Applying Computer Algebra Systems with SAT Solvers to the Williamson Conjecture

Curtis Bright[a], Ilias Kotsireas[b], Vijay Ganesh[a]

[a] *University of Waterloo*
[b] *Wilfrid Laurier University*

Abstract

We employ tools from the fields of symbolic computation and satisfiability checking—namely, computer algebra systems and SAT solvers—to study the Williamson conjecture from combinatorial design theory and increase the bounds to which Williamson matrices have been enumerated. In particular, we completely enumerate all Williamson matrices of even order up to and including 70 which gives us deeper insight into the behaviour and distribution of Williamson matrices. We find that, in contrast to the case when the order is odd, Williamson matrices of even order are quite plentiful and exist in every even order up to and including 70. As a consequence of this and a new construction for 8-Williamson matrices we construct 8-Williamson matrices in all odd orders up to and including 35. We additionally enumerate all Williamson matrices whose orders are divisible by 3 and less than 70, finding one previously unknown set of Williamson matrices of order 63.

Keywords: Williamson matrices; Boolean satisfiability; SAT solvers; Exhaustive search; Autocorrelation

1. Introduction

In recent years SAT solvers have been used to solve or make progress on mathematical conjectures which have otherwise resisted solution from some of the world's best mathematicians. Some prominent problems which fit into this trend include the Erdős discrepancy conjecture, which was open for 80 years and had a special case solved by Konev and Lisitsa (2014); the Ruskey–Savage conjecture, which has been open for 25 years and had a special case solved by Zulkoski et al. (2015); the Boolean Pythagorean triples problem, which was open for 30 years and solved by Heule et al. (2016); and the determination of the fifth Schur number, which was open for 100 years and solved by Heule (2018). Although these are problems which arise in completely separate fields and have no obvious connection to propositional satisfiability checking, nevertheless SAT solvers were found to be extremely effective at pushing the state-of-the-art and sometimes absolutely crucial in the problem's ultimate solution.

In this paper we apply a SAT solver to the Williamson conjecture from combinatorial design theory. Our work is similar in spirit to the aforementioned works but we would like to highlight two main differences. Firstly, we employ an approach inspired by SMT

(SAT modulo theories) solvers and use a SAT solver that is able to learn conflict clauses through a piece of code specifically tailored to the problem domain. This code encodes domain-specific knowledge that an off-the-shelf SAT solver would otherwise not be able to exploit. This framework is not limited to any specific domain; any external library or 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.

Secondly, similar in style to (Zulkoski et al., 2015) we incorporate functionality from computer algebra systems to increase the efficiency of the search in what we call the "SAT+CAS" paradigm. This approach of combining computer algebra systems with SAT or SMT solvers was also independently proposed at the conference ISSAC by 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.

We describe the Williamson conjecture, its history, and state the necessary properties of Williamson matrices that we require in Section 2. In particular, we derive a new version of Williamson's product theorem that applies to Williamson matrices of even order (Theorem 15). We give an overview of the SAT+CAS paradigm in Section 3, describe our SAT+CAS method in Section 4, and give a summary of our results in Section 5. The present work is an extension of our previous work (Bright et al., 2018a) that enumerated Williamson matrices of even order up to order 64. The present work extends this enumeration to order 70 and extends the method to enumerate Williamson matrices with orders divisible by 3. In doing so, we find a previously undiscovered set of Williamson matrices of order 63, the first new set of Williamson matrices of odd order discovered since one of order 43 was found over ten years ago by Holzmann et al. (2008). Additionally, we improve our treatment of equivalence checking (see Section 4.6), identify a new equivalence operation that applies to Williamson matrices of even order (see Section 2.2), derive a new doubling construction for Williamson matrices (Theorem 17), and a new construction for 8-Williamson matrices (Theorem 18). Using this construction we construct 8-Williamson matrices in all odd orders $n \leq 35$, improving on the result of Kotsireas and Koukouvinos (2009) that constructed 8-Williamson matrices in all odd orders $n \leq 29$. Finally, in Section 6 we use our experience developing systems that combine SAT solvers with computer algebra systems to give some guidelines about the kind of problems for which an approach is likely to be effective.

2. The Williamson conjecture

Williamson (1944) introduced the matrices which now bear his name (see Section 2.1) while developing a method of constructing Hadamard matrices. Hadamard matrices are square matrices with ± 1 entries and pairwise orthogonal rows; they have a long history and many applications such as to error-correcting codes (Bose and Shrikhande, 1959). The *Hadamard conjecture* states that Hadamard matrices exist for all orders divisible by 4. Williamson's construction has been extensively used to construct Hadamard matrices in many different orders and the *Williamson conjecture* states that it can be used to construct a Hadamard matrix of any order divisible by 4; Turyn (1972) states it as follows:

> Only a finite number of Hadamard matrices of Williamson type are known so far; it has been conjectured that one such exists of any order $4t$.

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 et al., 1962). These Williamson matrices were consequently used to construct a Hadamard matrix of order $4 \cdot 23 = 92$ (Cooper, 2013). (In some older works the Hadamard matrix constructed in this way was itself referred to as a Williamson matrix but we follow modern convention and do not use this terminology.) Williamson matrices are also studied for their elegant mathematical properties and their relationship to other mathematical conjectures (Schmidt, 1999).

Although Williamson defined his matrices for both even and odd orders, most subsequent work has focused on the odd case. A complete enumeration of Williamson matrices was completed for all odd orders up to 23 by Baumert and Hall (1965). A enumeration in orders 25 and 27 was completed by Sawade (1977) but this enumeration was later found to be incomplete by Đoković (1995), who gave a complete enumeration in the order 25 as well as (in a previous paper) the orders 29 and 31 (Đoković, 1992). The orders 33 and 39 were claimed to be completely enumerated by Koukouvinos and Kounias (1988, 1990) but these searches were demonstrated to be incomplete when a complete enumeration of the orders 33, 35, and 39 was completed by Đoković (1993). Most recently, all odd orders up to 59 were enumerated by Holzmann et al. (2008) and the order 61 was enumerated by Lang and Schneider (2012).

Historically, less attention was paid to the even order cases, although generalizations of Williamson matrices were explicitly constructed in even orders by Wallis (1974) as well as Agayan and Sarukhanyan (1981). Williamson matrices were constructed in all even orders up to 22 by Kotsireas and Koukouvinos (2006), up to 34 by Bright et al. (2016), and up to 42 by Zulkoski et al. (2017). Kotsireas and Koukouvinos (2006) provided a exhaustive search up to order 18 but otherwise these works did not contain a complete enumerations. A complete enumeration in the even orders up to 44 was given by Bright (2017) and this was extended to order 64 by Bright et al. (2018a).

One reason why more attention has traditionally been given to the odd order case is due to the fact that if it was possible to construct Williamson matrices in all odd orders this would resolve the Hadamard conjecture. On the other hand, constructing Williamson matrices in all even orders would not resolve the Hadamard conjecture because 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.

The Williamson conjecture was shown to be false by Đoković (1993) who showed that such matrices do not exist in order 35. Later, when an enumeration of Williamson matrices for odd orders $n < 60$ was completed (Holzmann et al., 2008) it was found 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) works in orders 61 and 63.

In this paper we provide for the first time a complete enumeration of Williamson matrices in the orders 63, 66, 68, 69, and 70. In particular, we show that Williamson matrices exist in every even order up to 70. This leads us to state what we call the *even Williamson conjecture*:

Conjecture 1. *Williamson matrices exist in every even order.*

The fact that Williamson matrices of even order turn out to be somewhat plentiful gives some evidence for the truth of Conjecture 1. Though we do not know how to prove Conjecture 1 our enumeration could potentially uncover structure in Williamson matrices which might then be exploited in a proof of the conjecture.

