PhD Research Proposal

A SAT+CAS system for checking math conjectures

Curtis Bright

University of Waterloo

March 14, 2016

Motivation

> *The research areas of SMT [SAT-Modulo-Theories] solving and symbolic computation are quite disconnected. On the one hand, SMT solving has its strength in efficient techniques for exploring Boolean structures, learning, combining solving techniques, and developing dedicated heuristics, but its current focus lies on easier theories and it makes use of symbolic computation results only in a rather naive way.*

Erica Ábrahám[1]

[1] Building bridges between symbolic computation and satisfiability checking. *ISSAC 2015*.

Satisfiability checking

Problem statement

Given a logical formula, determine if it is satisfiable.

- A logical formula is an expression involving Boolean variables and logical connectives such as $\wedge$, $\vee$, $\neg$.
- A formula is satisfiable if there exists an assignment to the variables which make the formula true.

Satisfiability checking

Example

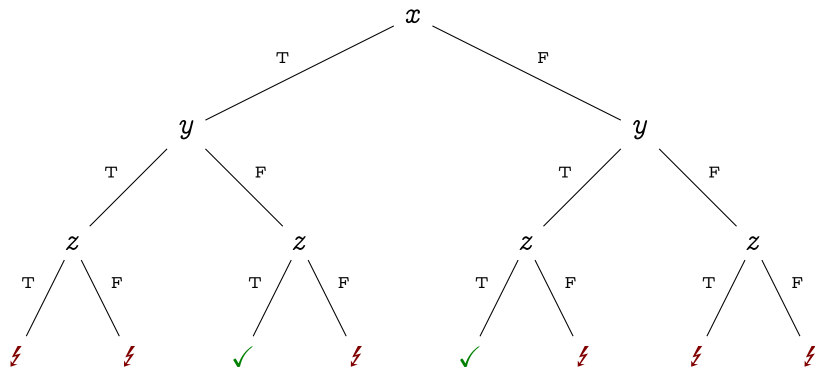
Is $(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y) \wedge z$ satisfiable?

Satisfiability checking

Example

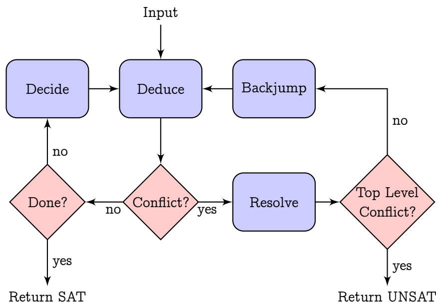
Is $(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y) \wedge z$ satisfiable?

Yes, as shown by a tree of possible assignments:



DPLL algorithm

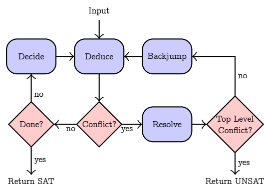
- Deduce: Simplify clauses to detect conflicts and infer new values of variables.
- Decide: Choose an unassigned variable and assign it a value.
- Resolve: If a conflict occurs, learn a clause prohibiting the current assignment and *backjump* (undo variable choices leading to the conflict).



DPLL algorithm

Example

Is $(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee \neg u) \wedge z \wedge (\neg y \vee u)$ satisfiable?



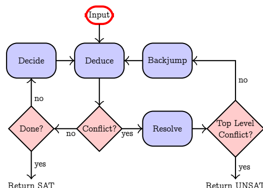
DPLL algorithm

Example

Is $(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee \neg u) \wedge z \wedge (\neg y \vee u)$ satisfiable?

Initial clauses

- $x \vee y \vee \neg z$
- $\neg x \vee \neg y \vee \neg u$
- z
- $\neg y \vee u$

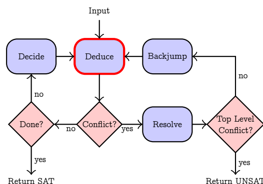


DPLL algorithm

Example

Is $(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee \neg u) \wedge z \wedge (\neg y \vee u)$ satisfiable?

Initial clauses



- $x \vee y \vee \neg z$
- $\neg x \vee \neg y \vee \neg u$
- z
- $\neg y \vee u$

Deduce

From clause 3, z must be true. Simplified clauses:

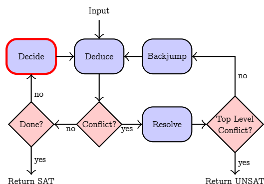
- $x \vee y$
- $\neg x \vee \neg y \vee \neg u$
- $\neg y \vee u$

DPLL algorithm

Decide

Choose x to be true.

