Breaking Symmetries with Lex Implications*

Michael Codish[1], Thorsten Ehlers[2], Graeme Gange[3],
Avraham Itzhakov[1], and Peter J. Stuckey[3,4]

[1] Department of Computer Science, Ben-Gurion University of the Negev, Israel
[2] Department of Computer Science, Kiel University, Germany
[3] Department of Computing and Information Systems, The University of Melbourne, Australia
[4] Data61 CSIRO, Australia

Abstract. Breaking symmetries is crucial when solving hard combinatorial problems. A common way to eliminate symmetries in CP/SAT is to add symmetry breaking constraints. Ideally, symmetry breaking constraints should be complete and compact. The aim of this paper is to find compact and complete symmetry breaks applicable when solving hard combinatorial problems using CP/SAT approach. In particular: graph search problems and matrix model problems where symmetry breaks are often specified in terms of lex constraints. We show that sets of lex constraints can be expressed with only a small portion of their inner lex implications which are a particular form of Horn clauses. We exploit this fact and compute a compact encoding of the row-wise LexLeader and state of the art partial symmetry breaking constraints. We illustrate the approach for graph search problems and matrix model problems.

1 Introduction

When solving hard combinatorial problems, symmetry breaks play a crucial role. When seeking solutions, the size of the search space is significantly reduced if symmetries are eliminated. The search space can be explored more efficiently when avoiding paths that lead to symmetric solutions and avoiding also those that lead to symmetric non-solutions.

This paper deals with *variable symmetry* in constraint staisfaction problems (CSP) where symmetry is a permutation defined over a set of variables that preserves solutions. Given a CSP with variables $x_1, ..., x_n$, we say that σ is a symmetry if for every assignment $\mu = \{x_1 = i_1, ..., x_n = i_n\}$, μ is a solution if and only if $\{x_1 = i_{\sigma(1)}, ..., x_n = i_{\sigma(n)}\}$ is also a solution.

One common approach to eliminate symmetries is to introduce symmetry breaking constraints [1–4] which rule out isomorphic solutions thus reducing the size of the search space while preserving the set of solutions. Ideally, a symmetry breaking constraint is satisfied by a single member of each equivalence class of

* Supported by the Israel Science Foundation, grant 625/17 and the German Federal Ministry of Education and Research, combined project 01IH15006A.

solutions, in which case it is said to be *complete*. However, computing such symmetry breaking constraints is, most often, intractable [2]. In practice, symmetry breaking constraints are often *partial*, and typically rule out some, but not all of the symmetries in the search. As noted in the survey by Walsh [5], often a few simple constraints rule out most of the symmetries.

In many cases, symmetry breaking constraints, complete or partial, are expressed in terms of lex constraints on the variables of the problem. Each lex constraint, corresponds to one symmetry σ, and restricts the search space to consider assignments which are lexicographically smaller than their permuted form obtained according to σ. Typical examples are: graph search problems [6] where rows and columns of the Boolean adjacency matrix can be reordered by some permutation, and matrix models where rows and columns can be reordered by a pair of permutations. For matrix models common partial symmetry breaks described in terms of lex constraints include doubleLex [7] (denoted also lex^2 in [8]) and snake-lex [9]. For graph search problems, Codish *et al.* [6] introduce partial symmetry breaks (denoted sb_ℓ and sb^*_ℓ) which are refinements of doubleLex for adjacency matrices.

Complete symmetry breaks can be obtained, for both types of problems, by introducing a lex constraint for each reordering of the combinatorial object (graph or matrix) [10]. For matrices, the reordering takes place by permuting rows and columns [7]. For graphs, symmetric solutions can be obtained by permutations of the vertices, which corresponds to simultaneously permuting both rows and columns of the adjacency matrix. However, the number of lex constraints is overwhelming.

The aim of this research is to find compact and complete symmetry breaks applicable when solving hard combinatorial problems. In particular: graph search problems and matrix model problems.

In previous work, Itzhakov and Codish [11] present complete and compact symmetry breaks for graphs based on so-called canonizing sets of permutations where each permutation represents a lex constraint. Their approach is based on the observation that many of the lex constraints expressed in terms of all permutations (of rows and columns of the adjacency matrix) are redundant. Itzhakov and Codish [11] compute compact symmetry breaking constraints for graphs with 10 or less vertices. They observe, for example, that for 10 vertices 7,853 lex constraints suffice to provide a complete symmetry break instead of the $10! = 3{,}628{,}800$ constraints introduced by the definition. Itzhakov and Codish [11] report that this symmetry break takes 4 days to compute.

Heule [12] poses the question: How expensive is it to break all graph symmetries? Heule seeks an answer in terms of the number of clauses in a CNF representation of the corresponding symmetry breaking constraint. For up to $n = 5$ vertices, Heule computes size-optimal compact and complete symmetry breaks. A size-optimal complete symmetry break for graphs with 5 vertices consists of only 12 clauses. In contrast, the symmetry break computed by Itzhakov and Codish consists of 7 lex constraints, which can be encoded in 83 clauses. For $5 < n \leq 8$ vertices, Heule computes complete symmetry breaks which are

significantly smaller than those computed by Itzhakov and Codish but which are not determined to be optimal. For 8 vertices, the complete symmetry break computed by Heule consists of 956 clauses (and takes two days to compute). The complete symmetry break computed by Itzhakov and Codish consists of 135 lex constraints (2724 clauses) and as reported in [11] takes 6 minutes to compute.

