Complex Golay Pairs up to Length 28: A Search via Computer Algebra and Programmatic SAT

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

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

Abstract

We use techniques from the fields of computer algebra and satisfiability checking to develop a new algorithm to search for complex Golay pairs. We implement this algorithm and use it to perform a complete search for complex Golay pairs of lengths up to 28. In doing so, we find that complex Golay pairs exist in the lengths 24 and 26 but do not exist in the lengths 23, 25, 27, and 28. This independently verifies work done by F. Fiedler in 2013 and confirms the 2002 conjecture of Craigen, Holzmann, and Kharaghani that complex Golay pairs of length 23 don't exist. Our algorithm is based on the recently proposed SAT+CAS paradigm of combining SAT solvers with computer algebra systems to efficiently search large spaces specified by both algebraic and logical constraints. The algorithm has two stages: first, a fine-tuned computer program uses functionality from computer algebra systems and numerical libraries to construct a list containing every sequence which could appear as the first sequence in a complex Golay pair up to equivalence. Second, a programmatic SAT solver constructs every sequence (if any) that pair off with the sequences constructed in the first stage to form a complex Golay pair. This extends work originally presented at the International Symposium on Symbolic and Algebraic Computation (ISSAC) in 2018; we discuss and implement several improvements to our algorithm that enabled us to improve the efficiency of the search and increase the maximum length we search from length 25 to 28.

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

1. Introduction

The sequences which are now referred to *Golay sequences* or *Golay pairs* were first introduced by Golay (1949) in a groundbreaking paper on multislit spectrometry. Later, Golay (1961) made a detailed study of their elegant mathematical properties and in this paper he referred to them as *complementary series*:

> Regardless of past or possible future applications, the writer has found these complementary series mathematically appealing...

Since then, Golay pairs and their generalizations have been widely studied and applied to a huge and surprisingly varied number of problems in engineering. For example, they

have been applied to encoding acoustic surface waves (Tseng, 1971), spread-spectrum systems (Krone and Sarwate, 1984), optical time domain reflectometry (Nazarathy et al., 1989), CDMA networks (Seberry et al., 2002), medical ultrasounds (Nowicki et al., 2003), train wheel detection (Donato et al., 2004), radar systems (Li et al., 2008), and wireless networks (Lomayev et al., 2017). Golay pairs consist of two sequences and the property that defines them (roughly speaking) is the fact that one sequence's "correlation" with itself is the inverse of the other sequence's "correlation" with itself; see Definition 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 $i \coloneqq \sqrt{-1}$. 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 *quadriphase*, *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 in several different contexts. Initially they were studied in the context of signal processing where some researchers ran searches for "polyphase complementary codes" in the process of studying signal encoding methods. In particular, Sivaswamy (1978) found a complex Golay pair of length 3 and Frank (1980) found complex Golay pairs of length 5 and 13 and stated that 7, 9, 11, 15, and 17 were not complex Golay numbers. Complex Golay pairs were also introduced by Craigen (1994) (independently of the above works) in order to expand the orders of Hadamard matrices attainable via ordinary Golay pairs—that are only known to exist in the lengths $2^{a+b+c} \cdot 5^b \cdot 13^c$ for integers $a, b, c \geq 0$. Craigen also proved that if m and n are complex Golay numbers then $2mn$ is a complex Golay number.

An exhaustive search for complex Golay pairs up to length 13 was performed by Holzmann and Kharaghani (1994). This search verified the remark of Frank (1980) that 7 and 9 are not complex Golay numbers but found that 11 *is* in fact a complex Golay number. Later an exhaustive search up to length 19 was performed by Craigen, Holzmann, and Kharaghani (2002) showing that 14, 15, 17, and 19 are not complex Golay numbers. In addition they reported that 21 was not a complex Golay number and conjectured that 23 was not a complex Golay number.

Another line of research is to provide explicit constructions for complex Golay pairs and a number of results of this form have been published as well. Fiedler, Jedwab, and Parker (2008a,b) provided a construction which explained the existence of all known complex Golay pairs whose lengths were a power of 2, including complex Golay pairs of length 16 discovered by Li and Chu (2005) to not fit into a construction given by Davis and Jedwab (1999). Gibson and Jedwab (2011) 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. This table was produced by the mathematician Frank Fiedler, who described his enumeration method in a subsequent paper (Fiedler, 2013) where he also reported that 27 and 28 are not complex Golay numbers.