- $x \vee y$
- $\neg x \vee \neg y \vee \neg u$
- $\neg y \vee u$

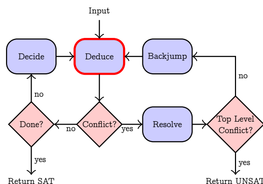


DPLL algorithm

Decide

Choose x to be true.

- $x \vee y$
- $\neg x \vee \neg y \vee \neg u$
- $\neg y \vee u$



Deduce

Propagate the fact that x is true. Simplified clauses:

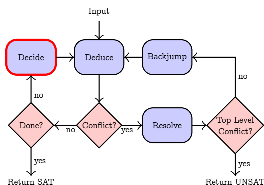
- $\neg y \vee \neg u$
- $\neg y \vee u$

DPLL algorithm

Decide

Choose y to be true.

- $\neg y \vee \neg u$
- $\neg y \vee u$

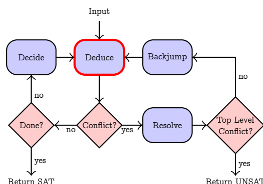


DPLL algorithm

Decide

Choose y to be true.

- $\neg y \vee \neg u$
- $\neg y \vee u$

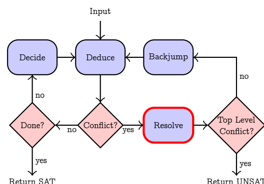


Deduce

Propagate the fact that y is true. Simplified clauses:

- $\neg u$
- u

DPLL algorithm



Decide

Choose y to be true.

- $\neg y \vee \neg u$
- $\neg y \vee u$

Deduce

Propagate the fact that y is true. Simplified clauses:

- $\neg u$
- u

Resolve

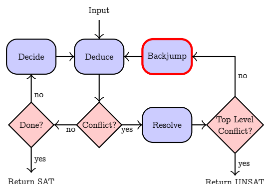
Conflict: No way to choose u. At least one variable assignment must change, so we learn the clause $\neg z \vee \neg x \vee \neg y$.

DPLL algorithm

Backjump

Clauses after undoing the last variable choice:

- $\neg y \vee \neg u$
- $\neg y \vee u$
- $\neg z \vee \neg x \vee \neg y$

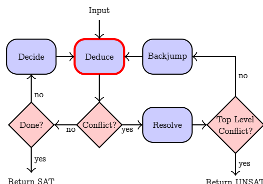


DPLL algorithm

Backjump

Clauses after undoing the last variable choice:

- $\neg y \vee \neg u$
- $\neg y \vee u$
- $\neg z \vee \neg x \vee \neg y$



Deduce

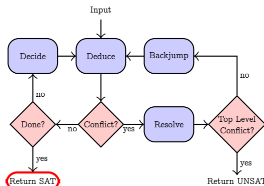
Since x and z are true, clause 3 simplifies to $\neg y$ and y must be false. Simplified clauses: $\emptyset$

DPLL algorithm

Backjump

Clauses after undoing the last variable choice:

- $\neg y \vee \neg u$
- $\neg y \vee u$
- $\neg z \vee \neg x \vee \neg y$



Deduce

Since x and z are true, clause 3 simplifies to $\neg y$ and y must be false. Simplified clauses: $\emptyset$

Satisfying assignment

Take x true, y false, and z true.

SAT-Modulo-Theories (SMT)

It is possible to consider the satisfiability problem for different types of logical formulas, e.g., those of first-order logic over various theories.

- theory of strings
- array theory
- bitvector theory
- theories of arithmetic
  - integer or real
  - linear or nonlinear

SAT-Modulo-Theories (SMT)

Example

Is the formula $x^2 < 0 \lor x^2 > 1$ satisfiable in the integer theory of arithmetic?

SAT-Modulo-Theories (SMT)

Example

Is the formula $x^2 < 0 \vee x^2 > 1$ satisfiable in the integer theory of arithmetic?

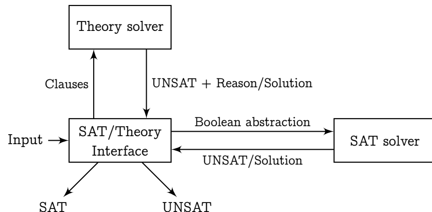
Yes, by taking $x = 2$.