Frisch and Harvey illustrate in [13] the redundancies in a complete symmetry break in a three-by-two matrix. They show how to simplify the 11 lex constraints expressing all reorderings of rows and columns. The resulting symmetry break has 8 simplified lex constraints.

In this paper we note the standard decomposition of a lex constraint of the form $x_1 \dots x_n \leq_{lex} y_1 \dots y_n$ to a conjunction of Horn clauses of the form

$$(x_1 = y_1), \dots, (x_k = y_k) \rightarrow x_{k+1} \leq y_{k+1}$$

where the literals on the left side are equalities between the variables in the lex constraint and $1 \leq k < n$ [14]. We call clauses of this form *lex implications*. We observe that many of the lex implications in the decomposition of the complete symmetry breaks derived in [11] are redundant. This enables to significantly reduce the size of the symmetry breaking constraints. For example, for $n = 10$ vertices, a complete symmetry break is obtained in [11] with 7,853 lex constraints. These decompose to 248,604 lex implications which can be reduced (removing implications implied by the others) to a complete symmetry break expressed using only 21,844 lex implications. For 5 vertices, the complete symmetry break computed in [11] involves 7 lex constraints which decompose to 41 lex implications. These can be reduced to 14 non-redundant lex implications and 40 clauses, cf. Example 3. For 8 vertices, the complete symmetry break computed in [11] involves 135 lex constraints which decompose to 2006 lex implications (2724 clauses). These can be reduced to 387 non-redundant lex implications (1077 clauses), c.f. Table 2.

Given the observation that so many lex implications are redundant, we then pose the direct question: how many lex implications are required to express a complete symmetry break on a graph with n nodes. We generate such symmetry breaks directly and not by reducing the complete symmetry breaks presented in [11]. A compact and complete symmetry break for graphs with 8 vertices can be computed in about 1 minute. A compact and complete symmetry break for 10 vertices can now be computed in roughly 3 hours (in contrast to 4 days as reported in [11]. Moreover, we compute a compact and complete symmetry break for graphs with 11 vertices. This symmetry break consists in 274,109 lex implications (280,049 clauses) and is computed in 8 days.

The technique of representing conjunctions of lex constraints in terms of non-redundant lex implications works also for matrix models where symmetry breaks are also defined in terms of lex constraints (swapping rows and columns) and also for partial symmetry breaks for graphs and matrix models.

We analyze the standard doubleLex constraint and observe that it has no redundancies when represented as lex implications. However when using extensions which we call lex^+ and lex^* (also known as swapNext and swapAny re-

spectively [15]) there are many redundancies. We give compact versions of these (without the redundancies).

The rest of the paper is organized as follows. In the next section we introduce notation and basic concepts. In Section 3 we illustrate the effect of removing redundancy in encoding symmetry breaking constraints. In Section 4 we define our approach to efficiently generating compact and complete symmetry breaking constraints, and illustrate the effectiveness on graphs. Finally in Section 6 we conclude.

The computations throughout the paper are performed using the finite-domain constraint compiler BEE [16] which compiles constraints to CNF, and solves it applying an underlying SAT solver. We use Glucose 4.0 [17] as the underlying SAT solver except where specified that we used Clasp 3.1.3 [18]. All computations were performed on an Intel E8400 core, clocked at 2 GHz, able to run a total of 12 parallel threads. Each of the cores in the cluster has computational power comparable to a core on a standard desktop computer. Each SAT instance is run on a single thread, and all running times reported in this paper are CPU times.

2 Preliminaries

In this paper, we consider Boolean formulas φ which encode the set of solutions of combinatorial problems. In many cases, there are variable symmetries in the solution space of such problems. This is, given a solution $x_i = v_i$ there often exist permutations π such that $x_{\pi(i)} = v_i$ is an isomorphic solution [4].

When enumerating the solutions for a particular problem, it is often preferable to consider only non-isomorphic solutions. Furthermore, if one wishes to prove that no solutions for some problem exist, breaking symmetries often allows for smaller proofs.

In order to decrease the size of the solution space, one approach is to use constraints ψ which break symmetries in the solution space of φ. This is, for every solution x which satisfies φ, there exists an isomorphic solution x' which satisfies $\varphi \wedge \psi$. Symmetry breaking constraints which rule out all but one solution from each equivalence class are called complete. Symmetry breaking constraints which rule out some but not all isomorphic solutions from an equivalence class are called partial [2].

We consider finite and simple graphs (undirected with no self loops). The set of simple graphs on n nodes is denoted $\mathcal{G}_n$. We assume that the vertex set of a graph, $G = (V, E)$, is $V = \{1, \ldots, n\}$ and represent G by its $n \times n$ Boolean adjacency matrix. We often let G denote both the graph itself and also its adjacency matrix.

Graph search problems are about the search for a graph which satisfies a given set of constraints, φ, or to determine that no such graph exists. In this setting the unknown graph is represented by an adjacency matrix consisting of Boolean variables which are constrained by φ. Often graph search problems

$$A = \begin{bmatrix} 0 & a & b & c & d \\ a & 0 & e & f & g \\ b & e & 0 & h & i \\ c & f & h & 0 & j \\ d & g & i & j & 0 \end{bmatrix} \qquad \pi(A) = \begin{bmatrix} 0 & a & e & f & g \\ a & 0 & b & c & d \\ e & b & 0 & h & i \\ f & c & h & 0 & j \\ g & d & i & j & 0 \end{bmatrix}$$

Fig. 1. The 5×5 Boolean adjacency matrix G and $\pi(G)$ for $\pi = (2, 1, 3, 4, 5)$.

are about the search for the set of all graphs, modulo graph isomorphism, that satisfy the given constraints.

