedures could not handle the reasoning with respect to the Boolean component of a formula [51], so procedures which combine theory reasoning with Boolean satisfiability were matured over the last three decades [33] [36] [32] [4] [19] to leverage the strengths of both sides. This combination of procedures are called Satisfiability Modulo Theories or SMT.

Satisfiability Modulo Theories is the problem of deciding the satisfiability of a propositional formula $\phi$ with respect to a background theory $T$ [7]. In other words the problem of finding assignment $\mu$ in $T$ that satisfy the formula $\phi$. SMT deals with formulas of combination of atoms in propositional logic and others in one or more background theories, as shown in the next example.

\[(x \geq 2) \land (x \leq 0 \lor y \geq 0) \land A \land B\] (2.16)

$x \geq 2$, $x \leq 0$ and $y \geq 0$ are atoms in the theory of linear arithmetic ($T - atoms$). $x$ and $y$ are integers.

SMT solving is addressed by integrating a SAT solver with one or more theory solvers ($T - Solvers$). There are two approaches to achieve this integration, the first is to translate the input formula into equi-satisfiable proportional formula then use an off-the-shelf SAT solver to produce a satisfiable assignment. A theory solver ($T - Solver$) is used to check the validity the assignment with respect to $T$, this method is called the Eager approach. The second method and the widely adapted by the SMT community is called the Lazy approach [10] [51], in which the $T - atoms$ in the input formula are abstracted and fed to the SAT solver to produce an assignment. The $T - Solver$ keeps checking the assignment while it is being built for any conflicts in $T$ ($T - Inconsistency$). In case of $T - Inconsistency$ the $T - Solver$ generates a conflict clause (lemma) then adds it to the formula and feed it back to the SAT solver. Using the previous formula (2.16) as an example:
**SMT solver input:**

\[(x \geq 2) \land (x \leq 0 \lor y \geq 0) \land A \land B\]  \hspace{1cm} (2.17)

**Abstarcting \(T\) – atoms:**

\[c = (x \geq 2), \quad d = (x \leq 0) \text{ and } e = (y \geq 0)\]  \hspace{1cm} (2.18)

**SAT solver input:**

\[c \land (d \lor e) \land A \land B\]  \hspace{1cm} (2.19)

**SAT solver returns a partial model:**

\[c = \text{true}, \quad d = \text{true}\]  \hspace{1cm} (2.20)

**\(T\) – solver checks the partial model for any \(T\) – Inconsistency:**

\[(x \geq 2) \text{ and } (x \leq 0) \text{ are in Conflict}\]  \hspace{1cm} (2.21)

So the partial model \((c = \text{true}, d = \text{true})\) is \(T\) – Inconsistent  \hspace{1cm} (2.22)

**\(T\) – solver notifies the SAT solver about the conflict**

The lemma \(\neg (c \land d)\) is added to the formula  \hspace{1cm} (2.23)

**SAT solver Backjumps and adjusts the model**

\hspace{1cm} (2.24)

2.3.1 SMTLIB

SMT-LIB is the standard modelling language for Satisfiability Modulo Theories. It was introduced by the SMT-LIB initiative [7] to serve as a standard benchmarking suite for their annual SMT solvers competition [13] and a standard language and interfaces for the different SMT solvers [8]. The SMT-LIB standard promotes ease of parsing over human readability, that is because SMT-LIB code is meant to be generated by automated modelling tools, also to make it easier for the solvers to parse the code so easier for their developers to adopt the standard [7].

2.3.2 SMT Applications

The modular concept of SMT solvers enables them to easily support new theories which could open the doors for more applications. The scope of applications of SMT solvers currently ranges over software testing and verifications, model checking and theorem proving. Companies like Microsoft developing their own SMT solver [17] and currently using
it as a main tool for their software verification and unit test generation [18]. Another well known innovative company, SRI International [2], rely on the integration of their SMT solver Yices to build their theorem proving, model checking and probabilistic consistency tools [22]. Alongside the previous examples, the SMT solver Yices proved to be very competitive on a suit of benchmarks of problems related to scheduling, optimisation, design and others [12].

2.4 Symmetry in Satisfiability

A symmetry in satisfiability can be defined as permutations of a set of assignments of a formula which preserve its satisfiability state. For instance, if $M$ is a set of all assignments that satisfies the formula $\phi$, then any bijection $f(M)$ that maintains the satisfiability of $\phi$ is a symmetry of $M$. Symmetries arises in models of many satisfiability problems, where pairs of values or variables in a formula can be interchanged without affecting its satisfiability state [40] [50]. For example, because of the property of commutativity $(\neg A \lor B) \land (B \lor \neg C)$ has a symmetry between its variables $A$ and $B$, which maps $A \rightarrow B$ and $B \rightarrow A$. This type of symmetry is known as variable symmetry. Another type of symmetry appears when the interchangeability is possible between values. For instance, in $(\neg A \lor B) \land (A \lor \neg C) \land (\neg B \lor C)$ we could flip any assignments of $A$ and $B$ with no effect on the outcome of the formula. Variable and value symmetries could exist simultaneously in the same model, this case is known as the mixed symmetry.

Symmetries in satisfiability solving create redundant paths to solutions as well as to non-solutions in the search space. Eliminating Symmetries could save a solver time and resources spent in exploring those paths in search of a solution [25] [49] [15]. There are two main approaches to symmetry breaking in satisfiability [31] [24], both are based on adding extra constraints to prune the redundant branches of the search tree. The main difference between the two lays on when to add the symmetry breaking constraints. One approach adds a set of symmetry breaking constraints to the problem’s model before starting the search, it is called static symmetry breaking. The second, which is known as dynamic symmetry breaking, is based on trying to guide the search process away form symmetries by adding the appropriate symmetry breaking constraints during search.
2.4.1 Symmetry Braking and the Lexicographic Order Constraint

A well known case of symmetries in combinatorial problems is the interchangeability between rows and between columns of matrices of decision variables. An $n \times m$ size matrix has a $n! \times m!$ possible symmetries among each pair of rows and pair columns. The following example shows the possible rows and columns permutations for a $2 \times 3$ array.

\[
\begin{pmatrix}
a & b & c \\
d & e & f
\end{pmatrix}
\begin{pmatrix}
b & a & c \\
e & d & f
\end{pmatrix}
\begin{pmatrix}
c & b & a \\
f & e & d
\end{pmatrix}
\begin{pmatrix}
a & c & b \\
d & f & e
\end{pmatrix}
\begin{pmatrix}
c & a & b \\
f & d & e
\end{pmatrix}
\begin{pmatrix}
b & c & a \\
e & f & d
\end{pmatrix}
\begin{pmatrix}
d & e & f \\
\end{pmatrix}
\begin{pmatrix}
e & d & f \\
b & a & c
\end{pmatrix}
\begin{pmatrix}
f & e & d \\
\end{pmatrix}
\begin{pmatrix}
d & f & e \\
\end{pmatrix}
\begin{pmatrix}
f & d & e \\
\end{pmatrix}
\begin{pmatrix}
e & f & d \\
\end{pmatrix}
\begin{pmatrix}
a & b & c \\
\end{pmatrix}
\begin{pmatrix}
b & a & c \\
\end{pmatrix}
\begin{pmatrix}
c & b & a \\
\end{pmatrix}
\begin{pmatrix}
a & c & b \\
\end{pmatrix}
\begin{pmatrix}
c & a & b \\
\end{pmatrix}
\begin{pmatrix}
b & c & a \\
\end{pmatrix}
\]

Row and column symmetries are closely related to some real world problems, specifically whenever matrices are employed to model a problem, which is a common practice in scheduling problems for instance.

The Lexicographic order constraint has been proved useful in breaking certain kinds of symmetries in matrices of decision variables. To break all row and column symmetries Crawford et al [15] introduced what is now known as the lex-leader constraints. An example provided by Frisch et al [26] shows how to apply row-wise lex-leader on a $2 \times 3$ array like the one in the previous example, it is built on the idea that an order of a matrix must be lexicographically less than or equal that all of its permutations, to do so, a matrix is transformed into a single row of elements, starting from Left-Right-Top-Down our previous $2 \times 3$ matrix becomes $[a, b, c, d, e, f]$ and the symmetry breaking constraints are presented as follows:

\[
\begin{pmatrix}
a & b & c & d & e & f
\end{pmatrix} \leq_{lex} \begin{pmatrix}
b & a & c & e & d & f
\end{pmatrix}
\]

\[
\begin{pmatrix}
a & b & c & d & e & f
\end{pmatrix} \leq_{lex} \begin{pmatrix}
c & b & a & f & e & d
\end{pmatrix}
\]

\[
\begin{pmatrix}
a & b & c & d & e & f
\end{pmatrix} \leq_{lex} \begin{pmatrix}
a & c & b & d & f & e
\end{pmatrix}
\]

\[
\begin{pmatrix}
a & b & c & d & e & f
\end{pmatrix} \leq_{lex} \begin{pmatrix}
c & a & b & f & d & e
\end{pmatrix}
\]
\[
\begin{bmatrix}
a & b & c & d & e & f \\
\end{bmatrix} \leq_{\text{Lex}}
\begin{bmatrix}
b & c & a & e & f & d \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
a & b & c & d & e & f \\
\end{bmatrix} \leq_{\text{Lex}}
\begin{bmatrix}
d & e & f & a & b & c \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
a & b & c & d & e & f \\
\end{bmatrix} \leq_{\text{Lex}}
\begin{bmatrix}
e & d & f & b & a & c \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
a & b & c & d & e & f \\
\end{bmatrix} \leq_{\text{Lex}}
\begin{bmatrix}
f & e & d & c & b & a \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
a & b & c & d & e & f \\
\end{bmatrix} \leq_{\text{Lex}}
\begin{bmatrix}
d & f & e & a & c & b \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
a & b & c & d & e & f \\
\end{bmatrix} \leq_{\text{Lex}}
\begin{bmatrix}
f & d & e & c & a & b \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
a & b & c & d & e & f \\
\end{bmatrix} \leq_{\text{Lex}}
\begin{bmatrix}
e & f & d & b & c & a \\
\end{bmatrix}
\]

This will eliminate all the 11 permutations from the previous example retaining only the following assignment:

\[
\begin{pmatrix}
a & b & c \\
d & e & f \\
\end{pmatrix}
\]

Although Crawford’s method is complete, i.e., breaks all row and column symmetries in \( n \times m \) matrices, it is impractical specially on large matrices, because it produces \((n! \times m!) – 1\) symmetry breaking constraints, which could add burdens that outweigh the benefit of any potential search space pruning.

Based on Crawford’s work and by introducing number of symmetry breaking (SB) constraints linear to number of rows and columns in matrices, Shlyakhter [53] and Flener et al [23] independently managed to break not all but a great percentage of symmetries in matrices. This is done by adding a Lex constraint between each pair of neighbouring rows and columns, this introduces \((n – 1) + (m – 1)\) number of SB constraints. Again, using the matrix from the previous examples, this method can be presented as follows:
\[
\begin{pmatrix}
a & b & c \\
d & e & f \\
\end{pmatrix}
\]

\[\leq_{Lex} \text{ between rows}\]

\[
\begin{bmatrix}
a & b & c \\
d & e & f \\
\end{bmatrix} \leq_{Lex} \begin{bmatrix}
a & b & c \\
d & e & f \\
\end{bmatrix}
\]

\[\leq_{Lex} \text{ between columns}\]

\[
\begin{bmatrix}
a & d \\
b & e \\
\end{bmatrix} \leq_{Lex} \begin{bmatrix}
a & d \\
b & e \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
b & e \\
c & f \\
\end{bmatrix} \leq_{Lex} \begin{bmatrix}
b & e \\
c & f \\
\end{bmatrix}
\]

We used this method to model all the symmetry breaking constraints we used in this research.

Lex constraint could be also used to break both variable and values symmetries at the same time [59].
Chapter 3

Encodings for the \textit{Lex} Ordering Constraint

This chapter presents nine different encodings for the \textit{Lex} Ordering Constraint. Each of which is drawn from the constraint solving literature or is a variant of such. Throughout, we consider a non-strict \textit{Lex} constraint between two vectors $A$ and $B$ of finite-domain variables. Both vectors are considered to be of length $n$. We write such constraint as $A \leq_{\text{lex}} B$. We assume that $n \geq 2$, because \textit{Lex} on two vectors of $n = 1$ is simply $A[1] \leq B[1]$.

We shall use the term mzn2smt to refer to Bofill’s method of translating a MiniZinc 1.6 [1] specification to SMT 1.2, it is done in two steps, starts by 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.
Each of the following subsections presents an encoding of the *Lex* constraint followed by the result of passing it through the mzn2smt pipeline. Since most of the encodings are non-CNF, we decided to measure their sizes by number of atom occurrences, rather than number of constraints. $T_1[i], T_2[i], \ldots$ are auxiliary Boolean arrays introduced by the mzn2smt. The index $i$ of these arrays ranges from 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

This encoding, which is considered by Frisch *et al* [27], 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.

$$\bigwedge_{i=1}^{n-1} \big( \bigwedge_{j=1}^i (A[j] = B[j]) \rightarrow (A[i+1] \leq B[i+1]) \big)$$

This encoding produces $\frac{(n^2+n)}{2}$ atom occurrences, which makes the produced formula grow quadratically.

