Enumeration of Complex Golay Pairs via Programmatic SAT

Curtis Bright
University of Waterloo

Ilias Kotsireas
Wilfrid Laurier University

Albert Heinle
University of Waterloo

Vijay Ganesh
University of Waterloo

ABSTRACT

We provide a complete enumeration of all complex Golay pairs of length up to 25, verifying that complex Golay pairs do not exist in lengths 23 and 25 but do exist in length 24. This independently verifies work done by F. Fiedler in 2013 [11] that confirms the 2002 conjecture of Craigen, Holzmann, and Kharaghani [8] that complex Golay pairs of length 23 don't exist. Our enumeration method relies on the recently proposed SAT+CAS paradigm of combining computer algebra systems with SAT solvers to take advantage of the advances made in the fields of symbolic computation and satisfiability checking. The enumeration proceeds in two stages: First, we use a fine-tuned computer program and functionality from computer algebra systems to construct a list containing all sequences which could appear as the first sequence in a complex Golay pair (up to equivalence). Second, we use a programmatic SAT solver to construct all sequences (if any) that pair off with the sequences constructed in the first stage to form a complex Golay pair.

KEYWORDS

Complex Golay pairs; Boolean satisfiability; SAT solvers; Exhaustive search; Autocorrelation

ACM Reference Format:
Curtis Bright, Ilias Kotsireas, Albert Heinle, and Vijay Ganesh. 2018. Enumeration of Complex Golay Pairs via Programmatic SAT. In *ISSAC '18: 2018 ACM International Symposium on Symbolic and Algebraic Computation, July 16–19, 2018, New York, NY, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3208976.3209006

1 INTRODUCTION

The sequences which are now referred to *Golay sequences* or *Golay pairs* were first introduced by Marcel Golay in his groundbreaking 1949 paper [18] on multislit spectrometry. He later formally defined them in a 1961 paper [17] where he referred to them as *complementary series*. Since then, Golay pairs and their generalizations have been widely studied for both their elegant theoretical properties and a surprising number of practical applications. For example, they have been applied to radar pulse compression [20], Wi-Fi networks [25], train wheel detection systems [10], optical time domain reflectometry [27], and medical ultrasounds [28]. Golay pairs consist of two sequences and the property that makes them special is, roughly speaking, the fact that one sequence's "correlation" with itself is the inverse of the other sequence's "correlation" with itself; see Definition 2.2 in Section 2 for the formal definition.

Although Golay defined his complementary series over an alphabet of $\{\pm 1\}$, later authors have generalized the alphabet to include nonreal roots of unity such as the fourth root of unity $i = \sqrt{-1}$. In this paper, we focus on the case where the alphabet is $\{\pm 1, \pm i\}$. In this case the resulting sequence pairs are sometimes referred to as *4-phase* or *quaternary* Golay pairs though we will simply refer to them as *complex* Golay pairs. If a complex Golay pair of length n exists then we say that n is a *complex Golay number*.

Complex Golay pairs have been extensively studied by many authors. They were originally introduced in 1994 by Craigen in order to expand the orders of Hadamard matrices attainable via (ordinary) Golay pairs [7]. In 1994, Holzmann and Kharaghani enumerated all complex Golay pairs up to length 13 [19]. In 2002, Craigen, Holzmann, and Kharaghani enumerated all complex Golay pairs to 19, reported that 21 was not a complex Golay number, and conjectured that 23 was not a complex Golay number [8].

In 2006, Fiedler, Jedwab, and Parker provided a construction which explained the existence of all known complex Golay pairs whose lengths were a power of 2 [12, 13], including complex Golay pairs of length 16 discovered by Li and Chu [23] to not fit into a construction given by Davis and Jedwab [9]. In 2010, Gibson and Jedwab provided a construction which explained the existence of all complex Golay pairs up to length 26 and gave a table that listed the total number of complex Golay pairs up to length 26 [16]. This table was produced by the mathematician Frank Fiedler, who described his enumeration method in a 2013 paper [11] where he also reported that 27 and 28 are not complex Golay numbers.

In this paper we give an enumeration method which can be used to verify the table produced by Fiedler that appears in Gibson and Jedwab's paper; this table contains counts for the total number of complex Golay pairs and the total number of sequences which appear as a member of a complex Golay pair. We implemented our method and obtained counts up to length 25 after about a day of computing on a cluster with 25 cores. The counts we obtain match those in Fiedler's table in each case, increasing the confidence that the enumeration was performed without error. In addition, we also provide counts for the total number of complex Golay pairs up to well-known equivalence operations [19] and explicitly make available the sequences online [6]. To our knowledge, this is the first time that explicit complex Golay pairs (and their counts up to equivalence) have been published for lengths larger than 19. Lastly, we publicly release our code for enumerating complex Golay pairs so that others may verify and reproduce our work; we were not

ISSAC '18, July 16–19, 2018, New York, NY, USA

ACM ISBN 978-1-4503-5550-6/18/07…$15.00
https://doi.org/10.1145/3208976.3209006

able to find any other code for enumerating complex Golay pairs which was publicly available.

Our result is of interest not only because of the verification we provide but also because of the method we use to perform the verification. The method proceeds in two stages. In the first stage, a fine-tuned computer program performs an exhaustive search among all sequences which could possibly appear as the first sequence in a complex Golay pair of a given length (up to an equivalence defined in Section 2). Several filtering theorems which we describe in Section 2 allow us to discard almost all sequences from consideration. To apply these filtering theorems we use functionality from the computer algebra system Maple [26] and the mathematical library FFTW [14]. After this filtering is completed we have a list of sequences of a manageable size such that the first sequence of every complex Golay pair of a given length (up to equivalence) appears in the list.

In the second stage, we use the programmatic SAT solver MapleSAT [24] to determine which sequences from the first stage (if any) can be paired up with another sequence to form a complex Golay pair. A SAT instance is constructed from each sequence found in the first stage such that the SAT instance is satisfiable if and only if the sequence is part of a complex Golay pair. Furthermore, in the case that the instance is satisfiable a satisfying assignment determines a sequence which forms the second half of a complex Golay pair.

This method combines both computer algebra and SAT solving and is of interest in its own right because it links the two previously separated fields of symbolic computation and satisfiability checking. Recently there has been interest in combining methods from both fields to solve computational problems as demonstrated by the SC^2 project [1, 2]. Our work fits into this paradigm and to our knowledge is the first application of a SAT solver to search for complex Golay pairs, though previous work exists which uses a SAT solver to search for other types of complementary sequences [3–5, 31].

