A SAT+CAS Method for Enumerating Williamson Matrices of Even Order

Curtis Bright[1], Ilias Kotsireas[2], and Vijay Ganesh[1]

[1] University of Waterloo
[2] Wilfrid Laurier University
{cbright, ikotsire, vganesh}@uwaterloo.ca

Abstract. We present for the first time a complete enumeration of Williamson matrices of even order $n < 45$. The enumeration method relies on the recently proposed SAT+CAS paradigm of coupling SAT solvers with computer algebra systems, thereby taking advantage of the advances made in both the field of satisfiability checking and the field of symbolic computation. Our results show that Williamson matrices of even order tend to be more abundant than those of odd orders; such matrices exist for every even order in which we performed a search but they do not exist for some odd orders.

1 Introduction

In 1944, the mathematician John Williamson introduced the type of matrix which now bear his name in the process of studying the Hadamard conjecture from combinatorial design theory [20]. This conjecture states that *Hadamard matrices*—matrices $H \in \{\pm 1\}^{n\times n}$ which satisfy $HH^T = nI_n$—exist for all orders n divisible by 4. Williamson defined a new collection of matrices which have since been extensively used to construct Hadamard matrices of many different orders, including some orders for which no other method has been able to work successfully. Williamson matrices have also been studied for their application to long-range digital communication systems and this motivated mathematicians from NASA's Jet Propulsion Laboratory to construct Williamson matrices of order 23 in 1961 [5].

Although Williamson defined his matrices for both even and odd orders, the even case has only been studied quite recently [7], though generalizations of Williamson matrices (sometimes referred to as Williamson-type matrices) have formerly been studied in even orders [18]. The algorithms used for enumerating Williamson matrices prior to 2016 (e.g., [4,13,16]) rely on properties only available in odd orders making it impossible to use them for enumerating Williamson matrices of even order. In light of this, it is interesting to develop algorithms for enumerating Williamson matrices which also work for even orders.

Unfortunately, it would not be possible to resolve the Hadamard conjecture by only studying Williamson matrices of even order, since Williamson matrices of even order can only be used to construct Hadamard matrices of orders which are divisible by 8. However, it is still not even known if Hadamard matrices exist for all orders divisible by 8, so nevertheless studying Williamson matrices of even order has the potential to shed light on the Hadamard conjecture as well.

On the other hand, if it was possible to show that Williamson matrices exist for all odd orders this would resolve the Hadamard conjecture, leading to the related conjecture that Williamson matrices exist in all odd orders. As Richard Turyn wrote [17]:

> It has been conjectured that an Hadamard matrix of this [Williamson] type might exist of every order $4t$, at least for t odd.

However, this conjecture was shown to be false by the mathematician Dragomir Đoković who showed that such matrices do not exist in order $t = 35$ [16]. Later, Holzmann, Kharaghani, and Tayfeh-Rezaie [13] showed that Williamson matrices also do not exist for orders 47, 53, and 59 but exist for all other odd orders under 60.

In this paper we provide for the first time[3] an enumeration of Williamson matrices in orders which are not odd. In doing so, we uncover an interesting but so far unexplained phenomenon that there tend to be more Williamson matrices in even orders than there are in odd orders. In fact, Williamson matrices exist for every even order in which we performed a search. In light of this, Turyn's remark stating that the Williamson conjecture should apply "at least for t odd" seems unnecessary. This leads us to propose what could be called the *updated Williamson conjecture*:

Conjecture 1. Williamson matrices of order t exist for all even t.

By itself, enumerating Williamson matrices of even order will never prove Conjecture 1. However, our enumeration could potentially uncover structure in Williamson sequences which might then be exploited to prove Conjecture 1, and if Williamson matrices of even order turn out to be very plentiful this gives some evidence for the truth of Conjecture 1.

The method we use to enumerate Williamson matrices of even order is based on the recently proposed SAT+CAS paradigm which uses tools and techniques from the fields of *satisfiability checking* and *symbolic computation*, as described in Section 3. As argued by the SC^2 project [2], these fields are complementary and by combining the tools of both fields (i.e., computer algebra systems and SAT solvers) in the right way we can solve problems more efficiently than we could by applying the tools of either field in isolation. Our method will be described in Section 4, followed by our results in Section 5. In particular, our results include the total number of Williamson matrices which exist in all even orders $n < 45$. These counts are given up to an equivalence which is described, along with many other properties of Williamson matrices, in Section 2.

2 Background

In this section we give the background on Williamson matrices and their properties which is necessary to understand the remainder of the paper.

[3] This content originally appeared in the PhD thesis of the first author [6].

2.1 Williamson matrices

The definition of Williamson matrices is motivated by the following theorem that Williamson used for constructing Hadamard matrices [20]:

Theorem 1. *Let $n \in \mathbb{N}$ and let A, B, C, $D \in \{\pm 1\}^{n\times n}$. Further, suppose that*

1. *A, B, C, and D are symmetric;*
2. *A, B, C, and D commute pairwise (i.e., $AB = BA$, $AC = CA$, etc.);*
3. *$A^2 + B^2 + C^2 + D^2 = 4nI_n$, where I_n is the identity matrix 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 Hadamard matrix of order $4n$.