Additionally, we point out that the existence of Williamson matrices of order $70 = 2 \cdot 35$ is especially interesting since 35 is the smallest order for which Williamson matrices do not exist. Using complex Hadamard matrices, Turyn (1970) showed the existence of Williamson matrices of odd order n implies the existence of Williamson matrices of orders $2^k n$ for $k = 1, 2, 3, 4$. Since Williamson matrices exist for all odd orders $n < 35$ Turyn's result implies that Williamson matrices exist for all even orders strictly less than 70 with the possible exceptions of 32 and 64. Since Williamson matrices of order 35 do not exist Turyn's result cannot be used to show the existence of Williamson matrices of order 70; the question of existence in order 70 was open until this paper.

We also determine that there are exactly two sets of Williamson matrices (up to the equivalence given in Section 2.2) of order 63. One of these falls under the aforementioned construction given by Turyn (1972) while the other is new and is the first newly discovered set of Williamson matrices in an odd order since one was found using an exhaustive search in order 43 by Holzmann et al. (2008). In order 69 our enumeration method produced just one set of Williamson matrices and that set falls under the construction given by Turyn.

2.1. Williamson matrices and sequences

We now give the background on Williamson matrices and their properties that are necessary to understand the remainder of the paper. The definition of Williamson matrices is motivated by the following theorem used for constructing Hadamard matrices by Williamson (1944).

Theorem 2. *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 3. *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 2 are known as a quadruple of *Williamson matrices* in honour 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 $1 \leq i < n$. 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 4. *The* periodic autocorrelation function *of the sequence $A = [a_0, \dots, a_{n-1}]$ 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.,

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

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

Definition 5. *Four symmetric sequences A, B, C, $D \in \{\pm 1\}^n$ are called a* Williamson sequence quadruple *(or simply* Williamson sequences*) if they satisfy*

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

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

It is straightforward to see that there is an equivalence between such sequences and Williamson matrices (Bright et al., 2016, §3.2) because the first row of the matrix A^2 is exactly the sequence PAF_A. Therefore, for the remainder of this paper we will work directly with these sequences instead of Williamson matrices.

2.2. Williamson equivalences

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

E1. (Reorder) Reorder the sequences A, B, C, D in any way.
E2. (Negate) Negate all the entries of any of A, B, C, or D.
E3. (Shift) If n is even, cyclically shift all the entries in any of A, B, C, or D by an offset of $n/2$.
E4. (Permute entries) Apply an automorphism of the cyclic group C_n to all the indices of the entries of each of A, B, C, and D simultaneously.
E5. (Alternating negation) If n is even, negate every second entry in each of A, B, C, and D simultaneously.

These equivalence operations are well known (Holzmann et al., 2008) except for the shift and alternating negation operations which have not traditionally been used because they only apply when n is even. In fact, they were overlooked until a careful examination of the sequences produced by our enumeration method.

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

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

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

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

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

Theorem 7. *Four symmetric sequences A, B, C, $D \in \{\pm 1\}^n$ are Williamson sequences if and only if*

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

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

Corollary 8. *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 8 to apply to more than one sequence at a time.

Corollary 9. *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.*

Additionally, a quadruple of sequences A, B, C, D that are not Williamson must necessarily fail the PSD test for some value of s.

Corollary 10. *If four sequences A, B, C, $D \in \{\pm 1\}^n$ are not Williamson sequences then* $\mathrm{PSD}_A(s) + \mathrm{PSD}_B(s) + \mathrm{PSD}_C(s) + \mathrm{PSD}_D(s) > 4n$ *for some s.*

Proof. By Parseval's theorem the average value of $\mathrm{PSD}_A(s) + \mathrm{PSD}_B(s) + \mathrm{PSD}_C(s) + \mathrm{PSD}_D(s)$ for $s = \{0, \dots, n-1\}$ is $4n$. Thus either this sum is the constant $4n$ (in which case the sequences are Williamson) or there is some s for which this sum is larger than $4n$. □

2.4. Compression

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

Definition 11. *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.*

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

Theorem 12. *If A, B, C, D are Williamson sequences of order n then*

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

and

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

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

Corollary 13. *If A, B, C, D are Williamson sequences of order n then*

$$R_A^2 + R_B^2 + R_C^2 + R_D^2 = 4n \tag{$**$}$$

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 12. □

2.5. Williamson's product theorem

Williamson (1944) proved the following theorem:

Theorem 14. *If A, B, C, D are Williamson sequences of odd order n then*

$$a_i b_i c_i d_i = -a_0 b_0 c_0 d_0 \qquad \textit{for } 1 \le i < n/2.$$

We prove a version of this theorem for even n:

Theorem 15. *If A, B, C, D are Williamson sequences of even order $n = 2m$ then*

$$a_i b_i c_i d_i = a_{i+m} b_{i+m} c_{i+m} d_{i+m} \qquad \text{for } 0 \le i < m.$$

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 16. *If A', B', C', D' are the 2-compressions of a set of Williamson sequences then $A' + B' + C' + D' \equiv [0, \dots, 0] \pmod 4$.*

Proofs of Theorem 15 and Corollary 16 are available in the appendix.

2.6. Doubling construction

We now give a simple construction that generates Williamson sequences of order $2n$ from Williamson sequences of odd order n using the following three operations on sequences $A = [a_0, \dots, a_{n-1}]$ and $B = [b_0, \dots, b_{n-1}]$:

1. Negation. Individually negate each entry of A, i.e., $-A \coloneqq [-a_0, \dots, -a_{n-1}]$.
2. Shift. Cyclically shift the entries of A by an offset of $(n-1)/2$, i.e.,

$$A' \coloneqq [a_{(n-1)/2}, \dots, a_{n-1}, a_0, a_1, \dots, a_{(n-3)/2}].$$

3. Interleave. Interleave the entries of A and B in a perfect shuffle, i.e.,

$$A ⧢ B \coloneqq [a_0, b_0, a_1, b_1, \dots, a_{n-1}, b_{n-1}].$$

Our doubling construction is captured by the following theorem.

Theorem 17. *Let A, B, C, D be Williamson sequences of odd order n. Then*

$$A ⧢ B',\ (-A) ⧢ B',\ C ⧢ D',\ (-C) ⧢ D'$$

are Williamson sequences of order $2n$.

We remark that a single set of Williamson sequences of order n can often be used to generate more than one set of Williamson sequences of order $2n$ by applying equivalence operations to the quadruple A, B, C, D before using the construction. For example, the single inequivalent set of Williamson sequences of order 5 can be used to generate both inequivalent sets of Williamson sequences of order 10 using this construction with an appropriate reordering of A, B, C, D.

We also remark that this doubling construction can be reversed in the sense that if Williamson sequences of order $2n$ exist for n odd then symmetric sequences $X_1, \dots, X_8 \in \{\pm 1\}^n$ can be constructed that satisfy the Williamson property

$$\sum_{i=1}^{8} \mathrm{PAF}_{X_i}(s) = 0 \qquad \text{for } s = 1, \dots, n-1.$$

We call such sequences *8-Williamson sequences* because they form the first rows of *8-Williamson matrices* as defined by for example Kotsireas and Koukouvinos (2006). Note that the equivalence operations of Section 2.2 also define an equivalence on 8-Williamson sequences so long as they are written to apply to 8 sequences instead of 4.

Theorem 18. *Let A, B, C, D be Williamson sequences of order $2n$ with n odd and write*

$$A = A_1 ⧢ A_2',\ B = B_1 ⧢ B_2',\ C = C_1 ⧢ C_2',\ D = D_1 ⧢ D_2'.$$

Then A_1, A_2, B_1, B_2, C_1, C_2, D_1, D_2 are 8-Williamson sequences of order n.