A strict ordering can be obtained by adding the constraint $(A[n-1] = B[n-1]) \rightarrow (A[n] < B[n])$ to the encoding and reducing the range of $i$ to $n-2$ in the conjunction.

After translation using mzn2smt:

$$A[1] \leq B[1] \quad (3.1)$$
$$1 \leq i \leq n-1 \quad T_1[i] \iff (A[i] = B[i]) \quad (3.2)$$
$$1 \leq i \leq n-1 \quad T_2[i] \iff (A[i+1] \leq B[i+1]) \quad (3.3)$$
$$1 \leq i \leq n-2 \quad \bigwedge_{j=1}^{i+1} T_1[j] \quad (3.4)$$
$$-T_1[1] \lor T_2[1] \quad (3.5)$$
$$1 \leq i \leq n-2 \quad -T_3[i] \lor T_2[i+1] \quad (3.6)$$

This translation also has an $O(n^2)$ growth and produces $\frac{(n^2+13n)}{2} + 9$ atom occurrences.
A strict ordering can be obtained by adding the constraint $T_2[n - 1] \Leftrightarrow (A[n] < B[n])$ to the encoding and reducing the range of $i$ to $n - 2$ in (3.3).

### 3.2 The AND Decomposition Encoding using Common Sub-expression Elimination

Common Sub-expression Elimination helps in reducing formula size by substituting any recurring parts of the formula with variables. For example, by applying $Lex$ using the AND encoding between vectors $A$ and $B$, both of a size 4, we get the following formula:

\[(A[1] \leq B[1])\] (3.7)

The above formula has a quadratic size growth, which can be eliminated using the Boolean array $X[i]$ to perform CSE. The following resulting formula has a linear growth:

\[(A[1] \leq B[1])\] (3.11)

The following encoding, which we call AND CSE, is similar to the AND encoding 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 (3.20). The purpose of this encoding is to compare performance between using the nested loops as in line (3.4) in the previous encoding and this approach.
The AND encoding using eliminating common sub-expressions using the Boolean array $X[i]$: 

\begin{align*}
1 \leq i \leq n-2 &\quad X[i+1] \iff (X[i] \land (A[i+1] = B[i+1])) & (3.20) \\
1 \leq i \leq n-1 &\quad X[i] \rightarrow (A[i+1] \leq B[i+1]) & (3.21)
\end{align*}

This formula produces $5n-5$ atom occurrences. A strict ordering can be obtained by adding the constraint $X[n-1] \rightarrow (A[n] < B[n])$ to the encoding and changing the range of $i$ in (3.21) to $n-2$.

After translation using mzn2smt:

\begin{align*}
1 \leq i \leq n-2 &\quad T_1[i] \iff (A[i+1] = B[i+1]) & (3.24) \\
1 \leq i \leq n-1 &\quad T_2[i] \iff (A[i+1] \leq B[i+1]) & (3.25) \\
1 \leq i \leq n-2 &\quad X[i+1] \iff (X[i] \land T_1[i]) & (3.26) \\
1 \leq i \leq n-1 &\quad \neg X[i] \lor T_2[i] & (3.27)
\end{align*}

This formula produces $9n-11$ atom occurrences. A strict ordering can be obtained by adding the constraint $T_2[n-1] \equiv (A[n] < B[n])$ to the encoding and reducing the range of $i$ to $n-2$ in (3.25).

### 3.3 The OR Decomposition Encoding

This encoding, also considered by Frisch et al [27], decomposes the $Lex$ constraint into a formula of smaller constraints disjoined together, i.e., a DNF formula. It is traditionally known as the OR decomposition.
\[(A[1] < B[1]) \lor \]
\[
\left( \bigvee_{i=1}^{n-1} (A[i] = B[j]) \land (A[i+1] < B[i+1]) \right) \lor
\]
\[
\left( \bigwedge_{i=1}^{n} (A[i] = B[i]) \right) \tag{3.28}
\]
\[
\left( \bigwedge_{i=1}^{n} (A[i] = B[i]) \right) \tag{3.29}
\]

This encoding produces \( \frac{(n^2 + 3n)}{2} \) atom occurrences, which, like AND encoding, makes the produced formula grow quadratically. A strict ordering can be obtained by removing (3.29) from the above formula.

After translation using mzn2smt:

\[
1 \leq i \leq n \quad T_1[i] \leftrightarrow (A[i] = B[i]) \tag{3.30}
\]
\[
1 \leq i \leq n \quad T_2[i] \leftrightarrow (A[i] < B[i]) \tag{3.31}
\]
\[
1 \leq i \leq n - 1 \quad T_3[i] \leftrightarrow \bigwedge_{j=1}^{i} T_1[j] \land T_2[i+1] \tag{3.32}
\]
\[
T_3[n] \leftrightarrow \bigwedge_{i=1}^{n} T_1[i] \tag{3.33}
\]
\[
(\bigvee_{i=1}^{n-1} T_3[i]) \lor T_2[1] \lor T_3[n] \tag{3.34}
\]

This encoding produces \( \frac{(n^2 + 15n)}{2} \) atom occurrences. A strict ordering can be obtained by removing \( T_3[n] \) from (3.34).

### 3.4 The OR Decomposition Encoding using Common Sub-expression Elimination

This OR decomposition, we call it 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 1 to \( n \),
it is used to eliminate common sub-expressions as shown in the following formula.

\[(A[1] < B[1]) \lor \]
\[\bigvee_{i=1}^{n-1} (X[i] \land (A[i+1] < B[i+1])) \lor X[n]\]  
(3.35)

(3.36)

\[1 \leq i \leq n-1 \quad X[i+1] \iff (X[i] \land (A[i+1] = B[i+1]))\]  
(3.37)

This encoding produces 5\(n - 1\) atom occurrences. A strict ordering can be obtained by removing \(X[n]\) from (3.35).

After translation the OR using mzn2smt we get:

\[1 \leq i \leq n \quad X[1] \iff (A[1] = B[1])\]  
(3.38)

\[1 \leq i \leq n-1 \quad T_2[i] \iff (A[i+1] = B[i+1])\]  
(3.39)

\[1 \leq i \leq n \quad T_3[i] \iff (A[i] < B[i])\]  
(3.40)

\[1 \leq i \leq n-1 \quad X[i+1] \iff (X[i] \land T_2[i])\]  
(3.41)

\[1 \leq i \leq n-1 \quad T_4[i] \iff (X[i] \land T_3[i+1])\]  
(3.42)

\[1 \leq i \leq n-1 \quad T_4[i] \iff T_3[i+1] \lor X[n]\]  
(3.43)

This encoding produces 13\(n - 7\) atom occurrences. A strict ordering can be obtained by removing \(X[n]\) from (3.43).

### 3.5 The Recursive OR Decomposition

This variant of the OR encoding, presented by Gent et al. [30], decomposes \(Lex\) into a set of nested ORs and ANDs, unwinding them produces the same OR encoding. We produced this encoding using a recursion.


We get the above encoding using the following constraints. We introduced the boolean array \(X\), of a size \(n\), to eliminate common sub-expressions and to simulate a recursion.
The Recursive OR Decomposition produces $2n$ atom occurrences. It can be made strict by changing $(A[n] \leq B[n])$ in (3.45) to $(A[n] < B[n])$.

The mzn2smt translation of the recursive OR is as follows:

\[
X[1] \quad \begin{equation} \tag{3.47} \end{equation}
\]

\[
1 \leq i \leq n \quad T_1[i] \Leftrightarrow (A[1] = B[1]) \quad \begin{equation} \tag{3.48} \end{equation}
\]

\[
1 \leq i \leq n \quad T_2[i] \Leftrightarrow (A[1] < B[1]) \quad \begin{equation} \tag{3.49} \end{equation}
\]

\[
1 \leq i \leq n \quad X[n] \Leftrightarrow (T_1[n] \lor T_2[n]) \quad \begin{equation} \tag{3.50} \end{equation}
\]

\[
1 \leq i \leq n - 1 \quad T_3[i] \Leftrightarrow (X[i] \Leftrightarrow T_2[i]) \quad \begin{equation} \tag{3.51} \end{equation}
\]

\[
1 \leq i \leq n - 1 \quad T_4[i] \Leftrightarrow (T_1[n - i] \Leftrightarrow X[n - i + 1]) \quad \begin{equation} \tag{3.52} \end{equation}
\]

\[
1 \leq i \leq n - 1 \quad T_4[i] \lor T_3[n - i] \quad \begin{equation} \tag{3.53} \end{equation}
\]

This translation produces $12n - 4$ atom occurrences. It can be made strict by removing $T_1[n]$ from (3.50).

\section*{3.6 The Arithmetic Lex Encoding}

Another way of encoding Lex constraint is using an arithmetic constraint. This constraint considered by Frisch et al \cite{arithmetic_lex}, it 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 1 to $d$.

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

The mzn2smt translation produces exactly 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 $<$. 

### 3.7 Harvey Lex Encoding

This alternative arithmetic encoding is introduced by Frisch et al [27] who attribute it to Warwick Harvey. The general formula of this encoding is:


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 $n - 1$.

$$X[1]$$

$$X[n] \Leftrightarrow (A[n] < (B[n] + 1))$$

$$0 \leq i \leq n - 2 \quad X[n - i - 1] \Leftrightarrow (A[n - i - 1] < (B[n - i - 1] + \text{Bool2Int}(X[n - i])))$$

This encoding produces $2n - 1$ atom occurrences. We get a strict version by changing $B[n] + 1$ to $B[n] + 0$ in (3.55). The 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 $n$.
\[ X[1] \]
\[ X[n] \Leftrightarrow ((A[n] - B[n]) \leq 0) \] (3.57)
\[ 1 \leq i \leq n \quad \text{int}[i] \leq 1 \] (3.58)
\[ 1 \leq i \leq n \quad \text{int}[i] \geq 0 \] (3.59)
\[ 1 \leq i \leq n - 1 \quad X[i + 1] \rightarrow (\text{int}[i] = 1) \] (3.60)
\[ 1 \leq i \leq n - 1 \quad \neg X[i + 1] \rightarrow (\text{int}[i] = 0) \] (3.61)
\[ 1 \leq i \leq n - 1 \quad X[n - i] \Leftrightarrow ((A[n - i] - B[n - i] - \text{int}[i]) \leq -1) \] (3.62)

This translation produces \(8n - 3\) number of atom occurrences. We get a strict version by changing \(((A[n] - B[n]) \leq 0)\) to \(((A[n] - B[n]) < 0)\) in (3.58).

### 3.8 Alpha Lex Encoding

This encoding was introduced by Gent et al [30], we called it 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:

\[ 1 \leq i \leq n \]
\[ 1 \leq j \leq i \]
\[ \text{if } \alpha[i] = \text{true then } A[j] = B[j] \] (3.64)
\[ \text{if } (\alpha[i] = \text{true and } \alpha[i + 1] = \text{false}) \text{ then } A[i + 1] < B[i + 1] \] (3.65)

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 behaves in a way similar to the AND encoding. Line (3.66) is to guarantee that $A[1] \leq B[1]$ and if $A[1] = B[1]$ then $\alpha[1] = true$, which in turn, implies that the next index of $A$ is less than or equal the next index of $B$, this goes on till the end of the vectors. The difference between the two encodings appears with the first occurrence of $A[i] < B[i]$, where ALPHA uses the constraints in lines (3.67) and (3.69) to make the values of $A[i]$ and $B[i]$ after the first occurrence of $A[i] < B[i]$ insignificant to the problem.

This encoding can be changed to a strict $Lex$ by adding the constraint $\neg \alpha[n+1]$ to the formula. The encoding produces $9n - 6$ atom occurrences.

Next is the encoding’s mzn2smt translation. Here we use $\alpha'$ instead of $\alpha$ because the mzn2smt translator changed the range of $\alpha$ from $0 \leq i \leq n$ to $1 \leq i \leq n + 1$.

This translation produces $18n - 3$ atom occurrences and can be changed to a strict $Lex$ by adding the constraint $\neg \alpha[n+1]$. 

27
3.9 Alpha M \textit{Lex} Encoding

This decomposition, which we call Alpha M, is the default decomposition used by the Minizinc 1.6 [1]. Like the previous Alpha encoding it uses a Boolean array as a bookkeeping mechanism for relations between the corresponding values in both vectors. The index of the Alpha array ranges from 1 to $n+1$.

\begin{align}
\alpha[1] & \quad (3.81) \\
1 \leq i \leq n & \quad \alpha[i] \Leftrightarrow (((A[i] < B[i]) \lor \alpha[i+1]) \land (A[i] \leq B[i])) \quad (3.82)
\end{align}


This encoding produces $4n+1$ number of atom occurrences and we can obtain a strict version by adding $\neg\alpha[n+1]$ to the constraints.

