A SAT+CAS Approach to Finding Good Matrices: New Examples and Counterexamples

Curtis Bright
University of Waterloo

Dragomir Ž. Đoković
University of Waterloo

Ilias Kotsireas
Wilfrid Laurier University

Vijay Ganesh
University of Waterloo

Abstract

We enumerate all circulant good matrices with odd orders divisible by 3 up to order 70. As a consequence of this we find a previously overlooked set of good matrices of order 27 and a new set of good matrices of order 57. We also find that circulant good matrices do not exist in the orders 51, 63, and 69, thereby finding three new counterexamples to the conjecture that such matrices exist in all odd orders. Additionally, we prove a new relationship between the entries of good matrices and exploit this relationship in our enumeration algorithm. Our method applies the SAT+CAS paradigm of combining computer algebra functionality with modern SAT solvers to efficiently search large spaces which are specified by both algebraic and logical constraints.

1 Introduction

In 2002, circulant "good" matrices were searched for in all odd orders n with $n < 40$ (Georgiou, Koukouvinos, and Stylianou 2002) and were found to exist in all such orders. This gave evidence to the conjecture that such matrices actually exist in *all* odd orders. The mathematician George Szekeres studied circulant good matrices in terms of an equivalent type of object which he termed E-sequences and in his classic paper (Szekeres 1988) he mentioned this hypothesis as being worthwhile to study using computers:

> ... it is conceivable that E-sequences exist for all $n = 2m + 1$, $m \geq 1$ and it is worth testing this hypothesis at least for those orders which are accessible to present day computers...

Unfortunately, this conjecture was recently shown to be false in (Đoković and Kotsireas 2018) where it was shown that the orders $n = 41$, 47, and 49 are counterexamples, i.e., no circulant good matrices of orders 41, 47, and 49 exist.

In this paper we find the additional three larger counterexamples $n = 51$, 63, and 69. In addition, we verify the claim of Đoković and Kotsireas that there are exactly four inequivalent sets of circulant good matrices of order 45 and determine that there is a single set of inequivalent circulant good matrices of order 57. We also find a previously undiscovered set of good matrices of order 27 demonstrating that Szekeres' claimed exhaustive search (Szekeres 1988) was incomplete.

The matrices that are now known as good matrices were first used in the PhD thesis of Jennifer Seberry Wallis (Wallis 1970); we give their formal definition in Section 2. She gave a construction using good matrices that enabled the construction of *Hadamard matrices*—square matrices with ± 1 entries and whose rows are pairwise orthogonal. In her thesis she gave examples of good matrices in all odd orders up to 15 as well as order 19 and shortly later gave an example in order 23 (Wallis 1971). Subsequently, (Hunt 1972) ran a complete search in the odd orders up to 21 and gave examples in order 25, (Szekeres 1988) ran a complete search up to order 31, (Đoković 1993) gave examples in orders 33 and 35, (Georgiou, Koukouvinos, and Stylianou 2002) ran a complete search up to order 39 and most recently (Đoković and Kotsireas 2018) ran a complete search up to order 49. The result of these searches have shown that good matrices exist in all odd orders up to 39 as well as 43 and 45. Additionally, (Đoković 1993) gave a construction showing they exist in order 127.

In this paper we extend the search to all odd orders divisible by 3 up to order 69. Note that increasing n by 2 increases the size of the search space (at least to a first approximation) by a factor of 16, meaning that the size of the search space in order 69 is an enormous 2^{40} times larger than the search space in order 49, the largest order which has previously been exhaustively searched. There are two primary reasons our approach is able to scale to such orders:

1. We use a number of theoretical mathematical results to cut down the search space, including filtering theorems, recently proven compression theorems, and a new product theorem given in this paper.
2. We use state-of-the-art programmatic SAT solvers, computer algebra systems, and mathematical libraries to very efficiently search spaces specified by both logical and algebraic constraints.

This approach of coupling computer algebra systems and SAT solvers was first proposed in 2015 at the conferences CADE and ISSAC (Zulkoski, Ganesh, and Czarnecki 2015; Ábrahám 2015) and has since appeared in papers at the conferences IJCAI, ISSAC, and the journal of automated reasoning (Zulkoski, Ganesh, and Czarnecki 2016; Bright et

al. 2018; Zulkoski et al. 2017) and is the aim of the SC^2 project (Ábrahám et al. 2016). In particular, the SAT+CAS method in this paper is an adaption of one presented at AAAI (Bright, Kotsireas, and Ganesh 2018).

Furthermore, we employ a SAT solver which can learn clauses *programmatically*, through a piece of code compiled with the SAT solver and which makes calls to an external mathematical library. This approach permits one to efficiently encode constraints that would be much too cumbersome to encode with native SAT clauses, as we will see in Section 3. For example, it is much faster and easier to compute a discrete Fourier transform using a numerical library or computer algebra system than it would be to encode a DFT circuit using logical clauses, the format normally accepted by SAT solvers.

A detailed description of our enumeration algorithm will be given in Section 4, followed by our results in Section 5. In particular, we explicitly give two new examples of good matrices, one in order 27 which has gone undetected since 1988 and one in order 57, an order which had previously been out of reach of exhaustive search algorithms; the largest order searched prior to our work was 49. We also provide a table enumerating the number of sets of good matrices in all odd orders divisible by 3 up to 69. These counts are given up to an equivalence which is described, along with the required background on good matrices, in Section 2.

2 Background

In this section we define good matrices and review the properties and theorems of good matrices that we use in our enumeration algorithm.