In this paper we give an enumeration method that can be used to verify Fiedler's table giving the total number of complex Golay pairs up to length 28. We implemented

our method and obtained counts up to length 28 after about 8.5 months of CPU time. 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 and explicitly publish the sequences on our website `uwaterloo.ca/mathcheck`. 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 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 that could possibly appear as the first sequence in a complex Golay pair of a given length (up to the equivalence defined in Proposition 8 of Section 2). Several filtering theorems that 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 (Monagan et al., 2005) and the mathematical library FFTW (Frigo and Johnson, 2005) as we describe in Section 3. 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 developed by Liang et al. (2017) to determine which sequences from the first stage (if any) can be paired up with another sequence to form a complex Golay pair. A programmatic SAT instance is constructed from each sequence found in the first stage such that the instance is satisfiable if and only if the sequence is part of a complex Golay pair. Furthermore, when the instance is satisfiable the assignment produced by the SAT solver determines the second sequence of a complex Golay pair.

The "SAT+CAS" method that we use 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 outlined in the invited talk at ISSAC by Ábrahám (2015) and demonstrated by the SC^2 (satisfiability checking + symbolic computation) project initiated by Ábrahám et al. (2016). 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 like those defining Williamson matrices (Bright et al., 2016; Zulkoski et al., 2017; Bright, 2017).

A preliminary version of our algorithm and results was presented at the conference ISSAC (Bright et al., 2018b). We have since made several improvements to our algorithm that allow us to extend our exhaustive search from all lengths $n \leq 25$ to all lengths $n \leq 28$. We describe our implementation, give timings for our searches, and compare the timings with those of our previous algorithm in Section 4. The new timings are about an order of magnitude faster on the same hardware, demonstrating the effectiveness of our improvements. In particular, our new algorithm incorporates the following updates:

More efficient filtering. The preprocessing stage of our algorithm uses a filtering condition (Corollary 14 of Section 2) to remove a large number of sequences from consideration. In our original algorithm we applied this condition using a large number of equally-spaced points z along the unit circle. In our new algorithm we apply this condition significantly less often but filter approximately the same number of sequences (and sometimes even more) by showing how to only use the condition for the z which are the most likely to work.

More efficient joining. The first stage of our algorithm requires searching through two lists and finding elements of the first that can be joined with elements of the second. In our previous algorithm this was done in a totally brute-force manner, i.e., every item in the first list was paired with every item of the second list to see if they could be joined. In our new algorithm we sort the lists using a custom ordering and show how to find the elements that can be joined via a number of linear scans through the sorted lists, thereby eliminating the need to consider all possible pairs.

Increased filtering. Because the improved filtering method outlined above is so much faster than the previous filtering method we were able to add another round of filtering while the SAT instances were being generated. The result is that approximately 40% of the SAT instances generated by our previous algorithm can now be quickly filtered before even calling the SAT solver.

Additionally we include more background details in this version of the paper. For example, we add Section 2.1 that gives an alternative definition of complex Golay pairs that is often useful and we show that this definition is equivalent with the first definition.

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. We let $\overline{z}$ denote the complex conjugate of z (which is just the multiplicative inverse of z for z on the unit circle) and if A is a sequence we let $\overline{A}$ denote the sequence containing the conjugates of A.

Definition 1 (cf. Kotsireas (2013))**.** *The* nonperiodic autocorrelation function *of a sequence* $A = [a_0, \ldots, a_{n-1}]$ *of length* n *is*

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

Definition 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 $\{\pm 1, \pm i\}^n$ (as we assume throughout this paper) then $N_A(0) + N_B(0) = 2n$ by the definition of the nonperiodic autocorrelation function and the fact that $z\overline{z} = 1$ if z is ± 1 or $\pm i$, explaining why $s \neq 0$ in Definition 2.

Example 3. $([1, 1, -1], [1, i, 1])$ *is a complex Golay pair of length* 3 *since the first sequence A has autocorrelations* $N_A(1) = 0$ *and* $N_A(2) = -1$ *and the second sequence B has autocorrelations* $N_B(1) = 0$ *and* $N_B(2) = 1$.