2 BACKGROUND ON COMPLEX GOLAY PAIRS

In this section we present the background necessary to describe our method for enumerating complex Golay pairs. First, we require some preliminary definitions to define what complex Golay pairs are. Let $\overline{x}$ denote the complex conjugate of x (this is just the multiplicative inverse of x when x is ± 1 or $\pm i$).

Definition 2.1 (cf. [22]). The *nonperiodic autocorrelation function* of a sequence $A = [a_0, \ldots, a_{n-1}] \in \mathbb{C}^n$ of length $n \in \mathbb{N}$ is

$$N_A(s) := \sum_{k=0}^{n-s-1} a_k \overline{a_{k+s}}, \qquad s = 0, \ldots, n-1.$$

Definition 2.2. A pair of sequences (A, B) with A and B in $\{\pm 1, \pm i\}^n$ are called a *complex Golay pair* if the sum of their nonperiodic autocorrelations is a constant zero for $s \neq 0$, i.e.,

$$N_A(s) + N_B(s) = 0 \qquad \text{for} \qquad s = 1, \ldots, n-1.$$

Note that if A and B are in $\{\pm 1, \pm i\}^n$ then $N_A(0) + N_B(0) = 2n$ by the definition of the complex nonperiodic autocorrelation function and the fact that $x\overline{x} = 1$ if x is ± 1 or $\pm i$, explaining why $s \neq 0$ in Definition 2.2.

Example 2.3. $([1, 1, -1], [1, i, 1])$ is a complex Golay pair.

2.1 Equivalence operations

There are certain invertible operations which preserve the property of being a complex Golay pair when applied to a sequence pair (A, B). These are summarized in the following proposition.

Proposition 2.4 (cf. [8]). *Let* $([a_0, \ldots, a_{n-1}], [b_0, \ldots, b_{n-1}])$ *be a complex Golay pair. The following are then also complex Golay pairs:*

E1. *(Reversal)* $([a_{n-1}, \ldots, a_0], [b_{n-1}, \ldots, b_0])$.
E2. *(Conjugate Reverse A)* $([\overline{a_{n-1}}, \ldots, \overline{a_0}], [b_0, \ldots, b_{n-1}])$.
E3. *(Swap)* $([b_0, \ldots, b_{n-1}], [a_0, \ldots, a_{n-1}])$.
E4. *(Scale A)* $([ia_0, \ldots, ia_{n-1}], [b_0, \ldots, b_{n-1}])$.
E5. *(Positional Scaling)* $(i \star A, i \star B)$ *where* $c \star (x_0, \ldots, x_{n-1}) := (x_0, cx_1, c^2x_2, \ldots, c^{n-1}x_{n-1})$.

Definition 2.5. We call two complex Golay pairs (A, B) and (A', B') *equivalent* if (A', B') can be obtained from (A, B) using the transformations described in Proposition 2.4.

2.2 Useful properties and lemmas

In this subsection we prove some useful properties that complex Golay pairs satisfy and which will be exploited by our method for enumerating complex Golay pairs. The first lemma provides a fundamental relationship that all complex Golay pairs must satisfy. To conveniently state it we use the following definition.

Definition 2.6 (cf. [8]). The *Hall polynomial* of the sequence $A := [a_0, \ldots, a_{n-1}]$ is defined to be $h_A(z) := a_0 + a_1 z + \cdots + a_{n-1}z^{n-1} \in \mathbb{C}[z]$.

Lemma 2.7 (cf. [29]). *Let* (A, B) *be a complex Golay pair. For every* $z \in \mathbb{C}$ *with* $|z| = 1$*, we have*

$$|h_A(z)|^2 + |h_B(z)|^2 = 2n.$$

Proof. Since $|z| = 1$ we can write $z = e^{i\theta}$ for some $0 \leq \theta < 2\pi$. Similar to the fact pointed out in [21], using Euler's identity one can derive the following expansion:

$$|h_A(z)|^2 = N_A(0) + 2\sum_{j=1}^{n-1}\Big(\mathrm{Re}(N_A(j))\cos(\theta j) + \mathrm{Im}(N_A(j))\sin(\theta j)\Big).$$

Since A and B form a complex Golay pair, by definition one has that $\mathrm{Re}(N_A(j) + N_B(j)) = 0$ and $\mathrm{Im}(N_A(j) + N_B(j)) = 0$ and then

$$|h_A(z)|^2 + |h_B(z)|^2 = N_A(0) + N_B(0) = 2n. \qquad \square$$

This lemma is highly useful as a condition for filtering sequences which could not possibly be part of a complex Golay pair, as explained in the following corollary.

Corollary 2.8. *Let* $A \in \mathbb{C}^n$*,* $z \in \mathbb{C}$ *with* $|z| = 1$*, and* $|h_A(z)|^2 > 2n$*. Then* A *is not a member of a complex Golay pair.*

Proof. Suppose the sequence A was a member of a complex Golay pair whose other member was the sequence B. Since $|h_B(z)|^2 \geq 0$, we must have $|h_A(z)|^2 + |h_B(z)|^2 > 2n$, in contradiction to Lemma 2.7. $\square$

In [11], Fiedler derives the following extension of Lemma 2.7. Let A_{even} be identical to A with the entries of odd index replaced by zeros and let A_{odd} be identical to A with the entries of even index replaced by zeros.