To begin, we recall the definitions of symmetric, skew, and circulant matrices. Let X be a square matrix of order n with entries $x_{i,j}$ with $0 \leq i, j < n$. We say that X is *symmetric* if all entries satisfy $x_{i,j} = x_{j,i}$, that X is *skew* if diagonal entries are 1 and non-diagonal entries satisfy $x_{i,j} = -x_{j,i}$, and that X is *circulant* if all entries satisfy $x_{i,j} = x_{i-1,j-1}$ with indices taken mod n as necessary.

Good matrices

Definition 1. *Four matrices A, B, C, $D \in \{\pm 1\}^{n \times n}$ are known as* good matrices *if they have the following properties.*

(a) They are pairwise amicable (XY^T is symmetric for $X, Y \in \{A, B, C, D\}$).

(b) A is skew and B, C, D are symmetric.

(c) $AA^T + B^2 + C^2 + D^2$ is the identity matrix scaled by $4n$.

The primary reason that good matrices have attracted sustained interest for almost 50 years is because of the following theorem of (Wallis 1970) which provides a construction for skew Hadamard matrices.

Theorem 1. *Let A, B, C, D be good matrices of order n. Then*

$$\begin{bmatrix} A & B & C & D \\ -B & A & D & -C \\ -C & -D & A & B \\ -D & C & -B & A \end{bmatrix}$$

is a skew Hadamard matrix of order $4n$.

Skew Hadamard matrices are conjectured to exist in all orders $4n$ with $n \geq 1$ and much effort has gone into constructing them in as many orders as possible; the current smallest unknown order is $n = 69$ and the previous smallest unknown order $n = 47$ was solved ten years ago by (Đoković 2008).

From a computational perspective the issue with good matrices is that the search space is too large to effectively search. The standard remedy for this is to focus on *circulant* matrices for which the search can be performed more effectively. For example, in (Awyzio and Seberry 2015) good matrices are defined as in Definition 1 except that condition (a) is replaced with the strictly stronger condition

(a′) A, B, C, D are circulant.

Technically matrices which satisfy conditions (a′), (b), and (c) are not good matrices but one can recover good matrices from them simply by reversing the order of the rows of B, C, and D (thereby making them amicable with A). Since using (a′) is convenient to state our results we will employ it for the remainder of this paper.

We often consider good matrices in terms of their first rows $(a_0, \ldots, a_{n-1})$, $(b_0, \ldots, b_{n-1})$, $(c_0, \ldots, c_{n-1})$, and $(d_0, \ldots, d_{n-1})$ which we call their *defining rows*. Since A is skew and B, C, and D are symmetric, the defining rows of a set of good matrices must satisfy $a_0 = 1$ and

$$a_i = -a_{n-i}, \quad b_i = b_{n-i}, \quad c_i = c_{n-i}, \quad d_i = d_{n-i}$$

for $1 \leq i < n/2$. Furthermore, we often use A, B, C, and D to denote the defining rows themselves, in which case we say that A is a skew sequence and that B, C, and D are symmetric sequences.

Equivalence operations

The following three invertible operations are well-known and can be applied to a set of good matrices A, B, C, D to produce another set of good matrices we consider *equivalent* to A, B, C, D.

1. Reorder B, C, D in any way.
2. Negate any of B, C, D.
3. Apply an automorphism of $\mathbb{Z}_n = \{0, \ldots, n-1\}$ to the indices of the defining rows of each of A, B, C, and D.

For example, the negation equivalence operation implies that any set of good matrices is equivalent to one with $b_0 = c_0 = d_0 = 1$.

Filtering theorems

The search space for good matrices of order n is enormous. Even if one takes into account the symmetry properties of the defining rows and fixes the first entries to be positive, each defining row contains $\lfloor n/2 \rfloor$ unspecified entries. Since each unspecified entry can be one of two values, there are a total of $2^{4\lfloor n/2 \rfloor} = 4^{n-1}$ possibilities in the search space.

In order to cut down the size of the search space so that we can run exhaustive searches in larger orders we need filtering theorems. One of the most powerful filtering theorems is based on the following which gives an alternative characterization of good matrices (Đoković and Kotsireas 2015).

Theorem 2. *Let A, B, C, D be sequences of length n with A skew and B, C, D symmetric. Then A, B, C, D are the defining rows of a set of good matrices if and only if for all integers k we have*

$$\mathrm{PSD}_A(k) + \mathrm{PSD}_B(k) + \mathrm{PSD}_C(k) + \mathrm{PSD}_D(k) = 4n.$$

Here $\mathrm{PSD}_X(k) \coloneqq \left|\sum_{j=0}^{n-1} x_j \omega^{jk}\right|^2$ *with* $\omega \coloneqq \exp(2\pi i/n)$ *is the power spectral density of* $X = [x_0, \ldots, x_{n-1}]$.

This is important because of the following well-known corollary:

Corollary 3. *Let A, B, C, D be the defining rows of a set of good matrices of order n. If S is a subset of $\{A, B, C, D\}$ then $\sum_{X \in S} \mathrm{PSD}_X(k) \leq 4n$ for all integers k.*

For example, this corollary says that if $\mathrm{PSD}_A(k) > 4n$ for some k then A cannot be part of a set of good matrices. In other words, we can filter A and ignore it in our enumeration method. A second well-known corollary of Theorem 2 comes from setting $k = 0$ in the statement of the theorem:

Corollary 4. *Let A, B, C, D be the defining rows of a set of good matrices of odd order n. Then*

$$\mathrm{sum}(B)^2 + \mathrm{sum}(C)^2 + \mathrm{sum}(D)^2 = 4n - 1$$

where $\mathrm{sum}(X)$ *denotes the sum of entries in* X.