To make the search for such matrices more tractable, and in particular to make condition 2 trivial, Williamson also required the matrices A, B, C, D to be circulant matrices, as defined below.

Definition 1. *An $n \times n$ matrix $A = (a_{ij})$ is circulant if $a_{ij} = a_{0,(j-i) \bmod n}$ for all i and $j \in \{0, \dots, n-1\}$.*

Circulant matrices A, B, C, D which satisfy the conditions of Theorem 1 are known as *Williamson matrices* in honor of Williamson. Since Williamson matrices are circulant they are defined in terms of their first row $[x_0, \dots, x_{n-1}]$ and since they are symmetric this row must be a symmetric sequence, i.e., satisfy $x_i = x_{n-i}$ for $1 \le i < n$. Given these facts, it is often convenient to work in terms of sequences rather than matrices. When working with sequences in this context the following function becomes very useful.

Definition 2. *The* periodic autocorrelation function *of $A = [a_0, \dots, a_{n-1}]$ is the function given by*

$$\mathrm{PAF}_A(s) := \sum_{k=0}^{n-1} a_k a_{(k+s) \bmod n}.$$

We also use PAF_A to refer to a sequence containing the values of the above function (which has period n), i.e.,

$$\mathrm{PAF}_A := \big[\mathrm{PAF}_A(0), \dots, \mathrm{PAF}_A(n-1)\big].$$

This function allows us to easily give a definition of Williamson matrices in terms of sequences.

Definition 3. *Four symmetric sequences A, B, C, $D \in \{\pm 1\}^n$ are called* Williamson sequences *if they satisfy the equations*

$$\mathrm{PAF}_A(s) + \mathrm{PAF}_B(s) + \mathrm{PAF}_C(s) + \mathrm{PAF}_D(s) = 0 \qquad (1)$$

for $s = 1, \dots, \lfloor n/2 \rfloor$.

It is straightforward to see that there is an equivalence between such sequences and Williamson matrices (e.g., see [6, §3.1]) so for the remainder of this paper we will work with these sequences instead of Williamson matrices. Note that by PAF symmetry and periodicity one has that Williamson sequences satisfy (1) for all $s \not\equiv 0 \pmod{n}$. In the remaining case when $s \equiv 0 \pmod{n}$ one trivially has that

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

2.2 Williamson equivalences

Given a Williamson sequence A, B, C, D of even order n, there are four types of invertible operations which can be applied to produce another Williamson sequence. These operations allow us to define *equivalence classes* of Williamson sequences. If a single Williamson sequence is known it is easy to generate all Williamson sequences in the same equivalence class, so it suffices to search for Williamson sequences up to these equivalence operations.

1. (Reorder) Reorder the sequences A, B, C, D in any way.
2. (Negate) Negate all the entries of any of A, B, C, or D.
3. (Shift) Cyclically shift all the entries in any of A, B, C, or D by an offset of $n/2$.
4. (Permute entries) Apply an automorphism of the cyclic group C_n to all the entries of each of A, B, C, and D simultaneously.

These equivalence operations are well-known [13] except for the shift operation which has not traditionally been used because it only applies when n is even. In fact, it was overlooked until our enumeration method produced many sequences which were cyclic shifts of each other.

2.3 Fourier analysis

In this section we give an alternative characterization of Williamson sequences using concepts from Fourier analysis. First, we recall the definition of the discrete Fourier transform and use it to state the definition of the power spectral density.

Definition 4. *The* discrete Fourier transform *of the sequence $A = [a_0, \dots, a_{n-1}]$ is the function*

$$\mathrm{DFT}_A(s) \coloneqq \sum_{k=0}^{n-1} a_k e^{2\pi i k s/n}$$

for $s = 0, \dots, n-1$. Equivalently, we may also consider the discrete Fourier transform to be a sequence containing the values of the above function, i.e.,

$$\mathrm{DFT}(A) \coloneqq \big[\mathrm{DFT}_A(0), \dots, \mathrm{DFT}_A(n-1)\big].$$

Definition 5. *The* power spectral density *of the sequence $A = [a_0, \dots, a_{n-1}]$ is the function*

$$\mathrm{PSD}_A(s) \coloneqq \bigl|\mathrm{DFT}_A(s)\bigr|^2.$$

Equivalently, we may also consider the power spectral density to be a sequence containing the values of the above function, i.e.,

$$\mathrm{PSD}_A \coloneqq \bigl[\mathrm{PSD}_A(0), \dots, \mathrm{PSD}_A(n-1)\bigr].$$

Note that $\mathrm{DFT}_A(s)$ is equal to the complex conjugate of $\mathrm{DFT}_A(n-s)$ and so we have that $\mathrm{PSD}_A(s) = \mathrm{PSD}(n-s)$.