Lemma 2.9 (cf. [11]). *Let (A, B) be a complex Golay pair. For every $z \in \mathbb{C}$ with $|z| = 1$, we have*

$$|h_{A_{\text{even}}}(z)|^2 + |h_{A_{\text{odd}}}(z)|^2 + |h_{B_{\text{even}}}(z)|^2 + |h_{B_{\text{odd}}}(z)|^2 = 2n.$$

Proof. The proof proceeds as in the proof of Lemma 2.7, except that one instead obtains that $|h_{A_{\text{even}}}(z)|^2 + |h_{A_{\text{odd}}}(z)|^2$ is equal to

$$N_A(0) + 2 \sum_{\substack{j=1 \\ j \text{ even}}}^{n-1} \Big(\text{Re}(N_A(j)) \cos(\theta j) + \text{Im}(N_A(j)) \sin(\theta j) \Big). \quad \square$$

Corollary 2.10. *Let $A \in \mathbb{C}^n$, $z \in \mathbb{C}$ with $|z| = 1$, and $|h_{A'}(z)|^2 > 2n$ where A' is either A_{even} or A_{odd}. Then A is not a member of a complex Golay pair.*

Proof. If either $|h_{A_{\text{even}}}(z)|^2 > 2n$ or $|h_{A_{\text{odd}}}(z)|^2 > 2n$ then the identity in Lemma 2.9 cannot hold. $\square$

The next lemma is useful because it allows us to write $2n$ as the sum of four integer squares. It is stated in [19] using a different notation; we use the notation $\text{resum}(A)$ and $\text{imsum}(A)$ to represent the real and imaginary parts of the sum of the entries of A. For example, if $A := [1, i, -i, i]$ then $\text{resum}(A) = \text{imsum}(A) = 1$.

Lemma 2.11 (cf. [19]). *Let (A, B) be a complex Golay sequence pair. Then*

$$\text{resum}(A)^2 + \text{imsum}(A)^2 + \text{resum}(B)^2 + \text{imsum}(B)^2 = 2n.$$

Proof. Using Lemma 2.7 with $z = 1$ we have

$$|\text{resum}(A) + \text{imsum}(A)i|^2 + |\text{resum}(B) + \text{imsum}(B)i|^2 = 2n.$$

Since $|\text{resum}(X) + \text{imsum}(X)i|^2 = \text{resum}(X)^2 + \text{imsum}(X)^2$ the result follows. $\square$

The next lemma provides some normalization conditions which can be used when searching for complex Golay pairs up to equivalence. Since all complex Golay pairs (A', B') which are equivalent to a complex Golay pair (A, B) can easily be generated from (A, B), it suffices to search for complex Golay pairs up to equivalence.

Lemma 2.12 (cf. [11]). *Let (A', B') be a complex Golay pair. Then (A', B') is equivalent to a complex Golay pair (A, B) with $a_0 = a_1 = b_0 = 1$ and $a_2 \in \{\pm 1, i\}$.*

Proof. We will transform a given complex Golay sequence pair (A', B') into an equivalent normalized one using the equivalence operations of Proposition 2.4. To start with, let $A := A'$ and $B := B'$.

First, we ensure that $a_0 = 1$. To do this, we apply operation E4 (scale A) enough times until $a_0 = 1$.

Second, we ensure that $a_1 = 1$. To do this, we apply operation E5 (positional scaling) enough times until $a_1 = 1$; note that E5 does not change a_0.

Third, we ensure that $a_2 \neq -i$. If it is, we apply operation E1 (reversal) and E2 (conjugate reverse A) which has the effect of keeping $a_0 = a_1 = 1$ and setting $a_2 = i$.

Last, we ensure that $b_0 = 1$. To do this, we apply operation E3 (swap) and then operation E4 (scale A) enough times so that $a_0 = 1$ and then operation E3 (swap) again. This has the effect of not changing A but setting $b_0 = 1$. $\square$

2.3 Sum-of-squares decomposition types

A consequence of Lemma 2.11 is that every complex Golay pair generates a decomposition of $2n$ into a sum of four integer squares. In fact, it typically generates several decompositions of $2n$ into a sum of four squares. Recall that $i \star A$ denotes positional scaling by i (operation E5) on the sequence A. If (A, B) is a complex Golay pair then applying operation E5 to this pair k times shows that $(i^k \star A, i^k \star B)$ is also a complex Golay pair. By using Lemma 2.11 on these complex Golay pairs one obtains the fact that $2n$ can be decomposed as the sum of four integer squares as

$$\text{resum}(i^k \star A)^2 + \text{imsum}(i^k \star A)^2 + \text{resum}(i^k \star B)^2 + \text{imsum}(i^k \star B)^2.$$

For $k > 3$ this produces no new decompositions but in general for $k = 0$, 1, 2, and 3 this produces four distinct decompositions of $2n$ into a sum of four squares.

With the help of a computer algebra system (CAS) one can enumerate every possible way that $2n$ may be written as a sum of four integer squares. For example, when $n = 23$ one has $0^2 + 1^2 + 3^2 + 6^2 = 2 \cdot 23$ and $1^2 + 2^2 + 4^2 + 5^2 = 2 \cdot 23$ as well as all permutations of the squares and negations of the integers being squared. During the first stage of our enumeration method only the first sequence of a complex Golay pair is known, so at that stage we cannot compute its whole sums-of-squares decomposition. However, it is still possible to filter some sequences from consideration based on analyzing the two known terms in the sums-of-squares decomposition.

For example, say that A is the first sequence in a potential complex Golay pair of length 23 with $\text{resum}(A) = 0$ and $\text{imsum}(A) = 5$. We can immediately discard A from consideration because there is no way to chose the resum and imsum of B to complete the sums-of-squares decomposition of $2n$, i.e., there are no integer solutions (x, y) of $0^2 + 5^2 + x^2 + y^2 = 2n$.

3 ENUMERATION METHOD

In this section we describe in detail the method we used to perform a complete enumeration of all complex Golay pairs up to length 25. Given a length n our goal is to find all $\{\pm 1, \pm i\}$ sequences A and B of length n such that (A, B) is a complex Golay pair.

3.1 Preprocessing: Enumerate possibilities for A_{even} and A_{odd}

The first step of our method uses Fiedler's trick of considering the entries of A of even index separately from the entries of A of odd index. There are approximately $n/2$ nonzero entries in each of A_{even} and A_{odd} and there are four possible values for each nonzero entry. Therefore there are approximately $2 \cdot 4^{n/2} = 2^{n+1}$ possible sequences to check in this step. Additionally, by Lemma 2.12 we may assume the first nonzero entry of both A_{even} and A_{odd} is 1 and that the second nonzero entry of A_{even} is not $-i$, decreasing the number of sequences to check in this step by more than a factor of 4. It is quite feasible to perform a brute-force search through all such sequences when $n \approx 30$.

