The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

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

Curtis Bright
University of Waterloo

Ilias Kotsireas
Wilfrid Laurier University

Vijay Ganesh
University of Waterloo

Abstract

We present for the first time an exhaustive enumeration of Williamson matrices of even order $n < 65$. The search method relies on the novel SAT+CAS paradigm of coupling SAT solvers with computer algebra systems so as to take advantage of the advances made in both the field of satisfiability checking and the field of symbolic computation. Additionally, we use a programmatic SAT solver which allows conflict clauses to be learned programmatically, through a piece of code specifically tailored to the domain area. Prior to our work, Williamson matrices had only been enumerated for odd orders $n < 60$, so our work increases the bounds that Williamson matrices have been enumerated up to and provides the first enumeration of Williamson matrices of even order. Our results show that Williamson matrices of even order tend to be much more abundant than those of odd orders. In particular, Williamson matrices exist for every even order $n < 65$ but do not exist in orders 35, 47, 53, and 59.

1 Introduction

In 1944 the mathematician John Williamson introduced the matrices which now bear his name in the process of studying the Hadamard conjecture from combinatorial design theory (Williamson 1944). This conjecture states that *Hadamard matrices*—square $n \times n$ matrices H with ± 1 entries where HH^T is the identity matrix times n—exist for all orders n divisible by 4. Williamson defined a new type of matrix which has been extensively used to construct Hadamard matrices in many different orders n. Williamson matrices have also found use in digital communication systems and this motivated mathematicians from NASA's Jet Propulsion Laboratory to construct Williamson matrices of order 23 while developing codes allowing the transmission of signals over a long range (Baumert, Golomb, and Hall 1962).

Although Williamson defined his matrices for both even and odd orders (see Section 2), almost all subsequent work has focused on the odd case. Williamson matrices were recently constructed in all even orders up to 42 (Bright et al. 2016; Zulkoski et al. 2017) but these works did not contain a complete enumeration. We are not aware of any other constructions of Williamson matrices in even orders, though generalizations of Williamson matrices have been constructed in even orders (Wallis 1974). The algorithms used for enumerating Williamson matrices prior to 2016 (e.g., (Baumert and Hall 1965; Sawade 1977; Koukouvinos and Kounias 1988; Đoković 1993; Holzmann, Kharaghani, and Tayfeh-Rezaie 2008)) have all used properties only available in odd orders. In light of this, it is interesting to develop algorithms for enumerating Williamson matrices which work in even orders.

Unfortunately, it would not be possible to resolve the Hadamard conjecture by only studying Williamson matrices of even order, since Hadamard matrices constructed using Williamson matrices of even order have 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 prove 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 the mathematician Richard Turyn wrote (Turyn 1972):

> 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 $t = 35$ (Đoković 1993). Later, a complete enumeration of Williamson matrices for odd orders $n < 60$ was completed (Holzmann, Kharaghani, and Tayfeh-Rezaie 2008). This showed that Williamson matrices also do not exist for orders 47, 53, and 59 but exist for all other odd orders under 65 since Turyn's construction (Turyn 1972; Lang and Schneider 2012) works in orders 61 and 63.

In this paper we provide for the first time an enumeration of Williamson matrices in orders which are not odd (preliminary results originally appeared in the PhD thesis of the first author (Bright 2017a)). 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 matrices 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 the tools and techniques from the fields of *satisfiability checking* and *symbolic computation*, as described in Section 3. Such an approach was recently presented at the conferences CADE and IJCAI (Zulkoski, Ganesh, and Czarnecki 2015; 2016) and was used to improve the best known bounds in certain graph theoretic conjectures. The approach was also independently proposed at the conference ISSAC (Ábrahám 2015). More recently, it has been argued by the SC^2 project (Ábrahám et al. 2016) that the fields of satisfiability checking and symbolic computation are complementary and combining the tools of both fields (i.e., SAT solvers and computer algebra systems) in the right way can solve problems more efficiently than could be done by applying the tools of either field in isolation, and our work provides evidence for this view.