The following theorem first shown by Wiener [19] and Khinchin [14] is of central importance to us because it provides a relationship between the PAF and PSD values.

Theorem 2 (Wiener–Khinchin). *For every sequence $A \in \mathbb{C}^n$ we have that*

$$\mathrm{PSD}_A = \mathrm{DFT}\bigl(\mathrm{PAF}_A\bigr).$$

The Wiener–Khinchin theorem allows us to give the following alternative characterization of Williamson sequences which can be viewed as a definition of Williamson sequences which does not make reference to PAF values.

Corollary 1. *Four symmetric sequences A, B, C, $D \in \{\pm 1\}^n$ are Williamson sequences if and only if*

$$\mathrm{PSD}_A(s) + \mathrm{PSD}_B(s) + \mathrm{PSD}_C(s) + \mathrm{PSD}_D(s) = 4n \tag{2}$$

for $s = 1, \dots, \lfloor n/2 \rfloor$.

Proof. If A, B, C, D are Williamson sequences then by definition and symmetry of the PAF we have that

$$\mathrm{PAF}_A + \mathrm{PAF}_B + \mathrm{PAF}_C + \mathrm{PAF}_D = [4n, 0, \dots, 0].$$

Applying the discrete Fourier transform to both sides of this and using the Wiener–Khinchin theorem along with the linearity of the DFT we have that

$$\mathrm{PSD}_A + \mathrm{PSD}_B + \mathrm{PSD}_C + \mathrm{PSD}_D = [4n, 4n, \dots, 4n].$$

The reverse direction is proven in a similar manner except that one applies the inverse of the discrete Fourier transform.

Corollary 2. *If $\mathrm{PSD}_A(s) > 4n$ for any value s then A cannot be part of a Williamson sequence.*

Proof. Since PSD values are nonnegative, if $\mathrm{PSD}_A(s) > 4n$ then the relationship (2) cannot hold and thus A cannot be part of a Williamson sequence.

Similarly, one can easily extend Corollary 2 to apply to more than one sequence at a time, as in the following corollary.

Corollary 3. *If $\mathrm{PSD}_A(s) + \mathrm{PSD}_B(s) > 4n$ for any value of s then A and B do not occur together in a Williamson sequence and if $\mathrm{PSD}_A(s) + \mathrm{PSD}_B(s) + \mathrm{PSD}_C(s) > 4n$ for any value of s then A, B, and C do not occur together in a Williamson sequence.*

2.4 Compression

As in the work [10] we now introduce the notion of *compression*.

Definition 6. *Let $A = [a_0, a_1, \dots, a_{n-1}]$ be a sequence of length $n = dm$ and set*

$$a_j^{(d)} = a_j + a_{j+d} + \cdots + a_{j+(m-1)d}, \qquad j = 0, \dots, d-1.$$

*Then we say that the sequence $A^{(d)} = [a_0^{(d)}, a_1^{(d)}, \dots, a_{d-1}^{(d)}]$ is the m-*compression *of A.*

The following theorem tells us that the PSD values of a compression of A will be a subset of the PSD values of A.

Theorem 3. *Let $A = [a_0, a_1, \dots, a_{n-1}]$ be a sequence of length $n = dm$. Then*

$$\operatorname{PSD}_{A^{(d)}}(s) = \operatorname{PSD}_A(ms).$$

Proof. By the Wiener–Khinchin theorem and the fact that $m/n = 1/d$ we have

$$\operatorname{PSD}_A(ms) = \sum_{k=0}^{n-1} \operatorname{PAF}_A(k) e^{2\pi iks/d}.$$

Since $\{e^{2\pi iks/d}\}_k$ has period d, this is equal to

$$\sum_{k=0}^{d-1} \sum_{l=0}^{m-1} \operatorname{PAF}_A(k + ld) e^{2\pi iks/d}.$$

Also note that $\operatorname{PAF}_{A^{(d)}}(k) = \sum_{l=0}^{m-1} \operatorname{PAF}_A(k + ld)$, so this becomes

$$\sum_{k=0}^{d-1} \operatorname{PAF}_{A^{(d)}}(k) e^{2\pi iks/d}$$

which is $\operatorname{PSD}_{A^{(d)}}(s)$ by the Wiener–Khinchin theorem.

For example, the PSD values of a 2-compression of a sequence A will be the entries of PSD_A which have even index. Thus, the compression of a sequence will have fewer PSD values than the original sequence has, however, the values which it does have will be PSD values from the original sequence. It follows that the PSD properties of Williamson sequences are invariant under compression and we have the following corollary.

Corollary 4. *If A, B, C, D is a Williamson sequence of order n then*

$$\operatorname{PSD}_{A'} + \operatorname{PSD}_{B'} + \operatorname{PSD}_{C'} + \operatorname{PSD}_{D'} = [4n, \dots, 4n]$$

for any compression A', B', C', D' of that Williamson sequence.