SMT Solvers

First, translate the given formula into a propositional one:

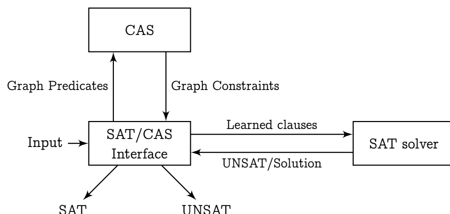
$$\overbrace{x^2 < 0}^{a} \vee \overbrace{x^2 > 1}^{b} \quad \text{becomes} \quad a \vee b$$

A SAT solver can then find a satisfying assignment (set a true).
A *theory solver* needs to be queried to determine if the assignment yields a solution of the original formula (and if not, why not).
Here the theory solver can produce the clause $\neg a$ (i.e., $x^2 \geqslant 0$).



The MathCheck System

- Uses SAT and CAS functionality to finitely verify or counterexample conjectures in mathematics[2].
- Verified two conjectures in graph theory to new bounds.
- Similar to a SMT solver with the theory solver replaced by a CAS.



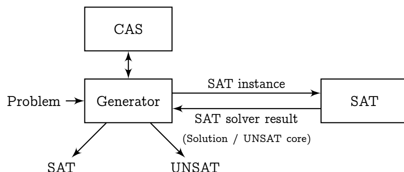
Authors

Edward Zulkoski, Vijay Ganesh, Krzysztof Czarnecki

[2] MathCheck: A Math Assistant via a Combination of Computer Algebra Systems and SAT Solvers. *CADE 2015*.

The MathCheck2 System

- Also uses SAT and CAS functionality to finitely verify or counterexample conjectures in mathematics[3].
- Used to study conjectures in combinatorial design theory about the existence of Hadamard matrices.



Authors

Curtis Bright, Vijay Ganesh, Albert Heinle, Ilias Kotsireas, Saeed Nejati, Krzysztof Czarnecki

[3] MathCheck2: A SAT+CAS Verifier for Combinatorial Conjectures. Submitted to *IJCAR 2016*.

Contributions

- Demonstration of usefulness of employing SAT to combinatorial conjectures.
- Three general techniques for improving the search.
- Verification that Williamson matrices of order 35 do not exist.
- Description of an algorithm for finding Williamson matrices of a given order (or showing none exist).
- Found 160 Hadamard matrices not in the library of the CAS Magma.

Experimental Results

The result that Williamson matrices of order 35 do not exist was shown in under 9 hours of computation time on SHARCNET[4]. This was first shown by Đoković[5], who requested an independent verification.

MathCheck2 was also able to find Williamson matrices for all orders $n < 35$.

[4] 64-bit AMD Opteron processors running at 2.2 GHz

[5] Williamson matrices of order $4n$ for $n = 33, 35, 39$. *Discrete Mathematics.*

Hadamard matrices

- square matrix with ± 1 entries
- any two distinct rows are orthogonal

Example

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & 1 & 1 & -1 \end{bmatrix}$$

Conjecture

An $n \times n$ Hadamard matrix exists for any n a multiple of 4.

Verifying H is Hadamard

When H is of order n, want to have

$$HH^{\mathrm{T}} = nI_n.$$

Need to verify that $\binom{n}{2}$ inner products are 0.

Example

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & 1 & 1 & -1 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

Williamson Matrices

- $n \times n$ matrices A, B, C, D
- entries ± 1
- symmetric, circulant
- $A^2 + B^2 + C^2 + D^2 = 4nI_n$

Example

$$A = B = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \qquad C = D = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

Williamson construction

A Hadamard matrix of order $4n$ can be constructed from Williamson matrices.

Symmetric and Circulant Matrices

Such matrices are defined by their first $\lceil \frac{n+1}{2} \rceil$ entries so we may refer to them as if they were sequences.