Furthermore, our method uses a SAT solver which can learn conflict clauses *programmatically*, i.e., through a piece of custom code supplied to the SAT solver. This code encodes domain-specific knowledge that an off-the-shelf SAT solver would otherwise not be able to exploit. This general framework is not limited to any specific domain; any external library or CAS function can be used as long as it is callable by the SAT solver. As we will see in Section 3, the clauses that are learned in this fashion can enormously cut down the search space as well as the solver's runtime.

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 $n < 65$. These counts are given up to an equivalence which is described, along with many other properties of Williamson matrices, including a new version of Williamson's product theorem, 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.

Williamson matrices and sequences

The definition of Williamson matrices is motivated by the following theorem that Williamson used for constructing Hadamard matrices (Williamson 1944):

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 $j \in \{0, \ldots, n-1\}$.*

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 $[x_0, \ldots, x_{n-1}]$ and since they are symmetric this row must be a symmetric sequence, i.e., satisfy $x_i = x_{n-i}$ for $1 \leq i < n$. 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 the sequence $A = [a_0, \ldots, a_{n-1}]$ is the function given by*

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

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

$$\mathrm{PAF}_A \coloneqq \big[\mathrm{PAF}_A(0), \ldots, \mathrm{PAF}_A(n-1)\big].$$

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*

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

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

It is straightforward to see that there is an equivalence between such sequences and Williamson matrices (Bright et al. 2016, §3.4) and so for the remainder of this paper we will work directly with these sequences instead of Williamson matrices.

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 (Holzmann, Kharaghani, and Tayfeh-Rezaie 2008) 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 with an offset of $n/2$.

Fourier analysis

We now give an alternative definition of Williamson sequences using concepts from Fourier analysis. First, we define the power spectral density of a sequence.

Definition 4. *The* power spectral density *of the sequence* $A = [a_0, \ldots, a_{n-1}]$ *is the function*

$$\mathrm{PSD}_A(s) \coloneqq \left|\mathrm{DFT}_A(s)\right|^2$$

where DFT_A *is the* discrete Fourier transform *of* A*, i.e.,* $\mathrm{DFT}_A(s) \coloneqq \sum_{k=0}^{n-1} a_k e^{2\pi iks/n}$*. Equivalently, we may also consider the power spectral density to be a sequence containing the values of the above function, i.e.,*

$$\mathrm{PSD}_A \coloneqq \left[\mathrm{PSD}_A(0), \ldots, \mathrm{PSD}_A(n-1)\right].$$

It now follows by (Đoković and Kotsireas 2015, Theorem 2) that Williamson sequences have the following alternative definition.

Theorem 2. *Four symmetric sequences* A*,* B*,* C*,* $D \in \{\pm 1\}^n$ *are Williamson sequences if and only if*

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

for $s = 0, \ldots, \lfloor n/2 \rfloor$.

Corollary 1. *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 $(*)$ cannot hold and thus A cannot be part of a Williamson sequence. □

Similarly, one can extend this so-called PSD test in Corollary 1 to apply to more than one sequence at a time:

Corollary 2. *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.*

Compression

As in the work (Đoković and Kotsireas 2015) we now introduce the notion of *compression*.

Definition 5. *Let* $A = [a_0, a_1, \ldots, a_{n-1}]$ *be a sequence of length* $n = dm$ *and set*

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

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

From (Đoković and Kotsireas 2015, Theorem 3) we have the following result.

Theorem 3. *If* A*,* B*,* C*,* D *is a Williamson sequence of order* n *then*

$$\mathrm{PAF}_{A'} + \mathrm{PAF}_{B'} + \mathrm{PAF}_{C'} + \mathrm{PAF}_{D'} = [4n, 0, \ldots, 0]$$

and

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

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

Corollary 3. *If* A*,* B*,* C*,* D *is a Williamson sequence of order* n *then*

$$R_A^2 + R_B^2 + R_C^2 + R_D^2 = 4n \qquad (**)$$

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 the result follows by Theorem 3. □

Williamson's product theorem

Williamson (Williamson 1944) proved the following theorem:

Theorem 4. *If* A*,* B*,* C*,* D *is a Williamson sequence of odd order* n *then* $a_i b_i c_i d_i = -a_0 b_0 c_0 d_0$ *for* $1 \le i < n/2$*.*

We prove a version of this theorem for even n:

Theorem 5. *If* A*,* B*,* C*,* D *is a Williamson sequence of even order* $n = 2m$ *then* $a_i b_i c_i d_i = a_{i+m} b_{i+m} c_{i+m} d_{i+m}$ *for* $0 \le i < m$*.*

Although this theorem is not an essential part of our algorithm it improves its efficiency by allowing us to cut down the size of the search space. Our algorithm uses the theorem in the following form:

Corollary 4. *If* A'*,* B'*,* C'*,* D' *is a* 2*-compression of a Williamson sequence then* $A' + B' + C' + D' \equiv [0, \ldots, 0] \pmod 4$.

Proofs of Theorem 5 and Corollary 4 are available on the arXiv (Bright 2017b).

3 The SAT+CAS paradigm

The idea of combining SAT solvers with computer algebra systems originated independently in two works published in 2015: In a paper at the conference CADE entitled "MATHCHECK: A Math Assistant via a Combination of Computer Algebra Systems and SAT Solvers" (Zulkoski, Ganesh, and Czarnecki 2015) and in an invited talk at the conference ISSAC entitled "Building Bridges between Symbolic Computation and Satisfiability Checking" (Ábrahám 2015).

The CADE paper describes a tool called MATHCHECK which combines the general-purpose search capability of SAT solvers with the domain-specific knowledge of computer algebra 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.

As evidence for the power of this paradigm, they used MATHCHECK to improve the best known bounds in two conjectures in graph theory.

Independently, the computer scientist Erika Ábrahám observed that the fields of satisfiability checking and symbolic computation share many common aims but in practice are quite separated, with little communication between the fields:

> ...collaboration between symbolic computation and SMT [SAT modulo theories] solving is still (surprisingly) quite restricted...

Furthermore, she outlined reasons why combining the insights from both fields had the potential to solve certain problems more efficiently than would be otherwise possible. To this end, the SC^2 project (Ábrahám et al. 2016) was started with the aim of fostering collaboration between the two communities.

Programmatic SAT

The idea of a *programmatic* SAT solver was introduced in the paper (Ganesh et al. 2012). 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; such a method was used in (Bright et al. 2016). However, one could also use the equivalent definition given in Theorem 2. This alternate definition has the advantage that it becomes easy to apply Corollaries 1 and 2, which allows one to filter many sequences from consideration and greatly speed up the search. Because of this, our method will use the constraints $(*)$ from Theorem 2 to encode the definition of Williamson sequences in our SAT instances.

However, encoding the equations in $(*)$ would be extremely cumbersome to do using CNF clauses, because of the involved nature of computing the PSD values. However, the equations $(*)$ are easy to express programmatically—as long as one has a method of computing the PSD values. This can be done 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 Theorem 2 and the filtering criteria of Corollaries 1 and 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 Boolean variables defining the sequences A, B, C, and D (where true encodes 1 and false encodes -1). In the case when the partial assignment can be ruled out using Corollaries 1 or 2, 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 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 encoding the entries in sequence A have been assigned; if so, add A to the set S and compute PSD_A, otherwise skip to the next step. When $\mathrm{PSD}_A(s) > 4n$ 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
$$\neg(a_0^{\text{cur}} \wedge a_1^{\text{cur}} \wedge \cdots \wedge a_{n-1}^{\text{cur}}) \equiv \neg a_0^{\text{cur}} \vee \neg a_1^{\text{cur}} \vee \cdots \vee \neg a_{n-1}^{\text{cur}}$$
where a_i^{cur} is the literal a_i when a_i is currently assigned to true and is the literal $\neg a_i$ when a_i is currently assigned to false.
3. Check if all the variables encoding the entries in sequence B have been assigned; if so, add B to the set S and compute PSD_B. When 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 record the sequences in an auxiliary file and 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 search is completed the auxiliary file will contain all sequences which passed the PSD tests and thus all Williamson sequences will be in this list (verifying a sequence is in fact Williamson can be done using Definition 3). 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 steps 2–4 but in practice such a clause is learned quite frequently and this more than makes up for the overhead of computing the PSD values (this accounted for about 20% of the SAT solver's runtime in our experiments). The programmatic approach was essential for the largest orders which we were able to solve; see Table 2 in Section 5 for a comparision between the running times of a SAT solver using the CNF and programmatic encodings. However, it was much too slow to be able perform the enumeration by itself.