Since $\mathrm{sum}(X)$ is an integer when X has integer entries this corollary tells us that every set of good matrices gives a decomposition of $4n - 1$ into a sum of three integer squares which is a rather restrictive condition. For example, when n is 69 there are only three ways to write $4n - 1$ as a sum of exactly three squares of nonnegative integers, namely,

$$1^2 + 7^2 + 15^2, \quad 5^2 + 5^2 + 15^2, \quad \text{and} \quad 5^2 + 9^2 + 13^2.$$

One must also consider decompositions with squares of negative integers but the negative integers can be found with the following simple (often unstated) lemma. It gives a criterion to determine the signs of $\mathrm{sum}(B)$, $\mathrm{sum}(C)$, and $\mathrm{sum}(D)$.

Lemma 5. *Let $(a_0, \ldots, a_{n-1})$, $(b_0, \ldots, b_{n-1})$, $(c_0, \ldots, c_{n-1})$, $(d_0, \ldots, d_{n-1})$ be the defining rows of a set of good matrices A, B, C, D of odd order n with $b_0 = c_0 = d_0 = 1$. Then*

$$\mathrm{sum}(B) \equiv \mathrm{sum}(C) \equiv \mathrm{sum}(D) \equiv n \pmod 4.$$

Compression

Although the filtering theorems greatly decrease the number of possibilities for the defining rows of good matrices there are still typically a large number of possibilities which are not filtered by Corollaries 3 and 4. An effective way of shrinking the search space even farther is to apply properties that "compressions" of good matrices must satisfy. Such theorems are only applicable in composite orders; we will assume that the order n is a multiple of 3. To our knowledge this is the first time compression has been used in the search for good matrices and it allows us to search larger orders than other methods. For example, (Đoković and Kotsireas 2018) use the aforementioned filtering corollaries but do not apply compression and are only able to search orders up to 49.

The *3-compression* of a sequence $X = (x_0, \ldots, x_{n-1})$ of length n is the sequence X' of length $m \coloneqq n/3$ whose kth entry is $x_k + x_{k+m} + x_{k+2m}$. Since we are concerned with the case where the entries of X are ± 1 the entries of X' are either ± 1 or ± 3. Compression is useful because (Đoković and Kotsireas 2015) showed that Theorem 2 holds for compressions of the defining rows of good matrices.

Theorem 6. *Let A, B, C, D be the defining rows of a set of good matrices of order n and let A', B', C', D' be compressions of those rows. Then for all integers k we have*

$$\mathrm{PSD}_{A'}(k) + \mathrm{PSD}_{B'}(k) + \mathrm{PSD}_{C'}(k) + \mathrm{PSD}_{D'}(k) = 4n.$$

A new product theorem

Our study of good matrices uncovered the following relationship that the entries of good matrices must satisfy. This is an analog of the product theorem that was proven by John Williamson for Williamson matrices (Williamson 1944).

Theorem 7. *Let $(a_0, \ldots, a_{n-1})$, $(b_0, \ldots, b_{n-1})$, $(c_0, \ldots, c_{n-1})$, $(d_0, \ldots, d_{n-1})$ be the defining rows of a set of good matrices A, B, C, D of odd order n. Then $a_k b_k c_k d_k = -a_{2k \bmod n} b_0 c_0 d_0$ for all $1 \leq k < n$.*

Let $\bar{x} \coloneqq (1 - x)/2$, i.e., the mapping $x \mapsto \bar{x}$ is the group isomorphism between $\mathbb{Z}_4^* = \{\pm 1\}$ and $\mathbb{Z}_2 = \{0, 1\}$. Our enumeration method uses Theorem 7 in the following form.

Corollary 8. *Let $(a_0, \ldots, a_{n-1})$, $(b_0, \ldots, b_{n-1})$, $(c_0, \ldots, c_{n-1})$, $(d_0, \ldots, d_{n-1})$ be the defining rows of a set of good matrices A, B, C, D of odd order n with $b_0 = c_0 = d_0 = 1$. Then for $1 \leq k < n/2$ we have in $\mathbb{Z}_2$*

$$\bar{a}_k + \bar{a}_{2k} + \bar{b}_k + \bar{c}_k + \bar{d}_k = 1$$

and when $n = 3m$ we have $\bar{b}_m + \bar{c}_m + \bar{d}_m = 1$.

Proofs of Lemma 5, Theorem 7, and Corollaries 3, 4, and 8 are published at uwaterloo.ca/mathcheck.

3 The SAT+CAS paradigm

The SAT+CAS paradigm is a new method of solving certain kinds of computational problems which harnesses the power of two fields of computer science: satisfiability checking and symbolic computation. The paradigm is particularly useful for problems that have a significant search component and a rich mathematical component.

Briefly, this is due to the fact that modern SAT solvers have extremely efficient search procedures but lack the domain knowledge to effectively deal with rich mathematical concepts. On the other hand, computer algebra systems like MAPLE, MATHEMATICA, and SAGEMATH as well as special-purpose numerical libraries can very effectively deal with rich mathematics.

The fact that satisfiability checking and symbolic computation had great potential synergy was first pointed out by Erika Ábrahám in an invited talk at the conference ISSAC in 2015 (Ábrahám 2015). At almost the same time this synergy was demonstrated by the system MATHCHECK presented at the conference CADE (Zulkoski, Ganesh, and Czarnecki 2015). The system MATHCHECK coupled a SAT solver with a computer algebra system and solved open cases of two

conjectures in graph theory and was later extended to solve open cases in combinatorial conjectures (Zulkoski et al. 2017). Since then, the SC^2 project (Ábrahám et al. 2016) has organized an annual workshop on this topic for the last three years.