The set of permutations $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ is denoted S_n. Permutations act on adjacency matrices in the natural way: If A is the adjacency matrix of a graph G and π is a permutation, then $\pi(A)$ is the adjacency matrix obtained by simultaneously permuting with π the rows and columns of A.

Two graphs $G_1, G_2 \in \mathcal{G}_n$ are isomorphic, denoted $G_1 \approx G_2$, if there exists a permutation $\pi \in S_n$ such that $G_1 = \pi(G_2)$. Sometimes we write $G_1 \approx_\pi G_2$ to emphasize that π is the permutation such that $G_1 = \pi(G_2)$. For sets of graphs H_1, H_2, we say that $H_1 \approx H_2$ if for every $G_1 \in H_1$ (likewise in H_2) there exists $G_2 \in H_2$ (likewise in H_1) such that $G_1 \approx G_2$.

We consider an ordering on graphs, defined by representing their adjacency matrices as strings. Because adjacency matrices are symmetric with zeroes on the diagonal, it suffices to focus on the upper triangle parts of the matrices [19].

Definition 1 (ordering graphs). *Let $G_1, G_2 \in \mathcal{G}_n$ and let s_1, s_2 be the strings obtained by concatenating the rows of the upper triangular parts of their corresponding adjacency matrices A_1, A_2 respectively. Then, $G_1 \preceq G_2$ if and only if $s_1 \preceq_{lex} s_2$. We also write $A_1 \preceq A_2$.*

A classic complete symmetry break for graphs is the LexLeader constraint [10] defined as follows:

Definition 2 (LexLeader). $\textsc{LexLeader}(n) = \bigwedge \{\ G \preceq \pi(G) \mid \pi \in S_n\ \}$ *where G is an $n \times n$ matrix of Boolean variables with 0's on the diagonal and such that $G_{ij} = G_{ji}$ for all $1 \leq i < j \leq n$. Sometimes we write $\textsc{LexLeader}_G(n)$ to make G explicit.*

Example 1 *Consider graphs on 5 nodes. Figure 1 depicts the adjacency matrix A of such graphs, where $a \ldots j$ denote Boolean variables (on the left side). For the permutation $\pi = (2, 1, 3, 4, 5)$, $\pi(A)$, is detailed on the right side of Figure 1. A complete symmetry break can be created by using the* LexLeader *constraint, which requires $5! = 120$ lex constraints, one for each permutation in S_5.*

The constraint $G \preceq \pi(G)$ is expressed as the lex constraint $abcdefghij \preceq_{lex} aefgbcdhij$, which can be simplified to $bcd \preceq_{lex} efg$.

In fact, it is sufficient to consider only some of the LexLeader constraints. In [11], a refinement procedure was used which adds non-redundant lex constraints until a complete symmetry break has been reached.

$$
\begin{array}{rr}
\pi_1 = (5,3,4,2,1) & abcefgi \preceq_{lex} ijghebc \\
\pi_2 = (2,1,5,3,4) & bcdefhi \preceq_{lex} gefdbij \\
\pi_3 = (1,3,2,4,5) & afg \preceq_{lex} bhi \\
\pi_4 = (1,2,4,3,5) & bei \preceq_{lex} cfj \\
\pi_5 = (2,4,1,5,3) & abcdefgi \preceq_{lex} fagecjhb \\
\pi_6 = (2,3,1,5,4) & abcdfgh \preceq_{lex} eagfihd \\
\pi_7 = (1,2,5,3,4) & bcefhi \preceq_{lex} dbgeij
\end{array}
$$

Fig. 2. A complete symmetry break for graphs with 5 nodes expressed in terms of 7 lex constraints derived from corresponding permutations.

$$
\begin{array}{lll}
a \leq b & (T_{a=h}) \Rightarrow c \leq i & (T_{b=c} \wedge T_{e=f}) \Rightarrow i \leq j \\
b \leq c & (T_{b=f}) \Rightarrow c \leq e & (T_{a=b} \wedge T_{f=h}) \Rightarrow g \leq i \\
c \leq d & (T_{c=d}) \Rightarrow f \leq g & (T_{b=g} \wedge T_{c=e}) \Rightarrow d \leq f \\
b \leq f & (T_{a=b}) \Rightarrow f \leq h & (T_{b=f} \wedge T_{c=e} \wedge T_{d=g}) \Rightarrow i \leq j \\
 & (T_{b=c}) \Rightarrow e \leq f & (T_{b=e} \wedge T_{c=g} \wedge T_{d=f}) \Rightarrow h \leq i
\end{array}
$$

Fig. 3. A complete symmetry break for graphs with 5 nodes expressed in terms of 14 lex implications. These clauses were computed directly, they are not derived from the lex constraints presented in Figure 2.

Example 2 *A complete symmetry break for graph search problems on* 5 *nodes can be expressed in terms of the 7 permutations detailed in Figure 2 (on the left) which give rise to the corresponding lex constraints (on the right). All of the* $5! = 120$ *lex constraints used by the* LexLeader*(5) constraint are implied by these 7 lex constraints.*

It is well known that lex constraints can be decomposed into lex implications. Using Tseytin variables $T_{x=y} \leftrightarrow x = y$ and replacing $a \leq b$ by $(\neg a \vee b)$, these are Horn clauses. The question we ask in this paper is: How many lex implications are required to represent a complete symmetry break on graphs?