4 Our enumeration algorithm

We now give a complete description of our method which enumerates all Williamson sequences of a given even order n.

Step 1: Generate possible sum-of-squares decompositions

First, note that by Corollary 3 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 equation $(**)$. Because we only care about Williamson sequences up to equivalence, we add the inequalities

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

to the Diophantine equation; 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.

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 whose rowsum does not appear in any possible solution (R_A, R_B, R_C, R_D) of the sum-of-squares Diophantine equation $(**)$. 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.

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. For each solution (R_A, R_B, R_C, R_D) of the sum-of-squares Diophantine equation $(**)$ 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 R_A (and the other lists 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.

Step 4: Match the compressions

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 $R_A^2 + R_B^2 + R_C^2 + R_D^2$. Thus, by trying all possible sum-of-squares decompositions and 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 of order n. By Theorem 3, a necessary condition for A, B, C, D to be a Williamson sequence is that

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

where A', B', C', D' are the 2-compressions of A, B, C, D. Therefore, one could perform this step by enumerating all $(A', B', C', D') \in L_A \times L_B \times L_C \times L_D$ and outputting those whose PSDs sum to $[4n, \ldots, 4n]$ as a potential 2-compression of a Williamson sequence. However, there will typically be far too many elements of $L_A \times L_B \times L_C \times L_D$ to try in a reasonable amount of time.

Instead, we will enumerate all $(A', B') \in L_A \times L_B$ and $(C', D') \in L_C \times L_D$ and use a string sorting technique (Kotsireas, Koukouvinos, and Pardalos 2010) to find which (A', B') and (C', D') can be matched together to form potential 2-compressions of Williamson sequences. To determine which pairs can be matched together we use the necessary condition from Theorem 3 in a slightly rewritten form,

$$\mathrm{PAF}_{A'} + \mathrm{PAF}_{B'} = [4n, 0, \ldots, 0] - (\mathrm{PAF}_{C'} + \mathrm{PAF}_{D'}).$$

Our matching procedure outputs a list of the (A', B', C', D') which satisfy this condition, and therefore output a list of potential 2-compressions of Williamson sequences.

In detail, our matching procedure performs the following steps:

1: **initialize** L_{AB} and L_{CD} to empty lists
2: **for** $(A', B') \in L_A \times L_B$ **do**
3: **if** $\mathrm{PSD}_{A'}(s) + \mathrm{PSD}_{B'}(s) < 4n$ for all s **then**
4: **add** $\mathrm{PAF}_{A'} + \mathrm{PAF}_{B'}$ to L_{AB}
5: **for** $(C', D') \in L_C \times L_D$ **do**
6: **if** $\mathrm{PSD}_{C'}(s) + \mathrm{PSD}_{D'}(s) < 4n$ for all s **then**
7: **add** $[4n, 0, \ldots, 0] - (\mathrm{PAF}_{C'} + \mathrm{PAF}_{D'})$ to L_{CD}
8: **for** each X common to both L_{AB} and L_{CD} **do**
9: **output** (A', B') and (C', D') which X was generated from in an auxiliary file

Line 8 can be done efficiently by sorting the lists L_{AB} and L_{CD} and then performing a linear scan through the sorted lists to find the elements common to both lists. Line 9 can be done efficiently if with each element in the lists L_{AB} and L_{CD} we also keep track of a pointer to the sequences (A', B') or (C', D') that the element was generated from in line 4 or 7. Also in line 9 we only output sequences for which $A' + B' + C' + D'$ is the zero vector mod 4 as this is an invariant of all 2-compressed Williamson sequences by Corollary 4.