Finally, the following property from [7] will be useful.

Lemma 1. *If A is a sequence of length $n = dm$ with ± 1 entries, then the entries of the m-compression of A are congruent to $m \pmod 2$ and have absolute value at most m.*

If one uses Corollary 4 and Lemma 1 with maximal (i.e., n-) compression then obtains the following result.

Corollary 5. *If A, B, C, D is a Williamson sequence of order n then*

$$R_A^2 + R_B^2 + R_C^2 + R_D^2 = 4n \quad \textit{and} \quad R_A \equiv R_B \equiv R_C \equiv R_D \equiv n \pmod 2, \quad (3)$$

where R_X denotes the rowsum of X.

Proof. Let X' be the n-compression of $X \in \{\pm 1\}^n$, i.e., X' is a sequence with one entry whose value is R_X. Note that $\mathrm{PSD}_{X'} = [R_X^2]$, so by Corollary 4 one derives the first equation in (3). The second equation is derived by noting that Lemma 1 on X with $m = n$ implies that $R_X \equiv n \pmod 2$.

3 The SAT+CAS paradigm

The SAT+CAS paradigm is a novel methodology for solving problems which originated independently in two works published in 2015:

1. A paper at the *Conference on Automated Deduction* (CADE) by Edward Zulkoski, Vijay Ganesh, and Krzysztof Czarnecki [21] entitled "MathCheck: A Math Assistant via a Combination of Computer Algebra Systems and SAT Solvers".
2. An invited talk at the *International Symposium on Symbolic and Algebraic Computation* (ISSAC) by Erika Ábrahám [1] entitled "Building Bridges between Symbolic Computation and Satisfiability Checking".

The CADE paper describes the tool MathCheck as combining the search capability of SAT solvers with the domain knowledge of CAS systems. The paper made the case that MathCheck

> . . . combines the efficient search routines of modern SAT solvers, with the expressive power of CAS, thus complementing both.

Indeed, SAT solvers contain some of the best general-purpose search procedures ever developed. While they do not perform well for all applications, the CADE paper showed that they can be made more efficient if they are supplied with appropriate domain-specific knowledge, such as the knowledge available in a CAS.

Independently from the work done on MathCheck the computer scientist Erika Ábrahám made the observation in her ISSAC 2015 invited talk that the symbolic computation and satisfiability checking communities have similar goals but the way in which they approach and solve problems is rather different. She remarked that

…collaboration between symbolic computation and SMT solving is still (surprisingly) quite restricted…

and made the case that the communities would benefit from increased mutual discussion. Furthermore, she argued that developing algorithms and tools which combine the strengths and insights from both these fields is a promising line of research which could be beneficial to both communities.

3.1 Programmatic SAT

The idea of a *programmatic* SAT solver was introduced in the paper [12]. A programmatic SAT solver can generate conflict clauses programmatically, i.e., by a piece of code which runs as the SAT solver carries out its search. Such a SAT solver can learn clauses which are more useful than the conflict clauses which it learns by default. Not only can this make the SAT solver's search more efficient, it allows for increased expressiveness as many types of constraints which are awkward to express in a conjunctive normal form format can naturally be expressed using code. Additionally, it allows one to compile *instance-specific* SAT solvers which are tailored to solving one specific type of instance. In this framework instances no longer have to solely consist of a set of clauses in conjunctive normal form. Instead, instances can consist of both a set of CNF clauses and a piece of code which encodes constraints which are too cumbersome to be written in CNF format.

As an example of this, consider the case of searching for Williamson sequences using a SAT solver. One could encode Definition 3 in CNF format by using Boolean variables to represent the entries in the Williamson sequences and by using binary adders to encode the summations; this was the method used in [7]. However, one could also use the equivalent definition given in Corollary 1. This alternate definition has the advantage that it becomes easy to apply Corollaries 2 and 3, which allows one to filter many sequences from consideration and greatly speed up the search. Because of this, our method will use the constraints (2) from Corollary 1 to encode the definition of Williamson sequences in our SAT instances.

However, encoding the equations in (2) would be extremely cumbersome to do using CNF clauses, because of the involved nature of efficiently computing the PSD values. However, the equations (2) are easy to express programmatically, as long as one has a method of computing the PSD values. Furthermore, the PSD values can be computed efficiently using the fast Fourier transform which is available in many computer algebra systems and mathematical libraries.

Thus, our SAT instances will not use CNF clauses to encode the defining property of Williamson sequences but instead encode those clauses programmatically. This is done by writing a *callback function* which is compiled with the SAT solver and programmatically expresses the constraints in Corollary 1, as well as the filtering criteria in Corollaries 2 and 3.