Example 3 *In order to break all symmetries on graphs with* 5 *nodes, it is sufficient to consider the* 14 *lex implications depicted in Figure 3. As the Tseytin variables only occur on the left-hand side of the implications, it is sufficient to encode them using two clauses as in* $(x \wedge y) \Rightarrow T_{x=y}$ *and* $(\neg x \wedge \neg y) \Rightarrow T_{x=y}$*[20]. Thus, this symmetry break can be encoded using* 40 *clauses.*

In some cases, the redundancy of lex implications can be seen directly. The lex constraint $abcefgi \preceq_{lex} ijghebc$ *from Example 2 implies* $a \leq i$*, which is redundant with respect to the lex implications from Figure 3: If* $a = 1$*, this implies* $b{=}c{=}d{=}f{=}1$ *by the inequalities on the left-hand side. This again implies* $h = 1$ *by the lex implication* $(T_{a=b}) \Rightarrow f \leq h$*, and* $i = 1$ *by* $(T_{a=h}) \Rightarrow c \leq i$*. Checking the redundancy of other lex implications often requires a case analysis.*

In this paper we consider several partial symmetry breaks for graphs and for matrix models which can be defined in terms of specific sets of permutations. We denote by S_n^{adj} and by S_n^{pair} the sets of permutations on $\{1,\ldots,n\}$ which swap a single adjacent pair $(i, i+1)$ for $1 \leq i < n$ and respectively a single pair (i,j) for $1 \leq i < j \leq n$.

The partial symmetry breaks sb_ℓ and sb_ℓ^* presented in [6] for graphs are defined as follows where A is an $n \times n$ adjacency matrix of Boolean variables:

$$\mathsf{sb}_\ell(A) = \bigwedge_{\pi \in S_n^{adj}} (A \leq \pi(A)) \quad \text{and} \quad \mathsf{sb}_\ell^*(A) = \bigwedge_{\pi \in S_n^{pair}} (A \leq \pi(A))$$

Thus, they can be encoded using $\mathcal{O}(n)$ and $\mathcal{O}(n^2)$ lex constraints, respectively.

Definition 3 (ordering matrices). *Let M_1, M_2 be a pair of $m \times n$ matrices of Boolean variables and let s_1, s_2 be the strings obtained by concatenating their rows respectively. Then, $M_1 \preceq M_2$ if and only if $s_1 \preceq_{lex} s_2$.*

For a $m \times n$ matrix M of Boolean variables and permutations $\pi_1 \in S_m$, $\pi_2 \in S_n$, let $\pi_1^{rows}(M)$ denote the matrix obtained by permuting the rows of M by π_1 and let $\pi_2^{cols}(M)$ denote the matrix obtained by permuting the columns of M by π_2. The doubleLex symmetry break, also denoted lex^2 [7] is defined by

$$lex^2(M) = \bigwedge \Big(\{\ (M \leq \pi^{rows}(M)) \,\big|\, \pi \in S_m^{adj}\ \} \bigcup \{\ (M \leq \pi^{cols}(M)) \,\big|\, \pi \in S_n^{adj}\ \} \Big)$$

It enforces rows and columns to be sorted lexicographically and it can be encoded using $\mathcal{O}(n+m)$ lex constraints. We also consider extensions of lex^2, denoted lex$^+$ and lex*.

$$lex^+(M) = lex^2(M) \wedge \bigwedge \{\ (M \leq \pi_1^{rows}\pi_2^{cols}(M)) \,\big|\, \pi_1 \in S_m^{adj}, \pi_2 \in S_n^{adj}\ \}$$

$$lex^*(M) = lex^2(M) \wedge \bigwedge \{\ (M \leq \pi_1^{rows}\pi_2^{cols}(M)) \,\big|\, \pi_1 \in S_m^{pair}, \pi_2 \in S_n^{pair}\ \}$$

Encoding these symmetry breaks requires $\mathcal{O}(nm)$ and $\mathcal{O}(n^2m^2)$ lex constraints, respectively.

In [8], the authors suggest combining lex^2 with additional constraints which enforce that the first row is lexicographically smaller than every permutation of every other row, and call this symmetry break allPerm. It can be implemented to run in linear time, however, encoding it statically into a SAT formula requires $\mathcal{O}(n!m)$ lex constraints. As we will show in Section 3, most of these constraints are actually redundant.

Table 1 illustrates the relative power of several symmetry breaks for Boolean matrix models. We consider matrices of size $n \times n$ and report the number of solutions for each of the symmetry breaks. The smaller the solution, the more precise the symmetry break. The symmetry breaks are detailed from weakest (left) to strongest (right). The left column, titled "None" has the most solutions and corresponds to imposing no symmetry break. The right column, titled "Complete" has the least solutions and corresponds to imposing a complete symmetry break. This column is obtained as OEIS sequence A002724 [21]. We observe that allPerm is only slightly stronger than lex^2 and weaker than lex$^+$. This is surprising, as lex$^+$ is polynomial in size whilst allPerm is exponential.

Table 1. The number of solutions for Boolean matrix models with various symmetry breaks.

n	None	lex^2	allPerm	lex^+	lex^*	Complete
3	2^9	45	41	37	36	36
4	2^{16}	650	520	366	330	317
5	2^{25}	24,520	17,128	8,659	6,779	5,624
6	2^{36}	2,625,117	1,616,074	602,813	391,532	251,610
7	2^{49}	836,488,618	458,375,316	139,268,908	73,720,859	33,642,660

3 Removing Redundant Constraints

In [11], the authors generated complete symmetry breaks for graph problems. They aimed for a small set of permutations which is canonizing, i.e. lex constraints derived from them create a complete symmetry break. They found that while generating such sets, some of the lex constraints became redundant and could be removed. Thus, after generating a complete symmetry break, they removed as many lex constraints as possible, and derived a set of non-redundant lex constraints. They furthermore noted that the set of permutations required for a complete symmetry break for graph problems on n nodes has significantly less than $n!$ elements.