We apply Corollary 2.10 to every possibility for A_{even} and A_{odd}. There are an infinite number of possible $z \in \mathbb{C}$ with $|z| = 1$, so we do not attempt to apply Corollary 2.10 using all such z. Instead we try a sufficiently large number of z so that in the majority of cases for which a z exists with $|h_{A'}(z)|^2 > 2n$ (where A' is either A_{even}

or A_{odd}) we in fact find such a z. In our implementation we chose to take z to be $e^{2\pi ij/N}$ where $N \coloneqq 2^{14}$ and $j = 0, \ldots, N-1$.

At the conclusion of this step we will have two lists: one list L_{even} of the A_{even} which were not discarded and one list L_{odd} of the A_{odd} which were not discarded.

3.2 Stage 1: Enumerate possibilities for A

We now enumerate all possibilities for A by joining the possibilities for A_{even} with the possibilities for A_{odd}. For each $A_1 \in L_{\text{odd}}$ and $A_2 \in L_{\text{even}}$ we form the sequence A by letting the kth entry of A be either the kth entry of A_1 or A_2 (whichever is nonzero). Thus the entries of A are either ± 1 or $\pm i$ and therefore A is a valid candidate for the first sequence of a complex Golay pair of length n.

At this stage we now use the filtering result of Corollary 2.8 and the sums-of-squares decomposition result of Lemma 2.11 to perform more extensive filtering on the sequences A which we formed above. In detail, our next filtering check proceeds as follows: Let $R_k \coloneqq \text{resum}(i^k \star A)$ and $I_k \coloneqq \text{imsum}(i^k \star A)$. By using a Diophantine equation solver we check if the Diophantine equations

$$R_k^2 + I_k^2 + x^2 + y^2 = 2n$$

are solvable in integers (x, y) for $k = 0, 1, 2, 3$. As explained in Section 2.3, if any of these equations have no solutions then A cannot be a member of a complex Golay pair and can be ignored. Secondly, we use Corollary 2.8 with z chosen to be $e^{2\pi ij/N}$ for $j = 0, \ldots, N-1$ where $N \coloneqq 2^7$ (we use a smaller value of N than in the preprocessing step because in this case there are a larger number of sequences which we need to apply the filtering condition on).

If A passes both filtering conditions then we add it to a list L_{A} and try the next value of A until no more possibilities remain. At the conclusion of this stage we will have a list of sequences L_{A} which could potentially be a member of a complex Golay pair. By construction, the first member of all complex Golay pairs (up to the equivalence described in Lemma 2.12) of length n will be in L_{A}.

3.3 Stage 2: Construct the second sequence B from A

In the second stage we take as input the list L_{A} generated in the first stage, i.e., a list of the sequences A that were not filtered by any of the filtering theorems we applied. For each $A \in L_{\text{A}}$ we attempt to construct a second sequence B such that (A, B) is a complex Golay pair. We do this by generating a SAT instance which encodes the property of (A, B) being a complex Golay pair where the entries of A are known and the entries of B are unknown and encoded using Boolean variables. Because there are four possible values for each entry of B we use two Boolean variables to encode each entry. Although the exact encoding used is arbitrary, we fixed the following encoding in our implementation, where the variables v_{2k} and v_{2k+1} represent b_k, the kth entry of B:

v_{2k}	v_{2k+1}	b_k
F	F	1
F	T	-1
T	F	i
T	T	$-i$

To encode the property that (A, B) is a complex Golay pair in out SAT instance we add the conditions which define (A, B) to be a complex Golay pair, i.e.,

$$N_A(s) + N_B(s) = 0 \qquad \text{for} \qquad s = 1, \ldots, n-1.$$

These equations could be encoded using clauses in conjunctive normal form (for example by constructing logical circuits to perform complex multiplication and addition and then converting those circuits into CNF clauses). However, we found that a much more efficient and convenient method was to use a *programmatic* SAT solver.

The concept of a programmatic SAT solver was first introduced in [15] where a programmatic SAT solver was shown to be more efficient than a standard SAT solver when solving instances derived from RNA folding problems. More recently, a programmatic SAT solver was also shown to be useful when searching for Williamson matrices [5]. Generally, programmatic SAT solvers perform well when there is domain-specific knowledge about the problem being solved that cannot easily be encoded into SAT instances directly but can be used to learn facts about potential solutions which can help guide the solver in its search.

Concretely, a programmatic SAT solver is compiled with a piece of code which encodes a property that a solution of the SAT instance must satisfy. Periodically the SAT solver will run this code while performing its search and if the current partial assignment violates a property that is expressed in the provided code then a conflict clause is generated encoding this fact. The conflict clause is added to the SAT solver's database of learned clauses where it is used to increase the efficiency of the remainder of the search. The reason these clauses can be so useful is because they can encode facts which the SAT solver would have no way of learning otherwise, since the SAT solver has no knowledge of the domain of the problem.

Not only does this paradigm allow the SAT solver to perform its search more efficiently, it also allows instances to be much more expressive. Under this framework SAT instances do not have to consist solely of Boolean formulas in conjunctive normal form (the typical format of SAT instances) but can consist of clauses in conjunctive normal form combined with a piece of code that *programmatically* expresses clauses. This extra expressiveness is also a feature of SMT solvers, though SMT solvers typically require more overhead to use. Additionally, one can compile *instance-specific* programmatic SAT solvers which are tailored to perform searches for a specific class of problems.

For our purposes we use a programmatic SAT solver tailored to search for sequences B that when paired with a given sequence A form a complex Golay pair. Each instance will contain the $2n$ variables $v_0, \ldots, v_{2n-1}$ that encode the entries of B as previously specified. In detail, the code given to the SAT solver does the following:

(1) Compute and store the values $N_A(k)$ for $k = 1, \ldots, n-1$.
(2) Initialize s to $n-1$. This will be a variable which controls which autocorrelation condition we are currently examining.
(3) Examine the current partial assignment to v_0, v_1, v_{2n-2}, and v_{2n-1}. If all these values have been assigned then we can determine the values of b_0 and b_{n-1}. From these values we compute $N_B(s) = b_0\overline{b_{n-1}}$. If $N_A(s) + N_B(s) \neq 0$ then (A, B) cannot be a complex Golay pair (regardless of the values of $b_1, \ldots, b_{n-2}$) and therefore we learn a conflict clause

which says that b_0 and b_{n-1} cannot both be assigned to their current values. More explicitly, if v_k^{cur} represents the literal v_k when v_k is currently assigned to true and the literal $\neg v_k$ when v_k is currently assigned to false we learn the clause

$$\neg(v_0^{\text{cur}} \wedge v_1^{\text{cur}} \wedge v_{2n-2}^{\text{cur}} \wedge v_{2n-1}^{\text{cur}}).$$

(4) Decrement s by 1 and repeat the previous step, computing $N_B(s)$ if the all the b_k which appear in its definition have known values. If $N_A(s) + N_B(s) \neq 0$ then learn a clause preventing the values of b_k which appear in the definition of $N_B(s)$ from being assigned the way that they currently are. Continue to repeat this step until $s = 0$.
(5) If all values of B are assigned but no clauses have been learned then output the complex Golay pair (A, B). If an exhaustive search is desired, learn a clause which prevents the values of B from being assigned the way they currently are; otherwise learn nothing and return control to the SAT solver.