To illustrate the usefulness of the SAT+CAS paradigm in the search for good matrices specifically, consider Corollaries 3, 4, and 8 from Section 2. These corollaries all state various properties that good matrices must satisfy but each property arises from a different branch of mathematics. Consequently, each property is best dealt with using systems which have been optimized to deal with that particular branch:

- Corollary 3 is a statement in Fourier analysis and is best checked using a system which has been designed to efficiently compute Fourier transforms.
- Corollary 4 (and Lemma 5) are number theoretic statements and possibilities for the values $\mathrm{sum}(B)$, $\mathrm{sum}(C)$, and $\mathrm{sum}(D)$ can be found by using a system which has been designed to solve quadratic Diophantine systems.
- Corollary 8 is a statement using arithmetic in $\mathbb{Z}_2$ and can efficiently be encoded directly in a SAT instance.

The programmatic SAT paradigm

A programmatic SAT solver, introduced in (Ganesh et al. 2012), is a new kind of SAT solver which allows solving satisfiability problems with more expressiveness than the kinds of problems that can be solved with a standard SAT solver. Briefly, a programmatic SAT solver will use some custom code to examine the current partial assignment and if it cannot be extended into a full assignment (based on some known theorem from the problem domain) a clause will be learned encoding that fact. Because one can think of "calling" the programmatic SAT solver with the custom code this code is known as a "callback" function.

Programmatic SAT solvers are conceptually similar to SMT (SAT modulo theories) solvers and in some ways could be considered a "poor man's SMT". However, they also differ from SMT solvers in some key ways:

Simplicity. SMT solvers are necessarily more complicated than SAT solvers because they solve problems in first-order logic instead of problems in propositional logic. This increase in complexity is justifiable for many problems but some problems contain mostly just propositional logic with a handful of "theory lemmas" that are cumbersome to state in propositional logic but can be given to a programmatic SAT solver as necessary to help guide its search.

Flexibility. Typically SMT solvers only support theories which are fixed in advance; for example, the SMT-LIB standard (Barrett, Fontaine, and Tinelli 2016) specifies a fixed number of theories and a common input and output language for those theories. However, not all problems can naturally be expressed in those theories; e.g., the current SMT-LIB standard does not include theories involving Fourier transforms (or transcendental functions or complex numbers). In contrast, a programmatic SAT solver can learn theory lemmas derived using Fourier transforms so long as an appropriate CAS or library can be called.

Tailored solving. Programmatic SAT solvers can be tailored to solve specific classes of problems (in addition to using the same state-of-the-art techniques which make modern SAT solving so efficient). For example, in this paper we develop a SAT solver tailored to searching for good matrices. In principle SMT solvers could work in the same way but we are not aware of any SMT solvers that offer a programmatic interface like this.

A programmatic SAT encoding of good matrices

We now describe how our programmatic SAT solver encoding of good matrices works in detail. We encode the entries of the defining rows of a set of good matrices A, B, C, D of order n using $4n$ Boolean variables where we let true values denote 1 and false values denote -1. By abuse of notation we use the same names for the Boolean variables and the ± 1 entries that they represent but it will be clear from the context if we are referring to a Boolean or integer value.

In fact, using the symmetry constraints from Section 2 we only need to define the $2(n+1)$ variables a_i, b_i, c_i, d_i that have indices i with $0 \leq i < n/2$. In what follows we implicitly use this whenever necessary; i.e., any variables with indices larger than $n/2$ are used for clarity only.

A programmatic SAT solver includes a callback function that is run whenever the SAT solver's usual conflict analysis fails to find a conflict. The callback function examines the current partial assignment and learns clauses that block the current assignment (and extensions of the current assignment) from the search in the future.

If x is a variable that appears in the current partial assignment we let x^{cur} denote either the literal x or $\neg x$, whichever is true under the current assignment. The callback function used in our encoding of good matrices works by encoding the property given in Corollary 3 and its steps are described in detail below.

1. Let S be a list containing the sequences $(a_0, \ldots, a_{n-1})$, $(b_0, \ldots, b_{n-1})$, $(c_0, \ldots, c_{n-1})$, $(d_0, \ldots, d_{n-1})$ that have all their variables currently assigned.
2. If S is empty then return control to the SAT solver.
3. Let X be the first sequence in S and compute the power spectral density of X. If $\mathrm{PSD}_X(k) > 4n$ for some k then learn a clause blocking the sequence X from the search in the future. Explicitly, learn the clause
$$\neg x_0^{\text{cur}} \vee \neg x_1^{\text{cur}} \vee \cdots \vee \neg x_{n-1}^{\text{cur}}$$
and return control to the SAT solver.
4. If S contains at least two sequences then let Y be the second sequence in S and compute the power spectral density of Y. If $\mathrm{PSD}_X(k) + \mathrm{PSD}_Y(k) > 4n$ for some k then learn a clause blocking the sequences X and Y from occurring together again in the future. Explicitly, learn the clause
$$\neg x_0^{\text{cur}} \vee \cdots \vee \neg x_{n-1}^{\text{cur}} \vee \neg y_0^{\text{cur}} \vee \cdots \vee \neg y_{n-1}^{\text{cur}}$$
and return control to the SAT solver.

5. If S contains at least three sequences then let Z be the third sequence in S and compute the power spectral density of Z. If $\mathrm{PSD}_X(k) + \mathrm{PSD}_Y(k) + \mathrm{PSD}_Z(k) > 4n$ for some k then learn a clause blocking the sequences X, Y, and Z from occurring together again in the future and return control to the SAT solver.
6. If S contains four sequences then we know every entry of A, B, C, and D. If the relationship in Theorem 2 holds then record A, B, C, and D as rows that define good matrices. Whether or not the sequences define good matrices learn a clause blocking these sequences from occurring together again in the future.