After translation using mzn2smt:

\begin{align}
\alpha[1] & \quad (3.83) \\
1 \leq i \leq n & \quad T_1[i] \Leftrightarrow (A[i] \leq B[i]) \quad (3.84) \\
1 \leq i \leq n & \quad T_2[i] \Leftrightarrow (A[i] < B[i]) \quad (3.85) \\
1 \leq i \leq n & \quad T_3[i] \Leftrightarrow (T_2[i] \lor \alpha[i+1]) \quad (3.86) \\
1 \leq i \leq n & \quad \alpha[i] \Leftrightarrow (T_3[i] \land T_1[i]) \quad (3.87)
\end{align}

This translation produces $10n+1$ number of atom occurrences and could be changed to a strict \textit{Lex} by also adding the constraint $\neg\alpha[n+1]$.
Chapter 4

Experimental Results

We evaluated eight of the decompositions on solving a suite of instances of the Social Golfers Problem (SGP) and Balanced Incomplete Block Designs Problem (BIBD), problems 010 and 028 in CSPLib [38]. Both problems are well known for their highly symmetric models. The arithmetic decomposition is not evaluated because it is impractical to do so. We also run inference tests on a set of unsatisfiable instances of enforcing \textit{Lex} between two long vectors. To chose the benchmarking SMT solvers, we ran performance comparison tests between four SMT solvers, Yices1, Yices2 [22], Z3 [17] and CVC4 [9]. We decided to use the two with the best performances and the best output format, these were Yices2 and Z3.

We ran two sets of benchmarks to evaluate the decompositions. The first, is by directly translating each encoding to SMT using the C# programming language then benchmark them. In the second, we were aiming to test the effect of using constraint reification on the SMT decompositions of \textit{Lex}. Constraint reification was the method chosen by Bofill \textit{et al} to translate CSP problems into SMT [12], which alongside the SMT solver Yices2, proved competitive against some leading CSP solvers. Reifying a constraint $C$ is reformulating it to the form of $(b \iff C)$, where $b$ is a boolean proposition. We used two off-the-shelf tools, Minizinc’s mzn2fzn [1] and Bofill’s fzn2smt to get the reified constraints translations for the decompositions.

Both sets of benchmarks share the same code for the benchmarking problems, the difference is only in the code related to the different \textit{Lex} constraint encodings.

We made the direct translation using a C# language code. As for the mzn2smt translation, the MiniZinc implementation includes a set of libraries to decompose global constraints and made to be called from MiniZinc models. We created a similar MiniZinc
global library for each of the eight \textit{Lex} decompositions, then we called them from the benchmarking problem’s code. We modified a MiniZinc model for the problems by adding a symmetry breaking based on lexicographical orderings constraint.

Frisch \textit{et al} \cite{27} proved that the conjunction of AND and OR encodings facilitates a stronger Generalised Arc Consistency (GAC) than each separately, so we decided to include \textit{AND} $\land$ \textit{OR} in the Long Vectors instances tests and call it \textit{ANDOR}. We did not do \textit{ANDOR} tests on the SGP and the BIBD because the size of the \textit{ANDOR} caused Out-of-Memory problems on most of the instances.

All benchmarks were run on a Windows PC with Intel i7 1.8Ghz processor and 8GB of RAM and using Yices2 v2.2.1 and Z3 v4.3.0 SMT solvers on the SGP, the BIBD problems and on the unsatisfiable instances.

We noticed that different orders of an SMT file have different solving times. To get a mean value of solving times, we decided to run 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. To generate the 30 samples we made a shuffling script. Each figure in the tables from table 4.1 to 4.24 represents an average of 30 values for each encoding on each instance. Figure 4.1 demonstrates the process of obtaining the samples. We set a time-out of 300 seconds for each run for both the SMT solver.

![Figure 4.1: Generating the Samples](image)

Apart from the instance 8-4-4 of the SGP, we only used satisfiable instances for both the SGP and the BIBD. Instances of the SGP were obtained from \cite{35}, as for the BIBD we made our own. The instances were chosen to represent different levels of difficulty.

## 4.1 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 than 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 \(m = g \times s\) 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 Week2, Player1 and Player3 meet in the first group, Player2 and Player5 meet in the second group and Player4 and Player6 meet in the third.

<table>
<thead>
<tr>
<th>Week1</th>
<th>Week2</th>
<th>Week3</th>
<th>Player</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Player1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>Player2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>Player3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>Player4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>Player5</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>Player6</td>
</tr>
</tbody>
</table>

The Social Golfer is known for its highly symmetric models. For example, in the previous solution of the instance 3-2-3 of the SGP, we could swap any two rows or two columns and still get a valid solution. The solution below is a result of swapping the first and last rows.

<table>
<thead>
<tr>
<th>Week1</th>
<th>Week2</th>
<th>Week3</th>
<th>Player</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>Player1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>Player2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>Player3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>Player4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>Player5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Player6</td>
</tr>
</tbody>
</table>

We use the Lex constraint to break two groups of symmetries in the problem; symmetries in weeks (columns) and symmetries in players (rows). 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 [41] 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:

\[
1 \leq \text{group} \leq g \quad 1 \leq \text{week} \leq w \quad \left( \sum_{\text{player}=1}^{m} \text{Bool2Int}(\text{Schedule[\text{player}, \text{week}]} = \text{group})) = s \right.
\]
Where \( \text{Bool2Int}() \) is Boolean to integer converter function.

Each pair of players meet at most once

\[
1 \leq pa \leq m \quad 1 \leq pb \leq m
\]
\[
1 \leq wa \leq w \quad 1 \leq wb \leq w
\]

where \( pa \neq pb \land wa \neq wb \)

\((\text{Schedule}\{pa, wa\} \neq \text{Schedule}\{pb, wa\}) \lor \)
\((\text{Schedule}\{pa, wb\} \neq \text{Schedule}\{pb, wb\})\)

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 \( \text{Schedule}[] \) it is clear that symmetries can happen between weeks and between players. To break symmetry between weeks we put \( \text{lex} \) constraint ordering between each two neighbouring columns and the same is done for players.

\text{Lex} \text{ constraint on weeks:}

\[
1 \leq \text{week} \leq w - 1 \quad [\text{Schedule}\{\text{player}, \text{week}\} \mid \text{player} \in 1..m] \leq_{\text{lex}} \]
\[
[\text{Schedule}\{\text{player}, \text{week} + 1\} \mid \text{player} \in 1..m]
\]

\text{Lex} \text{ constraint on players:}

\[
1 \leq \text{player} \leq m - 1 \quad [\text{Schedule}\{\text{player}, \text{week}\} \mid \text{week} \in 1..w] \leq_{\text{lex}} \]
\[
[\text{Schedule}\{\text{player} + 1, \text{week}\} \mid \text{week} \in 1..w]
\]

4.1.1 SGP Results using Yices2

The results of the SGP using Yices2 are arranged in four tables, two for timings for the directly translated instances and their corresponding numbers of decisions, and similar two for the mzn2smt translation. From Tables 4.1 and 4.4, apart from the Recursive OR (ROR) and Harvey’s, all other encoding show similar performance with very minor differences between timings. ROR’s average timing is greatly reduced by its timeing-out on the instance 5-3-7, and a smaller effect by the instances 6-3-5 and 5-3-6, otherwise it performed better than all of the other instances, and on most of the larger instances.
Also, from the two tables, it is clear that using constraint reification greatly improved ROR and Harvey’s timings, with not much effect on the others. Also, CSE made a negligible difference in cases of the AND and the OR. Tables 4.3 and 4.6 show that, in general, ROR is much faster in making decisions than all the others on most of the instances. Decisions are assigning values to variables, and they depend on the solver’s ability to make certain inferences. Those faster decisions imply that the ROR encoding facilitates solving the instances using faster or fewer inferences compared to the other encodings. We compute decision rates by dividing the average number of decisions by the average solution time for each encoding on each instance.

<table>
<thead>
<tr>
<th>Instances</th>
<th>AND</th>
<th>AND</th>
<th>OR</th>
<th>OR</th>
<th>ROR</th>
<th>Alpha</th>
<th>AlphaM</th>
<th>Harvey</th>
<th>No Lex</th>
</tr>
</thead>
<tbody>
<tr>
<td>G-S-W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-3-5</td>
<td>0.27</td>
<td>0.28</td>
<td>0.27</td>
<td>0.29</td>
<td>0.17</td>
<td>0.27</td>
<td>0.25</td>
<td>0.42</td>
<td>0.38</td>
</tr>
<tr>
<td>5-3-6</td>
<td>1.93</td>
<td>1.88</td>
<td>2.07</td>
<td>1.53</td>
<td>9.66</td>
<td>1.91</td>
<td>1.78</td>
<td>3.13</td>
<td>164.85</td>
</tr>
<tr>
<td>5-3-7</td>
<td>12.36</td>
<td>13.59</td>
<td>13.65</td>
<td>16.18</td>
<td>300.00</td>
<td>10.73</td>
<td>17.84</td>
<td>32.51</td>
<td>265.53</td>
</tr>
<tr>
<td>6-3-5</td>
<td>0.26</td>
<td>0.27</td>
<td>0.29</td>
<td>0.34</td>
<td>0.17</td>
<td>0.27</td>
<td>0.26</td>
<td>0.48</td>
<td>0.15</td>
</tr>
<tr>
<td>6-3-6</td>
<td>1.54</td>
<td>1.76</td>
<td>1.94</td>
<td>1.88</td>
<td>0.97</td>
<td>1.69</td>
<td>1.83</td>
<td>3.18</td>
<td>1.69</td>
</tr>
<tr>
<td>6-3-7</td>
<td>7.27</td>
<td>7.26</td>
<td>7.24</td>
<td>6.98</td>
<td>12.50</td>
<td>6.94</td>
<td>6.93</td>
<td>10.56</td>
<td>119.99</td>
</tr>
<tr>
<td>6-4-4</td>
<td>0.72</td>
<td>0.69</td>
<td>0.65</td>
<td>0.66</td>
<td>0.42</td>
<td>0.65</td>
<td>0.68</td>
<td>1.11</td>
<td>1.15</td>
</tr>
<tr>
<td>6-4-5</td>
<td>4.81</td>
<td>5.06</td>
<td>5.04</td>
<td>4.72</td>
<td>2.96</td>
<td>4.74</td>
<td>4.64</td>
<td>6.34</td>
<td>9.25</td>
</tr>
<tr>
<td>8-4-4</td>
<td>1.12</td>
<td>1.03</td>
<td>1.21</td>
<td>1.25</td>
<td>0.62</td>
<td>1.16</td>
<td>1.00</td>
<td>1.71</td>
<td>0.88</td>
</tr>
<tr>
<td>8-4-5</td>
<td>10.41</td>
<td>10.57</td>
<td>9.89</td>
<td>10.78</td>
<td>6.68</td>
<td>10.08</td>
<td>10.56</td>
<td>21.26</td>
<td>23.56</td>
</tr>
<tr>
<td>8-4-6</td>
<td>82.84</td>
<td>82.16</td>
<td>81.60</td>
<td>78.15</td>
<td>36.75</td>
<td>90.06</td>
<td>82.68</td>
<td>127.09</td>
<td>57.63</td>
</tr>
</tbody>
</table>

| Arith-mean | 11.23| 11.32| 11.26| 11.16| 33.72| 11.68| 11.68  | 18.89  | 58.63  |
| Geo-mean   | 2.73 | 2.78 | 2.84 | 2.86 | 3.01 | 2.71  | 2.76   | 4.66   | 7.80   |

Table 4.1: Average solution time (in seconds) for instances of the SGP using the direct translation and Yices2 SMT solver

<table>
<thead>
<tr>
<th>Instances</th>
<th>AND</th>
<th>AND</th>
<th>OR</th>
<th>OR</th>
<th>ROR</th>
<th>Alpha</th>
<th>AlphaM</th>
<th>Harvey</th>
</tr>
</thead>
<tbody>
<tr>
<td>G-S-W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-3-5</td>
<td>8</td>
<td>10</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>8</td>
<td>8</td>
<td>13</td>
</tr>
<tr>
<td>5-3-6</td>
<td>41</td>
<td>39</td>
<td>41</td>
<td>34</td>
<td>79</td>
<td>34</td>
<td>35</td>
<td>50</td>
</tr>
<tr>
<td>5-3-7</td>
<td>135</td>
<td>114</td>
<td>99</td>
<td>133</td>
<td>2,163</td>
<td>92</td>
<td>141</td>
<td>192</td>
</tr>
<tr>
<td>6-3-5</td>
<td>10</td>
<td>12</td>
<td>11</td>
<td>12</td>
<td>16</td>
<td>9</td>
<td>7</td>
<td>18</td>
</tr>
<tr>
<td>6-3-6</td>
<td>50</td>
<td>51</td>
<td>54</td>
<td>57</td>
<td>62</td>
<td>54</td>
<td>56</td>
<td>82</td>
</tr>
<tr>
<td>6-3-7</td>
<td>163</td>
<td>158</td>
<td>162</td>
<td>164</td>
<td>296</td>
<td>177</td>
<td>152</td>
<td>199</td>
</tr>
<tr>
<td>6-4-4</td>
<td>22</td>
<td>23</td>
<td>20</td>
<td>19</td>
<td>29</td>
<td>22</td>
<td>20</td>
<td>31</td>
</tr>
<tr>
<td>6-4-5</td>
<td>122</td>
<td>127</td>
<td>128</td>
<td>134</td>
<td>136</td>
<td>125</td>
<td>114</td>
<td>132</td>
</tr>
<tr>
<td>8-4-4</td>
<td>66</td>
<td>62</td>
<td>72</td>
<td>65</td>
<td>98</td>
<td>61</td>
<td>59</td>
<td>98</td>
</tr>
<tr>
<td>8-4-5</td>
<td>467</td>
<td>405</td>
<td>347</td>
<td>449</td>
<td>607</td>
<td>429</td>
<td>409</td>
<td>727</td>
</tr>
<tr>
<td>8-4-6</td>
<td>2,154</td>
<td>2,446</td>
<td>2,443</td>
<td>2,311</td>
<td>2,135</td>
<td>2,552</td>
<td>2,209</td>
<td>3,051</td>
</tr>
</tbody>
</table>