Proofs of Theorems 17 and 18 are available in the appendix. Similar constructions have been described for complementary sets of sequences by Tseng and Liu (1972) but to our knowledge these constructions are new in the context of Williamson sequences.

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" by Zulkoski et al. (2015) and in an invited talk at the conference ISSAC entitled "Building Bridges between Symbolic Computation and Satisfiability Checking" by Ábrahám (2015). The paradigm was also anticipated by Jovanović and de Moura (2012) who used CAS techniques in SAT-like search algorithms.

The CADE paper describes a tool called MATHCHECK that 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.

3.1. Programmatic SAT and SMT

Not all constraints can easily be expressed in SAT instances. To deal with this, sophisticated *SMT solvers* were developed that can determine the satisfiability of formulas in first-order logic with respect to certain logical theories (Barrett et al., 2009). SMT solvers are often based on the Davis–Putnam–Logemann–Loveland algorithm (modulo theories) by Ganzinger et al. (2004) denoted DPLL(T) where T is a theory of first-order logic. In this framework SAT solvers handle the Boolean aspects of a formula while T solvers handle the theory aspects of a formula. Ábrahám (2015) gives a number of requirements that a theory solver should satisfy, and the SMT-LIB standard by Barrett et al. (2016) describes many different theories used by SMT solvers along with a common input and output language for those theories.

Programmatic SAT solvers were introduced by Ganesh et al. (2012) and are simply a variant of DPLL(T) where the T solvers are replaced by arbitrary user-specified code. However, programmatic SAT solvers only deal with Boolean logic and not first-order logic, similar to the "SAT modulo SAT" solvers by Bayless et al. (2013). Additionally, the code provided to a programmatic SAT solver can be specialized to individual formulas, while the T solvers in DPLL(T) deal with a specific theory of first-order logic that is typically fixed in advance. For our purposes in this paper we will use a CAS as a theory solver and therefore our usage of programmatic SAT can be considered an instance of DPLL(CAS).

A programmatic SAT solver can generate conflict clauses programmatically, i.e., by a piece of code that runs as the SAT solver carries out its search. Such a SAT solver can learn clauses that are more useful than the conflict clauses that 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 that 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 that encodes constraints that 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 5 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 by Bright et al. (2016). However, one could also use the equivalent definition given in Theorem 7. This alternate definition has the advantage that it becomes easy to apply Corollaries 8 and 9, 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 7 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* that is compiled with the SAT solver and programmatically expresses the constraints in Theorem 7 and the filtering criteria of Corollaries 8 and 9.

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

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 8 or 9, 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 would traditionally 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 $S := \emptyset$. 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
$$\neg(a_0^{\mathrm{cur}} \wedge a_1^{\mathrm{cur}} \wedge \cdots \wedge a_{n-1}^{\mathrm{cur}}) \equiv \neg a_0^{\mathrm{cur}} \vee \neg a_1^{\mathrm{cur}} \vee \cdots \vee \neg a_{n-1}^{\mathrm{cur}}$$
where a_i^{cur} is the literal a_i when a_i is currently assigned to true and is the literal $\neg a_i$ 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 that 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 5. (If the PSDs were computed exactly this is not necessary by Corollary 10 but with floating-point arithmetic there is a slight chance that a PSD test failure was not detected.) 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 set of Williamson sequences. 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 that we were able to solve; see Table 2 in Section 5 for a comparison between the running times of a SAT solver

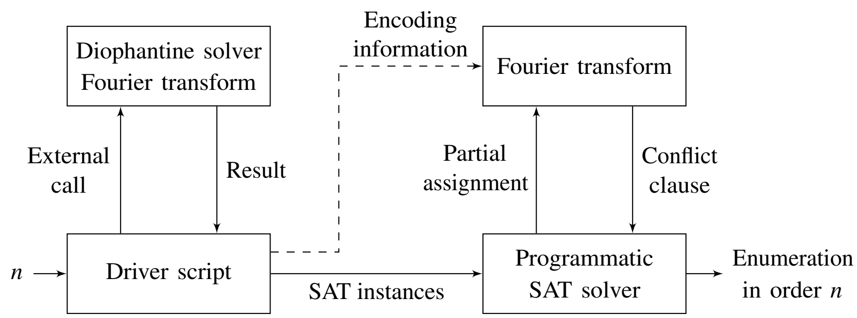

Figure 1: Outline of our algorithm for enumerating Williamson sequences of order n. The boxes on the left correspond to the preprocessing which encodes and decomposes the original problem into SAT instances. The boxes on the right correspond to an SMT-like setup where the system that computes the discrete Fourier transform takes on the role of the theory solver.

using the CNF and programmatic encodings. However, the programmatic encoding was much too slow to be able perform the enumeration by itself and a more sophisticated enumeration algorithm was required (using the programmatic encoding as a subroutine).

4. Our enumeration algorithm

A high-level summary of the components of our enumeration algorithm is given in Figure 1. We require two kinds of functions from computer algebra systems or mathematical libraries, namely, one that can solve the quadratic Diophantine equation (∗∗) and one that can compute the discrete Fourier transform of a sequence.

In the following description we have step 1 handled by the Diophantine equation solver, steps 2–4 handled by the driver script, and step 5 handled by the programmatic SAT solver. The driver script is responsible for constructing the SAT instances and passing them off to the programmatic SAT solver. It also implicitly passes encoding information to the system responsible for performing the programmatic Williamson encoding described in Section 3.2, i.e., the system needs to know which variables in the SAT instance correspond to which Williamson sequence entries, but this is fixed in advance.

We now give a complete description of our method that enumerates all Williamson sequences of a given order n divisible by 2 or 3. Let $m \in \{2, 3\}$ be the smallest prime divisor of n.

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

First, note that by Corollary 13 every set of Williamson sequences 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 quadruple can be transformed into another quadruple that satisfies these inequalities by applying the reorder and/or negating equivalence operations.

4.2. Step 2: Generate possible Williamson sequence quadruple members

Next, we form a list of the sequences that could possibly appear as a member of a Williamson sequence quadruple 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 that 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 that remain after this process form a list of the sequences that could possibly appear as a member of a set of Williamson sequences. At this stage we could generate all Williamson sequences quadruples of order n by trying all ways of grouping the possible sequences X into quadruples and filtering those that 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 for small n.

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 (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 m-compressions of the sequences X generated in step 2 that 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 if n is even and $\{\pm 3, \pm 1\}$-sequences if n is odd since they are m-compressions of the sequences X which are $\{\pm 1\}$-sequences.

4.4. Step 4: Match the compressions

By construction, the lists L_A, L_B, L_C, and L_D contain all possible m-compressions of the members of Williamson sequence quadruples 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 m-compressions of Williamson sequence quadruples of order n. By Theorem 12, a necessary condition for A, B, C, D to be Williamson sequences is that

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

where A', B', C', D' are the m-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 $[4n, \dots, 4n]$ as a potential m-compression of a Williamson sequence quadruple. 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 by Kotsireas et al. (2010) to find which (A', B') and (C', D') can be matched together to form potential m-compressions of Williamson sequences.

To determine which pairs can be matched together we use the necessary condition from Theorem 12 in a slightly rewritten form,

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

Our matching procedure outputs a list of the (A', B', C', D') which satisfy this condition, and therefore output a list of potential m-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** $[4n, 0, \dots, 0] - (\mathrm{PAF}_{C'} + \mathrm{PAF}_{D'})$ 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 if n is even 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 16.

4.5. Step 5: Uncompress the matched compressions

It is now necessary to find the Williamson sequences, if any, which when compressed by a factor of m produce one of the sequence quadruples 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 satisfiability instance and use a SAT solver's combinatorial search facilities to search for solutions to the uncompression problem.

We 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 sequence quadruples 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 distinct variables

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

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 that we generate will contain $2n + 4$ variables when n is even and $2n + 2$ variables when n is odd.