For each A in the list L_A from stage 1 we run a SAT solver with the above programmatic code; the list of all outputs (A, B) in step (5) shown above now form a complete list of complex Golay pairs of length n up to the equivalence given in Lemma 2.12. In fact, since Lemma 2.12 says that we can set $b_0 = 1$ we can assume that both v_0 and v_1 are always set to false. In other words, we can add the two clauses $\neg v_0$ and $\neg v_1$ into our SAT instance without omitting any complex Golay pairs up to equivalence.

3.4 Postprocessing: Enumerating all complex Golay pairs

At the conclusion of the second stage we have obtained a list of complex Golay pairs of length n such that every complex Golay pair of length n is equivalent to some pair in our list. However, because we have not accounted for all the equivalences in Section 2.1 some pairs in our list may be equivalent to each other. In some sense such pairs should not actually be considered distinct, so to count how many distinct complex Golay pairs exist in length n we would like to find and remove pairs which are equivalent from the list. Additionally, to verify the counts given in [16] it is necessary to produce a list which contains *all* complex Golay pairs. We now describe an algorithm which does both, i.e., it produces a list of all complex Golay pairs as well as a list of all inequivalent complex Golay pairs.

In detail, our algorithm performs the following steps:

(1) Initialize Ω_{all} to be the set of complex Golay pairs generated in stage 2. This variable will be a set that will be populated with and eventually contain all complex Golay pairs of length n.
(2) Initialize Ω_{inequiv} to be the empty set. This variable will be a set that will be populated with and eventually contain all inequivalent complex Golay pairs of length n.
(3) For each (A, B) in Ω_{all}:
  (a) If (A, B) is already in Ω_{inequiv} then skip this (A, B) and proceed to the next pair (A, B) in Ω_{all}.
  (b) Initialize Γ to be the set containing (A, B). This variable will be a set that will be populated with and eventually contain all complex Golay pairs equivalent to (A, B).
  (c) For every γ in Γ add $\text{E1}(\gamma), \ldots, \text{E5}(\gamma)$ to Γ. Continue to do this until every pair in Γ has been examined and no new pairs are added to Γ.
  (d) Add (A, B) to Ω_{inequiv} and add all pairs in Γ to Ω_{all}.

After running this algorithm listing the members of Ω_{all} gives a list of all complex Golay pairs of length n and listing the members of Ω_{inequiv} gives a list of all inequivalent complex Golay pairs of length n. At this point we can also construct the complete list of sequences which appear in any complex Golay pair of length n. To do this it suffices to add A and B to a new set Ω_{seqs} for each $(A, B) \in \Omega_{\text{all}}$.

3.5 Optimizations

Although the method described will correctly enumerate all complex Golay pairs of a given length n, for the benefit of potential implementors we mention a few optimizations which we found helpful.

In stage 1 we check if Diophantine equations of the form

$$R^2 + I^2 + x^2 + y^2 = 2n \qquad (*)$$

are solvable in integers (x, y) where R and I are given. CAS functions like `PowersRepresentations` in MATHEMATICA or `nsoks` in MAPLE [30] can determine all ways of writing $2n$ as a sum of four integer squares. From this information we construct a Boolean two dimensional array D such that $D_{|R|,|I|}$ is true if and only if $(*)$ has a solution, making the check for solvability a fast lookup. In fact, one need only construct the lookup table for R and I with $R + I \equiv n \pmod 2$ as the following lemma shows.

LEMMA 3.1. *Suppose R and I are the* resum *and* imsum *of a sequence $X \in \{\pm 1, \pm i\}^n$. Then $R + I \equiv n \pmod 2$.*

PROOF. Let $\#_c$ denote the number of entries in X with value c. Then

$$R + I = (\#_1 - \#_{-1}) + (\#_i - \#_{-i}) \equiv \#_1 + \#_{-1} + \#_i + \#_{-i} \pmod 2$$

since $-1 \equiv 1 \pmod 2$. The quantity on the right is n since there are n entries in X. □

In stage 1 we check if $|h_A(z)|^2 > 2n$ where $z = e^{2\pi ij/N}$ for $j = 0, \ldots, N-1$ with $N = 2^7$. However, we found that it was more efficient to not check the condition for each j in ascending order (i.e., for each z in ascending complex argument) but to first perform the check on points z with larger spacing between them. In our implementation we first assigned N to be 2^3 and performed the check for odd $j = 1, 3, \ldots, N-1$. Following this we doubled N and again performed the check for odd j, proceeding in this matter until all points z had been checked. (This ignores checking the condition when $z = i^k$ for some k but that is desirable since in those cases $|h_A(i^k)|^2 = \text{resum}(i^k \star A)^2 + \text{imsum}(i^k \star A)^2$ and the sums-of-squares condition is a strictly stronger filtering method.)

In the preprocessing step and stage 1 it is necessary to evaluate the Hall polynomial $h_{A'}$ or h_A at roots of unity $z = e^{2\pi ij/N}$ and determine its squared absolute value. The fastest way we found of doing this used the discrete Fourier transform. For example, let A' be the sequence A_{even}, A_{odd}, or A under consideration but padded

with trailing zeros so that A' is of length N. By definition of the discrete Fourier transform we have that

$$\mathrm{DFT}(A') = \left[h_{A'}\left(e^{2\pi ij/N}\right)\right]_{j=0}^{N-1}.$$

Thus, we determine the values of $|h_{A'}(z)|^2$ by taking the squared absolute values of the entries of $\mathrm{DFT}(A')$. If $|h_{A'}(z)|^2 > 2n$ for some z then by Corollary 2.8 or Corollary 2.10 we can discard A' from consideration. To guard against potential inaccuracies introduced by the algorithms used to compute the DFT we actually ensure that $|h_{A'}(z)|^2 > 2n + \epsilon$ for some tolerance ϵ which is small but larger than the accuracy that the DFT is computed to (e.g., $\epsilon = 10^{-3}$).