Examples ($n = 5$ and 6)

$$\begin{bmatrix} a_0 & a_1 & a_2 & a_2 & a_1 \\ a_1 & a_0 & a_1 & a_2 & a_2 \\ a_2 & a_1 & a_0 & a_1 & a_2 \\ a_2 & a_2 & a_1 & a_0 & a_1 \\ a_1 & a_2 & a_2 & a_1 & a_0 \end{bmatrix} \qquad \begin{bmatrix} a_0 & a_1 & a_2 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_1 & a_2 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_1 & a_2 & a_3 \\ a_3 & a_2 & a_1 & a_0 & a_1 & a_2 \\ a_2 & a_3 & a_2 & a_1 & a_0 & a_1 \\ a_1 & a_2 & a_3 & a_2 & a_1 & a_0 \end{bmatrix}$$

symmetric conditions — circulant conditions

Naive Hadamard Encoding

The property "H is a Hadamard matrix" can be expressed as a logical formula.

Each entry of H is represented using a *Boolean variable* encoding with $\mathrm{BV}(1) = \mathsf{true}$ and $\mathrm{BV}(-1) = \mathsf{false}$.

Multiplication becomes XNOR under this encoding, i.e.,

$$\mathrm{BV}(x \cdot y) = \neg(\mathrm{BV}(x) \oplus \mathrm{BV}(y)) \qquad \text{for } x, y \in \{\pm 1\}.$$

Naive Hadamard Encoding

Arithmetic formula encoding

$$\sum_{k=0}^{n-1} h_{ik} \cdot h_{jk} = 0 \qquad \text{for all } i \neq j.$$

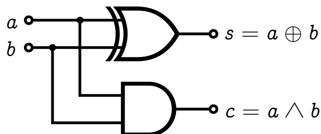
Boolean variable encoding

Using 'product' variables $p_{ijk} \coloneqq \neg(\mathrm{BV}(h_{ik}) \oplus \mathrm{BV}(h_{jk}))$ this becomes the cardinality constraints

$$\left(\#\ p_{ijk} \text{ true in } \{p_{ijk}\}_{k=0}^{n-1}\right) = \frac{n}{2} \qquad \text{for all } i \neq j.$$

Naive Hadamard Encoding

A *binary adder* consumes Boolean values and produces Boolean values; when thought of as bits, the outputs contain the binary representation of how many inputs were true.



Binary adder encoding

In order to encode the cardinality constraints, we use a network of binary adders with $\{p_{ijk}\}_{k=0}^{n-1}$ as inputs.
The output will be $\lfloor \log_2 n \rfloor + 1$ new variables which store the count of how many inputs are true.

Williamson Encoding

Similar to general encoding, but only using the variables

$$\{(a_i, b_i, c_i, d_i)\}_{i=0}^{\lceil (n-1)/2 \rceil}.$$

Because of the symmetric and circulant properties, only need to verify that the first $\lceil \frac{n-1}{2} \rceil$ off-diagonal entries of

$$A^2 + B^2 + C^2 + D^2$$

are zero. This condition can be rewritten using *periodic autocorrelation*.

Periodic Autocorrelation Function

The *periodic autocorrelation function* of the sequence A is

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

Periodic and symmetric properties

- $\mathrm{PAF}_A(s) = \mathrm{PAF}_A(s \bmod n)$
- $\mathrm{PAF}_A(s) = \mathrm{PAF}_A(n - s)$

Periodic Autocorrelation Function

Example

The periodic autocorrelation of $A = [1, 1, -1, -1, 1]$ is given by:

$$
\begin{aligned}
\mathrm{PAF}_A(0) &= 1^2 + 1^2 + (-1)^2 + (-1)^2 + 1^2 &&= 5 \\
\mathrm{PAF}_A(1) &= 1^2 + (-1) + (-1)^2 + (-1) + 1^2 &&= 1 \\
\mathrm{PAF}_A(2) &= (-1) + (-1) + (-1) + (-1) + 1^2 &&= -3 \\
\mathrm{PAF}_A(3) &= (-1) + 1^2 + (-1) + (-1) + (-1) &&= -3 \\
\mathrm{PAF}_A(4) &= 1^2 + 1^2 + (-1) + (-1)^2 + (-1) &&= 1
\end{aligned}
$$

Periodic Autocorrelation Function

Example

The periodic autocorrelation of $A = [1, 1, -1, -1, 1]$ is given by:

$$
\begin{aligned}
\mathrm{PAF}_A(0) &= 1^2 + 1^2 + (-1)^2 + (-1)^2 + 1^2 &&= 5 \\
\mathrm{PAF}_A(1) &= 1^2 + (-1) + (-1)^2 + (-1) + 1^2 &&= 1 \\
\mathrm{PAF}_A(2) &= (-1) + (-1) + (-1) + (-1) + 1^2 &&= -3 \\
\mathrm{PAF}_A(3) &= (-1) + 1^2 + (-1) + (-1) + (-1) &&= -3 \\
\mathrm{PAF}_A(4) &= 1^2 + 1^2 + (-1) + (-1)^2 + (-1) &&= 1
\end{aligned}
$$