3.2 Programmatic Williamson encoding

We now describe in detail our programmatic encoding of Williamson sequences. The encoding takes the form of a piece of code which examines a partial assignment to the variables defining the sequences A, B, C, and D. In the case when the partial assignment can be ruled out using Corollaries 2 or 3, a conflict clause is returned which encodes a reason why the partial assignment no longer needs to be considered. If the sequences actually form a Williamson sequence then they are recorded in an auxiliary file; at this point the solver can return SAT and stop, though our implementation continues the search because we are not just searching for a single solution and want to do a complete enumeration of the space.

The programmatic callback function does the following:

1. Initialize $S \coloneqq \emptyset$. This variable will be a set which contains the sequences whose entries are all currently assigned.
2. Check if all the variables which define the entries in sequence A have been assigned; if so, add A to S and compute PSD_A, otherwise skip to the next step. If $\mathrm{PSD}_A(s) > 4n$ for some value of s then learn a clause prohibiting the entries of A from being assigned the way they currently are, i.e., learn the clause
$$\lnot(a_0^{\mathrm{cur}} \land a_1^{\mathrm{cur}} \land \cdots \land a_{n-1}^{\mathrm{cur}}) \equiv \lnot a_0^{\mathrm{cur}} \lor \lnot a_1^{\mathrm{cur}} \lor \cdots \lor \lnot a_{n-1}^{\mathrm{cur}}$$
where a_i^{cur} is the literal a_i when a_i is currently assigned to true and is the literal $\lnot a_i$ when a_i is currently assigned to false.
3. Check if all the variables which define the entries in sequence B have been assigned; if so, add B to S and compute PSD_B. If there is some s such that $\sum_{X\in S} \mathrm{PSD}_X(s) > 4n$ then learn a clause prohibiting the values of the sequences in S from being assigned the way they currently are.
4. Repeat the last step again twice, once with B replaced with C and then again with B replaced with D.
5. If all the variables in sequences A, B, C, and D are assigned then examine the values
$$\mathrm{PSD}_A(s) + \mathrm{PSD}_B(s) + \mathrm{PSD}_C(s) + \mathrm{PSD}_D(s)$$
for $s = 1, \ldots, \lfloor n/2 \rfloor$. If this value is always $4n$ then record the sequences in an auxiliary file, otherwise they are not Williamson sequences and can be discarded. In either case, learn a clause prohibiting the values of the sequences from being assigned the way they currently are so that this assignment is not examined again.

After the SAT solver has completed its search the auxiliary file will contain a list of the Williamson sequences which were found during the search. Note that the clauses learned by this function allow the SAT solver to execute the search significantly faster than would be possible using a brute-force technique. As a rough estimate of the benefit, note that there are approximately $2^{n/2}$ possibilities for each member A, B, C, D in a Williamson sequence. If no clauses are learned in steps 2–4 then the SAT solver will examine all $2^{4(n/2)}$ total possibilities. Conversely, if a clause is always learned in step 2 then the SAT solver will only need to examine the $2^{n/2}$ possibilities for A. Of course, one will not always learn a clause in step 2, but in practice such a clause is learned quite frequently and this more than makes up for the small overhead of computing the PSD values.

4 Our enumeration algorithm

In this section we give a complete description of our method which enumerates all Williamson sequences of a given order n when n is assumed to be even.

4.1 Step 1: Generate possible sum-of-squares decompositions

First, note that by Corollary 5 every Williamson sequence gives rise to a decomposition of $4n$ into a sum of four squares. We query a computer algebra system such as Maple or Mathematica to get all possible solutions of the Diophantine system (3). Because we only care about Williamson sequences up to equivalence, we also add the inequalities

$$0 \leq R_A \leq R_B \leq R_C \leq R_D$$

to the Diophantine system; it is clear that any Williamson sequence can be transformed into another Williamson sequence which satisfies these inequalities by applying the reorder and/or negate equivalence operations.

Example 1. When $n = 44$ there are exactly two solutions to the sum-of-squares Diophantine system, namely

$$R_A = 0, R_B = 4, R_C = 4, R_D = 12 \quad \text{and} \quad R_A = 2, R_B = 6, R_C = 6, R_D = 10.$$

4.2 Step 2: Generate possible Williamson sequence members

Next, we form a list of the sequences which could possibly appear as a member of a Williamson sequence of order n. To do this, we examine every symmetric sequence $X \in \{\pm 1\}^n$. For all such X we compute PSD_X and ignore those which satisfy $\mathrm{PSD}_X(s) > 4n$ for some s. We also ignore those X for which R_X does not appear in any possible solution (R_A, R_B, R_C, R_D) of the sum-of-squares Diophantine system (3). The sequences X which remain after this process form a list of the sequences which could possibly appear as a member of a Williamson sequence. At this stage we could generate all Williamson sequences of order n by trying all ways of grouping the possible sequences X into quadruples and filtering those which are not Williamson. However, because of the large number of ways in which this grouping into quadruples can be done this is not feasible to do except in the case when n is very small.