Say that (A', B', C', D') is one of the m-compressions generated in step 4. By the definition of m-compression, we have that $a'_i = a_i + a_{i+n/2}$ if n is even and $a'_i = a_i + a_{i+n/3} + a_{i+2n/3}$ if n is odd. Since $-3 \leq a'_i \leq 3$ there are seven possibilities we must consider for each a'_i.

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

$$a_i \wedge a_{i+n/3} \wedge a_{i+2n/3}$$

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

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

to our SAT instance.

Case 3. If $a'_i = 1$ then exactly one of the entries a_i, $a_{i+n/3}$, and $a_{i+2n/3}$ must be -1. Thinking of the entries as Boolean variables, we add the clauses

$$(\neg a_i \vee \neg a_{i+n/3} \vee \neg a_{i+2n/3}) \wedge (a_i \vee a_{i+n/3}) \wedge (a_i \vee a_{i+2n/3}) \wedge (a_{i+n/3} \vee a_{i+2n/3})$$

to our SAT instance. These clauses specify in conjunctive normal form that exactly one of the variables a_i, $a_{i+n/3}$, and $a_{i+2n/3}$ is false.

Case 4. 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. These clauses specify in conjunctive normal form that exactly one of the variables a_i and $a_{i+n/2}$ is true.

Case 5. If $a'_i = -1$ then exactly one of the entries a_i, $a_{i+n/3}$, and $a_{i+2n/3}$ must be 1. Thinking of the entries as Boolean variables, we add the clauses

$$(a_i \vee a_{i+n/3} \vee a_{i+2n/3}) \wedge (\neg a_i \vee \neg a_{i+n/3}) \wedge (\neg a_i \vee \neg a_{i+2n/3}) \wedge (\neg a_{i+n/3} \vee \neg a_{i+2n/3})$$

to our SAT instance. These clauses specify in conjunctive normal form that exactly one of the variables a_i, $a_{i+n/3}$, and $a_{i+2n/3}$ is true.

Case 6. 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 7. If $a'_i = -3$ then we must have $a_i = -1$, $a_{i+n/3} = -1$, and $a_{i+2n/3} = -1$. Thinking of the entries as Boolean variables, we add the clauses

$$\neg a_i \wedge \neg a_{i+n/3} \wedge \neg a_{i+2n/3}$$

to our SAT instance.

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 set of Williamson sequences. 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 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 set of Williamson sequences and performs an exhaustive search to find all solutions. By construction, every Williamson sequence quadruple of order n will have its m-compression generated in step 4, making this search totally exhaustive (up to the discarded equivalences).

4.6. Postprocessing: Remove equivalent Williamson sequences

After step 5 we have produced a list of all the Williamson sequences of order n that have a certain sum-of-squares decompositions. We chose the decompositions in such a way that every Williamson sequence quadruple will be equivalent to at least one decomposition but we have not dealt with all equivalences E1–E5. Therefore some Williamson sequences that we generate may be equivalent to each other. Removing equivalences can be performed in step 5 but since the SAT instances are generally solved in parallel we wait until all SAT instances have been solved for simplicity.

For the purpose of counting the total number of inequivalent Williamson sequences that exist in order n it is necessary to examine each quadruple in the list generated in step 5 and determine if it is equivalent to another quadruple in the list. This can be done by repeatedly applying the equivalence operations from Section 2.2 on the quadruples in the list and discarding those which are equivalent to a previously found set of Williamson sequences. However, this can be inefficient because there are typically a large number of Williamson sequence quadruples in each equivalence class. Instead, a more efficient way of testing for equivalence is to define a single representative in each equivalence class that is easy to compute. Then two sets of Williamson sequences can be tested for equivalence by testing that their representatives are equal.

As a first step in defining a single representative in each equivalence class of Williamson sequence quadruples we first consider only the equivalence operations E1, E2, and E3 (reorder, negate, and shift). The operations E2 and E3 apply to individual sequences X and there are up to four sequences which could be generated using E2 and E3, namely, X, E2(X), E3(X), and E2(E3(X)). Let M_X be the lexicographic minimum of these four sequences. Given a Williamson sequence quadruple (A, B, C, D), we compute (M_A, M_B, M_C, M_D) and then use operation E1 on the sequences in the quadruple to sort those sequences in increasing lexicographic order. The resulting quadruple is the lexicographic minimum of all quadruples equivalent to (A, B, C, D) using the operations E1, E2, and E3 and is therefore a unique single representative of the equivalence class which we denote $M_{(A,B,C,D)}$.

Next, consider the equivalence operation E4 (permute entries). Let σ be an automorphism of the cyclic group C_n and let $\sigma(X)$ be the sequence whose ith entry is $x_{\sigma(i)}$. Then the lexicographic minimum of the set

$$S_{(A,B,C,D)} \coloneqq \{\, M_{(\sigma(A),\sigma(B),\sigma(C),\sigma(D))} : \sigma \in \operatorname{Aut}(C_n) \,\}$$

is the lexicographic minimum of all quadruples equivalent to (A, B, C, D) using the operations E1, E2, E3, and E4. (This is due to the fact that E4 commutes with E1, E2, and E3, so it is always possible to find the global lexicographic minimum by first trying all possible ways of applying E4 and only afterwards considering E1, E2, and E3.)

Finally, if n is even we consider the equivalence operation E5 (alternating negation). The lexicographic minimum of the set $S_{(A,B,C,D)} \cup S_{\text{E5}(A,B,C,D)}$ will be the lexicographic minimum of all quadruples equivalent to (A, B, C, D) and is therefore a single unique representative of the equivalence class. (Again, this is due to the fact that E5 commutes with the other operations so it is always possible to find the global lexicographic minimum by first trying all possible ways of applying E5 before applying the other operations.)

4.7. Optimizations

While the procedure just described will correctly enumerate all Williamson sequences of a given even order n, there are a few optimizations that can be used to improve the efficiency of the search. Note that in step 3 we have not generated *all* possible m-compression quadruples; we only generate those quadruples that have rowsums (R_A, R_B, R_C, R_D) that 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 that can be removed; if σ is an automorphism of the cyclic group C_n then (A, B, C, D) is a Williamson sequence quadruple if and only if $(\sigma(A), \sigma(B), \sigma(C), \sigma(D))$ is a Williamson sequence quadruple (with $\sigma(X)$ defined so that its ith entry is $x_{\sigma(i)}$). Thus if both A and $\sigma(A)$ are in the list 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 representative for A that was kept.

Similarly, in step 5 one can ignore any SAT instance that can be transformed into another SAT instance using the equivalence operations from Section 2.2. 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.

In the programmatic Williamson encoding we can often learn shorter clauses with a slight modification of the procedure described in Section 3.2. Instead of checking $\sum_{X \in S} \operatorname{PSD}_X(s) > 4n$ directly we instead find the smallest subset S' of S such that $\sum_{X \in S'} \operatorname{PSD}_X(s) > 4n$ (if such a subset exists). This is done by sorting the values of $\operatorname{PSD}_X(s)$ and performing the check using the largest values $\operatorname{PSD}_X(s)$ before considering the smaller values. For example, if $\operatorname{PSD}_B(s) > \operatorname{PSD}_A(s)$ then we would check $\operatorname{PSD}_B(s) > 4n$ before checking $\operatorname{PSD}_A(s) + \operatorname{PSD}_B(s) > 4n$.