Table 4.2: Average number of decisions (divided by 1000) for instances of the SGP using the direct translation and Yices2 SMT solver
Table 4.3: Average number of decisions per millisecond for instances of the SGP using the direct translation and Yices2 SMT solver

<table>
<thead>
<tr>
<th>Instances</th>
<th>AND</th>
<th>AND</th>
<th>OR</th>
<th>OR</th>
<th>ROR</th>
<th>Alpha</th>
<th>AlphaM</th>
<th>Harvey</th>
</tr>
</thead>
<tbody>
<tr>
<td>G-S-W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-3-5</td>
<td>0.27</td>
<td>0.25</td>
<td>0.28</td>
<td>0.26</td>
<td>0.17</td>
<td>0.25</td>
<td>0.27</td>
<td>0.33</td>
</tr>
<tr>
<td>5-3-6</td>
<td>1.40</td>
<td>1.79</td>
<td>1.70</td>
<td>2.02</td>
<td>1.46</td>
<td>1.60</td>
<td>1.59</td>
<td>1.94</td>
</tr>
<tr>
<td>5-3-7</td>
<td>12.53</td>
<td>13.08</td>
<td>11.07</td>
<td>13.94</td>
<td>265.0</td>
<td>13.47</td>
<td>14.77</td>
<td>16.71</td>
</tr>
<tr>
<td>6-3-5</td>
<td>0.27</td>
<td>0.27</td>
<td>0.25</td>
<td>0.26</td>
<td>0.16</td>
<td>0.26</td>
<td>0.29</td>
<td>0.32</td>
</tr>
<tr>
<td>6-3-6</td>
<td>1.63</td>
<td>1.64</td>
<td>1.70</td>
<td>1.79</td>
<td>1.03</td>
<td>1.69</td>
<td>1.55</td>
<td>1.87</td>
</tr>
<tr>
<td>6-3-7</td>
<td>6.77</td>
<td>6.72</td>
<td>7.14</td>
<td>7.32</td>
<td>28.92</td>
<td>6.10</td>
<td>6.84</td>
<td>6.81</td>
</tr>
<tr>
<td>6-4-4</td>
<td>0.63</td>
<td>0.63</td>
<td>0.62</td>
<td>0.63</td>
<td>0.47</td>
<td>0.66</td>
<td>0.64</td>
<td>0.68</td>
</tr>
<tr>
<td>6-4-5</td>
<td>5.32</td>
<td>4.82</td>
<td>5.01</td>
<td>4.53</td>
<td>3.14</td>
<td>4.97</td>
<td>4.88</td>
<td>4.89</td>
</tr>
<tr>
<td>8-4-4</td>
<td>1.08</td>
<td>0.98</td>
<td>1.11</td>
<td>1.17</td>
<td>0.67</td>
<td>1.09</td>
<td>1.14</td>
<td>1.20</td>
</tr>
<tr>
<td>8-4-5</td>
<td>10.88</td>
<td>10.80</td>
<td>10.42</td>
<td>9.73</td>
<td>6.77</td>
<td>10.76</td>
<td>9.63</td>
<td>10.80</td>
</tr>
<tr>
<td>8-4-6</td>
<td>85.45</td>
<td>87.43</td>
<td>79.19</td>
<td>90.63</td>
<td>41.51</td>
<td>84.18</td>
<td>93.65</td>
<td>83.29</td>
</tr>
<tr>
<td>Arith-mean</td>
<td>11.48</td>
<td>11.67</td>
<td>10.77</td>
<td>12.03</td>
<td>31.85</td>
<td>11.37</td>
<td>12.29</td>
<td>11.71</td>
</tr>
<tr>
<td>Geo-mean</td>
<td>2.66</td>
<td>2.67</td>
<td>2.65</td>
<td>2.77</td>
<td>2.95</td>
<td>2.66</td>
<td>2.73</td>
<td>2.97</td>
</tr>
</tbody>
</table>

Table 4.4: Average solution time (in seconds) for instances of the SGP using mzn2smt translation and Yices2 SMT solver

<table>
<thead>
<tr>
<th>Instances</th>
<th>AND</th>
<th>AND</th>
<th>OR</th>
<th>OR</th>
<th>ROR</th>
<th>Alpha</th>
<th>AlphaM</th>
<th>Harvey</th>
</tr>
</thead>
<tbody>
<tr>
<td>G-S-W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-3-5</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>9</td>
<td>14</td>
<td>8</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>5-3-6</td>
<td>32</td>
<td>34</td>
<td>37</td>
<td>38</td>
<td>48</td>
<td>35</td>
<td>34</td>
<td>40</td>
</tr>
<tr>
<td>5-3-7</td>
<td>129</td>
<td>113</td>
<td>94</td>
<td>123</td>
<td>629</td>
<td>115</td>
<td>161</td>
<td>128</td>
</tr>
<tr>
<td>6-3-5</td>
<td>11</td>
<td>10</td>
<td>10</td>
<td>11</td>
<td>16</td>
<td>10</td>
<td>11</td>
<td>13</td>
</tr>
<tr>
<td>6-3-6</td>
<td>52</td>
<td>59</td>
<td>63</td>
<td>56</td>
<td>63</td>
<td>59</td>
<td>55</td>
<td>61</td>
</tr>
<tr>
<td>6-3-7</td>
<td>150</td>
<td>139</td>
<td>174</td>
<td>172</td>
<td>181</td>
<td>128</td>
<td>162</td>
<td>169</td>
</tr>
<tr>
<td>6-4-4</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>22</td>
<td>31</td>
<td>22</td>
<td>23</td>
<td>21</td>
</tr>
<tr>
<td>6-4-5</td>
<td>139</td>
<td>106</td>
<td>121</td>
<td>131</td>
<td>136</td>
<td>129</td>
<td>127</td>
<td>125</td>
</tr>
<tr>
<td>8-4-4</td>
<td>67</td>
<td>61</td>
<td>61</td>
<td>69</td>
<td>109</td>
<td>62</td>
<td>71</td>
<td>67</td>
</tr>
<tr>
<td>8-4-5</td>
<td>431</td>
<td>407</td>
<td>420</td>
<td>375</td>
<td>652</td>
<td>446</td>
<td>425</td>
<td>472</td>
</tr>
<tr>
<td>8-4-6</td>
<td>2,235</td>
<td>2,507</td>
<td>2,155</td>
<td>2,521</td>
<td>2,645</td>
<td>2,290</td>
<td>2,728</td>
<td>2,483</td>
</tr>
</tbody>
</table>

Table 4.5: Average number of decisions (divided by 1000) for instances of the SGP using mzn2smt translation and Yices2 SMT solver
Table 4.6: Average number of decisions per millisecond for instances of the SGP using mzn2smt translation and Yices2 SMT solver

3.5

4.1.2 SGP Results using Z3

Similar to SGP using Yices2, apart from the ROR and Harvey encodings, the results here show similar performances of Z3 on all instances. The dominance of the ROR encoding is much evident here. Z3 results also supports that ROR facilitates a faster decision rate. CSE did not make a noticeable difference in cases of the AND and the OR. Constraint reification slightly helped Harvey’s, but had contrary small effect on the others.

Table 4.7: Average solution time (in seconds) for instances of the SGP using the direct translation and Z3 SMT solver

<table>
<thead>
<tr>
<th>Instances</th>
<th>AND</th>
<th>AND</th>
<th>OR</th>
<th>OR</th>
<th>ROR</th>
<th>Alpha</th>
<th>AlphaM</th>
<th>Harvey</th>
<th>No Lex</th>
</tr>
</thead>
<tbody>
<tr>
<td>G-S-W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-3-5</td>
<td>1.17</td>
<td>1.14</td>
<td>1.17</td>
<td>1.19</td>
<td>0.44</td>
<td>1.18</td>
<td>1.14</td>
<td>1.44</td>
<td>0.27</td>
</tr>
<tr>
<td>5-3-6</td>
<td>6.80</td>
<td>7.26</td>
<td>7.63</td>
<td>8.19</td>
<td>3.05</td>
<td>7.67</td>
<td>8.06</td>
<td>18.17</td>
<td>5.26</td>
</tr>
<tr>
<td>5-3-7</td>
<td>44.86</td>
<td>51.42</td>
<td>44.17</td>
<td>56.74</td>
<td>34.52</td>
<td>48.16</td>
<td>42.15</td>
<td>98.45</td>
<td>173.12</td>
</tr>
<tr>
<td>6-3-5</td>
<td>2.30</td>
<td>2.24</td>
<td>2.43</td>
<td>2.35</td>
<td>0.61</td>
<td>2.38</td>
<td>2.33</td>
<td>2.66</td>
<td>0.38</td>
</tr>
<tr>
<td>6-3-6</td>
<td>4.37</td>
<td>4.33</td>
<td>4.62</td>
<td>4.72</td>
<td>1.03</td>
<td>4.48</td>
<td>4.48</td>
<td>5.39</td>
<td>0.70</td>
</tr>
<tr>
<td>6-3-7</td>
<td>15.67</td>
<td>15.70</td>
<td>19.15</td>
<td>19.59</td>
<td>4.77</td>
<td>17.98</td>
<td>15.86</td>
<td>25.36</td>
<td>5.31</td>
</tr>
<tr>
<td>6-4-4</td>
<td>3.40</td>
<td>3.45</td>
<td>3.79</td>
<td>3.68</td>
<td>0.83</td>
<td>3.53</td>
<td>3.47</td>
<td>4.03</td>
<td>0.57</td>
</tr>
<tr>
<td>6-4-5</td>
<td>9.53</td>
<td>10.29</td>
<td>10.09</td>
<td>10.45</td>
<td>2.29</td>
<td>9.56</td>
<td>9.67</td>
<td>15.69</td>
<td>2.55</td>
</tr>
<tr>
<td>8-4-4</td>
<td>10.94</td>
<td>11.18</td>
<td>12.74</td>
<td>12.77</td>
<td>1.99</td>
<td>11.40</td>
<td>11.18</td>
<td>12.71</td>
<td>1.20</td>
</tr>
<tr>
<td>8-4-5</td>
<td>31.64</td>
<td>31.82</td>
<td>38.22</td>
<td>38.04</td>
<td>3.97</td>
<td>30.58</td>
<td>32.17</td>
<td>38.81</td>
<td>2.77</td>
</tr>
<tr>
<td>8-4-6</td>
<td>59.38</td>
<td>65.06</td>
<td>58.76</td>
<td>50.47</td>
<td>9.92</td>
<td>69.05</td>
<td>66.38</td>
<td>42.88</td>
<td>8.24</td>
</tr>
<tr>
<td>Arith-Mean</td>
<td>17.28</td>
<td>18.54</td>
<td>18.43</td>
<td>18.93</td>
<td>5.76</td>
<td>18.72</td>
<td>17.90</td>
<td>24.14</td>
<td>18.22</td>
</tr>
<tr>
<td>Instances</td>
<td>AND</td>
<td>AND</td>
<td>OR</td>
<td>OR</td>
<td>ROR</td>
<td>Alpha</td>
<td>AlphaM</td>
<td>Harvey</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>-----</td>
<td>-----</td>
<td>----</td>
<td>----</td>
<td>-----</td>
<td>-------</td>
<td>--------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>G-S-W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-3-5</td>
<td>1.36</td>
<td>1.31</td>
<td>1.70</td>
<td>1.83</td>
<td>1.32</td>
<td>1.32</td>
<td>1.38</td>
<td>2.15</td>
<td></td>
</tr>
<tr>
<td>5-3-7</td>
<td>113.56</td>
<td>132.28</td>
<td>114.73</td>
<td>148.88</td>
<td>176.81</td>
<td>117.69</td>
<td>107.17</td>
<td>196.62</td>
<td></td>
</tr>
<tr>
<td>6-3-5</td>
<td>2.13</td>
<td>2.01</td>
<td>2.84</td>
<td>2.36</td>
<td>1.90</td>
<td>1.89</td>
<td>2.22</td>
<td>3.14</td>
<td></td>
</tr>
<tr>
<td>6-3-6</td>
<td>3.53</td>
<td>3.34</td>
<td>4.65</td>
<td>4.77</td>
<td>3.51</td>
<td>3.47</td>
<td>3.71</td>
<td>5.64</td>
<td></td>
</tr>
<tr>
<td>6-3-7</td>
<td>22.68</td>
<td>22.65</td>
<td>34.05</td>
<td>32.32</td>
<td>17.83</td>
<td>26.35</td>
<td>22.98</td>
<td>47.79</td>
<td></td>
</tr>
<tr>
<td>6-4-4</td>
<td>2.29</td>
<td>2.30</td>
<td>3.66</td>
<td>3.22</td>
<td>2.57</td>
<td>2.32</td>
<td>2.43</td>
<td>3.78</td>
<td></td>
</tr>
<tr>
<td>6-4-5</td>
<td>7.47</td>
<td>8.31</td>
<td>9.60</td>
<td>10.38</td>
<td>6.24</td>
<td>7.53</td>
<td>7.95</td>
<td>17.98</td>
<td></td>
</tr>
<tr>
<td>6-4-6</td>
<td>22.68</td>
<td>22.65</td>
<td>34.05</td>
<td>32.32</td>
<td>17.83</td>
<td>26.35</td>
<td>22.98</td>
<td>47.79</td>
<td></td>
</tr>
<tr>
<td>6-4-5</td>
<td>22.68</td>
<td>22.65</td>
<td>34.05</td>
<td>32.32</td>
<td>17.83</td>
<td>26.35</td>
<td>22.98</td>
<td>47.79</td>
<td></td>
</tr>
<tr>
<td>8-4-4</td>
<td>4.94</td>
<td>5.01</td>
<td>8.64</td>
<td>8.80</td>
<td>5.77</td>
<td>4.86</td>
<td>5.08</td>
<td>8.78</td>
<td></td>
</tr>
<tr>
<td>8-4-5</td>
<td>18.96</td>
<td>20.52</td>
<td>35.14</td>
<td>34.69</td>
<td>11.70</td>
<td>17.41</td>
<td>21.56</td>
<td>35.85</td>
<td></td>
</tr>
<tr>
<td>8-4-6</td>
<td>56.55</td>
<td>49.70</td>
<td>64.47</td>
<td>68.98</td>
<td>37.27</td>
<td>47.00</td>
<td>49.42</td>
<td>80.34</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.8: Average number of decisions (divided by 1000) for instances of the SGP using the direct translation and Z3 SMT solver