Periodic Autocorrelation Function

Example

The periodic autocorrelation of $A = [1, 1, -1, -1, 1]$ is given by:

$$
\begin{aligned}
\mathrm{PAF}_A(0) &= 1^2 + 1^2 + (-1)^2 + (-1)^2 + 1^2 &&= 5 \\
\mathrm{PAF}_A(1) &= 1^2 + (-1) + (-1)^2 + (-1) + 1^2 &&= 1 \\
\mathrm{PAF}_A(2) &= (-1) + (-1) + (-1) + (-1) + 1^2 &&= -3 \\
\mathrm{PAF}_A(3) &= (-1) + 1^2 + (-1) + (-1) + (-1) &&= -3 \\
\mathrm{PAF}_A(4) &= 1^2 + 1^2 + (-1) + (-1)^2 + (-1) &&= 1
\end{aligned}
$$

Periodic Autocorrelation Function

Example

The periodic autocorrelation of $A = [1, 1, -1, -1, 1]$ is given by:

$$
\begin{aligned}
\mathrm{PAF}_A(0) &= 1^2 + 1^2 + (-1)^2 + (-1)^2 + 1^2 &&= 5 \\
\mathrm{PAF}_A(1) &= 1^2 + (-1) + (-1)^2 + (-1) + 1^2 &&= 1 \\
\mathrm{PAF}_A(2) &= (-1) + (-1) + (-1) + (-1) + 1^2 &&= -3 \\
\mathrm{PAF}_A(3) &= (-1) + 1^2 + (-1) + (-1) + (-1) &&= -3 \\
\mathrm{PAF}_A(4) &= 1^2 + 1^2 + (-1) + (-1)^2 + (-1) &&= 1
\end{aligned}
$$

Periodic Autocorrelation Function

Example

The periodic autocorrelation of $A = [1, 1, -1, -1, 1]$ is given by:

$$
\begin{aligned}
\mathrm{PAF}_A(0) &= 1^2 + 1^2 + (-1)^2 + (-1)^2 + 1^2 && = 5 \\
\mathrm{PAF}_A(1) &= 1^2 + (-1) + (-1)^2 + (-1) + 1^2 && = 1 \\
\mathrm{PAF}_A(2) &= (-1) + (-1) + (-1) + (-1) + 1^2 && = -3 \\
\mathrm{PAF}_A(3) &= (-1) + 1^2 + (-1) + (-1) + (-1) && = -3 \\
\mathrm{PAF}_A(4) &= 1^2 + 1^2 + (-1) + (-1)^2 + (-1) && = 1
\end{aligned}
$$

Williamson Encoding

The sth entry of

$$A^2 + B^2 + C^2 + D^2$$

is

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

To verify A, B, C, D are Williamson matrices, we want to verify that this is 0 for $s = 1, \dots, \lceil \frac{n-1}{2} \rceil$.

Compression

The *m-compression* of a sequence $A = [a_0, \ldots, a_{n-1}]$ of length $n = dm$ is the sequence of length d

$$A^{(d)} \coloneqq \left[a_0^{(d)}, \ldots, a_{d-1}^{(d)}\right] \qquad \text{where } a_j^{(d)} \coloneqq \sum_{k=0}^{m-1} a_{j+kd}.$$

Example

The sequence $A = [a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9]$ has the 5-compression

$$\left[a_0 + a_2 + a_4 + a_6 + a_8, \quad a_1 + a_3 + a_5 + a_7 + a_9\right]$$

and the 2-compression

$$\left[a_0 + a_5, \quad a_1 + a_6, \quad a_2 + a_7, \quad a_3 + a_8, \quad a_4 + a_9\right].$$

Compression

The *m-compression* of a sequence $A = [a_0, \ldots, a_{n-1}]$ of length $n = dm$ is the sequence of length d

$$A^{(d)} \coloneqq \left[a_0^{(d)}, \ldots, a_{d-1}^{(d)}\right] \qquad \text{where } a_j^{(d)} \coloneqq \sum_{k=0}^{m-1} a_{j+kd}.$$

Example

The sequence $A = [a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9]$ has the 5-compression

$$\left[a_0 + a_2 + a_4 + a_6 + a_8, \quad a_1 + a_3 + a_5 + a_7 + a_9\right]$$

and the 2-compression

$$\left[a_0 + a_5, \quad a_1 + a_6, \quad a_2 + a_7, \quad a_3 + a_8, \quad a_4 + a_9\right].$$

Useful Properties of Compressed Sequences

Lemma 1

Let A be a ± 1-sequence of length $n = dm$.
The entries of the m-compression of A have absolute value at most m and have the same parity as m.