Here, we consider the set of lex implications derived from a set of lex constraints rather than the lex constraints themselves. We show that even if the set of lex constraints is non-reducible, many of the lex implications are redundant and can be removed. The approach of removing redundant lex implications applies both to complete and partial symmetry breaks.

Table 2 shows the size of different symmetry breaks for graphs, both in terms of lex implications, and in terms of the number of clauses (in parentheses). This includes clauses required for encoding the Tseytin variables. For each symmetry break, the table details the impact of removing redundant lex implications. The columns titled "orig" denote the size of the symmetry breaks before reduction (obtained by decomposing the lex constraints), and the columns titled "red" denote the size of the symmetry breaks after removing redundant lex implications.

The constraints sb_ℓ and sb^*_ℓ are partial symmetry breaks introduced in [6]. Interestingly, sb_ℓ does not contain any redundant lex implications, whereas roughly 65% of the lex implications of sb^*_ℓ are redundant. For the right-most column, we took canonizing sets from [11], translated them into lex implications, and removed redundant clauses. Although the set of lex constraints does not contain any redundant constraints, more than 90% of the lex implications could be removed for $n = 10$ nodes. Furthermore, it is noteworthy that on small graphs, there are more clauses for the Tseytin encoding than for the symmetry break. For larger graphs, most of the clauses are lex implications.

Table 3 illustrates the reduction in size of symmetry breaks for Boolean matrix models of size $n \times n$. Here we focus on the number of lex implications

Table 2. Number of lex implications and clauses (in parentheses), before and after the reduction, for partial symmetry breaks sb_ℓ, sb^*_ℓ (on graphs) and for a complete symmetry break (for graphs) based on canonizing sets.

	sb_ℓ	sb^*_ℓ		canonizing	
n	orig & red	orig	red	orig	red
3	2(2)	3(3)	2(2)	3(2)	3(2)
4	6(12)	12(24)	6(12)	6(12)	6(12)
5	12(28)	30(70)	13(33)	41(83)	21(55)
6	20(50)	60(150)	24(72)	70(156)	38(118)
7	30(78)	105(273)	40(136)	302(580)	108(374)
8	42(112)	168(448)	62(232)	2006(2724)	387(1077)
9	56(152)	252(684)	91(367)	17059(18311)	2366(3600)
10	72(198)	360(990)	128(548)	248604(250582)	21844(23814)

Table 3. Number of lex implications for different symmetry breaks for matrix models before and after reduction.

	lex^2		lex^+		lex^*		allPerm	
n	orig	reduced	orig	reduced	orig	reduced	orig	reduced
3	12	12	28	16	64	16	48	13
4	24	24	78	37	294	45	312	32
5	40	40	168	77	968	112	2440	71
6	60	60	310	141	2560	252	21660	148
7	84	84	516	235	5808	532	211764	310
8	112	112	798	365	11774	1048	—	—
9	144	144	1168	536	21904	1944	—	—
10	180	180	1638	755	38088	3413	—	—

in the symmetry break (before and after reduce). In the table we consider the symmetry breaks lex^2, lex^+, lex^* and allPerm which are described in Section 2.

We observe that DoubleLex (lex^2) does not contain any redundant implications. On the contrary, more than half of the lex implications from lex^+ are redundant and approximately 90% of the lex implications in lex^* are redundant. With regards to the allPerm symmetry break proposed in [8], the constraint itself is huge. We did not generate it for matrices larger than 7×7 for which 99.85% of the lex implications are redundant. This huge size makes it hard to compute a reduced set of lex implications, we refrained from investing computational resources for the reduction of allPerm on matrices of size 8×8 and larger.

How the reduce works

Basically, our algorithm iterates over the set of lex implications, and checks for each of them if they are redundant. This is done by removing them from the

Algorithm 1 Ranking Redundant Constraints

$rank(c) = 0 \forall c \in \varphi$
for all $c \in \varphi$ **do**
  $\varphi' = (\varphi \setminus \{c\}) \cup \neg c$
  if UNSAT(φ') **then**
    Let $\psi \subseteq \varphi'$ such that $\psi \wedge \neg c \equiv \bot$
    for all $c' \in \psi$ **do**
      $rank(c') \leftarrow rank(c') + 1$
Sort clauses by their ranks, small ranks first.

Algorithm 2 Removing Redundant Constraints

Rank clauses with Algorithm 1
for all $c \in \varphi$ in ascending order **do**
  $\varphi' = (\varphi \setminus \{c\}) \cup \neg c$
  if UNSAT(φ') **then**
    $\varphi = \varphi \setminus \{c\}$

formula, and checking if there is a solution which would be forbidden by this clause, as shown in Algorithm 2. If this is not the case, the clause is redundant and can be removed. Contrary to other approaches like the one presented in [22], we run a full SAT search to determine if a lex implication is actually redundant or not. This allows for removing more clauses. Furthermore, the number of clauses which can actually be removed depends on the order in which clauses are checked. Some clauses are more helpful as they contribute to making other clauses redundant. Thus, we run our reduction in two phases. The first phase is shown in Algorithm 1. Here, we check if a clause c is redundant. If this is the case, we compute a subset $\psi \subseteq \varphi'$ of clauses which makes c redundant, and increase the ranking of all clauses within this set. The rationale is that removing these clauses is more likely make other clauses no longer redundant, and so increase the size of the final symmetry break.