Step 5: Uncompress the matched 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 matched compressions which we generated. To do this, we formulate the uncompression problem as a Boolean SAT instance and use a SAT solver's fine-tuned search facilities 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,

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

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

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

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 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 clauses

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

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 clauses

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

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 clauses

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

to our SAT instance. Note that these clauses specify 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.

For each (A', B', C', D') generated in Step 4 we generate a SAT instance which contains the clauses specified above. We then solve the SAT instances with a programmatic SAT solver whose programmatic clause generator specifies that any satisfying assignment of the instance encodes a Williamson sequence and performs an exhaustive search to find all solutions. By construction, every Williamson sequence of order n will have its 2-compression generated in Step 4, making this search totally exhaustive (up to the discarded equivalences).

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 on the Williamson sequences in the list and discarding those which are equivalent to a previously found Williamson sequence.

Optimizations

While the procedure just described will correctly enumerate all Williamson sequences of a given even order n, there are a few optimizations which can be used to improve the efficiency of the search. Note that in Step 3 we have not generated *all* possible 2-compression quadruples; we only generate those quadruples that have rowsums (R_A, R_B, R_C, R_D)

which correspond to solutions of $(**)$, and we use the negation and reordering equivalence operations to cut down the number of possible rowsums necessary to check. However, there still remain equivalences which can be removed; if ϕ is an automorphism of C_n then (A, B, C, D) is a Williamson quadruple if and only if $(\phi(A), \phi(B), \phi(C), \phi(D))$ is a Williamson quadruple. Thus if both A and $\phi(A)$ are in the list X generated in Step 2 we can remove one from consideration. Unfortunately, we cannot do the same in the lists for B, C, and D, since it is not possible to know which representatives for B, C, and D to keep, as the representatives must match with the A that was kept.

Similarly, in Step 5 one can ignore any SAT instance which can be transformed into another SAT instance using the equivalence operations from Section 2. In this case the solutions in the ignored SAT instance will be equivalent to those in the SAT instance associated to it through the equivalence transformation.

5 Results

We implemented the algorithm described in Section 4 (including all optimizations) and ran it on even orders $n < 65$. Step 1 was completed using the computer algebra system MAPLE. Steps 2–4 and 6 were completed using C++ code which used the library FFTW (Frigo and Johnson 2005) for computing PSD values. Step 5 was completed using MAPLESAT (Liang et al. 2016) modified to support a programmatic interface and also used FFTW for computing PSD values. Since FFTW provides no guarantee on the accuracy of the values it returns, the PSD values which were used to remove sequences from consideration were double-checked using Definition 4 in high precision.

Our computations were performed on a cluster of 64-bit AMD Opteron 2.2 GHz processors limited to 6 GB of memory and running CentOS 6.7. Timings for running our entire algorithm (in hours) are given in Table 1, and timings for the running of the SAT solver alone are given in Table 2. The bottleneck of our method for large n was the matching procedure described in Step 4, which requires enumerating and then sorting a very large number of vectors. For example, when $n = 64$ and $R_A = R_B = 8$ there were over 26.6 billion vectors added to L_{AB}. 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 (denoted by $\#W_n$). An explicit enumeration of these Williamson sequences is available online at doi.org/10.5281/zenodo.825339.

n	Time (h)	# inst.	$\#W_n$
2	0.00	1	1
4	0.00	1	1
6	0.00	1	1
8	0.00	1	1
10	0.00	2	2
12	0.00	3	3
14	0.00	3	7
16	0.00	5	6
18	0.00	22	40
20	0.00	14	27
22	0.00	22	27
24	0.00	40	80
26	0.00	24	38
28	0.00	78	99
30	0.00	281	268
32	0.00	70	200
34	0.01	214	160
36	0.01	1013	691
38	0.00	360	87
40	0.02	4032	1898
42	0.03	2945	561
44	0.02	1163	378
46	0.11	1538	97
48	0.20	4008	12528
50	1.04	3715	500
52	2.24	4535	1071
54	6.30	25798	979
56	2.19	18840	40502
58	33.19	9908	140
60	55.00	256820	7235
62	159.95	19418	117
64	149.57	34974	95504

Table 1: A summary of the running time, number of SAT instances used, and number of inequivalent Williamson sequences generated in each even order $2 \leq n \leq 64$.