When n is odd we use Theorem 14 to provide additional information to the SAT solver. For simplicity, suppose we fix $a_0 = 1$; as shown in (Bright, 2017, §3.1.2) this can be done by fixing the sign of rowsum(A) to not necessarily be positive but to satisfy rowsum(A) $\equiv n \pmod 4$. Briefly, this is because the symmetry of A implies that the rowsum of A will always be congruent to $a_0 + n - 1 \pmod 4$. Also fixing the values $b_0 = c_0 = d_0 = 1$, Theorem 14 says that

$$a_k b_k c_k d_k = -1 \qquad \text{for } k = 1, \dots, (n-1)/2.$$

Thinking of the entries as Boolean variables, we can encode the multiplicative constraint in conjunctive normal form as

$$(a_k \lor b_k \lor c_k \lor d_k) \land (\lnot a_k \lor \lnot b_k \lor c_k \lor d_k) \land \cdots \land (\lnot a_k \lor \lnot b_k \lor \lnot c_k \lor \lnot d_k)$$

(that is, all the clauses on the four variables a_k, b_k, c_k, d_k with an even number of negative literals). We add these clauses for $k = 1, \dots, (n-1)/2$ into each SAT instance generated in each odd order n.

5. Results

We implemented the algorithm described in Section 4, including all optimizations, and ran it in all orders $n \leq 70$ divisible by 2 or 3. Step 1 was completed using the script NSOKS (Riel, 2006) in the computer algebra system MAPLE 18. Steps 2–4 and the postprocessing were completed using C++ code which used the library FFTW 3.3.6-pl2 by Frigo and Johnson (2005) for computing PSD values. Step 5 was completed using MAPLESAT (Liang et al., 2017) modified to support a programmatic interface and also used FFTW for computing PSD values. Since FFTW introduces some floating-point errors in the values it returns, when checking the PSD values of A we actually ensure that $\mathrm{PSD}_A(s) > 4n + \epsilon$ for some ϵ which is small but larger than the accuracy of the discrete Fourier transform used, e.g., $\epsilon = 10^{-2}$. Our computations were performed on a cluster of 64-bit Intel Xeon E5-2683V4 2.1 GHz processors limited to 6 GB of memory and running CentOS 7.

Timings for running our entire algorithm (in hours) in even orders 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 even 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 that we generated in each order, as well as the total number of sets of Williamson sequences that were found up to equivalence (denoted by $\#W_n$). The counts for $\#W_n$ are not identical to those given in (Bright et al., 2018a) because that work did not use the equivalence operation E5 (alternating negation) but the results up to order 64 (the largest order previously solved) are otherwise identical.

Table 3 contains the same information as Table 1 except in the odd orders. The bottleneck for our algorithm in these orders was the uncompression step (since uncompressing by a factor of 3 is more challenging than uncompressing by a factor of 2), i.e., solving the SAT instances. The counts for $\#W_n$ in these cases exactly match those given by Holzmann et al. (2008) up to 59, the largest order they solved. We found one

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	5
16	0.00	5	6
18	0.00	22	23
20	0.00	14	17
22	0.00	22	15
24	0.00	40	72
26	0.00	24	26
28	0.00	78	83
30	0.00	281	150
32	0.00	70	152
34	0.00	214	91
36	0.00	1013	477
38	0.00	360	50
40	0.01	4032	1499
42	0.02	2945	301
44	0.01	1163	249
46	0.03	1538	50
48	0.09	4008	9800
50	0.45	3715	275
52	0.78	4535	926
54	3.00	25798	498
56	0.98	18840	40315
58	15.97	9908	73
60	27.14	256820	4083
62	64.74	19418	61
64	65.52	34974	69960
66	764.96	109566	262
68	593.77	122150	1113
70	957.96	71861	98

Table 1: A summary of the running time in hours, number of SAT instances used, and number of inequivalent sets of Williamson sequences generated in each even order $n \leq 70$.

	SAT Solving Time (hours)	
n	CNF encoding	Programmatic
30	0.01	0.00
32	0.01	0.00
34	0.03	0.00
36	0.36	0.00
38	0.12	0.00
40	2.01	0.01
42	4.31	0.01
44	5.59	0.00
46	8.65	0.01
48	18.78	0.02
50	53.12	0.02
52	–	0.05
54	–	0.18
56	–	0.18
58	–	0.13
60	–	9.56
62	–	0.45
64	–	0.94
66	–	5.14
68	–	17.18
70	–	8.07

Table 2: The total time spent running MapleSAT in each even order $30 \leq n \leq 70$ using the CNF encoding and the programmatic encoding. A timeout of 100 hours was used.

n	Time (h)	# inst.	$\#W_n$
3	0.00	1	1
9	0.00	3	3
15	0.00	8	4
21	0.00	30	7
27	0.00	172	6
33	0.01	364	5
39	0.05	1527	1
45	1.12	15542	1
51	4.57	17403	2
57	61.26	58376	1
63	1670.95	466561	2
69	8162.50	600338	1

Table 3: A summary of the running time in hours, number of SAT instances used, and number of inequivalent sets of Williamson sequences generated in each odd order n divisible by 3 and less than 70.

previously unknown Williamson sequence of order 63 using 466,561 3-compressed quadruples. We give this Williamson sequence here explicitly, with '+' representing 1, '-' representing -1, and each sequence member on a new line:

```
+----+-+++-+-++++--++--+-----+-++-+-----+--++--++++-+-+++-+----
+--++-+-+--++++----+-++-++---+----+---++-++-+----++++--+-+-++--
+--++-+++-+++---++-----+-+---+----+---+-+-----++---+++-+++-++--
+++++-++----+++-+-++---+-++++-+--+-++++-+---++-+-+++----++-++++
```

We also used our enumeration of Williamson sequences of order $2n$ for $n \leq 35$ along with Theorem 18 to explicitly construct 8-Williamson sequences in all odd orders $n \leq 35$. Table 4 contains the counts of how many inequivalent sets of 8-Williamson sequences that can be constructed in this fashion (this does not count the total number of 8-Williamson sequences in order n, only those that can be constructed via the construction of Theorem 18). We explicitly give one example of an 8-Williamson sequence of order 35, with '+' representing 1 and '-' representing -1:

```
++++-+++--+-+----++----+-+--+++-+++ +++---+-+++-++--+--+--++-+++-+---++
++-+-+++-+-----++++++-----+-+++-+-+ ++-+--+--++---+-++++-+---++--+--+-+
++---++-+-+--+--------+--+-+-++---+ ++---++-+-+--+--------+--+-+-++---+
+--+++-----+---+-++-+---+-----+++-- +---++--++++-++-+--+-++-++++--++---
```

These sequences can be used to generate a Hadamard matrix of order $8 \cdot 35 = 280$; for details see Kotsireas and Koukouvinos (2006, 2009).

Additionally, our code and an explicit enumeration of all the Williamson sequences and 8-Williamson sequences that we constructed have been made available on our website `uwaterloo.ca/mathcheck`.

n	1	3	5	7	9	11	13	15	17
$\#8W_n \geq$	1	1	1	4	13	10	18	129	79
n	19	21	23	25	27	29	31	33	35
$\#8W_n \geq$	43	280	48	257	486	71	58	240	78

Table 4: The number of inequivalent sets of 8-Williamson sequences generated using Theorem 18 in each odd order $n \leq 35$ (and therefore a lower bound on $\#8W_n$, the number of inequivalent sets of 8-Williamson sequences in order n).