If only a single instance of good matrices is desired then one can stop searching in Step 6 if a satisfying A, B, C, D is found. However, we learn a blocking clause and continue the search because we want to provide an exhaustive search.

The programmatic approach was essential to our algorithm because it allowed us to easily and efficiently apply Corollary 3 which would otherwise be difficult to apply in a SAT instance. Corollary 3 was also extremely effective; the SAT solver was usually able to learn a clause in Step 3 when enough variables were assigned. This enormously cut down the search space—the SAT solver could often learn conflicts with just n variables instead of conflicts with all $4n$ variables and in practice this appeared to give us an exponential speedup in n.

However, using this method alone was not sufficient for us to derive results beyond order 21 and a more efficient enumeration method (using the programmatic encoding as a subroutine) was required to scale to order 69. We describe this method in the next section.

4 Enumeration algorithm

Although SAT solvers are useful tools for searching large spaces they have their limits and the main reason that we were not able to derive any new results only using SAT solvers is because the search space is too large, even using the programmatic filtering encoding.

One way that the performance of SAT solvers can be improved on instances with large search spaces is by splitting the instance into many smaller instances of approximately equal difficulty. The cube-and-conquer paradigm (Heule, Kullmann, and Biere 2018) does this and has achieved some impressive successes such as solving the Boolean Pythagorean triples problem (Heule, Kullmann, and Marek 2016) and determining the fifth Schur number (Heule 2018).

Our enumeration algorithm will use a similar paradigm to cube-and-conquer except that instead of splitting the search space into *cubes* we split the search space into *compressions*. Not only does splitting via compressions increase the performance of the SAT solver it also allows for easy parallelization because instances generated in this way are independent. Additionally, we use Theorem 6 to apply filtering to the compressions which also has the effect of further decreasing the size of the search space.

We now describe our algorithm in four steps followed by some postprocessing. Given an odd order n divisible by 3 our algorithm enumerates all inequivalent circulant good matrices of order n with positive first entries. Each step is designed to search through a larger subspace than the previous step: step 1 finds every possible rowsum of a good matrix, step 2 finds every possible 3-compressed row of a good matrix, step 3 finds every possible 3-compressed quadruple of good matrices, and step 4 finds all good matrices up to equivalence.

Step 1: Find possible rowsums

If A, B, C, D are good matrices then from Corollary 4 and Lemma 5 the rowsums of the matrices B, C, and D must satisfy the quadratic Diophantine system

$$x^2 + y^2 + z^2 = 4n - 1$$
$$x \equiv y \equiv z \equiv n \pmod 4$$

where x, y, and z are the rowsums of B, C, and D.

Many computer algebra systems contain number theoretic functions that allow this Diophantine system to be easily solved. For example, the MATHEMATICA function `PowersRepresentations` or the MAPLE script NSOKS (Riel 2006).

Step 2: Find possible compressed rows

In this step we will generate sets S_{sk} and S_{sy} where S_{sk} will contain all possible 3-compressions of skew defining rows and S_{sy} will contain all possible 3-compressions of symmetric defining rows. These sets will be generated by an essentially brute-force enumeration through all skew or symmetric $\{\pm 1\}$-sequences of length n, using Corollaries 3 and 4 to filter sequences which cannot be a defining row.

Let $d \coloneqq \lfloor n/2 \rfloor$. In detail, we do the following:

1. For all $X \in \{\pm 1\}^d$:
   (a) Let A be the skew sequence whose first entry is 1 and next d entries are given by X, i.e., $A \coloneqq (1, x_0, \ldots, x_{d-1}, -x_{d-1}, \ldots, -x_0)$.
   (b) If $\mathrm{PSD}_A(k) \leq 4n$ for all k then add the 3-compression of A to S_{sk}.
   (c) Let B be the symmetric sequence whose first entry is 1 and next d entries are given by X, i.e., $B \coloneqq (1, x_0, \ldots, x_{d-1}, x_{d-1}, \ldots, x_0)$.
   (d) If $\mathrm{PSD}_B(k) \leq 4n$ for all k and $\mathrm{sum}(B)$ is one of the solutions x, y, z from step 1 then add the 3-compression of B to S_{sy}.

Step 3: Find possible compressed quadruples

In step 2 we generated all possible 3-compressions of the defining rows of good matrices. However, most quadruples of possible sequences found in step 2 will violate the relationship in Theorem 6 and can therefore be filtered. In this step we identify all quadruples of 3-compressions which can't be filtered and store them in a set S_{q}. We do this by examining all (A', B', C', D') in $S_{\mathrm{sk}} \times S_{\mathrm{sy}}^3$ and adding the quadruples that satisfy the relationship in Theorem 6 to S_{q}.

Step 4: Uncompress via programmatic SAT

In step 3 we generated all 3-compressed quadruples (A', B', C', D') that satisfy the relationship in Theorem 6. Given such a quadruple we need to determine if it is possible to *uncompress* the quadruple. In other words, we want to find all quadruples (A, B, C, D) of defining rows of good matrices (if any) whose 3-compression is (A', B', C', D'). To do this we formulate the uncompression problem as a programmatic SAT problem.

We already discussed our programmatic encoding of the constraint saying that A, B, C, D define rows of good matrices of order $n = 3m$. The remaining constraints can easily be expressed in the standard format accepted by SAT solvers (conjunctive normal form). We now discuss how to encode the constraint that they are 3-compressions of A', B', C', and D'. (For simplicity we only consider the compression constraints of $A' = (a'_0, \ldots, a'_{m-1})$ but the other constraints are analogous.)