In the preprocessing step before setting $N := 2^{14}$ we first set $N := n$ and perform the rest of the step as given. The advantage of first performing the check with a smaller value of N is that the discrete Fourier transform of A' can be computed faster. Although the check with $N = n$ is a less effective filter, it often succeeds and whenever it does it allows us to save time by not performing the more costly longer DFT.

In stage 1 our application of Corollary 2.8 requires computing $|h_A(z)|^2$ where $z = e^{2\pi ij/N}$ for $j = 0, \ldots, N-1$. Noting that

$$h_A(z) = h_{A_{\mathrm{even}}}(z) + h_{A_{\mathrm{odd}}}(z)$$

one need only compute $h_{A_{\mathrm{even}}}(z)$ and $h_{A_{\mathrm{odd}}}(z)$ for each each A_{even} and A_{odd} generated in the preprocessing step and once those are known $h_A(z)$ can be found by a simple addition.

In stage 2 one can also include properties that complex Golay sequences must satisfy in the code compiled with the programmatic SAT solver. As an example of this, we state the following proposition which was new to the authors and does not appear to have been previously published.

Proposition 3.2. *Let (A, B) be a complex Golay pair. Then*

$$a_k a_{n-k-1} b_k b_{n-k-1} = \pm 1 \qquad \text{for} \qquad k = 0, \ldots, n-1.$$

To prove this, we use the following simple lemma.

Lemma 3.3. *Let $c_k \in \mathbb{Z}_4$ for $k = 0, \ldots, n-1$. Then*

$$\sum_{k=0}^{n-1} i^{c_k} = 0 \qquad \text{implies} \qquad \sum_{k=0}^{n-1} c_k \equiv 0 \pmod 2.$$

Proof. Let $\#_c$ denote the number of c_k with value c. Note that the sum on the left implies that $\#_0 = \#_2$ and $\#_1 = \#_3$ because the 1s must cancel with the -1s and the is must cancel with the $-i$s. Then $\sum_{k=0}^{n-1} c_k = \#_1 + 2\#_2 + 3\#_3 \equiv \#_1 + \#_3 \equiv 2\#_1 \equiv 0 \pmod 2$. □

We now prove Proposition 3.2.

Proof. Let $c_k, d_k \in \mathbb{Z}_4$ be such that $a_k = i^{c_k}$ and $b_k = i^{d_k}$. Using this notation the multiplicative equation from Proposition 3.2 becomes the additive congruence

$$c_k + c_{n-k-1} + d_k + d_{n-k-1} \equiv 0 \pmod 2. \qquad (*)$$

Since (A, B) is a complex Golay pair, the autocorrelation equations give us

$$\sum_{k=0}^{n-s-1} \left(i^{c_k - c_{k+s}} + i^{d_k - d_{k+s}}\right) = 0$$

Total CPU Time in hours			
n	Preproc.	Stage 1	Stage 2
17	0.00	0.01	0.06
18	0.01	0.03	0.23
19	0.01	0.07	0.18
20	0.02	0.35	0.43
21	0.04	1.93	1.89
22	0.08	9.58	1.11
23	0.15	42.01	3.02
24	0.32	81.42	5.23
25	0.57	681.31	20.51

Table 1: The time used to run the various stages of our algorithm in lengths $17 \le n \le 25$.

for $s = 1, \ldots, n-1$. Using Lemma 3.3 and the fact that $-1 \equiv 1 \pmod 2$ gives

$$\sum_{k=0}^{n-s-1} \left(c_k + c_{k+s} + d_k + d_{k+s}\right) \equiv 0 \pmod 2$$

for $s = 1, \ldots, n-1$. With $s = n-1$ one immediately derives $(*)$ for $k = 0$. With $s = n-2$ and $(*)$ for $k = 0$ one derives $(*)$ for $k = 1$. Working inductively in this manner one derives $(*)$ for all k. □

In short, Proposition 3.2 tells us that an even number of a_k, a_{n-k-1}, b_k, and b_{n-k-1} are real for each $k = 0, \ldots, n-1$. For example, if exactly one of a_k and a_{n-k-1} is real then exactly one of b_k and b_{n-k-1} must also be real. In this case, using our encoding from Section 3.3 we can add the clauses

$$(v_{2k} \vee v_{2(n-k-1)}) \wedge (\neg v_{2k} \vee \neg v_{2(n-k-1)})$$

to our SAT instance. These clauses say that exactly one of v_{2k} and $v_{2(n-k-1)}$ is true.

4 RESULTS

In order to provide a verification of the counts from [16] we implemented the enumeration method described in Section 3. The preprocessing step was performed by a C program and used the mathematical library FFTW [14] for computing the values of $h_{A'}(z)$ as described in Section 3.5. Stage 1 was performed by a C++ program, used FFTW for computing the values of $h_A(z)$ and a Maple script [30] for determining the solvability of the Diophantine equations given in Section 3.3. Stage 2 was performed by the programmatic SAT solver MapleSAT [24]. The postprocessing step was performed by a Python script.

We ran our implementation on a cluster of machines running CentOS 7 and using Intel Xeon E5-2683V4 processors running at 2.1 GHz and using at most 300MB of RAM. To parallelize the work in each length n we split L_{odd} into 25 pieces and used 25 cores to complete stages 1 and 2 of the algorithm. Everything in the stages proceeded exactly as before except that in stage 1 the list L_{odd} was 25 times shorter than it would otherwise be, which allowed us to complete the first stages 20.7 times faster and the second stages 23.9 times faster. The timings for the preprocessing step and the two stages of our algorithm are given in Table 1; the timings for the postprocessing step were negligible. The times are given as

n	$\lvert L_{\text{even}}\rvert$	$\lvert L_{\text{odd}}\rvert$	$\lvert L_{\text{A}}\rvert$
1	1	–	1
2	3	1	3
3	3	1	1
4	3	4	3
5	12	4	5
6	12	16	14
7	39	16	12
8	48	64	36
9	153	64	44
10	153	204	120
11	561	252	101
12	645	860	465
13	2121	884	293
14	2463	3284	317
15	8340	3572	1793
16	9087	12116	923
17	31275	12824	3710
18	34560	46080	14353
19	117597	50944	10918
20	130215	173620	26869
21	446052	194004	116612
22	500478	667304	67349
23	1694865	732232	182989
24	1886568	2515424	313878
25	6447090	2727452	1211520