4.3 Step 3: Perform compression

In order to reduce the size of the problem so that the possible sequences generated in Step 2 can be grouped into quadruples we first compress the sequences using the process described in Section 2.4. For each solution (a, b, c, d) of the sum-of-squares Diophantine system we form four lists L_A, L_B, L_C, and L_D. The list L_A will contain the 2-compressions of the sequences X generated in Step 2 which have rowsum a (and the lists L_B, L_C, L_D will be defined in a similar manner). Note that the sequences in these lists will be $\{\pm 2, 0\}$-sequences since they are 2-compressions of the sequences X which are $\{\pm 1\}$-sequences.

4.4 Step 4: Match the compressions

By Corollary 4 a necessary condition for A, B, C, D to be a Williamson sequence is that

$$\mathrm{PSD}_{A'} + \mathrm{PSD}_{B'} + \mathrm{PSD}_{C'} + \mathrm{PSD}_{D'} = [4n, \dots, 4n]$$

where A', B', C', D' are the 2-compressions of A, B, C, D. By construction, the lists L_A, L_B, L_C, and L_D contain all possible 2-compressions of the members of Williamson sequences whose sum-of-squares decomposition is $a^2 + b^2 + c^2 + d^2$. Thus, by trying all ways of matching together the sequences from the lists L_A, L_B, L_C, L_D we can find all 2-compressions of Williamson sequences whose sum-of-squares decomposition is $a^2 + b^2 + c^2 + d^2$. In fact, some matchings can be eliminated without being explicitly considered, if for example two or three sequences in the matching have PSD values which sum to a number larger than $4n$.

In detail, our matching procedure performs the following steps:

```
1:  for $A' \in L_A$ do
2:      for $B' \in L_B$ do
3:          if $\max(\mathrm{PSD}_{A'} + \mathrm{PSD}_{B'}) > 4n$ then
4:              continue for loop with new $B'$
5:          for $C' \in L_C$ do
6:              if $\max(\mathrm{PSD}_{A'} + \mathrm{PSD}_{B'} + \mathrm{PSD}_{C'}) > 4n$ then
7:                  continue for loop with new $C'$
8:              for $D' \in L_D$ do
9:                  if $\mathrm{PSD}_{A'} + \mathrm{PSD}_{B'} + \mathrm{PSD}_{C'} + \mathrm{PSD}_{D'} = [4n, \dots, 4n]$ then
10:                     record $(A', B', C', D')$ in an auxiliary file
```

Note that the sequences in the lists have PSD values at most $4n$ by construction, so it is not necessary to make a check if $\max(\mathrm{PSD}_{A'}) > 4n$. At the conclusion of this matching procedure we will have a list of all the possible 2-compressions of Williamson sequences whose sum-of-squares decomposition is $a^2 + b^2 + c^2 + d^2$.

4.5 Step 5: Uncompress the 2-compressions

It is now necessary to find the Williamson sequences, if any, which when compressed by a factor of 2 produce one of the sequences generated in Step 4. In other words, we want to find a way to perform uncompression on the sequences which we generated. To do this, we formulate the uncompression problem as a Boolean SAT instance and use a SAT solver's sophisticated search ability to search for solutions to the uncompression problem.

We will use Boolean variables to represent the entries of the uncompressed Williamson sequences, with true representing the value of 1 and false representing the value of -1. Since Williamson sequences consist of four sequences of length n they contain a total of $4n$ entries, namely,

$$a_0, \dots, a_{n-1}, b_0, \dots, b_{n-1}, c_0, \dots, c_{n-1}, d_0, \dots, d_{n-1}.$$

However, because Williamson sequences are symmetric we actually only need to define the $2n+4$ distinct variables

$$a_0, \dots, a_{n/2}, b_0, \dots, b_{n/2}, c_0, \dots, c_{n/2}, d_0, \dots, d_{n/2}.$$

Any variable x_i with $i > n/2$ can simply be replaced with the equivalent variable x_{n-i}; in what follows we implicitly use this substitution when necessary. Thus, the SAT instances which we generate will contain $2n+4$ variables.

Say that (A', B', C', D') is one of the possible 2-compressions generated in Step 4. By the definition of 2-compression, we have that $a'_i = a_i + a_{i+n/2}$ and similarly for the entries of B', C', and D'. Since $a'_i \in \{\pm 2, 0\}$ there are three possibilities we must consider for each a'_i.

Case 1. If $a'_i = 2$ then we must have $a_i = 1$ and $a_{i+n/2} = 1$. Thinking of the entries as Boolean variables, we add the clause

$$a_i \wedge a_{i+n/2}$$

to our SAT instance.

Case 2. If $a'_i = -2$ then we must have $a_i = -1$ and $a_{i+n/2} = -1$. Thinking of the entries as Boolean variables, we add the clause

$$\neg a_i \wedge \neg a_{i+n/2}$$

to our SAT instance.