Consider the entry $a'_k = a_k + a_{k+m} + a_{k+2m}$. There are four possible cases:

1. $a'_k = 3$. In this case $a_k = a_{k+m} = a_{k+2m} = 1$ which is encoded as the three unit clauses
$$a_k, \quad a_{k+m}, \quad a_{k+2m}.$$
2. $a'_k = 1$. In this case two of $\{a_k, a_{k+m}, a_{k+2m}\}$ are true and one is false. This is encoded as the four clauses
$$\neg a_k \vee \neg a_{k+m} \vee \neg a_{k+2m},$$
$$a_k \vee a_{k+m}, \quad a_k \vee a_{k+2m}, \quad a_{k+m} \vee a_{k+2m}.$$
3. $a'_k = -1$. In this case one of $\{a_k, a_{k+m}, a_{k+2m}\}$ is true and two are false. This is encoded as the four clauses
$$a_k \vee a_{k+m} \vee a_{k+2m},$$
$$\neg a_k \vee \neg a_{k+m}, \quad \neg a_k \vee \neg a_{k+2m}, \quad \neg a_{k+m} \vee \neg a_{k+2m}.$$
4. $a'_k = -3$. In this case $a_k = a_{k+m} = a_{k+2m} = -1$ which is encoded as the three unit clauses
$$\neg a_k, \quad \neg a_{k+m}, \quad \neg a_{k+2m}.$$

Finally, we discuss how to encode Corollary 8. The simplest case of Corollary 8 tells us that an even number of $\{b_m, c_m, d_m\}$ are true. This is encoded as the four clauses

$$b_m \vee c_m \vee d_m, \quad \neg b_m \vee \neg c_m \vee d_m,$$
$$\neg b_m \vee c_m \vee \neg d_m, \quad b_m \vee \neg c_m \vee \neg d_m.$$

The general case of Corollary 8 is an XOR constraint of length 5. While SAT solvers are not good with arbitrary XOR constraints we found that these constraints were short enough that they could be effectively encoded as the sixteen clauses with an even number of negated literals, i.e.,

$$a_k \vee a_{2k} \vee b_k \vee c_k \vee d_k, \ldots, a_k \vee \neg a_{2k} \vee \neg b_k \vee \neg c_k \vee \neg d_k.$$

Postprocessing: Remove equivalent good matrices

After all the SAT instances generated in step 4 are solved we will have a list of all circulant good matrices of order n up to equivalence. However, some of them may be equivalent to each other since we have not encoded all of the equivalence operations of Section 2. In order to provide a list which contains only inequivalent good matrices it is necessary to test the matrices for equivalence and remove any that are found to be equivalent to one which has previously been found.

To do this, we define a canonical representative of each equivalence class of defining rows of good matrices. This way, to check if two good matrices are equivalent we compute the canonical representative of each and check if they are equal.

First, consider the first two equivalence operations (reorder and negate). Given a quadruple of defining rows (A, B, C, D) we apply the negation operation to set the first entry of each defining row to be positive. Then we apply the reorder operation to sort B, C, D in lexicographically increasing order. This gives us a canonical representative of each equivalence class formed by the first two equivalence operations which we denote by $M_{A,B,C,D}$.

Next, consider the third equivalence operation (permute indices). Let σ be an automorphism of $\mathbb{Z}_n$ and let σ act on (A, B, C, D) by permuting the indices of the defining rows. This operation commutes with the first two operations, so the lexicographic minimum of the set

$$S \coloneqq \{\, M_{\sigma(A,B,C,D)} : \sigma \in \mathrm{Aut}(\mathbb{Z}_n) \,\}$$

is the lexicographic minimum of all quadruples equivalent to (A, B, C, D) and is therefore canonical. We compute this by explicitly generating all elements of S and selecting the lexicographic minimum.

Optimizations

For the benefit of those who would like to implement our algorithm we now discuss some optimizations that we found useful.

In the programmatic encoding and in step 2 of our enumeration algorithm we need to check if there exists an integer k such that $\mathrm{PSD}_X(k) > 4n$ for a given sequence X. We only need to check k with $0 \le k < n/2$ because the symmetries of the Fourier transform on real input X imply that $\mathrm{PSD}_X(k) = \mathrm{PSD}_X(\pm k \bmod n)$. In fact, checking $k = 0$ is also unnecessary because $\mathrm{PSD}_X(0) = \mathrm{sum}(X)^2$ and step 1 will filter any sequences X with $\mathrm{sum}(X)^2 > 4n$. Also, when checking the PSD criterion for each k we sort the elements of S so that the sequences with the largest PSD values appear first, making it easier to learn short clauses.

In step 3, the matching procedure can be done very efficiently by a string sorting technique similar to the method used in (Kotsireas, Koukouvinos, and Pardalos 2010). First we enumerate all $(A', B') \in S_{\mathrm{sk}} \times S_{\mathrm{sy}}$ that satisfy $\mathrm{PSD}_{A'}(k) + \mathrm{PSD}_{B'}(k) \le 4n$ for all k and all $(C', D') \in S^2_{\mathrm{sy}}$ that satisfy $\mathrm{PSD}_{C'}(k) + \mathrm{PSD}_{D'}(k) \le 4n$ for all k. At this point we form one list that contains strings that contain the values of $\mathrm{PSD}_{A'}(k) + \mathrm{PSD}_{B'}(k)$ and another list that contains strings that contain the values of $4n - (\mathrm{PSD}_{C'}(k) + \mathrm{PSD}_{D'}(k))$ and look for strings which occur in both lists. This can be done by sorting the two lists and doing a linear scan through the lists to find duplicates. The fact that PSDs are real numbers can make this tricky, but we can deal solely with integers by applying the inverse

Fourier transform to the PSD values; this produces integer values known as PAF values and the relationship given in Theorem 6 becomes (for $0 < k < n$)

$$\mathrm{PAF}_{A'}(k) + \mathrm{PAF}_{B'}(k) + \mathrm{PAF}_{C'}(k) + \mathrm{PAF}_{D'}(k) = 0.$$

In fact, since this step was not a bottleneck we computed the PAF values using the slower but more straightforward relationship $\mathrm{PAF}_X(k) = \sum_{j=0}^{n-1} x_j x_{j+k \bmod n}$.