6. Conclusion and advice

In this paper we have shown the power of the SAT+CAS paradigm (i.e., the technique of applying the tools from the fields of satisfiability checking and symbolic computation) as well as the power and flexibility of the programmatic SAT approach. Our focus was applying the SAT+CAS paradigm to the Williamson conjecture from combinatorial design theory, but we believe the SAT+CAS paradigm shows promise to be applicable to many other problems and conjectures. In fact, the SAT+CAS paradigm has recently been used to enumerate complex Golay pairs (Bright et al., 2018b) and good matrices (Bright et al., 2019). However, the SAT+CAS paradigm is not something that can be effortlessly applied to problems or expected to be effective on all types of problems. Our experiments in this area allow us to offer some guidance about the kind of problems in which the SAT+CAS paradigm would work particularly well. In particular, Bright (2017) highlights the following properties of problems which makes them good candidates to study using the SAT+CAS paradigm:

1. *There is an efficient encoding of the problem into a Boolean setting.* Since the problem has to be translated into a SAT instance or multiple SAT instances the encoding should ideally be straightforward and easy to compute. Not only does this make the process of generating the SAT instances easier and less error-prone it also means that the SAT solver is executing its search through a domain which is closer to the original problem. In general, the more convoluted the encoding the less likely the SAT solver will be able to efficiently search the space. For example, in our application we were fortunate to be able to encode ± 1 values as Boolean variables.
2. *There is some way of splitting the Boolean formula into multiple instances using the knowledge from a CAS.* Of course, a SAT instance can always be split into multiple instances by hard-coding the values of certain variables and then generating instances which cover all possible assignments of those variables. However, this strategy is typically not a good way of splitting the search space. The instances generated in this fashion tend to have wildly unbalanced difficulties, with some very easy instances and some very hard instances, limiting the benefits of using many processors to search the space. Instead, the process of splitting using domain-specific knowledge allows instances which cannot be ruled out *a priori* to not even need to be generated because they encode some part of the search space which can be discarded based on domain-specific knowledge. For

example, in our application we only needed to generate SAT instances with a few possibilities for the rowsums of the sequences A, B, C, and D and could ignore all other possible rowsums.

3. *The search space can be split into a reasonable number of cases.* One of the disadvantages of using SAT solvers is that it can be difficult to tell how much progress is being made as the search is progressing. The process of splitting the search space allows one to get a better estimate of the progress being made, assuming the difficulty of the instances isn't extremely unbalanced. In our experience, splitting the search space into instances which can be solved relatively quickly worked well, assuming the number of instances isn't too large so that the overhead of calling the SAT solver is small. This allowed the space to be searched significantly faster (especially when using multiple processors) than a single instance would have taken to complete. In our application the order $n = 69$ required the most amount of splitting; in this case we split the search space into 600,338 SAT instances and each instance took an average of 48.8 seconds to solve.
4. *The SAT solver can learn something about the space as the search is running.* The efficiency of SAT solvers is in part due to the facts that they learn as the search progresses. It can often be difficult for a human to make sense of these facts but they play a vital role to the SAT solver internally and therefore a problem where the SAT solver can take advantage of its ability to learn nontrivial clauses is one in which the SAT+CAS paradigm is well suited for. For more sophisticated lemmas that the SAT solver would be unlikely to learn (because they rely on domain-specific knowledge) it is useful to learn clauses programmatically via the programmatic SAT idea (Ganesh et al., 2012). For example, the timings in Table 2 show how important the learned programmatic clauses were to the efficiency of the SAT solver.
5. *There is domain-specific knowledge which can be efficiently given to the SAT solver.* Domain-specific knowledge was found to be critical to solving instances of the problems besides those of the smallest sizes. The instances which were generated using naive encodings were typically only able to be solved for small sizes and all significant increases in the size of the problems past that point came from the usage of domain-specific knowledge. Of course, for the information to be useful to the solver there needs to be an efficient way for the solver to be given the information; it can be encoded directly in the SAT instances or generated on-the-fly using programmatic SAT functionality. For example, in our application we show how to encode Williamson's product theorem for odd orders n directly into the SAT instance in Section 4.7 and we show how to programmatically encode the PSD test in Section 3.2.
6. *The solutions of the problem lie in spaces which cannot be simply enumerated.* If the search space is highly structured and there exists an efficient search algorithm that exploits that structure then using this algorithm directly is likely to be a better choice. A SAT solver could also perform this search but would probably do so less efficiently; instead, SAT solvers have a relative advantage when the search space is less structured. For example, in our application we require searching for sequences whose compressions are equal to some given sequence and use the

PSD test to filter certain sequences from consideration. The space is specified by a number of simple but "messy" constraints and SAT solvers are good at dealing with that kind of complexity.

Perhaps the most surprising result of our work on the Williamson conjecture is our discovery 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 $18 \leq n \leq 70$ has $\#W_n > 10$ and there are some orders which contain thousands of inequivalent Williamson matrices. Part of this dichotomy can be explained by Theorem 17 which generates Williamson matrices of order $2n$ from Williamson matrices of odd order n. For example, the two classes of Williamson matrices of order 10 can be generated from the single class of Williamson matrices of order 5. However, this still does not fully explain the relative abundance of Williamson matrices in even orders. In particular, it cannot possibly explain why Williamson matrices exist in order 70 because Williamson matrices of order 35 do not exist.

Recently Acevedo and Dietrich (2019) discovered a remarkable construction for Williamson matrices using perfect quaternionic sequences. Their construction can be used to explain the existence of Williamson matrices of order 70 and we were able to use their method to construct 40 of the 98 inequivalent sets of Williamson matrices that we found in order 70.

Acknowledgements

This work was made possible because of the resources available on the petascale supercomputer Graham at the University of Waterloo, managed by Compute Canada and SHARCNET, the Shared Hierarchical Academic Research Computing Network. The authors thank Dragomir Ðoković informing us about the construction of Turyn (1970) using complex Hadamard matrices. We also thank the reviewers for their very detailed reading of our work and providing comments that greatly improved this article.

References

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, ISSAC 2015, Bath, United Kingdom, July 6–9, 2015. pp. 1–6.
URL https://doi.org/10.1145/2755996.2756636

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., 2016. SC^2: Satisfiability checking meets symbolic computation. In: Intelligent Computer Mathematics: 9th International Conference, CICM 2016, Bialystok, Poland, July 25–29, 2016, Proceedings. Springer International Publishing, Cham, pp. 28–43, http://www.sc-square.org/.
URL https://doi.org/10.1007/978-3-319-42547-4_3

Acevedo, S. B., Dietrich, H., 2019. New infinite families of Williamson Hadamard matrices. Australasian Journal of Combinatorics 73 (1), 207–219.

Agayan, S., Sarukhanyan, A., 1981. Recurrence formulas for the construction of Williamson-type matrices. Mathematical notes of the Academy of Sciences of the USSR 30 (4), 796–804.
URL `https://doi.org/10.1007/BF01137811`

Barrett, C., Fontaine, P., Tinelli, C., 2016. The Satisfiability Modulo Theories Library (SMT-LIB).
URL `http://www.smt-lib.org`

Barrett, C., Sebastiani, R., Seshia, S., Tinelli, C., 2009. Satisfiability modulo theories. In: Biere, A., Heule, M. J. H., van Maaren, H., Walsh, T. (Eds.), Handbook of Satisfiability. Vol. 185 of Frontiers in Artificial Intelligence and Applications. pp. 825–885.
URL `https://doi.org/10.3233/978-1-58603-929-5-825`

Baumert, L., Golomb, S. W., Hall, M., 1962. Discovery of an Hadamard matrix of order 92. Bull. Amer. Math. Soc. 68 (3), 237–238.
URL `https://doi.org/10.1090/S0002-9904-1962-10761-7`

Baumert, L., Hall, M., 1965. Hadamard matrices of the Williamson type. Mathematics of Computation 19 (91), 442–447.
URL `https://doi.org/10.2307/2003674`

Bayless, S., Val, C. G., Ball, T., Hoos, H. H., Hu, A. J., 2013. Efficient modular SAT solving for IC3. In: Jobstmann, B., Ray, S. (Eds.), 2013 Formal Methods in Computer-Aided Design. IEEE, pp. 149–156.
URL `https://doi.org/10.1109/FMCAD.2013.6679404`