<table>
<thead>
<tr>
<th>Instances</th>
<th>AND</th>
<th>AND</th>
<th>OR</th>
<th>OR</th>
<th>ROR</th>
<th>Alpha</th>
<th>AlphaM</th>
<th>Harvey</th>
</tr>
</thead>
<tbody>
<tr>
<td>G-S-W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>535</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>536</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>537</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>635</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>636</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>637</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>644</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>645</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>844</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>845</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>846</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 4.9: Average number of decisions per millisecond for instances of the SGP using the direct translation and Z3 SMT solver

<table>
<thead>
<tr>
<th>Instances</th>
<th>AND</th>
<th>AND</th>
<th>OR</th>
<th>OR</th>
<th>ROR</th>
<th>Alpha</th>
<th>AlphaM</th>
<th>Harvey</th>
<th>No Lex</th>
</tr>
</thead>
<tbody>
<tr>
<td>G-S-W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-3-5</td>
<td>1.16</td>
<td>1.19</td>
<td>1.28</td>
<td>1.27</td>
<td>0.43</td>
<td>1.26</td>
<td>1.17</td>
<td>1.35</td>
<td>0.27</td>
</tr>
<tr>
<td>5-3-6</td>
<td>8.40</td>
<td>7.62</td>
<td>7.63</td>
<td>8.23</td>
<td>3.56</td>
<td>10.13</td>
<td>7.70</td>
<td>10.41</td>
<td>5.26</td>
</tr>
<tr>
<td>5-3-7</td>
<td>44.20</td>
<td>42.19</td>
<td>53.84</td>
<td>54.60</td>
<td>29.89</td>
<td>41.99</td>
<td>49.93</td>
<td>53.69</td>
<td>173.12</td>
</tr>
<tr>
<td>6-3-5</td>
<td>2.31</td>
<td>2.28</td>
<td>2.51</td>
<td>2.44</td>
<td>0.60</td>
<td>2.43</td>
<td>2.36</td>
<td>2.57</td>
<td>0.38</td>
</tr>
<tr>
<td>6-3-6</td>
<td>4.58</td>
<td>4.51</td>
<td>4.92</td>
<td>4.86</td>
<td>1.05</td>
<td>4.60</td>
<td>4.52</td>
<td>5.02</td>
<td>0.70</td>
</tr>
<tr>
<td>6-3-7</td>
<td>16.46</td>
<td>16.50</td>
<td>20.12</td>
<td>18.05</td>
<td>5.08</td>
<td>17.03</td>
<td>15.88</td>
<td>20.16</td>
<td>5.31</td>
</tr>
<tr>
<td>6-4-4</td>
<td>3.37</td>
<td>3.41</td>
<td>3.71</td>
<td>3.78</td>
<td>0.87</td>
<td>3.55</td>
<td>3.40</td>
<td>3.71</td>
<td>0.57</td>
</tr>
<tr>
<td>6-4-5</td>
<td>9.77</td>
<td>9.74</td>
<td>11.29</td>
<td>11.65</td>
<td>2.43</td>
<td>10.73</td>
<td>9.81</td>
<td>10.39</td>
<td>2.55</td>
</tr>
<tr>
<td>6-4-6</td>
<td>11.02</td>
<td>11.02</td>
<td>12.65</td>
<td>13.49</td>
<td>2.10</td>
<td>12.10</td>
<td>11.85</td>
<td>12.16</td>
<td>1.20</td>
</tr>
<tr>
<td>6-4-6</td>
<td>31.81</td>
<td>31.46</td>
<td>39.30</td>
<td>39.33</td>
<td>4.04</td>
<td>31.11</td>
<td>31.61</td>
<td>34.75</td>
<td>2.77</td>
</tr>
<tr>
<td>6-4-6</td>
<td>65.38</td>
<td>69.76</td>
<td>46.14</td>
<td>52.66</td>
<td>10.68</td>
<td>66.54</td>
<td>67.83</td>
<td>57.28</td>
<td>8.24</td>
</tr>
<tr>
<td>Arith-Mean</td>
<td>18.04</td>
<td>18.15</td>
<td>18.49</td>
<td>19.12</td>
<td>5.52</td>
<td>18.32</td>
<td>18.73</td>
<td>19.23</td>
<td>18.22</td>
</tr>
<tr>
<td>Geo-Mean</td>
<td>9.50</td>
<td>9.43</td>
<td>10.21</td>
<td>10.38</td>
<td>2.53</td>
<td>9.97</td>
<td>9.61</td>
<td>10.56</td>
<td>2.38</td>
</tr>
</tbody>
</table>

Table 4.10: Average solution time (in seconds) for instances of the SGP using the mzn2smt translation and Z3 SMT solver
### Table 4.11: Average number of decisions (divided by 1000) for instances of the SGP using the mzn2smt translation and Z3 SMT solver

<table>
<thead>
<tr>
<th>Instances</th>
<th>AND</th>
<th>AND</th>
<th>OR</th>
<th>OR</th>
<th>ROR</th>
<th>Alpha</th>
<th>AlphaM</th>
<th>Harvey</th>
</tr>
</thead>
<tbody>
<tr>
<td>G-S-W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-3-5</td>
<td>1.23</td>
<td>1.27</td>
<td>1.95</td>
<td>1.86</td>
<td>1.24</td>
<td>1.38</td>
<td>1.29</td>
<td>1.83</td>
</tr>
<tr>
<td>5-3-6</td>
<td>16.67</td>
<td>14.83</td>
<td>15.80</td>
<td>17.37</td>
<td>15.54</td>
<td>20.11</td>
<td>14.90</td>
<td>20.86</td>
</tr>
<tr>
<td>5-3-7</td>
<td>100.71</td>
<td>97.57</td>
<td>133.10</td>
<td>139.69</td>
<td>145.75</td>
<td>96.25</td>
<td>119.15</td>
<td>125.43</td>
</tr>
<tr>
<td>6-3-5</td>
<td>1.99</td>
<td>2.06</td>
<td>3.09</td>
<td>2.81</td>
<td>1.77</td>
<td>1.96</td>
<td>2.14</td>
<td>2.60</td>
</tr>
<tr>
<td>6-3-6</td>
<td>3.38</td>
<td>3.56</td>
<td>5.24</td>
<td>4.97</td>
<td>3.45</td>
<td>3.48</td>
<td>3.57</td>
<td>4.48</td>
</tr>
<tr>
<td>6-3-7</td>
<td>23.81</td>
<td>24.28</td>
<td>36.47</td>
<td>30.91</td>
<td>19.56</td>
<td>25.99</td>
<td>24.02</td>
<td>35.19</td>
</tr>
<tr>
<td>6-4-4</td>
<td>2.19</td>
<td>2.16</td>
<td>3.47</td>
<td>3.59</td>
<td>2.40</td>
<td>2.50</td>
<td>2.33</td>
<td>3.15</td>
</tr>
<tr>
<td>6-4-5</td>
<td>7.94</td>
<td>7.59</td>
<td>11.89</td>
<td>12.56</td>
<td>6.28</td>
<td>8.53</td>
<td>7.78</td>
<td>9.13</td>
</tr>
<tr>
<td>8-4-4</td>
<td>4.93</td>
<td>4.87</td>
<td>8.98</td>
<td>10.90</td>
<td>5.54</td>
<td>5.26</td>
<td>5.21</td>
<td>6.94</td>
</tr>
<tr>
<td>8-4-5</td>
<td>20.25</td>
<td>18.83</td>
<td>35.88</td>
<td>35.83</td>
<td>11.07</td>
<td>17.15</td>
<td>17.80</td>
<td>24.99</td>
</tr>
<tr>
<td>8-4-6</td>
<td>50.73</td>
<td>47.27</td>
<td>73.33</td>
<td>67.63</td>
<td>38.04</td>
<td>49.24</td>
<td>48.99</td>
<td>63.10</td>
</tr>
</tbody>
</table>

### Table 4.12: Average number of decisions per millisecond for instances of the SGP using mzn2smt translation and Z3 SMT solver

<table>
<thead>
<tr>
<th>Instances</th>
<th>AND</th>
<th>AND</th>
<th>OR</th>
<th>OR</th>
<th>ROR</th>
<th>Alpha</th>
<th>AlphaM</th>
<th>Harvey</th>
</tr>
</thead>
<tbody>
<tr>
<td>G-S-W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>535</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>536</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>537</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>635</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>636</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>637</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>644</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>645</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>844</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>845</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>846</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.12: Average number of decisions per millisecond for instances of the SGP using mzn2smt translation and Z3 SMT solver
4.2 The Balanced Incomplete Block Design

The Balanced Incomplete Block Design is a classic combinatorial problem and it has some applications in design theory [5] [57]. The BIBD is the problem of finding a design of \( v \) distinct objects into \( b \) blocks in which each block has exactly \( k \) distinct objects, every object appears in \( r \) blocks and each two distinct objects appear together in \( \lambda \) blocks [48]. An instance of the BIBD is donated \((v, b, k, r, \lambda)\).

The BIBD MiniZinc model that we used is a part of the MiniZinc benchmark suite [1]. We modified the model by removing its symmetry breaking constraints and adding ones based on the \( \text{Lex} \) ordering constraint. The model uses a 0/1 \([v, b]\) matrix to hold the block designs. One possible solution for the instance \((7,7,4,4,2)\) could be represented by the following matrix, where columns are \textit{Blocks} and rows are \textit{Objects}. A value of 1 represents the occurrence of an \textit{Object}_v in a \textit{Block}_b while 0 represents its absence.