In step 4, before calling the SAT solver we partition the compressed quadruples into equivalence classes using the reorder and automorphism equivalence operations from Section 2 and keep one quadruple from each equivalence class. If any satisfiable SAT instances are removed by this process they will have equivalent solutions (also with positive first entries) in the instance that was kept.

5 Results

In this section we discuss our implementation of our enumeration algorithm, the software and hardware we used to run it, and the results that we achieved. Our code is available from our website uwaterloo.ca/mathcheck.

Step 1 of our algorithm was done using NSOKS in MAPLE (Riel 2006). Steps 2 and 3 were done with custom C++ code and the string sorting was done with the GNU core utility SORT. Step 4 was done with the programmatic SAT solver MAPLESAT (Liang et al. 2016) and the PSD values were computed using the library FFTW (Frigo and Johnson 2005). Because FFTW uses floating point values which are inherently inaccurate all comparisons were checked to a tolerance of $\epsilon = 10^{-2}$ which is small but larger than the precision of the FFT used. The computations were run on a cluster of 64-bit Opteron 2.2GHz and Xeon 2.6GHz processors running CentOS 6.9 and using 500MB of memory. By far the most computationally expensive step of our algorithm was running the SAT solver. This accounted for 99.9% of the total running time, underscoring the importance of using a state-of-the-art SAT solver such as MAPLESAT. This step was parallelized across 250 cores and completed in about two weeks. Despite step 4 being the bottleneck the other steps were found to be essential could not be removed without significantly increasing the computation time.

The timings for running our implementation in each odd order divisible by 3 up to order 70 are given in Table 1. This table also contains the number of SAT instances generated in each order and the number of inequivalent good matrices found (denoted by $\#G_n$). Our exhaustive search in order 27 produced 13 sets of inequivalent good matrices, though Szekeres' classic paper (Szekeres 1988) contains only 12 sets of such matrices and Seberry reports that this was checked by at least one other researcher (Seberry 1999). Comparing the matrices generated using our method with those in his paper shows that Szekeres missed a set of good matrices. Furthermore, we found a new set of good matrices of order 57; the two new sets of good matrices produced by our method are given in Figure 1.

n	Time (h)	# inst.	$\#G_n$
3	0.00	1	1
9	0.00	2	1
15	0.00	11	11
21	0.00	39	10
27	0.00	186	13
33	0.01	840	15
39	0.07	1934	5
45	1.91	19205	4
51	11.35	23611	0
57	233.56	102402	1
63	11115.70	808642	0
69	72366.85	918940	0

Table 1: The running time in hours, number of SAT instances used, and number of inequivalent good matrices found ($\#G_n$) in each odd order $n < 70$ divisible by 3.

6 Conclusion

In this paper we have demonstrated the applicability of the SAT+CAS and programmatic SAT paradigms to the problem of finding good matrices in combinatorics. This problem has been well-studied for almost 50 years and the algorithm we've developed using SAT solvers coupled with computer algebra systems and mathematical libraries is a practical method of enumerating circulant good matrices (at least those with orders divisible by 3). This is evidenced by the large gap between the prior state-of-the-art and the results of this paper—prior to this year the largest order enumerated was 39 and the orders up to 49 were only enumerated this year, while our algorithm was able to enumerate circulant good matrices of order 69. Since the search space grows exponentially in the order (approximately like 4^n) this is a much larger search space than the order 49 search space.

Additionally, we've demonstrated the effectiveness of our algorithm by constructing two new sets of good matrices, including one that escaped detection in a 1988 search (Szekeres 1988) and a double-check by Koukouvinos in 1999 (Seberry 1999). It is surprising that this set of good matrices was overlooked for so long though the fact that the double-check also failed to produce it likely dissuaded researchers from spending more computational resources to triple-check the same search in order 27.

Unfortunately, the fact that there are no known certificates for an exhaustive search means that it is difficult to verify that a search completed successfully. In our case, we relied on multiple pieces of software including MAPLE, MAPLESAT, FFTW, GNU SORT, and custom-written C++ code. A bug in any one of these programs could cause a good matrix to be overlooked. The fact that our code confirms (and even corrects) the results of previous searches gives some assurance that it is working as intended. However, verification of our nonexistence results from multiple independent sources would be ideal.

Acknowledgments

We thank the reviewers for their detailed comments which improved this paper's clarity. The computations were made possible by the high-performance computer clusters administered by Compute Canada and SHARCNET.

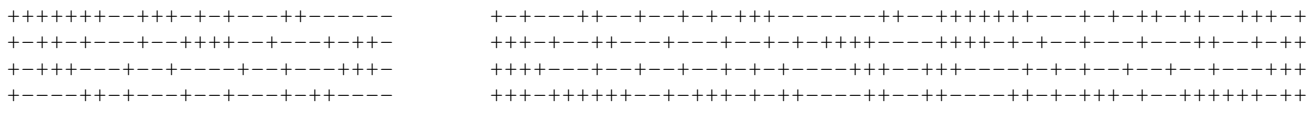

Figure 1: The defining rows of new good matrices of order 27 and order 57 where + encodes 1 and − encodes −1.