In the second stage, we sort the clauses by ranking, so clauses which were frequently the cause of redundancy appear as late as possible. We then reduce the set of lex implications using Algorithm 2.

4 Generating Compact and Complete Symmetry Breaks for Graphs

The LexLeader constraint which is a complete symmetry break defined in terms of all permutations of a graph can be expressed as a set of lex implications. Each lex constraint $G \leq_{lex} \pi(G)$, for a permutation π is decomposed to lex implications, as described in Section 2. Each implication is classified by two parameters: the length of the implication and the permutation from which the originating lex constraint was generated. The length of an implication is the number of atoms it contains which is between 1 and $\binom{n}{2}$ (the size of the upper

Algorithm 3 Generating Complete Implication Set

```
init: C ← { }
for k = 1 to (n choose 2) do
    while ∃π ∈ S_n, G ∈ G_n s.t (C ∧ ¬Imp_G(k, π)) is satisfiable do
        C ← C ∪ {Imp_G(k, π)}
    C ← reduce(C)
return C
```

triangle of the adjacency matrix). Formally, let A be an $n \times n$ matrix, $\pi \in S_n$ a permutation, and $1 \leq k \leq \binom{n}{2}$ an implication length. Let $x_1, \ldots, x_{\binom{n}{2}}$ and $y_1, \ldots, y_{\binom{n}{2}}$ be the upper triangle elements (row by row) of A and $\pi(A)$, respectively. The length k lex implication $Imp_A(k, \pi)$ is defined by

$$Imp_A(k, \pi) = (x_1 = y_1) \wedge \cdots \wedge (x_{k-1} = y_{k-1}) \Rightarrow x_k \leq y_k$$

Using this notation the classic LexLeader constraint for an $n \times n$ adjacency matrix A is equivalent to the following lex implication representation:

$$LexLeader_A(n) = \bigwedge_{\pi \in S_n} \bigwedge_{k=1}^{\binom{n}{2}} Imp_A(k, \pi)$$

In this work we generate a complete symmetry breaking constraint that is equivalent to LexLeader by repeatedly selecting lex implications from the definition of $LexLeader_A(n)$ which are not logically implied by those already selected. When no further lex implications can be selected we have found a complete symmetry break. Although we repeatedly select non-redundant lex implications, it is possible, because of the order of selection, that some of the implications in the set become redundant. For this we reason we perform a second pass to repeatedly remove redundant implications. This process is formalized as Algorithm 3 where we select implications according to their length, first the short ones, and then the longer ones. To derive a complete symmetry break for graphs with n vertices, for each $1 \leq k \leq \binom{n}{2}$ the algorithm repeatedly finds implications of the form $Imp_A(k, \pi)$ until the set obtained so far implies every implication of length k. To find a new implication of length k we check if there exists a permutation $\pi \in S_n$ and an $n \times n$ adjacency matrix A of Boolean variables such that C is satisfied, but there exists a lex implication $Imp_A(k, \pi)$ which is not satisfied where C denotes the conjunction of the implications selected so far. This process continues iterating for implications of all lengths, starting from short implications, $k = 1$, and finishing with the longest implications, $k = \binom{n}{2}$. In the algorithm we apply a reduce step to remove redundant implications after each increment of the value k.

Table 4 details the computation of compact complete symmetry breaks following Algorithm 3 (lex implications) and provides a comparison with those computed in [11] (canonizing), and the symmetry breaks from [12] (isolators). For each value of n (number of vertices) we detail the size of the symmetry break

Table 4. Computing compact and complete symmetry breaking constraints for graphs (time is in seconds except where indicated otherwise).

	canonizing			lex implications			Isolator from [12]	
	lex	clauses	time	imp	clauses	time	clauses	time
4	3	12	0.03	6	14	0.18	7	0.01
5	7	83	0.10	18	52	0.49	12	2.34
6	13	156	1.62	45	149	1.99	27	4.6 days
7	37	580	13.19	139	449	8.68	114	1555 days
8	135	2,724	345.37	447	1139	59.81	956	2 days
9	842	18,311	2.42 hr	2,496	3736	626.20	–	–
10	7,853	250,582	93.82 hr	22,542	24,512	3.10 hrs	–	–
11	–	–	–	274,109	277,075	8.44 days	–	–

derived and the time it took to compute it. For the symmetry breaks of [11], size is reported in the number of lex constraints ("lex") and also in the number of clauses in their encoding to CNF. For the symmetry breaks derived in this paper, size is reported in the number of lex implications ("imp") and also in the number of clauses in their encoding to CNF. Isolators are, by definition, sets of clauses. The times in the right column (Isolator) of the table are reported from [12]. These were obtained, for different values of n, using different techniques. Thus, the computation for 8 nodes is faster than the one for 7 nodes. For 7 nodes, Heule reports in [12], a computation involving 80,000 probes per round with 4 rounds at 7 minutes (average) per probe. This totals 1555 days of computation. The items denoted − indicate that the corresponding symmetry breaks cannot be computed. The only technique able to compute a symmetry break for graphs with 11 vertices is the technique presented in this paper. We note that this is the first time that a compact and complete symmetry break for graphs of size 11 has been computed.