<table>
<thead>
<tr>
<th>Block_1</th>
<th>Block_2</th>
<th>Block_3</th>
<th>Block_4</th>
<th>Block_5</th>
<th>Block_6</th>
<th>Block_7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\textit{Object}_1

\textit{Object}_2

\textit{Object}_3

\textit{Object}_4

\textit{Object}_5

\textit{Object}_6

\textit{Object}_7

The model implements the BIBD by using three constraints that represent the three BIBD rules mentioned earlier. Assuming \( M[\cdot,] \) is the BIBD matrix, since \( M[\cdot,] \) is a 0/1 matrix the BIBD rule of each \textit{Block}_b has exactly \( k \) distinct \textit{Objects} could be modelled as every column must sum to \( k \).

\[
1 \leq \text{Block} \leq b \quad (\sum_{\text{Object}=1}^{v} M[\text{Object}, \text{Block}]) = k
\]

Likewise, the second rule which states that every object must appear in \( r \) blocks becomes every row must sum to \( r \).
1 \leq \text{Object} \leq v \quad (\sum_{\text{Block}=1}^{b} M[\text{Object}, \text{Block}]) = r

The last rule, which restricts the number of Blocks that each two Objects could appear together to \( \lambda \) is modelled as the dot product of every pair of distinct rows must equal to \( \lambda \).

\[
1 \leq Oa < Ob \leq v \quad (\sum_{\text{Block}=1}^{b} (M[Oa, \text{Block}] \times M[Ob, \text{Block}])) = \lambda
\]

It is also worth mentioning that the values of both \( b \) and \( v \) can be obtained from \( k, r \) and \( \lambda \) using the formulas:

\[
b = \frac{\lambda \times v \times (v - 1)}{k \times (k - 1)}
\]

\[
r = \frac{\lambda \times (v - 1)}{k - 1}
\]

To break row and column symmetries in the BIBD model we used two Lex constraints, one on each two neighbouring rows (Objects) and similar one for columns (Blocks).

**Lex constraint on Object:**

\[
1 \leq \text{Object} \leq v - 1 \quad [M[\text{Object}, \text{Block}] \mid \text{Block} \in 1..b] \leq_{lex}
\]

\[
[M[\text{Object} + 1, \text{Block}] \mid \text{Block} \in 1..b]
\]

**Lex constraint on Blocks:**

\[
1 \leq \text{Blocks} \leq b - 1 \quad [M[\text{Object}, \text{Block}] \mid \text{Object} \in 1..v] \leq_{lex}
\]

\[
[M[\text{Object}, \text{Block} + 1] \mid \text{Object} \in 1..v]
\]
4.2.1 BIBD Results using Yices2

Similar to the SGP, the results of the BIBD using Yices2 are arranged in four tables, two for timings for the directly translated instances and their corresponding numbers of decisions, and likewise two for the mzn2smt translation. Here the difference in performances are much apparent. From Tables 4.13 and 4.16, ROR’s lead is much evident in both. Constraint reification helped in some cases of Harvey and OR, but its effect was opposite in cases of Alpha and OR CSE, it also seems that using constraint reification along side CSE reduced the performance of both AND and OR. Like in SGP results, here too, the ROR encoding in general had fastest decision rates.

<table>
<thead>
<tr>
<th>Instances</th>
<th>AND</th>
<th>AND</th>
<th>OR</th>
<th>OR</th>
<th>ROR</th>
<th>Alpha</th>
<th>AlphaM</th>
<th>Harvey</th>
<th>No Lex</th>
</tr>
</thead>
<tbody>
<tr>
<td>r, v, λ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7-3-2</td>
<td>0.11</td>
<td>0.13</td>
<td>0.26</td>
<td>0.24</td>
<td>0.11</td>
<td>0.12</td>
<td>0.10</td>
<td>0.18</td>
<td>0.08</td>
</tr>
<tr>
<td>8-4-3</td>
<td>0.39</td>
<td>0.30</td>
<td>0.84</td>
<td>0.80</td>
<td>0.31</td>
<td>0.28</td>
<td>0.29</td>
<td>0.69</td>
<td>0.16</td>
</tr>
<tr>
<td>8-4-4</td>
<td>0.28</td>
<td>0.38</td>
<td>0.41</td>
<td>0.34</td>
<td>0.07</td>
<td>0.23</td>
<td>0.11</td>
<td>299</td>
<td>300</td>
</tr>
<tr>
<td>9-3-1</td>
<td>0.22</td>
<td>0.19</td>
<td>0.45</td>
<td>0.44</td>
<td>0.10</td>
<td>0.18</td>
<td>0.18</td>
<td>0.29</td>
<td>0.04</td>
</tr>
<tr>
<td>11-5-2</td>
<td>1.29</td>
<td>1.27</td>
<td>10.4</td>
<td>4.59</td>
<td>1.48</td>
<td>1.04</td>
<td>1.08</td>
<td>1.91</td>
<td>0.34</td>
</tr>
<tr>
<td>13-3-1</td>
<td>17.1</td>
<td>14.2</td>
<td>31.7</td>
<td>35.6</td>
<td>11.9</td>
<td>16.5</td>
<td>20.3</td>
<td>24.5</td>
<td>9.96</td>
</tr>
<tr>
<td>13-4-1</td>
<td>3.13</td>
<td>3.53</td>
<td>6.86</td>
<td>6.72</td>
<td>2.63</td>
<td>3.88</td>
<td>4.97</td>
<td>2.25</td>
<td>0.29</td>
</tr>
</tbody>
</table>

Arith-mean | 2.23 | 2.87 | 7.28 | 6.96 | 2.38 | 3.18 | 3.86 | 47.12 | 44.41 |
Geo-mean    | 0.79 | 0.78 | 1.90 | 1.64 | 0.57 | 0.71 | 0.67 | 2.73  | 0.76  |

Table 4.13: Average solution time (in seconds) for instances of the BIBD using the direct translation and Yices2 SMT solver

<table>
<thead>
<tr>
<th>Instances</th>
<th>AND</th>
<th>AND</th>
<th>OR</th>
<th>OR</th>
<th>ROR</th>
<th>Alpha</th>
<th>AlphaM</th>
<th>Harvey</th>
<th>No Lex</th>
</tr>
</thead>
<tbody>
<tr>
<td>r, v, λ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7-3-2</td>
<td>1.48</td>
<td>1.45</td>
<td>3.38</td>
<td>3.38</td>
<td>2.76</td>
<td>2.201</td>
<td>2.939</td>
<td>2.693</td>
<td></td>
</tr>
<tr>
<td>8-4-3</td>
<td>2.356</td>
<td>2.410</td>
<td>6.051</td>
<td>5.927</td>
<td>4.071</td>
<td>2.555</td>
<td>2.704</td>
<td>5.314</td>
<td></td>
</tr>
<tr>
<td>8-4-4</td>
<td>1.087</td>
<td>1.232</td>
<td>2.310</td>
<td>2.203</td>
<td>577</td>
<td>956</td>
<td>289</td>
<td>233,823</td>
<td></td>
</tr>
<tr>
<td>9-3-1</td>
<td>1.532</td>
<td>1.639</td>
<td>4.641</td>
<td>4.330</td>
<td>2.822</td>
<td>2.016</td>
<td>1.987</td>
<td>3.293</td>
<td></td>
</tr>
<tr>
<td>13-3-1</td>
<td>25.950</td>
<td>25.316</td>
<td>84.718</td>
<td>81.233</td>
<td>44.060</td>
<td>30.539</td>
<td>33.849</td>
<td>68.440</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.14: Average number of decisions for instances of the BIBD using the direct translation and Yices2 SMT solver
Table 4.15: Average number of decisions per millisecond for instances of the BIBD using the direct translation and Yices2 SMT solver

<table>
<thead>
<tr>
<th>Instances</th>
<th>AND CSE</th>
<th>OR CSE</th>
<th>ROR Alpha</th>
<th>AlphaM Harvey</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-3-2</td>
<td>13</td>
<td>11</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>8-4-3</td>
<td>6</td>
<td>8</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8-4-4</td>
<td>4</td>
<td>3</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>9-3-1</td>
<td>7</td>
<td>9</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>11-5-2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>13-3-1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>13-4-1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 4.16: Average solution time (in seconds) for instances of the BIBD using mzn2smt translation and Yices2 SMT solver

<table>
<thead>
<tr>
<th>Instances</th>
<th>AND CSE</th>
<th>OR CSE</th>
<th>ROR Alpha</th>
<th>AlphaM Harvey</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-3-2</td>
<td>1,456</td>
<td>1,450</td>
<td>3,784</td>
<td>3,812</td>
</tr>
<tr>
<td>8-4-3</td>
<td>2,407</td>
<td>2,182</td>
<td>5,077</td>
<td>5,806</td>
</tr>
<tr>
<td>8-4-4</td>
<td>864</td>
<td>1,066</td>
<td>2,335</td>
<td>2,320</td>
</tr>
<tr>
<td>9-3-1</td>
<td>1,573</td>
<td>1,520</td>
<td>4,007</td>
<td>4,571</td>
</tr>
<tr>
<td>11-5-2</td>
<td>2,590</td>
<td>2,926</td>
<td>8,426</td>
<td>9,044</td>
</tr>
<tr>
<td>13-3-1</td>
<td>24,013</td>
<td>26,956</td>
<td>70,828</td>
<td>83,951</td>
</tr>
<tr>
<td>13-4-1</td>
<td>4,398</td>
<td>4,960</td>
<td>14,919</td>
<td>25,726</td>
</tr>
</tbody>
</table>

Table 4.17: Average number of decisions for instances of the BIBD using mzn2smt translation and Yices2 SMT solver

<table>
<thead>
<tr>
<th>Instances</th>
<th>AND CSE</th>
<th>OR CSE</th>
<th>ROR Alpha</th>
<th>AlphaM Harvey</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-3-2</td>
<td>1,456</td>
<td>1,450</td>
<td>3,784</td>
<td>3,812</td>
</tr>
<tr>
<td>8-4-3</td>
<td>2,407</td>
<td>2,182</td>
<td>5,077</td>
<td>5,806</td>
</tr>
<tr>
<td>8-4-4</td>
<td>864</td>
<td>1,066</td>
<td>2,335</td>
<td>2,320</td>
</tr>
<tr>
<td>9-3-1</td>
<td>1,573</td>
<td>1,520</td>
<td>4,007</td>
<td>4,571</td>
</tr>
<tr>
<td>11-5-2</td>
<td>2,590</td>
<td>2,926</td>
<td>8,426</td>
<td>9,044</td>
</tr>
<tr>
<td>13-3-1</td>
<td>24,013</td>
<td>26,956</td>
<td>70,828</td>
<td>83,951</td>
</tr>
<tr>
<td>13-4-1</td>
<td>4,398</td>
<td>4,960</td>
<td>14,919</td>
<td>25,726</td>
</tr>
</tbody>
</table>

Table 4.18: Average number of decisions per millisecond for instances of the BIBD using mzn2smt translation and Yices2 SMT solver

<table>
<thead>
<tr>
<th>Instances</th>
<th>AND CSE</th>
<th>OR CSE</th>
<th>ROR Alpha</th>
<th>AlphaM Harvey</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-3-2</td>
<td>1,456</td>
<td>1,450</td>
<td>3,784</td>
<td>3,812</td>
</tr>
<tr>
<td>8-4-3</td>
<td>2,407</td>
<td>2,182</td>
<td>5,077</td>
<td>5,806</td>
</tr>
<tr>
<td>8-4-4</td>
<td>864</td>
<td>1,066</td>
<td>2,335</td>
<td>2,320</td>
</tr>
<tr>
<td>9-3-1</td>
<td>1,573</td>
<td>1,520</td>
<td>4,007</td>
<td>4,571</td>
</tr>
<tr>
<td>11-5-2</td>
<td>2,590</td>
<td>2,926</td>
<td>8,426</td>
<td>9,044</td>
</tr>
<tr>
<td>13-3-1</td>
<td>24,013</td>
<td>26,956</td>
<td>70,828</td>
<td>83,951</td>
</tr>
<tr>
<td>13-4-1</td>
<td>4,398</td>
<td>4,960</td>
<td>14,919</td>
<td>25,726</td>
</tr>
</tbody>
</table>

Table 4.19: Average number of decisions per millisecond for instances of the BIBD using mzn2smt translation and Yices2 SMT solver

41
4.2.2 BIBD Results using Z3

Z3 did not perform well on most of the instances of the BIBD and Lex. Most the results in tables 4.19 and 4.16 are time-outs (>300 seconds). Using Harvey encoding on the directly translated instances, Z3 performed much better than the all the other encodings, though this was not the same on the mzn2smt translated instances. There are no decision tables here because the solver did not provide decisions count for the timed-out results. From the instances 7-3-2, 8-4-3 and 9-3-1 in both tables, constraint reification helped the AND and the OR but had an opposite effect on their CSE variants.