Case 3. If $a'_i = 0$ then we must have $a_i = 1$ and $a_{i+n/2} = -1$ or vice versa. Thinking of the entries as Boolean variables, we add the clause

$$(a_i \vee a_{i+n/2}) \wedge (\neg a_i \vee \neg a_{i+n/2})$$

to our SAT instance. Note that this clause specifies in conjunctive normal form that exactly one of the variables a_i and $a_{i+n/2}$ is true.

For each entry a'_i in A' we add the clauses from the appropriate case to the SAT instance, as well as add clauses from a similar case analysis for the entries from B', C', and D'. A satisfying assignment to the generated SAT instance provides an uncompression (A, B, C, D) of (A', B', C', D'). However, the uncompression need not be a Williamson sequence. To ensure that the solutions produced by the SAT solver are in fact Williamson sequences we additionally use the programmatic SAT Williamson encoding as described in Section 3.2.

For each (A', B', C', D') generated in Step 4 we generate a SAT instance which contains the set of clauses specified above as well as the programmatic clause generator which specifies that any satisfying assignment of the SAT instance encodes a Williamson sequence. We then solve the SAT instances using a SAT solver which performs an exhaustive search to find all the solutions in all the SAT instances.

As an optimization, one can ignore any SAT instance whose 2-compression (A', B', C', D') can be transformed into the 2-compression of another SAT instance using the equivalence operations from Section 2.2. In this case the solutions from the ignored SAT instance will have equivalent solutions in the equivalent SAT instance (generated by applying the same equivalence operations from Section 2.2).

4.6 Step 6: Remove equivalent Williamson sequences

After Step 5 we have produced a list of all the Williamson sequences of order n which have a certain sum-of-squares decompositions. We chose the decompositions in such a way that every Williamson sequence will be equivalent to one decomposition but this does not cover all possible equivalences, so some Williamson sequences which we generate may be equivalent to each other.

For the purpose of counting the total number of inequivalent Williamson sequences which exist in order n it is necessary to examine each Williamson sequence in the list and determine if it is equivalent to another Williamson sequence in the list. This can be done by repeatedly applying the equivalence operations from Section 2.2 on the Williamson sequences in the list and discarding those which are equivalent to a previously found Williamson sequence.

5 Results

We implemented the algorithm described in Section 4 and ran it on all even orders $n < 45$. Step 1 was completed using the computer algebra system MAPLE [9]. Steps 2–4 were completed using custom C++ code which used the library FFTW [11] for computing PSD values. Step 5 was completed using the SAT solver MAPLESAT [15] modified to support a programmatic interface and also used the library FFTW for computing PSD values on-the-fly. Step 6 was completed using custom C++ code.

Our results were run on the "Shared Hierarchical Academic Research Computing Network", a high-performance computing cluster known as SHARCNET [3] run by a consortium of 18 academic partners located in Ontario, Canada. Specifically, the cluster we used ran CentOS 6.7 and used 64-bit AMD Opteron processors running at 2.2 GHz. The timings for running our algorithm on each even order up to 44 are given in Table 1. We only give timings for Steps 4 and 5, as they were the only steps which required significant time to complete; the other steps ran in at most a few seconds, even on the largest orders. Table 1 also includes the number of SAT instances which we generated in each order, as well as the total number of Williamson sequences which were found up to equivalence[4] (denoted by $\#W_n$). An explicit enumeration of these Williamson sequences is available online [8].

n	Step 4 time	Step 5 time	# instances	$\#W_n$
2	0s	0.08s	1	1
4	0s	0.02s	1	1
6	0s	0.02s	1	1
8	0s	0.02s	1	1
10	0s	0.04s	2	2
12	0s	0.08s	3	3
14	0s	0.07s	3	7
16	0s	0.14s	5	6
18	0s	0.49s	22	40
20	0.01s	0.49s	21	27
22	0.01s	0.48s	22	27
24	0.03s	3.48s	176	80
26	0.42s	0.69s	24	38
28	0.59s	2.07s	78	99
30	8.15s	6.33s	281	268
32	3.58s	22.60s	1064	200
34	250.31s	5.48s	214	160
36	372.7s	66.26s	1705	691
38	4053.24s	10.97s	360	87
40	9121.86s	1727.02s	40924	1898
42	85017.28s	148.06s	2945	561
44	65492.84s	134.97s	1523	378

Table 1: A table summarizing the amount of time used to enumerate all Williamson sequences of order n, as well as the number of SAT instances generated and the total number of inequivalent Williamson sequences found (denoted by $\#W_n$).

6 Conclusion

In this paper we have shown the power of the SAT+CAS paradigm (i.e., the technique applying the tools from the fields of symbolic computation and satisfiability checking [2]) as well as the power and flexibility of the programmatic SAT approach [12]. We have done this by developing a programmatic SAT+CAS method to solve the long-standing problem of generating Williamson matrices of even order. This problem has been well-studied since 1944 in the odd order case and counts for the number of Williamson matrices up to equivalence have

[4] These results originally appeared in [6] but the notion of equivalence in that work did not include the shift equivalence operation.

been published for all odd orders up to 59 [13] but this is the first time counts for even orders have ever been published.