Lemma 2

The compression of a symmetric sequence is also symmetric.

Technique 1: Sum-of-squares Decomposition

Full compression

Let A, B, C, D be Williamson sequences with n-compressions A', B', C', D'. A theorem of Đoković–Kotsireas[6] says that

$$\mathrm{PAF}_{A'}(0) + \mathrm{PAF}_{B'}(0) + \mathrm{PAF}_{C'}(0) + \mathrm{PAF}_{D'}(0) = 4n$$

which is just

$$\mathrm{rowsum}(A)^2 + \mathrm{rowsum}(B)^2 + \mathrm{rowsum}(C)^2 + \mathrm{rowsum}(D)^2 = 4n.$$

Also, each rowsum has the same parity as n by Lemma 1.

[6] Compression of periodic complementary sequences and applications. *Designs, Codes and Cryptography*.

Technique 1: Sum-of-squares Decomposition

Why is this useful?

CAS functions exist which can determine all possible solutions of

$$w^2 + x^2 + y^2 + z^2 = 4n \qquad w, x, y, z \equiv n \pmod 2.$$

This tells us all possibilities for the rowsums of A, B, C, D.
We can then use binary adders to encode the constraints

$$\text{rowsum}(A) = w \qquad \text{rowsum}(B) = x$$
$$\text{rowsum}(C) = y \qquad \text{rowsum}(D) = z.$$

Technique 1: Sum-of-squares Decomposition

Example

When $n = 35$, there are exactly three ways to write $4n$ as a sum of four positive odd squares in ascending order:

$$1^2 + 3^2 + 3^2 + 11^2 = 4 \cdot 35$$
$$1^2 + 3^2 + 7^2 + 9^2 = 4 \cdot 35$$
$$3^2 + 5^2 + 5^2 + 9^2 = 4 \cdot 35$$

Williamson Equivalence Operations

1. Ordering

The Williamson matrices A, B, C, D can be re-ordered with impunity.
Given this, we may enforce the constraint

$$|\text{rowsum}(A)| \leqslant |\text{rowsum}(B)| \leqslant |\text{rowsum}(C)| \leqslant |\text{rowsum}(D)|.$$

Williamson Equivalence Operations

2. Negation

The entries in any Williamson matrix A, B, C, D may be negated without affecting the Williamson conditions.
Given this, we may enforce the constraint

$$0 \leqslant \text{rowsum}(X) \qquad \text{for } X = A, B, C, D.$$

Williamson Equivalence Operations

2. Negation

The entries in any Williamson matrix A, B, C, D may be negated without affecting the Williamson conditions.

Given this, we may enforce the constraint

$$0 \leqslant \operatorname{rowsum}(X) \qquad \text{for } X = A, B, C, D.$$

Alternatively, when n is odd, we can use

$$\operatorname{rowsum}(X) \equiv n \pmod 4 \qquad \text{for } X = A, B, C, D.$$

In this case, Williamson showed that $a_i b_i c_i d_i = -1$ for all $1 \leqslant i \leqslant n-1$.

Technique 2: Divide-and-conquer

For efficiency reasons, we want to partition the search space into subspaces. An effective way to do this is to have each subspace contain one possibility for the compressions of A, B, C, D.

The generator script uses Lemmas 1 and 2 to determine all possible compressions, and the ĐK theorem to remove possibilities whose uncompressions are necessarily invalid (for example, because their *power spectral density* is too large).

Power Spectral Density

The *power spectral density* of a sequence A is

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

where DFT_A is the *discrete Fourier transform* of A.