<table>
<thead>
<tr>
<th>Instances</th>
<th>AND</th>
<th>CSE</th>
<th>OR</th>
<th>CSE</th>
<th>ROR</th>
<th>Alpha</th>
<th>AlphaM</th>
<th>Harvey</th>
<th>No Lex</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-3-2</td>
<td>77.5</td>
<td>46.6</td>
<td>69.8</td>
<td>21.6</td>
<td>118</td>
<td>104</td>
<td>104</td>
<td>44</td>
<td>2.87</td>
</tr>
<tr>
<td>8-4-3</td>
<td>300</td>
<td>221</td>
<td>264</td>
<td>178</td>
<td>230</td>
<td>250</td>
<td>250</td>
<td>76</td>
<td>5.16</td>
</tr>
<tr>
<td>8-4-4</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>60</td>
<td>4.76</td>
</tr>
<tr>
<td>9-3-1</td>
<td>234</td>
<td>155</td>
<td>236</td>
<td>91.3</td>
<td>157</td>
<td>121</td>
<td>121</td>
<td>60</td>
<td>8.05</td>
</tr>
<tr>
<td>11-5-2</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>2.45</td>
<td>13.26</td>
</tr>
<tr>
<td>13-3-1</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>2.65</td>
<td>174.02</td>
</tr>
<tr>
<td>13-4-1</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>16.25</td>
</tr>
<tr>
<td>Arith-Mean</td>
<td>258</td>
<td>232</td>
<td>252</td>
<td>213</td>
<td>243</td>
<td>239</td>
<td>206</td>
<td>45.7</td>
<td>73.02</td>
</tr>
<tr>
<td>Geo-Mean</td>
<td>238</td>
<td>200</td>
<td>231</td>
<td>161</td>
<td>230</td>
<td>220</td>
<td>161</td>
<td>3.49</td>
<td>17.40</td>
</tr>
</tbody>
</table>

Table 4.19: Average solution time (in seconds) for instances of the BIBD using the direct translation and Z3 SMT solver

<table>
<thead>
<tr>
<th>Instances</th>
<th>AND</th>
<th>CSE</th>
<th>OR</th>
<th>CSE</th>
<th>ROR</th>
<th>Alpha</th>
<th>AlphaM</th>
<th>Harvey</th>
<th>No Lex</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-3-2</td>
<td>43.71</td>
<td>82.62</td>
<td>45.71</td>
<td>51.40</td>
<td>24.13</td>
<td>1364</td>
<td>13.81</td>
<td>52.73</td>
<td>2.87</td>
</tr>
<tr>
<td>8-4-3</td>
<td>262</td>
<td>287</td>
<td>174</td>
<td>223</td>
<td>182</td>
<td>164</td>
<td>117</td>
<td>138</td>
<td>5.16</td>
</tr>
<tr>
<td>8-4-4</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>8.05</td>
</tr>
<tr>
<td>9-3-1</td>
<td>175</td>
<td>191</td>
<td>91.3</td>
<td>103</td>
<td>123</td>
<td>72.2</td>
<td>42.3</td>
<td>35.1</td>
<td>4.76</td>
</tr>
<tr>
<td>11-5-2</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
</tr>
<tr>
<td>13-3-1</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>174.02</td>
</tr>
<tr>
<td>13-4-1</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>16.25</td>
</tr>
<tr>
<td>Arith-Mean</td>
<td>240</td>
<td>251</td>
<td>216</td>
<td>225</td>
<td>218</td>
<td>222</td>
<td>196</td>
<td>203</td>
<td>73.02</td>
</tr>
<tr>
<td>Geo-Mean</td>
<td>207</td>
<td>232</td>
<td>179</td>
<td>192</td>
<td>171</td>
<td>199</td>
<td>127</td>
<td>154</td>
<td>17.40</td>
</tr>
</tbody>
</table>

Table 4.20: Average solution time (in seconds) for instances of The BIBD using the mzn2smt translation and Z3 SMT solver
4.3 Long Vectors Instances

Our plan was to test the inference capabilities of the eight encodings by running unsatisfiable instances of both the SGP and the BIBD problems, but we could not manage to find suitable such instances. Apart from the instance 8-4-4 of the SGP, all the unsatisfiable instances we tested for both problems were either very easy and solved in a negligible time, or very hard which made them impractical to use in the benchmarks. So we made our own unsatisfiable instances as a simple model that enforces the Lex constraint on two vectors, $A$ and $B$, of a length $n$, both have the same integer domain of values of 1 to 4. We made the model unsatisfiable by making Lex fail at the last two items of both vectors, as shown in (4.2) and (4.3) in the following model.

\begin{align*}
A \leq_{lex} B \\
1 \leq i \leq n - 1 & \quad A[i] = 4 \\
\end{align*}

4.3.1 Long Vectors Results

Results for the Long Vectors tests are arranged in four tables (4.21, 4.22, 4.23 and 4.24) representing results for two SMT solvers, Yices2 and Z3, using the mzn2smt and the direct translations. Looking at the results in general, it is clear that the increasing formula size negatively affected the performance in the cases of AND, OR and ANDOR, and caused the Out-of-Memory problem. Common sub-expression elimination greatly helped in reducing formula size thus eliminated the Out-of-Memory problem, as it is clear in tables 4.25 and 4.26. Apart from AND CSE in Yices2 results, constraint reification had unfavourable effect on all encodings, and magnified the Out-of-Memory problem in AND, OR and ANDOR because of the additional variables the constraint reification introduces. The source of Out-of-Memory problem was the mzn2fzn tool that we used in our mzn2smt translation pipeline.
### Table 4.21: Average solution time (in seconds) for unsat instances of $Lex$ between two vectors using the direct translation and Yices2 SMT solver

<table>
<thead>
<tr>
<th>Vector Size</th>
<th>AND CSE</th>
<th>AND CSE</th>
<th>OR CSE</th>
<th>OR CSE</th>
<th>ROR</th>
<th>Alpha</th>
<th>AlphaM</th>
<th>Harvey</th>
<th>ANDOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>0.29</td>
<td>0.13</td>
<td>0.25</td>
<td>0.09</td>
<td>0.01</td>
<td>0.02</td>
<td><strong>0.01</strong></td>
<td>0.02</td>
<td>0.49</td>
</tr>
<tr>
<td>1000</td>
<td>1.30</td>
<td>0.60</td>
<td>1.02</td>
<td>0.35</td>
<td>0.02</td>
<td>0.03</td>
<td><strong>0.02</strong></td>
<td>0.04</td>
<td>2.07</td>
</tr>
<tr>
<td>2000</td>
<td>5.99</td>
<td>3.27</td>
<td>4.11</td>
<td>1.40</td>
<td>0.05</td>
<td>0.07</td>
<td><strong>0.04</strong></td>
<td>0.08</td>
<td>8.40</td>
</tr>
<tr>
<td>2500</td>
<td>9.98</td>
<td>5.69</td>
<td>6.50</td>
<td>2.16</td>
<td><strong>0.05</strong></td>
<td>0.08</td>
<td><strong>0.05</strong></td>
<td>0.09</td>
<td>15.91</td>
</tr>
<tr>
<td>3000</td>
<td>15.40</td>
<td>9.25</td>
<td>9.25</td>
<td>3.08</td>
<td><strong>0.07</strong></td>
<td>0.10</td>
<td><strong>0.07</strong></td>
<td>0.12</td>
<td>20.80</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>6.59</td>
<td>3.79</td>
<td>4.23</td>
<td>1.42</td>
<td><strong>0.04</strong></td>
<td>0.06</td>
<td><strong>0.04</strong></td>
<td>0.07</td>
<td>9.54</td>
</tr>
</tbody>
</table>

### Table 4.22: Average solution time (in seconds) for unsat instances of $Lex$ between two vectors using mzn2smt translation and Yices2 SMT solver

<table>
<thead>
<tr>
<th>Vector Size</th>
<th>AND CSE</th>
<th>AND CSE</th>
<th>OR CSE</th>
<th>OR CSE</th>
<th>ROR</th>
<th>Alpha</th>
<th>AlphaM</th>
<th>Harvey</th>
<th>ANDOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>OOM</td>
<td>0.09</td>
<td>0.26</td>
<td>0.12</td>
<td>0.11</td>
<td>0.10</td>
<td>0.08</td>
<td><strong>0.02</strong></td>
<td>0.33</td>
</tr>
<tr>
<td>1000</td>
<td>OOM</td>
<td>0.32</td>
<td>OOM</td>
<td>0.48</td>
<td>0.42</td>
<td>0.36</td>
<td>0.30</td>
<td><strong>0.04</strong></td>
<td>OOM</td>
</tr>
<tr>
<td>2000</td>
<td>OOM</td>
<td>1.26</td>
<td>OOM</td>
<td>2.03</td>
<td>1.73</td>
<td>1.71</td>
<td>1.19</td>
<td><strong>0.07</strong></td>
<td>OOM</td>
</tr>
<tr>
<td>2500</td>
<td>OOM</td>
<td>1.99</td>
<td>OOM</td>
<td>3.44</td>
<td>2.94</td>
<td>3.07</td>
<td>2.18</td>
<td><strong>0.09</strong></td>
<td>OOM</td>
</tr>
<tr>
<td>3000</td>
<td>OOM</td>
<td>3.00</td>
<td>OOM</td>
<td>5.89</td>
<td>4.73</td>
<td>5.09</td>
<td>3.09</td>
<td><strong>0.11</strong></td>
<td>OOM</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>1.33</td>
<td>2.39</td>
<td>1.99</td>
<td>2.06</td>
<td>1.37</td>
<td><strong>0.07</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 4.23: Average solution time (in seconds) for unsat instances of $Lex$ between two vectors using the direct translation and Z3 SMT solver

<table>
<thead>
<tr>
<th>Vector Size</th>
<th>AND CSE</th>
<th>AND CSE</th>
<th>OR CSE</th>
<th>OR CSE</th>
<th>ROR</th>
<th>Alpha</th>
<th>AlphaM</th>
<th>Harvey</th>
<th>ANDOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>0.53</td>
<td>0.14</td>
<td>0.34</td>
<td><strong>0.05</strong></td>
<td>0.10</td>
<td>0.33</td>
<td><strong>0.05</strong></td>
<td>0.11</td>
<td>0.63</td>
</tr>
<tr>
<td>1000</td>
<td>2.03</td>
<td>0.42</td>
<td>1.35</td>
<td><strong>0.09</strong></td>
<td>0.16</td>
<td>1.26</td>
<td>0.10</td>
<td>0.21</td>
<td>2.62</td>
</tr>
<tr>
<td>2000</td>
<td>7.68</td>
<td>1.40</td>
<td>4.73</td>
<td><strong>0.14</strong></td>
<td>0.21</td>
<td>4.19</td>
<td>0.16</td>
<td>0.38</td>
<td>9.27</td>
</tr>
<tr>
<td>2500</td>
<td>12.46</td>
<td>2.41</td>
<td>8.70</td>
<td><strong>0.20</strong></td>
<td>0.28</td>
<td>7.95</td>
<td>0.23</td>
<td>0.54</td>
<td>18.18</td>
</tr>
<tr>
<td>3000</td>
<td>31.39</td>
<td>3.57</td>
<td>13.34</td>
<td><strong>0.24</strong></td>
<td>0.33</td>
<td>11.87</td>
<td>0.27</td>
<td>0.64</td>
<td>27.86</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>10.82</td>
<td>1.59</td>
<td>5.69</td>
<td><strong>0.14</strong></td>
<td>0.21</td>
<td>5.12</td>
<td>0.16</td>
<td>0.38</td>
<td>11.71</td>
</tr>
</tbody>
</table>

### Table 4.24: Average solution time (in seconds) for unsat instances of $Lex$ between two vectors using mzn2smt translation and Z3 SMT solver

<table>
<thead>
<tr>
<th>Vector Size</th>
<th>AND CSE</th>
<th>AND CSE</th>
<th>OR CSE</th>
<th>OR CSE</th>
<th>ROR</th>
<th>Alpha</th>
<th>AlphaM</th>
<th>Harvey</th>
<th>ANDOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>0.35</td>
<td>0.17</td>
<td>0.21</td>
<td>0.08</td>
<td>0.27</td>
<td>0.39</td>
<td><strong>0.07</strong></td>
<td>0.12</td>
<td>0.42</td>
</tr>
<tr>
<td>1000</td>
<td>OOM</td>
<td>0.58</td>
<td>OOM</td>
<td>0.15</td>
<td>0.99</td>
<td>1.44</td>
<td><strong>0.13</strong></td>
<td>0.23</td>
<td>OOM</td>
</tr>
<tr>
<td>2000</td>
<td>OOM</td>
<td>2.67</td>
<td>OOM</td>
<td>0.34</td>
<td>4.10</td>
<td>5.98</td>
<td><strong>0.26</strong></td>
<td>0.50</td>
<td>OOM</td>
</tr>
<tr>
<td>2500</td>
<td>OOM</td>
<td>4.56</td>
<td>OOM</td>
<td>0.49</td>
<td>6.57</td>
<td>10.13</td>
<td><strong>0.32</strong></td>
<td>0.60</td>
<td>OOM</td>
</tr>
<tr>
<td>3000</td>
<td>OOM</td>
<td>17.39</td>
<td>OOM</td>
<td>0.63</td>
<td>9.55</td>
<td>14.21</td>
<td><strong>0.39</strong></td>
<td>0.74</td>
<td>OOM</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>5.07</td>
<td>0.34</td>
<td>4.39</td>
<td>6.43</td>
<td><strong>0.24</strong></td>
<td>4.43</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.3.2 Instances and Encoding Size

From the Long Vectors results, it is clear that formula size played a major role in affecting the performance of both solvers on some instances, this becomes more apparent with each increment in instances size. The smallest and with the best results is the formula that produced by the directly translated ROR encoding. Though this does not affect all encodings by the same factor. It seems that the AND encoding is more sensitive to formula size than the others, for instance a directly translated OR encoding performs twice as better as a formula of the same size of a directly translated AND. This is interesting because the AND encoding is closer to the natural SAT’s CNF than the OR which is basically a Disjunctive Normal Form (DNF). Both the AND and the OR have a quadratic formula growth.