References

Ábrahám, E.; Abbott, J.; Becker, B.; Bigatti, A. M.; Brain, M.; Buchberger, B.; Cimatti, A.; Davenport, J. H.; England, M.; Fontaine, P.; Forrest, S.; Griggio, A.; Kroening, D.; Seiler, W. M.; and Sturm, T. 2016. SC^2: Satisfiability checking meets symbolic computation. In *Intelligent Computer Mathematics: 9th International Conference, CICM 2016, Bialystok, Poland, July 25–29, 2016, Proceedings*, 28–43.

Ábrahám, E. 2015. Building bridges between symbolic computation and satisfiability checking. In *Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation*, 1–6. New York: ACM.

Awyzio, G., and Seberry, J. 2015. On good matrices and skew Hadamard matrices. In Colbourn, C. J., ed., *Algebraic Design Theory and Hadamard Matrices*, 13–28.

Barrett, C.; Fontaine, P.; and Tinelli, C. 2016. The Satisfiability Modulo Theories Library (SMT-LIB).

Bright, C.; Kotsireas, I.; Heinle, A.; and Ganesh, V. 2018. Enumeration of complex Golay pairs via programmatic SAT. In *Proceedings of the 2018 ACM International Symposium on Symbolic and Algebraic Computation*, 111–118.

Bright, C.; Kotsireas, I.; and Ganesh, V. 2018. A SAT+CAS method for enumerating Williamson matrices of even order. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2–7, 2018*, 6573–6580.

Đoković, D. Ž., and Kotsireas, I. S. 2015. Compression of periodic complementary sequences and applications. *Designs, Codes and Cryptography* 74(2):365–377.

Đoković, D. Ž., and Kotsireas, I. S. 2018. Goethals–Seidel difference families with symmetric or skew base blocks. *Mathematics in Computer Science* 12(4):373–388.

Đoković, D. Ž. 1993. Good matrices of order 33, 35 and 127 exist. *J. Combin. Math. Combin. Comput* 14:145–152.

Đoković, D. Ž. 2008. Skew-Hadamard matrices of orders 188 and 388 exist. *International Mathematical Forum* 3(22):1063–1068.

Frigo, M., and Johnson, S. G. 2005. The design and implementation of FFTW3. *Proceedings of the IEEE* 93(2):216–231.

Ganesh, V.; O'Donnell, C. W.; Soos, M.; Devadas, S.; Rinard, M. C.; and Solar-Lezama, A. 2012. Lynx: A programmatic SAT solver for the RNA-folding problem. In *International Conference on Theory and Applications of Satisfiability Testing*, 143–156. Springer.

Georgiou, S.; Koukouvinos, C.; and Stylianou, S. 2002. On good matrices, skew Hadamard matrices and optimal designs. *Comput. Statist. Data Anal.* 41(1):171–184.

Heule, M. J.; Kullmann, O.; and Biere, A. 2018. Cube-and-conquer for satisfiability. In *Handbook of Parallel Constraint Reasoning*. 31–59.

Heule, M. J.; Kullmann, O.; and Marek, V. W. 2016. Solving and verifying the Boolean Pythagorean triples problem via cube-and-conquer. In *International Conference on Theory and Applications of Satisfiability Testing*, 228–245.

Heule, M. J. 2018. Schur number five. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2–7, 2018*, 6598–6606.

Hunt, D. C. 1972. Skew-Hadamard matrices of order less than 100. In *Proceedings of the First Australian Conference on Combinatorial Mathematics*, 23–27.

Kotsireas, I. S.; Koukouvinos, C.; and Pardalos, P. M. 2010. An efficient string sorting algorithm for weighing matrices of small weight. *Optimization Letters* 4(1):29–36.

Liang, J. H.; Ganesh, V.; Poupart, P.; and Czarnecki, K. 2016. Exponential Recency Weighted Average Branching Heuristic for SAT Solvers. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 3434–3440.

Riel, J. 2006. NSOKS: A MAPLE script for writing n as a sum of k squares.

Seberry, J. 1999. Library of good matrices. www.uow.edu.au/~jennie/good.html.

Szekeres, G. 1988. A note on skew type orthogonal ± 1 matrices. In *Combinatorics, Colloquia Mathematica Societatis, Janos Bolyai*, volume 52, 489–498.

Wallis, J. S. 1970. *Combinatorial matrices*. Ph.D. Dissertation, La Trobe University.

Wallis, J. S. 1971. A skew-Hadamard matrix of order 92. *Bulletin of the Australian Mathematical Society* 5:203–204.

Williamson, J. 1944. Hadamard's determinant theorem and the sum of four squares. *Duke Mathematical Journal* 11(1):65–81.

Zulkoski, E.; Bright, C.; Heinle, A.; Kotsireas, I. S.; Czarnecki, K.; and Ganesh, V. 2017. Combining SAT solvers with computer algebra systems to verify combinatorial conjectures. *J. Autom. Reasoning* 58(3):313–339.

Zulkoski, E.; Ganesh, V.; and Czarnecki, K. 2015. MATHCHECK: A math assistant via a combination of computer algebra systems and SAT solvers. In Felty, A. P., and Middeldorp, A., eds., *Automated Deduction - CADE-25*, volume 9195 of *Lecture Notes in Computer Science*. 607–622.

Zulkoski, E.; Ganesh, V.; and Czarnecki, K. 2016. MATHCHECK: A math assistant via a combination of computer algebra systems and SAT solvers. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, 4228–4233.