Bose, R. C., Shrikhande, S. S., 1959. A note on a result in the theory of code construction. Information and Control 2 (2), 183–194.
URL `https://doi.org/10.1016/S0019-9958(59)90376-6`

Bright, C., 2017. Computational methods for combinatorial and number theoretic problems. Ph.D. thesis, University of Waterloo, Ontario, Canada.
URL `http://hdl.handle.net/10012/11761`

Bright, C., Đoković, D. Ž., Kotsireas, I., Ganesh, V., 2019. A SAT+CAS approach to finding good matrices: New examples and counterexamples. In: Hentenryck, P. V., Zhou, Z.-H. (Eds.), Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence. pp. 1435–1442.
URL `https://doi.org/10.1609/aaai.v33i01.33011435`

Bright, C., Ganesh, V., Heinle, A., Kotsireas, I. S., Nejati, S., Czarnecki, K., 2016. MATHCHECK2: A SAT+CAS verifier for combinatorial conjectures. In: Gerdt, V. P., Koepf, W., Seiler, W. M., Vorozhtsov, E. V. (Eds.), Computer Algebra in Scientific Computing - 18th International Workshop, CASC 2016, Bucharest, Romania,

September 19–23, 2016, Proceedings. Vol. 9890 of Lecture Notes in Computer Science. Springer International Publishing, Cham, pp. 117–133.
URL https://doi.org/10.1007/978-3-319-45641-6_9

Bright, C., Kotsireas, I., Ganesh, V., 2018a. A SAT+CAS method for enumerating Williamson matrices of even order. In: McIlraith, S. A., Weinberger, K. Q. (Eds.), Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2–7, 2018. pp. 6573–6580, preprint presented at the second SC^2 workshop in Kaiserslautern, Germany.
URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16625

Bright, C., Kotsireas, I. S., Heinle, A., Ganesh, V., 2018b. Enumeration of complex Golay pairs via programmatic SAT. In: Proceedings of the 2018 ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2018, New York, NY, USA, July 16–19, 2018. pp. 111–118.
URL https://doi.org/10.1145/3208976.3209006

Cooper, J., 2013. Hadamard matrix. JPL Slice of History Blog.
URL https://www.jpl.nasa.gov/blog/2013/8/hadamard-matrix

Đoković, D. Ž., 1992. Williamson matrices of orders $4 \cdot 29$ and $4 \cdot 31$. Journal of Combinatorial Theory, Series A 59 (2), 309–311.
URL https://doi.org/10.1016/0097-3165(92)90071-2

Đoković, D. Ž., 1993. Williamson matrices of order $4n$ for $n = 33, 35, 39$. Discrete mathematics 115 (1), 267–271.
URL https://doi.org/10.1016/0012-365X(93)90495-F

Đoković, D. Ž., 1995. Note on Williamson matrices of orders 25 and 37. Journal of Combinatorial Mathematics and Combinatorial Computing 18, 171–175.

Đoković, D. Ž., Kotsireas, I. S., 2015. Compression of periodic complementary sequences and applications. Designs, Codes and Cryptography 74 (2), 365–377.
URL https://doi.org/10.1007/s10623-013-9862-z

Frigo, M., Johnson, S. G., 2005. The design and implementation of FFTW3. Proceedings of the IEEE 93 (2), 216–231.
URL https://doi.org/10.1109/JPROC.2004.840301

Ganesh, V., O'Donnell, C. W., Soos, M., Devadas, S., Rinard, M. C., Solar-Lezama, A., 2012. Lynx: A programmatic SAT solver for the RNA-folding problem. In: Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17–20, 2012. Proceedings. pp. 143–156.
URL https://doi.org/10.1007/978-3-642-31612-8_12

Ganzinger, H., Hagen, G., Nieuwenhuis, R., Oliveras, A., Tinelli, C., 2004. DPLL(T): Fast decision procedures. In: International Conference on Computer Aided Verification. Springer, pp. 175–188.
URL https://doi.org/10.1007/978-3-540-27813-9_14

Hall, M., 1998. Combinatorial theory. Vol. 71. John Wiley & Sons.

Heule, M. J. H., 2018. Schur number five. In: McIlraith, S. A., Weinberger, K. Q. (Eds.), Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2–7, 2018. pp. 6598–6606.
URL `https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16952`

Heule, M. J. H., Kullmann, O., Marek, V. W., 2016. Solving and verifying the Boolean Pythagorean triples problem via cube-and-conquer. In: Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5–8, 2016, Proceedings. pp. 228–245.
URL `https://doi.org/10.1007/978-3-319-40970-2_15`

Holzmann, W. H., Kharaghani, H., Tayfeh-Rezaie, B., 2008. Williamson matrices up to order 59. Designs, Codes and Cryptography 46 (3), 343–352.
URL `https://doi.org/10.1007/s10623-007-9163-5`

Jovanović, D., de Moura, L., 2012. Solving non-linear arithmetic. In: International Joint Conference on Automated Reasoning. Springer, pp. 339–354.
URL `https://doi.org/10.1007/978-3-642-31365-3_27`

Konev, B., Lisitsa, A., 2014. A SAT attack on the Erdős discrepancy conjecture. In: Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14–17, 2014. Proceedings. pp. 219–226.
URL `https://doi.org/10.1007/978-3-319-09284-3_17`

Kotsireas, I., Koukouvinos, C., 2006. Constructions for Hadamard matrices of Williamson type. Journal of Combinatorial Mathematics and Combinatorial Computing 59, 17–32.

Kotsireas, I. S., Koukouvinos, C., 2009. Hadamard matrices of Williamson type: A challenge for computer algebra. Journal of Symbolic Computation 44 (3), 271–279.
URL `https://doi.org/10.1016/j.jsc.2007.05.004`

Kotsireas, I. S., Koukouvinos, C., Pardalos, P. M., 2010. An efficient string sorting algorithm for weighing matrices of small weight. Optimization Letters 4 (1), 29–36.
URL `https://doi.org/10.1007/s11590-009-0144-7`

Koukouvinos, C., 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.
URL `https://doi.org/10.1016/0012-365X(88)90040-4`

Koukouvinos, C., Kounias, S., 1990. There are no circulant symmetric Williamson matrices of order 39. Journal of Combinatorial Mathematics and Combinatorial Computing 7, 161–169.

Lang, W., Schneider, E., Jan 2012. Turyn type Williamson matrices up to order 99. Designs, Codes and Cryptography 62 (1), 79–84.
URL https://doi.org/10.1007/s10623-011-9492-2

Liang, J. H., Govind V. K., H., Poupart, P., Czarnecki, K., Ganesh, V., 2017. An empirical study of branching heuristics through the lens of global learning rate. In: Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28–September 1, 2017, Proceedings. pp. 119–135.
URL https://doi.org/10.1007/978-3-319-66263-3_8

Riel, J., 2006. NSOKS: A Maple script for writing *n* as a sum of *k* squares.
URL https://swmath.org/software/21060

Sawade, K., 1977. Hadamard matrices of order 100 and 108. Bulletin of Nagoya Institute of Technology (29), 147–153.
URL http://ci.nii.ac.jp/naid/40002775009/en/

Schmidt, B., 1999. Williamson matrices and a conjecture of Ito's. Designs, Codes and Cryptography 17 (1-3), 61–68.
URL https://doi.org/10.1023/A:1008398319853

Tseng, C.-C., Liu, C., 1972. Complementary sets of sequences. IEEE Transactions on Information theory 18 (5), 644–652.
URL https://doi.org/10.1109/TIT.1972.1054860

Turyn, R. J., 1970. Complex Hadamard matrices. In: Combinatorial structures and their applications. Gordon and Breach, pp. 435–437.