Table 2: The number of sequences A_{even}, A_{odd}, and A that passed the filtering conditions of our algorithm in lengths up to 25.

n	$\lvert\Omega_{\text{seqs}}\rvert$	$\lvert\Omega_{\text{all}}\rvert$	$\lvert\Omega_{\text{inequiv}}\rvert$
1	4	16	1
2	16	64	1
3	16	128	1
4	64	512	2
5	64	512	1
6	256	2048	3
7	0	0	0
8	768	6656	17
9	0	0	0
10	1536	12288	20
11	64	512	1
12	4608	36864	52
13	64	512	1
14	0	0	0
15	0	0	0
16	13312	106496	204
17	0	0	0
18	3072	24576	24
19	0	0	0
20	26880	215040	340
21	0	0	0
22	1024	8192	12
23	0	0	0
24	98304	786432	1056
25	0	0	0

Table 3: The number complex Golay pairs in lengths up to 25. The table counts the number of individual sequences, the number of pairs, and the number of pairs up to equivalence.

the total amount of CPU time used across all 25 cores. Our code is available online as a part of the MathCheck project and we have also made available the resulting enumeration of complex Golay pairs [6].

The sizes of the lists L_{even} and L_{odd} computed in the preprocessing step and the size of the list L_{A} computed in stage 1 are given in Table 2 for all lengths in which we completed a search. Without applying any filtering L_{A} would have size 4^n so Table 2 demonstrates the power of the criteria we used to perform filtering; typically far over 99.99% of possible sequences A are filtered from L_{A}. The generated SAT instances had $2n$ variables (encoding the entries $b_0, \ldots, b_{n-1}$), 2 unit clauses (encoding $b_0 = 1$), $2\lfloor n/2 \rfloor$ binary clauses (encoding Proposition 3.2), and $n - 1$ programmatic clauses (encoding Definition 2.2).

Finally, we provide counts of the total number of complex Golay pairs of length $n \leq 25$ in Table 3. The sizes of Ω_{seqs} and Ω_{all} match those from [16] in all cases and the size of Ω_{inequiv} matches those from [8] for $n \leq 19$ (the largest length they exhaustively solved).

Because [8, 11, 16] do not provide implementations or timings for the enumerations they completed it is not possible for us to compare the efficiency of our algorithm to previous algorithms. However, we note that the results in this paper did not require an exorbitant amount of computing resources. If one has access to 25 modern CPU cores then one can exhaustively enumerate all complex Golay pairs up to length 25 using our software in about a day and we estimate that increasing this to length 26 would take another week. We note that Fiedler's paper [11] enumerates complex Golay pairs to length 28. It is not clear whether this was accomplished using more computing resources or a more efficient algorithm, though we note that the preprocessing and stage 1 of our method is similar to Fiedler's method with some differences in the filtering theorems.

5 FUTURE WORK

Besides increasing the length to which complex Golay pairs have been enumerated there are a number of avenues for improvements which could be made in future work. As one example, we remark that we have not exploited the algebraic structure of complex Golay pairs revealed by Craigen, Holzmann, and Kharaghani [8]. In particular, those authors prove a theorem which implies that if $p \equiv 3 \pmod 4$ is a prime which divides n and A is a member of a complex Golay pair of length n then the polynomial h_A is not irreducible over $\mathbb{F}_p(i)$. Ensuring that this property holds could be added to the filtering conditions which were used in stage 1. In fact, the authors relate the factorization of h_A over $\mathbb{F}_p(i)$ to the factorization of h_B over $\mathbb{F}_p(i)$ for any complex Golay pair (A, B). This factorization could potentially be used to perform stage 2 more efficiently, possibly supplementing or replacing the SAT solver entirely, though it is unclear if such a method would perform better than our method in practice. In any case, it would not be possible to apply their theorem in all lengths (for example when n is a power of 2).

A second possible improvement could be to symbolically determine the value of z with $|z| = 1$ which maximizes $|h_{A'}(z)|^2$ in the preprocessing step. Once this value of z is known then A' can be filtered if $|h_{A'}(z)|^2 > 2n$ and if not then no other value of z needs to be tried. This would save evaluating $h_{A'}(z)$ at the points $z = e^{2\pi ij/N}$ for $j = 0, \ldots, N-1$ and would also increase the number of sequences which get filtered. However, it is unclear if this method would be beneficial in practice due to the overhead of maximizing $|h_{A'}(z)|^2$ subject to $|z| = 1$.

Another possible improvement could be obtained by deriving further properties like Proposition 3.2 that complex Golay pairs must satisfy. We have performed some preliminary searches for such properties; for example, consider the following property which could be viewed as a strengthening of Proposition 3.2:

$$a_k\overline{a_{n-k-1}} = (-1)^{n+1}b_k\overline{b_{n-k-1}} \qquad \text{for} \qquad k = 1, \ldots, n-2.$$

An examination of all complex Golay pairs up to length 25 reveals that they all satisfy this property except for a *single* complex Golay pair up to equivalence. The only pair which doesn't satisfy this property is equivalent to

$$\Big([1,1,1,-1,1,1,-1,1],[1,i,i,-1,1,-i,-i,-1]\Big)$$

and was already singled out in [13] for being special as the only known example of what they call a "cross-over" Golay sequence pair. Since a counterexample exists to this property there is no hope of proving it in general, but perhaps a suitable generalization could be proven.

ACKNOWLEDGEMENTS

This work was made possible by the facilities of the Shared Hierarchical Academic Research Computing Network (SHARCNET) and Compute/Calcul Canada. The authors would also like to thank the anonymous reviewers whose comments improved this article's clarity.