Table 5 demonstrates the impact of having more compact complete symmetry breaks. We detail the time to compute all graphs that satisfy each form of the complete symmetry break. Because the symmetry breaks are all complete, this number corresponds exactly to the number of non-isomorphic undirected graphs with n vertices (OEIS sequence number A000088) [21] which is detailed in the right column. We see from the table that enumerating graphs with more compact symmetry breaks significantly reduces the computation time.

5 An Application: Computing Ramsey Colorings $(4, 4; n)$

In this section we describe the impact of using compact and complete symmetry breaks. We consider a classic example of a graph search problem: the search for Ramsey graphs [23]. The graph $R(s, t; n)$ is a simple graph with n vertices, no clique of size s, and no independent set of size t. The Ramsey number $R(s, t)$ is the smallest number n for which there is no $R(s, t; n)$ graph. Table 6 reports

Table 5. Enumerating graphs using complete symmetry breaking methods: canonizing, lex-implications, and isolators (using Clasp where time in seconds unless indicated otherwise).

n	Canonizing	lex-imp's	Isolator [12]	graphs
4	0.00	0.00	0.00	11
5	0.00	0.00	0.00	34
6	0.00	0.00	0.00	156
7	0.00	0.00	0.00	1,044
8	0.27	0.11	0.04	12,346
9	33.34	3.65	–	274,668
10	5.78 hr.	542.25	–	12,005,168
11	–	2.69 days	–	1,018,997,864

on the search for all solutions for $R(4,4,n)$. For $n > 17$ there are no solutions and hence the Ramsey number $R(4,4) = 18$. The table compares three configurations: First, using the partial symmetry breaking predicate sb^*_ℓ defined in [6]. Second, using the complete canonizing symmetry breaks computed in [11]. Third, using the complete symmetry breaks computed in this paper. For each configuration we detail the size of the SAT encoding (clauses and variables), the time in seconds (except where indicated in hours) to find all solutions using a SAT solver, and the number of solutions found. The symmetry breaks based on canonizing permutations and on lex implications are both complete, so they both compute the exact number of solutions. In the upper part of the table, we use the corresponding complete symmetry breaks described in [11] (for $n \leq 10$) and in Section 4 of this paper (for $n \leq 11$). These symmetry breaks are *"instance independent"*. They apply to break symmetries for any graph search problem. For $12 \leq n \leq 17$ we compute *"instance dependent"* symmetry breaks. We refine Algorithm 3 to compute lex implications that break symmetries for the specific application to $R(4,4,n)$. This is, we restrict Algorithm 3 to consider only $R(4,4,n)$ graphs instead of all graphs in $\mathcal{G}_n$.

The lower part of Table 6 reports on the search for all solutions of $R(4,4,n)$ for $n \geq 12$ using these complete symmetry breaks. The items denoted $--$ indicate that the corresponding symmetry breaks cannot be computed within the timeout period (72 hours).

It can be seen that for $12 \leq n \leq 15$, in which the number of non-isomorphic solutions is large, these problem dependent symmetry breaks significantly improve the solving time over the partial symmetry break sb^*_ℓ. For $n = 16$, the symmetry break computed here is significantly stronger than sb^*_ℓ, allowing for only 2 instead of 94 solutions.

Using the lex-implications approach we were able to compute instance dependent symmetry breaks for all $R(4,4,n)$ instances whereas the computation of canonizing sets exceeded the timeout for three cases.

Table 6. Enumerating Ramsey graphs using sb^*_ℓ , canonizing sets and lex implications. (Clasp solver, computation time in seconds).

	sb^*_ℓ				canonizing sets			lex implications			
instance	cls	vars	sat	sols	cls	vars	sat	cls	vars	sat	exact
$R(4,4,4)$	22	10	0.00	9	17	9	0.00	68	21	0.00	9
$R(4,4,5)$	80	24	0.00	33	235	55	0.00	208	55	0.00	24
$R(4,4,6)$	195	48	0.00	178	315	72	0.00	495	120	0.00	84
$R(4,4,7)$	390	85	0.00	1,478	1,395	286	0.00	1,049	231	0.00	362
$R(4,4,8)$	690	138	0.03	16,919	10,885	2,177	0.04	2,099	406	0.02	2,079
$R(4,4,9)$	1,122	210	0.51	227,648	89,877	17,961	1.56	5,268	666	0.23	14,701
$R(4,4,10)$	1,715	304	9.97	2,891,024	1,406,100	281,181	149.15	26,922	1,035	5.43	103,706
$R(4,4,11)$	2,500	423	428.79	25,616,963	–	–	–	280,709	1,540	726.62	546,356
$R(4,4,12)$	3,510	570	5561.06	107,509,048	–	–	–	48,363	2,715	129.71	1,449,390
$R(4,4,13)$	4,780	748	29426.23	131,638,650	–	–	–	57,747	3,751	133.64	1,184,323
$R(4,4,14)$	6,347	960	8325.25	21,181,746	–	–	–	36,505	5,055	31.18	130,818
$R(4,4,15)$	8,250	1,209	281.79	144,663	183,985	36,356	26.21	30,855	6,669	55.95	640
$R(4,4,16)$	10,530	1,498	14.38	94	30,890	5,570	12.34	39,131	8,638	19.43	2
$R(4,4,17)$	13,230	1,830	5.63	4	15,255	2,235	7.51	49,953	11,010	20.69	1

6 Conclusion

We provided an analysis of the redundancy in symmetry breaking constraints for graphs and matrix models. Previous work had shown that many of the lex constraints in the LexLeader symmetry break are redundant. Here, we considered the decomposition of lex constraints in lex implications, and showed that many of them are redundant in complete symmetry breaks. This allowed us to reduce the size of complete symmetry breaks for graphs by an order of magnitude, and enabled us to compute a complete and compact symmetry break for graphs on 11 nodes.