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 satisfiability checking and symbolic computation) as well as the power and flexibility of the programmatic SAT approach. 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 been published for all odd orders up to 59 but prior to our work such counts were unavailable in the even case.

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 $18 \leq n \leq 64$ has $\#W_n > 10$ and there are some orders which contain thousands 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.

Acknowledgements

We thank the anonymous reviewers for their detailed comments which improved the exposition of this paper. This work was made possible by the facilities of SHARCNET, the

	SAT Solving Time (h)	
n	CNF encoding	Programmatic
34	0.04	0.00
36	0.55	0.00
38	0.19	0.00
40	3.00	0.02
42	6.69	0.01
44	9.00	0.01
46	14.57	0.01
48	33.64	0.03
50	98.44	0.03
52	–	0.08
54	–	0.31
56	–	0.29
58	–	0.24
60	–	16.47
62	–	0.83
64	–	1.62

Table 2: The total time spent running MAPLESAT in each order $34 \leq n \leq 64$ using the CNF encoding and the programmatic encoding. A timeout of 100 hours was used.

Shared Hierarchical Academic Research Computing Network and Compute/Calcul Canada.

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. *Intelligent Computer Mathematics: Proceedings CICM* 28–43.

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

Baumert, L., and Hall, M. 1965. Hadamard matrices of the Williamson type. *Mathematics of Computation* 19(91):442–447.

Baumert, L.; Golomb, S. W.; and Hall, M. 1962. Discovery of an Hadamard matrix of order 92. *Bull. Amer. Math. Soc.* 68(3):237–238.

Bright, C.; Ganesh, V.; Heinle, A.; Kotsireas, I. S.; Nejati, S.; and Czarnecki, K. 2016. 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*, 117–133.

Bright, C. 2017a. *Computational Methods for Combinatorial and Number Theoretic Problems*. Ph.D. Dissertation, University of Waterloo.

Bright, C. 2017b. A new form of Williamson's product theorem. https://arxiv.org/abs/1711.07056.

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. 1993. Williamson matrices of order $4n$ for $n = 33, 35, 39$. *Discrete mathematics* 115(1):267–271.

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 *Theory and Applications of Satisfiability Testing–SAT 2012*. Springer. 143–156.

Holzmann, W. H.; Kharaghani, H.; and Tayfeh-Rezaie, B. 2008. Williamson matrices up to order 59. *Designs, Codes and Cryptography* 46(3):343–352.

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.

Koukouvinos, C., and Kounias, S. 1988. Hadamard matrices of the Williamson type of order $4 \cdot m$, $m = p \cdot q$ an exhaustive search for $m = 33$. *Discrete mathematics* 68(1):45–57.

Lang, W., and Schneider, E. 2012. Turyn type Williamson matrices up to order 99. *Designs, Codes and Cryptography* 62(1):79–84.

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*, AAAI'16, 3434–3440. AAAI Press.

Sawade, K. 1977. Hadamard matrices of order 100 and 108. *Bulletin of Nagoya Institute of Technology* (29):147–153.

Turyn, R. J. 1972. An infinite class of Williamson matrices. *Journal of Combinatorial Theory, Series A* 12(3):319–321.

Wallis, J. S. 1974. Williamson matrices of even order. In Holton, D. A., ed., *Combinatorial Mathematics: Proceedings of the Second Australian Conference*. Berlin, Heidelberg: Springer Berlin Heidelberg. 132–142.

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

Zulkoski, E.; Bright, C.; Heinle, A.; Kotsireas, I.; Czarnecki, K.; and Ganesh, V. 2017. Combining SAT solvers with computer algebra systems to verify combinatorial conjectures. *Journal of Automated 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*. Springer International Publishing. 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, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, 4228–4233.