Example

The power spectral density of $A = [1, 1, -1, -1, 1]$ is given by:

$$\begin{aligned}
\mathrm{PSD}_A(0) &= 1^2 &&= 1 \\
\mathrm{PSD}_A(1) &\approx 3.236^2 &&= 10.472 \\
\mathrm{PSD}_A(2) &\approx (-1.236)^2 &&= 1.528 \\
\mathrm{PSD}_A(3) &\approx (-1.236)^2 &&= 1.528 \\
\mathrm{PSD}_A(4) &\approx 3.236^2 &&= 10.472
\end{aligned}$$

Đoković–Kotsireas Theorem

Let A, B, C, D be Williamson sequences. For all $s \in \mathbb{Z}$

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

and these still hold if A, B, C, D are replaced with their compressions.

Corollary

If $\mathrm{PSD}_X(s) > 4n$ for some s then X (or any sequence which compresses to X) cannot be a Williamson sequence.

Technique 2: Divide-and-conquer

For $n = 35$ with 7-compression, the following is one of 41 compressions which satisfy the ĐK condition:

$$
\begin{aligned}
A^{(5)} &= [\ \ 5, \ \ 1, -3, -3, \ \ 1\,] \\
B^{(5)} &= [-3, \ \ 3, -3, -3, \ \ 3\,] \\
C^{(5)} &= [-3, \ \ 1, -1, -1, \ \ 1\,] \\
D^{(5)} &= [\ \ 1, -3, -3, -3, -3\,]
\end{aligned}
$$

Technique 2: Divide-and-conquer

If n has more than one nontrivial factor it is possible to perform compression by both factors. This increases the number of subspaces, but decreases the size of each subspace.

Example

Using 5 and 7-compression on $n = 35$ lead to the following number of subspaces for each decomposition type:

Instance type	# subspaces
$1^2 + 3^2 + 3^2 + 11^2$	6960
$1^2 + 3^2 + 7^2 + 9^2$	8424
$3^2 + 5^2 + 5^2 + 9^2$	6290

Technique 3: UNSAT Core

When an instance is found to be unsatisfiable, some SAT solvers can generate an *UNSAT core* containing which of those variables lead to the UNSAT result. We can prune instances which set the same variables to the same values.

Example

The $n = 35$ instances contained 3376 variables but only 168 were set differently between instances (those which encode the rowsum and compression values).

Experimental Results

Timings on SHARCNET for Williamson orders $25 \leqslant n \leqslant 35$ are below. The number of SAT calls which successfully returned a result is in parenthesis. A hyphen denotes a timeout after 24h.

Order	Base	Sum-of-squares	Divide-and-conquer	UNSAT Core
25	317s (1)	1702s (4)	408s (179)	408s (179)
26	865s (1)	3818s (3)	61s (3136)	34s (1592)
27	5340s (1)	8593s (3)	1518s (14994)	1439s (689)
28	7674s (1)	2104s (2)	234s (13360)	158s (439)
29	-	21304s (1)	N/A	N/A
30	1684s (1)	36804s (1)	139s (370)	139s (370)
31	-	83010s (1)	N/A	N/A
32	-	-	96011s (13824)	95891s (348)
33	-	-	693s (8724)	683s (7603)
34	-	-	854s (732)	854s (732)
35	-	-	31816s (21674)	31792s (19356)

Future Work

1. Extend Hadamard results

- Search larger orders and find new inequivalent Hadamard matrices.
- Explore different construction types, such as Hadamard matrices with (one or two) circulant cores. These are defined with a similar number of variables as the Williamson construction and the second type is conjectured to exist for all orders $4n$ (Kotsireas et al.[7]).
- Extend our system to find *all* inequivalent matrices of a given order. Currently, the number of inequivalent Williamson matrices is known only for odd $n < 60$.

[7] Hadamard ideals and Hadamard matrices with two circulant cores. *European Journal of Combinatorics*.

Future Work

2. Support other conjectures

- Search for other combinatorial objects which can be defined using the autocorrelation function; Kotsireas lists at least 11 different types[8]:

number/type of sequences	defining property	name
2 binary	aper. autoc. 0	Golay sequences
2 binary	per. autoc. 0	Hadamard matrices
2 binary	per. autoc. 2	D-optimal matrices
2 binary	per. autoc. -2	Hadamard matrices
2 ternary	aper. autoc. 0	TCP
2 ternary	per. autoc. 0	Weighing matrices
3 binary	aper. autoc. const.	Normal sequences
4 binary	aper. autoc. 0	Base sequences
4 binary	aper. autoc. 0	Turyn type sequences
4 ternary	aper. autoc. 0	T-sequences
$2 \dots 12$ binary	per. autoc. zero	PCS