Turyn, R. J., 1972. An infinite class of Williamson matrices. Journal of Combinatorial Theory, Series A 12 (3), 319–321.
URL https://doi.org/10.1016/0097-3165(72)90095-7

Wallis, J. S., 1974. Williamson matrices of even order. In: Holton, D. A. (Ed.), Combinatorial Mathematics: Proceedings of the Second Australian Conference. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 132–142.
URL https://doi.org/10.1007/BFb0057387

Williamson, J., 1944. Hadamard's determinant theorem and the sum of four squares. Duke Math. J 11 (1), 65–81.
URL https://doi.org/10.1215/S0012-7094-44-01108-7

Zulkoski, E., Bright, C., Heinle, A., Kotsireas, I. S., Czarnecki, K., Ganesh, V., 2017. Combining SAT solvers with computer algebra systems to verify combinatorial conjectures. J. Autom. Reasoning 58 (3), 313–339.
URL https://doi.org/10.1007/s10817-016-9396-y

Zulkoski, E., Ganesh, V., Czarnecki, K., 2015. MATHCHECK: A math assistant via a combination of computer algebra systems and SAT solvers. In: Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany,

August 1–7, 2015, Proceedings. pp. 607–622.
URL `https://doi.org/10.1007/978-3-319-21401-6_41`

Appendix: Proofs

Theorem 15. *If A, B, C, D are Williamson sequences of even order $n = 2m$ then*

$$a_i b_i c_i d_i = a_{i+m} b_{i+m} c_{i+m} d_{i+m} \qquad \text{for } 0 \le i < m.$$

Proof. We can equivalently consider members of Williamson sequences to be elements of the group ring $\mathbb{Z}[C_n]$ where C_n is a cyclic group of order n with generator u. In such a formulation we have $X = x_0 + x_1 u + \cdots + x_{n-1} u^{n-1}$ and Williamson sequences are quadruples (A, B, C, D) whose members have ± 1 coefficients, whose coefficients form symmetric sequences of length n, and which satisfy

$$A^2 + B^2 + C^2 + D^2 = 4n.$$

Let $P_X = \sum_{x_i=1} u^i$ (with the sum over $0 \le i < n$) and let p_X denote the number of positive coefficients in X. As shown in (Hall, 1998, 14.2.20) we have that $P_A^2 + P_B^2 + P_C^2 + P_D^2$ is equal to

$$(p_A + p_B + p_C + p_D - n) \sum_{i=0}^{n-1} u^i + n. \tag{1}$$

Furthermore, by the fact that $P_X^2 \equiv \sum_{x_i=1} u^{2i} \pmod 2$, $P_A^2 + P_B^2 + P_C^2 + P_D^2$ is congruent to

$$\sum_{a_i=1} u^{2i} + \sum_{b_i=1} u^{2i} + \sum_{c_i=1} u^{2i} + \sum_{d_i=1} u^{2i} \pmod 2. \tag{2}$$

Now, if n is even then (1) reduces to

$$(p_A + p_B + p_C + p_D) \sum_{i=0}^{n-1} u^i \pmod 2$$

so all coefficients are the same mod 2. Since by (2) the coefficients with odd index are 0 mod 2, all coefficients in (1) and (2) must be 0 mod 2.

Note that $u^k = u^{2i}$ has exactly 2 solutions for given even k with $0 \le k < n$, namely, $i = k/2$ and $i = (k+n)/2$. Then (2) can be rewritten as

$$\sum_{a_{k/2}=1} u^k + \sum_{a_{(k+n)/2}=1} u^k + \cdots + \sum_{d_{(k+n)/2}=1} u^k \pmod 2$$

where the sums are over the even k with $0 \le k < n$. Since each coefficient must be 0 mod 2, there must be an even number of 1s among the entries $a_{k/2}, a_{(k+n)/2}, \ldots, d_{(k+n)/2}$ for each even k with $0 \le k < n$, i.e.,

$$a_{k/2} a_{(k+n)/2} b_{k/2} b_{(k+n)/2} c_{k/2} c_{(k+n)/2} d_{k/2} d_{(k+n)/2} = 1.$$

The required result is a rearrangement of this and rewriting with the definition $i = k/2$. $\square$

Corollary 16. *If A', B', C', D' are the 2-compressions of a set of Williamson sequences then* $A' + B' + C' + D' \equiv [0, \dots, 0] \pmod 4$.

Proof. Let N_+ and N_- denote the number of 1s and -1s in the eight Williamson sequence entries $a_i, b_i, c_i, d_i, a_{i+m}, b_{i+m}, c_{i+m}$, and d_{i+m}, where $0 \le i < m$. We have that $N_+ + N_- = 8$ and that $N_+ - N_- = a'_i + b'_i + c'_i + d'_i$ (the sum of the above eight Williamson sequence entries). Thus the ith entry of $A' + B' + C' + D'$ is $N_+ - N_- = N_+ - (8 - N_+) = 2N_+ - 8 \equiv 0 \pmod 4$ since Theorem 15 implies that N_+ must be even. □

Theorem 17. *Let A, B, C, D be Williamson sequences of odd order n. Then*

$$A ш B',\ (-A) ш B',\ C ш D',\ (-C) ш D'$$

are Williamson sequences of order $2n$.

Proof. The fact that the constructed sequences have ± 1 entries are of length $2n$ follows directly from the properties of the three types of operations used to generate them. The fact that they are symmetric follows from the fact that the sequences X which appear to the left of ш satisfy $x_k = x_{n-k}$ for $k = 1, \dots, n-1$ and the sequences Y which appear to the right of ш satisfy $y_k = y_{n-k-1}$ for $k = 0, \dots, n-1$ which are exactly the necessary properties for $X ш Y$ to be symmetric.

Let L be the list containing the constructed sequences of order $2n$. To show these sequences are Williamson we need to show that

$$\sum_{X \in L} \mathrm{PAF}_X(s) = 0$$

for $s = 1, \dots, n$. Using the properties that $\mathrm{PAF}_{-X}(s) = \mathrm{PAF}_X(s)$, $\mathrm{PAF}_{X'}(s) = \mathrm{PAF}_X(s)$, and $\mathrm{PAF}_{X ш Y}(s) = \mathrm{PAF}_X(s/2) + \mathrm{PAF}_Y(s/2)$ when s is even and in this range we obtain

$$\sum_{X \in L} \mathrm{PAF}_X(s) = 2 \sum_{X = A,B,C,D} \mathrm{PAF}_X(s/2) = 0$$

since A, B, C, D are Williamson. When s is odd we have that

$$\mathrm{PAF}_{(-X) ш Y}(s) = -\mathrm{PAF}_{X ш Y}(s)$$

and using this for $(X, Y) = (A, B')$ and (C, D') derives the desired property. □

Theorem 18. *Let A, B, C, D be Williamson sequences of order $2n$ with n odd and write*

$$A = A_1 ш A'_2,\ B = B_1 ш B'_2,\ C = C_1 ш C'_2,\ D = D_1 ш D'_2.$$

Then A_1, A_2, B_1, B_2, C_1, C_2, D_1, D_2 are 8-Williamson sequences of order n.

Proof. The fact that the constructed sequences are symmetric, have ± 1 entries, and are of order n follows directly from the construction and because the sequences they are constructed from are symmetric, have ± 1 entries, and are of order $2n$. Since A, B, C, D are Williamson we have that

$$\sum_{X = A,B,C,D} \mathrm{PAF}_X(2s) = 0 \qquad \text{for } s = 1, \dots, n-1.$$

Using the fact that $\mathrm{PAF}_{X ш Y}(2s) = \mathrm{PAF}_X(s) + \mathrm{PAF}_Y(s)$ and $\mathrm{PAF}_{Y'}(s) = \mathrm{PAF}_Y(s)$ this sum becomes exactly the Williamson property $\sum_{i=1}^{8} \mathrm{PAF}_{X_i}(s) = 0$. □