REFERENCES

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

[2] Erika Ábrahám, John Abbott, Bernd Becker, Anna M. Bigatti, Martin Brain, Bruno Buchberger, Alessandro Cimatti, James H. Davenport, Matthew England, Pascal Fontaine, Stephen Forrest, Alberto Griggio, Daniel Kroening, Werner M. Seiler, and Thomas Sturm. 2016. SC^2: Satisfiability Checking meets Symbolic Computation (Project Paper). In *Intelligent Computer Mathematics: 9th International Conference, CICM 2016, Bialystok, Poland, July 25–29, 2016, Proceedings*. Springer International Publishing, Cham, 28–43. http://www.sc-square.org/.

[3] Curtis Bright. 2017. *Computational Methods for Combinatorial and Number Theoretic Problems*. Ph.D. Dissertation. University of Waterloo.

[4] Curtis Bright, Vijay Ganesh, Albert Heinle, Ilias S. Kotsireas, Saeed Nejati, and Krzysztof Czarnecki. 2016. MathCheck2: A SAT+CAS Verifier for Combinatorial Conjectures. In *Computer Algebra in Scientific Computing - 18th International Workshop, CASC 2016, Bucharest, Romania, September 19–23, 2016, Proceedings*. 117–133.

[5] Curtis Bright, Ilias Kotsireas, and Vijay Ganesh. 2018. A SAT+CAS Method for Enumerating Williamson Matrices of Even Order. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*.

[6] Curtis Bright, Ilias Kotsireas, Albert Heinle, and Vijay Ganesh. 2018. Complex Golay Pairs via SAT. https://cs.uwaterloo.ca/~cbright/cgpsat/. Complex Golay pairs archived at https://zenodo.org/record/1246337, code available at https://bitbucket.org/cbright/mathcheck2.

[7] R. Craigen. 1994. Complex Golay sequences. *J. Combin. Math. Combin. Comput.* 15 (1994), 161–169.

[8] R. Craigen, W. Holzmann, and H. Kharaghani. 2002. Complex Golay sequences: structure and applications. *Discrete Math.* 252, 1-3 (2002), 73–89.

[9] James A Davis and Jonathan Jedwab. 1999. Peak-to-mean power control in OFDM, Golay complementary sequences, and Reed–Muller codes. *IEEE Transactions on Information Theory* 45, 7 (1999), 2397–2417.

[10] P. G. Donato, J. Urena, M. Mazo, and F. Alvarez. 2004. Train wheel detection without electronic equipment near the rail line. In *IEEE Intelligent Vehicles Symposium, 2004*. 876–880. https://doi.org/10.1109/IVS.2004.1336500

[11] Frank Fiedler. 2013. Small Golay sequences. *Advances in Mathematics of Communications* 7, 4 (2013).

[12] Frank Fiedler, Jonathan Jedwab, and Matthew G Parker. 2008. A Framework for the Construction of Golay Sequences. *IEEE Transactions on Information Theory* 54, 7 (2008), 3114–3129.

[13] Frank Fiedler, Jonathan Jedwab, and Matthew G Parker. 2008. A multi-dimensional approach to the construction and enumeration of Golay complementary sequences. *Journal of Combinatorial Theory, Series A* 115, 5 (2008), 753–776.

[14] Matteo Frigo and Steven G Johnson. 2005. The design and implementation of FFTW3. *Proc. IEEE* 93, 2 (2005), 216–231.

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

[16] Richard G Gibson and Jonathan Jedwab. 2011. Quaternary Golay sequence pairs I: Even length. *Designs, Codes and Cryptography* 59, 1-3 (2011), 131–146.

[17] Marcel Golay. 1961. Complementary series. *IRE Transactions on Information Theory* 7, 2 (1961), 82–87.

[18] Marcel J.E. Golay. 1949. Multi-slit spectrometry. *JOSA* 39, 6 (1949), 437–444.

[19] W. H. Holzmann and H. Kharaghani. 1994. A computer search for complex Golay sequences. *Australas. J. Combin.* 10 (1994), 251–258.

[20] Aamir Hussain, Zeashan H. Khan, Azfar Khalid, and Muhammad Iqbal. 2014. *A Comparison of Pulse Compression Techniques for Ranging Applications*. Springer Singapore, Singapore, 169–191. https://doi.org/10.1007/978-981-4585-36-1_5

[21] Hadi Kharaghani and Behruz Tayfeh-Rezaie. 2005. A Hadamard matrix of order 428. *Journal of Combinatorial Designs* 13, 6 (2005), 435–440.

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

[23] Ying Li and Wen Bin Chu. 2005. More Golay sequences. *IEEE Transactions on Information Theory* 51, 3 (2005), 1141–1145.

[24] Jia Hui Liang, Pascal Poupart, Krzysztof Czarnecki, and Vijay Ganesh. 2017. An Empirical Study of Branching Heuristics Through the Lens of Global Learning Rate. In *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 119–135.

[25] A. Lomayev, Y.P. Gagiev, A. Maltsev, A. Kasher, M. Genossar, and C. Cordeiro. 2017. Golay sequences for wireless networks. https://www.google.com/patents/US20170324461 US Patent App. 15/280,635.

[26] Michael B. Monagan, Keith O. Geddes, K. Michael Heal, George Labahn, Stefan M. Vorkoetter, James McCarron, and Paul DeMarco. 2005. *Maple 10 Programming Guide*. Maplesoft, Waterloo ON, Canada.

[27] Moshe Nazarathy, Steven A Newton, RP Giffard, DS Moberly, F Sischka, WR Trutna, and S Foster. 1989. Real-time long range complementary correlation optical time domain reflectometer. *Journal of Lightwave Technology* 7, 1 (1989), 24–38.

[28] A Nowicki, W Secomski, J Litniewski, I Trots, and PA Lewin. 2003. On the application of signal compression using Golay's codes sequences in ultrasound diagnostic. *Archives of Acoustics* 28, 4 (2003).

[29] Kenneth G Paterson. 2000. Generalized Reed–Muller codes and power control in OFDM modulation. *IEEE Transactions on Information Theory* 46, 1 (2000), 104–120.

[30] Joe Riel. 2006. nsoks: A Maple script for writing n as a sum of k squares. http://www.swmath.org/software/21060.

[31] Edward Zulkoski, Curtis Bright, Albert Heinle, Ilias S. Kotsireas, Krzysztof Czarnecki, and Vijay Ganesh. 2017. Combining SAT Solvers with Computer Algebra Systems to Verify Combinatorial Conjectures. *J. Autom. Reasoning* 58, 3 (2017), 313–339. https://doi.org/10.1007/s10817-016-9396-y