Furthermore, we analyzed partial symmetry breaks and the redundancies in them. While small symmetry breaks like sb_ℓ for graphs, and lex^2 for matrices do not contain any redundant lex implications, there are significant redundancies in their extensions sb^*_ℓ and lex^+, lex^*, respectively.

References

1. Puget, J.: On the satisfiability of symmetrical constrained satisfaction problems. In Komorowski, H.J., Ras, Z.W., eds.: Methodologies for Intelligent Systems, 7th International Symposium, ISMIS '93, Trondheim, Norway, June 15-18, 1993, Proceedings. Volume 689 of LNCS., Springer (1993) 350–361
2. Crawford, J.M., Ginsberg, M.L., Luks, E.M., Roy, A.: Symmetry-breaking predicates for search problems. In Aiello, L.C., Doyle, J., Shapiro, S.C., eds.: Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96), Cambridge, Massachusetts, USA, November 5-8, 1996., Morgan Kaufmann (1996) 148–159
3. Shlyakhter, I.: Generating effective symmetry-breaking predicates for search problems. Discrete Applied Mathematics **155**(12) (2007) 1539–1548
4. Walsh, T.: General symmetry breaking constraints. In Benhamou, F., ed.: Principles and Practice of Constraint Programming - CP 2006, 12th International Con-

ference, CP 2006, Nantes, France, September 25-29, 2006, Proceedings. Volume 4204 of LNCS., Springer (2006) 650–664
5. Walsh, T.: Symmetry breaking constraints: Recent results. In Hoffmann, J., Selman, B., eds.: Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada., AAAI Press (2012)
6. Codish, M., Miller, A., Prosser, P., Stuckey, P.J.: Breaking symmetries in graph representation. In Rossi, F., ed.: IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013, IJCAI/AAAI (2013) 510–516
7. Flener, P., Frisch, A.M., Hnich, B., Kiziltan, Z., Miguel, I., Pearson, J., Walsh, T.: Breaking row and column symmetries in matrix models. In Hentenryck, P.V., ed.: Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings. Volume 2470 of LNCS., Springer (2002) 462–476
8. Frisch, A.M., Jefferson, C., Miguel, I.: Constraints for breaking more row and column symmetries. In Rossi, F., ed.: Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings. Volume 2833 of LNCS., Springer (2003) 318–332
9. Grayland, A., Miguel, I., Roney-Dougal, C.M.: Snake lex: An alternative to double lex. In Gent, I.P., ed.: Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings. Volume 5732 of LNCS., Springer (2009) 391–399
10. Read, R.C.: Every one a winner or how to avoid isomorphism search when cataloguing combinatorial configurations. Ann. Discrete Math. **2** (1978) 107–120
11. Itzhakov, A., Codish, M.: Breaking symmetries in graph search with canonizing sets. Constraints **21**(3) (2016) 357–374
12. Heule, M.J.H.: The quest for perfect and compact symmetry breaking for graph problems. In Davenport, J.H., Negru, V., Ida, T., Jebelean, T., Petcu, D., Watt, S.M., Zaharie, D., eds.: 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2016, Timisoara, Romania, September 24-27, 2016, IEEE Computer Society (2016) 149–156
13. Frisch, A.M., Harvey, W.: Constraints for breaking all row and column symmetries in a three-by-two matrix. In: In Proceedings of SymCon03. (2003)
14. Frisch, A.M., Hnich, B., Kiziltan, Z., Miguel, I., Walsh, T.: Propagation algorithms for lexicographic ordering constraints. Artif. Intell. **170**(10) (2006) 803–834
15. Smith, B.: Symmetry breaking constraints in constraint programming. `http://ta.twi.tudelft.nl/wst/users/achill/MFOSymOpt2010/MFOSymOpt2010/Oberwolfach_Mini-Workshop_files/BarbaraMfoSlides.ppt` (2010) (slides published online).
16. Metodi, A., Codish, M., Stuckey, P.J.: Boolean equi-propagation for concise and efficient SAT encodings of combinatorial problems. J. Artif. Intell. Res. (JAIR) **46** (2013) 303–341
17. Audemard, G., Simon, L.: Glucose 4.0 SAT Solver. `http://www.labri.fr/perso/lsimon/glucose/`.
18. Gebser, M., Kaufmann, B., Schaub, T.: Conflict-driven answer set solving: From theory to practice. Artif. Intell. **187** (2012) 52–89
19. Cameron, R.D., Colbourn, C.J., Read, R.C., Wormald, N.C.: Cataloguing the graphs on 10 vertices. Journal of Graph Theory **9**(4) (1985) 551–562
20. Plaisted, D.A., Greenbaum, S.: A structure-preserving clause form translation. J. Symb. Comput. **2**(3) (1986) 293–304

21. : The on-line encyclopedia of integer sequences. published electronically at `http://oeis.org` (2010)
22. Fourdrinoy, O., Grégoire, É., Mazure, B., Sais, L.: Eliminating redundant clauses in SAT instances. In Hentenryck, P.V., Wolsey, L.A., eds.: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 4th International Conference, CPAIOR 2007, Brussels, Belgium, May 23-26, 2007, Proceedings. Volume 4510 of LNCS., Springer (2007) 71–83
23. Radziszowski, S.P.: Small Ramsey numbers. Electronic Journal of Combinatorics (1994) Revision #14: January, 2014.