[8] Algorithms and Metaheuristics for Combinatorial Matrices. *Handbook of Combinatorial Optimization.*

Future Work

2. Support other conjectures

- Some use the *aperiodic* autocorrelation function, defined by

$$\mathrm{AAF}_A(s) \coloneqq \sum_{k=0}^{n-s-1} a_k \bar{a}_{k+s} \qquad \text{for } s = 0, \ldots, n-1.$$

- For example, complex Golay sequences are two sequences $A, B \in \{\pm 1, \pm i\}^n$ which satisfy

$$\mathrm{AAF}_A(s) + \mathrm{AAF}_B(s) = 0 \qquad \text{for } s = 1, \ldots, n-1.$$

Future Work

3. Extend SAT solver programmatically

- Make custom modifications to the SAT solvers used to run domain-specific code tailored to each conjecture.
- Ganesh et al.[9] introduced a special API for programmatically influencing the behaviour of a SAT solver by generating problem-specific learned clauses as the search progresses. This approach was shown to be up to 100 times more efficient in the context of RNA folding problems.
- Benefits include increased expressiveness, efficiency, and better leverage of CAS functionality.

[9] Lynx: A programmatic SAT solver for the RNA-folding problem. *Theory and Applications of Satisfiability Testing–SAT 2012.*

Future Work

3. Extend SAT solver programmatically

Example 1

The Williamson instances required encoding constraints like $\text{rowsum}(A) = 1$ from which we can determine how many of $a_0, \ldots, a_{n-1}$ must be true.

We can have the SAT solver keep track of this count and backjump whenever a partial assignment is inconsistent with the constraint.

Williamson Equivalence Operations

3. Permuting entries

We can reorder the entries of the generating matrices with the rule $a_i \mapsto a_{ki \bmod n}$ where k is any number coprime with n, and similarly for b_i, c_i, d_i (the *same* reordering must be applied to each).

Future Work

3. Extend SAT solver programmatically

Example 2

Calling CAS functions from inside the SAT solver will allow theory-specific lemmas to be learned, such as those detecting symmetries and pruning isomorphic solutions.

The Williamson equivalence operation of permuting entries would be difficult to encode using propositional formulae but can be easily computed by a CAS.

Conclusions

We have...

- Presented the advantages of utilizing the power of SAT solvers in combination with domain specific knowledge and algorithms provided by computer algebra systems.
- Outlined how such a strategy can be useful for studying a wide variety of conjectures in combinatorics, and potential ways to improve such a strategy.
- Performed a requested verification of a nonexistence result using a new algorithm and techniques which generalize to other conjectures.
- Submitted new matrices to MAGMA's Hadamard database, including some generated by Williamson matrices of even order.

References

[1] Erika Ábrahám.
Building bridges between symbolic computation and satisfiability checking.
In *Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation*, pages 1–6. ACM, 2015.

[2] Vijay Ganesh, Charles W O'Donnell, Mate Soos, Srinivas Devadas, Martin C Rinard, and Armando Solar-Lezama.
Lynx: A programmatic SAT solver for the RNA-folding problem.
In *Theory and Applications of Satisfiability Testing–SAT 2012*, pages 143–156. Springer, 2012.

[3] Ilias S Kotsireas.
Algorithms and metaheuristics for combinatorial matrices.
In *Handbook of Combinatorial Optimization*, pages 283–309. Springer, 2013.

[4] Ilias S. Kotsireas, Christos Koukouvinos, and Jennifer Seberry.
Hadamard ideals and Hadamard matrices with two circulant cores.
European Journal of Combinatorics, 27(5):658–668, 2006.

[5] Dragomir Ž Đoković.
Williamson matrices of order $4n$ for $n = 33, 35, 39$.
Discrete mathematics, 115(1):267–271, 1993.

[6] Dragomir Ž Đoković and Ilias S Kotsireas.
Compression of periodic complementary sequences and applications.
Designs, Codes and Cryptography, 74(2):365–377, 2015.

[7] Edward Zulkoski, Vijay Ganesh, and Krzysztof Czarnecki.
MathCheck: A math assistant via a combination of computer algebra systems and SAT solvers.
In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25*, volume 9195 of *Lecture Notes in Computer Science*, pages 607–622. Springer International Publishing, 2015.