Our work reveals that there are typically many more Williamson matrices in even orders than there are in odd orders. In fact, every odd order n in which a search has been carried out has $\#W_n \leq 10$, while we have shown that every even order n between 18 and 44 has $\#W_n > 10$ and there are many orders which contain hundreds of inequivalent Williamson matrices. A theoretical reason which could explain this dichotomy would be interesting, though we currently know of no such reason. We hope that our work brings attention to this problem which could lead to a better understanding of the behaviour of Williamson matrices of even order.

References

1. Ábrahám, E.: Building bridges between symbolic computation and satisfiability checking. In: Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation. pp. 1–6. ACM, New York (2015)
2. 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., Sturm, T.: SC^2: Satisfiability Checking meets Symbolic Computation (Project Paper). In: Intelligent Computer Mathematics: 9th International Conference, CICM 2016, Bialystok, Poland, July 25–29, 2016, Proceedings. pp. 28–43. Springer International Publishing, Cham (2016), `http://www.sc-square.org/`
3. Bauer, M.: A Message from the Scientific Director. `https://www.sharcnet.ca/my/about/sd_message` (2016)
4. Baumert, L., Hall, M.: Hadamard matrices of the Williamson type. Mathematics of Computation 19(91), 442–447 (1965)
5. Baumert, L., Golomb, S.W., Hall, M.: Discovery of an Hadamard matrix of order 92. Bull. Amer. Math. Soc. 68(3), 237–238 (1962)
6. Bright, C.: Computational Methods for Combinatorial and Number Theoretic Problems. Ph.D. thesis, University of Waterloo (2017)
7. Bright, C., Ganesh, V., Heinle, A., Kotsireas, I.S., Nejati, S., Czarnecki, K.: MathCheck2: A SAT+CAS Verifier for Combinatorial Conjectures. In: Computer Algebra in Scientific Computing - 18th International Workshop, CASC 2016, Bucharest, Romania, September 19–23, 2016, Proceedings. pp. 117–133 (2016)
8. Bright, C., Kotsireas, I., Ganesh, V.: Enumeration of Williamson sequences of even order (Jul 2017), `https://doi.org/10.5281/zenodo.825339`
9. Char, B.W., Fee, G.J., Geddes, K.O., Gonnet, G.H., Monagan, M.B.: A tutorial introduction to Maple. Journal of Symbolic Computation 2(2), 179–200 (1986)
10. Đoković, D.Ž., Kotsireas, I.S.: Compression of periodic complementary sequences and applications. Designs, Codes and Cryptography 74(2), 365–377 (2015)
11. Frigo, M., Johnson, S.G.: The design and implementation of FFTW3. Proceedings of the IEEE 93(2), 216–231 (2005)
12. Ganesh, V., O'Donnell, C.W., Soos, M., Devadas, S., Rinard, M.C., Solar-Lezama, A.: Lynx: A programmatic SAT solver for the RNA-folding problem. In: Theory and Applications of Satisfiability Testing–SAT 2012, pp. 143–156. Springer (2012)
13. Holzmann, W.H., Kharaghani, H., Tayfeh-Rezaie, B.: Williamson matrices up to order 59. Designs, Codes and Cryptography 46(3), 343–352 (2008)

14. Khintchine, A.: Korrelationstheorie der stationären stochastischen prozesse. Mathematische Annalen 109(1), 604–615 (1934)
15. Liang, J.H., Ganesh, V., Poupart, P., Czarnecki, K.: Exponential Recency Weighted Average Branching Heuristic for SAT Solvers. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. pp. 3434–3440. AAAI'16, AAAI Press (2016), `http://dl.acm.org/citation.cfm?id=3016100.3016385`
16. Đoković, D.Ž.: Williamson matrices of order $4n$ for $n = 33, 35, 39$. Discrete mathematics 115(1), 267–271 (1993)
17. Turyn, R.J.: An infinite class of Williamson matrices. Journal of Combinatorial Theory, Series A 12(3), 319–321 (1972)
18. Wallis, J.S.: Williamson matrices of even order. In: Holton, D.A. (ed.) Combinatorial Mathematics: Proceedings of the Second Australian Conference, pp. 132–142. Springer Berlin Heidelberg, Berlin, Heidelberg (1974)
19. Wiener, N.: Generalized harmonic analysis. Acta mathematica 55(1), 117–258 (1930)
20. Williamson, J.: Hadamard's Determinant Theorem and the Sum of Four Squares. Duke Math. J 11(1), 65–81 (1944)
21. Zulkoski, E., Ganesh, V., Czarnecki, K.: MathCheck: A Math Assistant via a Combination of Computer Algebra Systems and SAT Solvers. In: Felty, A.P., Middeldorp, A. (eds.) Automated Deduction - CADE-25, Lecture Notes in Computer Science, vol. 9195, pp. 607–622. Springer International Publishing (2015)