<table>
<thead>
<tr>
<th>Vector Size</th>
<th>AND Size</th>
<th>AND CSE</th>
<th>OR Size</th>
<th>OR CSE</th>
<th>ROR Size</th>
<th>ROR CSE</th>
<th>Alpha Size</th>
<th>AlphaM</th>
<th>Harvey Size</th>
<th>ANDOR Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>125</td>
<td>2</td>
<td>125.75</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>251</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>500</td>
<td>5</td>
<td>501.50</td>
<td>5</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>1002</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>2001</td>
<td>10</td>
<td>2003.00</td>
<td>10</td>
<td>17</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>4004</td>
<td></td>
</tr>
<tr>
<td>2500</td>
<td>3126</td>
<td>12</td>
<td>3128.75</td>
<td>12</td>
<td>22</td>
<td>10</td>
<td>5</td>
<td>5</td>
<td>6255</td>
<td></td>
</tr>
<tr>
<td>3000</td>
<td>4501</td>
<td>15</td>
<td>4504.50</td>
<td>15</td>
<td>26</td>
<td>12</td>
<td>6</td>
<td>6</td>
<td>9006</td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>2051</td>
<td>9</td>
<td>2053</td>
<td>9</td>
<td>16</td>
<td>7</td>
<td>4</td>
<td>4</td>
<td>4104</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.25: Number of occurrences of atoms (divided by 1000) for the instances of $Lex$ between two vectors using the direct translation
4.4 BIBD All Solutions Benchmarks

After testing the encodings on large instances of two vectors, we decided to run further tests on hard instances of a problem. We choose to do this by trying to find all solutions for instances of the BIBD. This done by modifying our benchmarking script to run the Yices2 to solve an instance in a loop, and after each run that results in a solution the script feeds back the inverted solution to the instance to rule out that solution, this continues till the solver gives unsat, which means that all solutions has been found.

Tables 4.27 and 4.30 show the timing for those instances where Yices2 was able to find all solutions without timing-out. On the instance 13-3-1 Yices2 was unable to find all solutions before the time-out (300 seconds), so instead of the timings, we reported number of solutions found using each encoding in tables 4.28 and 4.31. As expected, without symmetry breaking finding all solutions would take longer time in most cases. Here, table 4.29 shows number of solutions found by Yices2 in around 300 seconds.

The results reflect close performances between AND, AND CSE, Alpha and AlphaM, with a marginal lead for AND in some cases. Apart form the cases of OR and ROR, the mzn2smt translation didn’t improve performance compared to the direct translation. Table 4.29 reflects the variation of difficulty levels between the instances we used.
<table>
<thead>
<tr>
<th>Instances</th>
<th>AND</th>
<th>AND</th>
<th>OR</th>
<th>OR</th>
<th>ROR</th>
<th>Alpha</th>
<th>AlphaM</th>
<th>Harvey</th>
</tr>
</thead>
<tbody>
<tr>
<td>r, v, λ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7-3-2</td>
<td>6.32</td>
<td>6.24</td>
<td>10.07</td>
<td>9.49</td>
<td>10.21</td>
<td>6.24</td>
<td>7.91</td>
<td>6.53</td>
</tr>
<tr>
<td>8-4-3</td>
<td>103.24</td>
<td>113.87</td>
<td>153.62</td>
<td>149.29</td>
<td>157.15</td>
<td>105.22</td>
<td>105.24</td>
<td>155.31</td>
</tr>
<tr>
<td>9-3-1</td>
<td>2.68</td>
<td>2.15</td>
<td>3.78</td>
<td>3.98</td>
<td>3.41</td>
<td>2.50</td>
<td>2.31</td>
<td>2.58</td>
</tr>
<tr>
<td>11-5-2</td>
<td>1.29</td>
<td>1.27</td>
<td>10.40</td>
<td>4.59</td>
<td>1.48</td>
<td>1.04</td>
<td>1.08</td>
<td>1.91</td>
</tr>
<tr>
<td>13-4-1</td>
<td>26.21</td>
<td>34.38</td>
<td>88.41</td>
<td>53.47</td>
<td>52.05</td>
<td>34.86</td>
<td>30.18</td>
<td>52.60</td>
</tr>
</tbody>
</table>

Arith-Mean | 27.95 | 31.58 | 53.26 | 44.16 | 44.86 | 29.97 | 29.34 | 43.79 |

Table 4.27: Average solution time (in seconds) for finding all solutions for instances of the BIBD using the direct translation and Yices2 SMT solver

<table>
<thead>
<tr>
<th>Instances</th>
<th>AND</th>
<th>AND</th>
<th>OR</th>
<th>OR</th>
<th>ROR</th>
<th>Alpha</th>
<th>AlphaM</th>
<th>Harvey</th>
</tr>
</thead>
<tbody>
<tr>
<td>r, v, λ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Solutions in 300s for 13-3-1</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>13</td>
<td>15</td>
<td>17</td>
<td>15</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 4.28: Average number of solutions found in 300 seconds for the instance 13-3-1 of the BIBD using the direct translation and Yices2 SMT solver

<table>
<thead>
<tr>
<th>Encoding</th>
<th>7-3-2</th>
<th>8-4-3</th>
<th>9-3-1</th>
<th>11-5-2</th>
<th>13-3-1</th>
<th>13-4-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Lex</td>
<td>641</td>
<td>309</td>
<td>426</td>
<td>148</td>
<td>14</td>
<td>62</td>
</tr>
</tbody>
</table>

Table 4.29: Number of solutions found in 300 seconds for instances of the BIBD without Lex constraint and using Yices2 SMT solver

<table>
<thead>
<tr>
<th>Instances</th>
<th>AND</th>
<th>AND</th>
<th>OR</th>
<th>OR</th>
<th>ROR</th>
<th>Alpha</th>
<th>AlphaM</th>
<th>Harvey</th>
</tr>
</thead>
<tbody>
<tr>
<td>r, v, λ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7-3-2</td>
<td>8.07</td>
<td>9.27</td>
<td>12.57</td>
<td>11.71</td>
<td>12.23</td>
<td>10.87</td>
<td>11.70</td>
<td>8.54</td>
</tr>
<tr>
<td>8-4-3</td>
<td>153.13</td>
<td>147.29</td>
<td>146.95</td>
<td>181.68</td>
<td>152.31</td>
<td>169.34</td>
<td>200.76</td>
<td>168.04</td>
</tr>
<tr>
<td>9-3-1</td>
<td>3.82</td>
<td>2.99</td>
<td>4.32</td>
<td>4.36</td>
<td>2.60</td>
<td>3.79</td>
<td>4.25</td>
<td>3.21</td>
</tr>
<tr>
<td>11-5-2</td>
<td>1.12</td>
<td>1.38</td>
<td>3.72</td>
<td>3.73</td>
<td>1.28</td>
<td>1.33</td>
<td>1.51</td>
<td>7.52</td>
</tr>
<tr>
<td>13-4-1</td>
<td>28.67</td>
<td>72.94</td>
<td>58.85</td>
<td>86.18</td>
<td>38.20</td>
<td>56.74</td>
<td>31.33</td>
<td>81.09</td>
</tr>
</tbody>
</table>

Arith-Mean | 38.96 | 46.77 | 45.28 | 57.53 | 41.32 | 48.41 | 49.91 | 53.68 |
Geo-Mean   | 10.87 | 13.27 | 17.72 | 19.72 | 11.88 | 13.94 | 13.65 | 19.49 |

Table 4.30: Average solution time (in seconds) for finding all solutions for instances of the BIBD using the mzn2smt translation and Yices2 SMT solver

<table>
<thead>
<tr>
<th>Instances</th>
<th>AND</th>
<th>AND</th>
<th>OR</th>
<th>OR</th>
<th>ROR</th>
<th>Alpha</th>
<th>AlphaM</th>
<th>Harvey</th>
</tr>
</thead>
<tbody>
<tr>
<td>r, v, λ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Solutions in 300s for 13-3-1</td>
<td>14</td>
<td>15</td>
<td>14</td>
<td>15</td>
<td>14</td>
<td>14</td>
<td>13</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.31: Average number of solutions found in 300 seconds for the instances 13-3-1 of the BIBD using the mzn2smt translation and Yices2 SMT solver
Chapter 5

Evaluation and Conclusions

Contrary to expectations, the results show that when searching for a single solution, not using any of the eight Lex encodings (No Lex) is better on most of the instances of the SGP and the BIBD we used. But still, using Lex would perform better in solving unsatisfiable instances or when all possible solutions were required as we saw in all solutions tests. In many cases, it is difficult to know whether an instance of a problem is satisfiable or not, this makes it difficult to have prior knowledge of the instances which would perform better without symmetry breaking using Lex. For that, we decided to evaluate the encodings by the differences their performances ignoring the performance of the No Lex.

The results, in general, show that the Recursive OR encoding has the best results on most of the tables. However, all the encodings preformed differently on different instances of the same problem, and even sometimes on different samples of the same instance. For example, on the SGP, the Recursive OR encoding did very well on 7 out of the 10 instances, but it was 100 times worse than its nearest competitor on one of those instance of the problem, it is also did not perform as well in all solutions test. This behaviour promotes the idea of using special algorithms for Lex [6] instead of the decomposition to tackle symmetry. Also, apart from the solving time, both SMT solvers did not produces statistics for most of the unsat long vectors instances we used, that is why there are no tables for number of decisions for the long vectors instances. The improvement in performance in the case of Harvey encoding when using the mzn2smt translation and Yices, supports Bofill’s observation [12] that more logical component over theory in a formula helps to improve solving time, though this true only in case of Yices2 but not with Z3 as it is clear from tables 4.19 and 4.20, this implies that this behaviour is dependant on the solving algorithm. The Out-of-Memory problem in the results demonstrated the limitations that
the AND, OR and the ANDOR encodings have when dealing with relativity sizeable problems. Finally, in the majority of cases, the performance of the directly translated instances was better than those ones which went through Bonfill’s fzn2smt translator.

5.1 Future Work

Without closely examining how SMT solvers approach the solving process, it is hard to get enough explanations from the results alone. A future line of research could be to study how SMT solvers internally handle each encoding, this can be done by examining the code of some the open source SMT solvers and by building a framework that monitors a solver during runtime. Alongside this, doing more benchmarks using different CSP problems and SMT solvers could help to find clearer patterns from the results.
# Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCP</td>
<td>Boolean Constraints Propagation</td>
</tr>
<tr>
<td>BIBD</td>
<td>Balanced Incomplete Block Design</td>
</tr>
<tr>
<td>CDCL</td>
<td>Conict Driven Clause Learning</td>
</tr>
<tr>
<td>CNF</td>
<td>Conjunctive Normal Form</td>
</tr>
<tr>
<td>CSE</td>
<td>Common Sub-expression Elimination</td>
</tr>
<tr>
<td>CSP</td>
<td>Constraint Satisfaction Problems</td>
</tr>
<tr>
<td>DNF</td>
<td>Disjunctive Normal Form</td>
</tr>
<tr>
<td>DPLL</td>
<td>Davis, Putnam, Logemann and Loveland</td>
</tr>
<tr>
<td>GAC</td>
<td>Generalised Arc Consistency</td>
</tr>
<tr>
<td>ROR</td>
<td>Recursive OR</td>
</tr>
<tr>
<td>SB</td>
<td>Symmetry Braking</td>
</tr>
<tr>
<td>SGP</td>
<td>Social Golfer Problem</td>
</tr>
</tbody>
</table>
References


[23] Pierre Flener, Alan M Frisch, Brahim Hnich, Zeynep Kiziltan, Ian Miguel, Justin Pearson, and Toby Walsh. Breaking row and column symmetries in matrix mod-


53


Index

applications, 1, 7, 9, 10, 13, 38

Boolean constraints, 1, 8, 10

Boolean formulas, 1

Boolean variable, 5, 6, 8

Boolean variables, 5

CNF, 1, 4, 6–9, 11, 12, 19, 45

Conjunctive Normal Form, 1, 6

constraint reification, 29, 33, 35, 40, 42, 43

Constraint Satisfaction Problems, 2, 12

constraints, 1–3, 10, 11, 14–19, 21, 23, 28–31, 38, 39

DPLL, 7, 9, 10

fzn2smt, 2, 3, 18, 25, 29

global constraints, 2, 29

lexicographic ordering, 1–3

mixed symmetry, 14

mzn2fzn, 3, 29, 43

mzn2smt, 18, 19, 21–25, 27–29, 48

optimisation, 11, 14

Out-of-Memory, 30, 43, 48

propositions, 1

SAT, 1, 6, 7, 9–12, 45

Satisfiability, 1, 5, 7, 8, 12, 14

Satisfiability Modulo Theories, 1, 11–13

scheduling, 1, 11, 14, 15

shuffling, 30

SMT, 1–4, 12–14, 18, 19, 25, 48, 49

Social Golfers Problem, 29, 30

symmetry braking, 3, 14

value symmetry, 14

variable symmetry, 14