2.1. Alternative definition

Instead of viewing complex Golay pairs as pairs of *sequences* it is also possible to view them as pairs of *polynomials*. If A is the sequence $[a_0, \ldots, a_{n-1}]$ we let $A(z)$ denote the *Hall polynomial* $a_0 + a_1 z + \ldots + a_{n-1} z^{n-1}$ which can be viewed as the finite generating function of A. This leads to the following alternative definition of complex Golay pairs.

Definition 4. *A pair of polynomials* $(A(z), B(z))$ *that have degrees* $n - 1$ *and coefficients in* $\{\pm 1, \pm i\}$ *are called a* complex Golay pair *if* $|A(z)|^2 + |B(z)|^2 = 2n$ *for all* z *on the unit circle.*

Example 5. $(1 + z - z^2, 1 + iz + z^2)$ *is a complex Golay pair of length* 3 *since*

$$\left|1 + z - z^2\right|^2 + \left|1 + iz + z^2\right|^2 = 6$$

for all z *on the unit circle.*

These two definitions can be seen to be equivalent using the following lemma that expresses the squared absolute values of a polynomial evaluated on the unit circle in terms of the autocorrelations of the sequence formed by the polynomial's coefficients.

Lemma 6. *Let* A *be a complex sequence of length* n. *Then*

$$|A(z)|^2 = N_A(0) + 2\,\mathrm{Re}\Big(\sum_{s=1}^{n-1} N_A(s) z^{-s}\Big).$$

for all z *on the unit circle.*

Proof. Since z is on the unit circle we have $\overline{z} = z^{-1}$, so $|A(z)|^2 = A(z)\overline{A(z)} = A(z)\overline{A}(z^{-1})$. Expanding this, we obtain that $|A(z)|^2$ is equal to

$$\sum_{k=0}^{n-1} \sum_{l=0}^{n-1} a_k \overline{a_l} z^{k-l}.$$

Collecting terms of z^s together for s from $-n + 1$ to $n - 1$ this becomes

$$\sum_{s=0}^{n-1} \Big(\sum_{j=s}^{n-1} a_j \overline{a_{j-s}}\Big) z^s + \sum_{s=1}^{n-1} \Big(\sum_{j=s}^{n-1} a_{j-s} \overline{a_j}\Big) z^{-s}$$

so the coefficient of z^s for positive s is $\overline{N_A(s)}$ and for negative s is $N_A(s)$. Using the fact that $\overline{N_A(s)} z^s = \overline{N_A(s) z^{-s}}$ this becomes

$$N_A(0) + \sum_{s=1}^{n-1} \Big(\overline{N_A(s) z^{-s}} + N_A(s) z^{-s}\Big).$$

and the desired result follows from the fact that $z + \overline{z} = 2\,\mathrm{Re}(z)$ for all complex z. □

For completeness we now demonstrate the equivalence of Definitions 2 and 4.

Theorem 7. *A pair (A, B) is a complex Golay pair in the sense of Definition 2 if and only if $(A(z), B(z))$ is a complex Golay pair in the sense of Definition 4.*

Proof. If (A, B) is a complex Golay pair in the sense of Definition 2 then by Lemma 6

$$|A(z)|^2 + |B(z)|^2 = N_A(0) + N_B(0) + 2\,\mathrm{Re}\Big(\sum_{s=1}^{n-1} (N_A(s) + N_B(z))z^{-s}\Big) = 2n$$

for all z on the unit circle, as required.

Conversely, if $(A(z), B(z))$ is a complex Golay pair in the sense of Definition 4 then by Lemma 6 and subtracting off $N_A(0) + N_B(0) = 2n$ we have

$$\sum_{s=1}^{n-1} \Big(N_A(s)z^{-s} + \overline{N_A(s)}z^{s} + N_B(z)z^{-s} + \overline{N_B(s)}z^{s}\Big) = 0$$

for all z on the unit circle. Since any nonzero rational function has a finite number of zeros in $\mathbb{C}$ the function given on the left hand side must be identically zero. In particular, the coefficients of z^{-s} for $s = 1, \ldots, n-1$ must be zero and we derive $N_A(s) + N_B(s) = 0$ for $s = 1, \ldots, n-1$ as required. □

2.2. 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 8 (cf. Craigen et al. (2002))**.** *Let $([a_0, \ldots, a_{n-1}], [b_0, \ldots, b_{n-1}])$ be a complex Golay pair. Then the following are 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 $c \star A$ denotes the sequence of coefficients of the polynomial $A(cz)$, i.e.,* $[a_0, ca_1, c^2a_2, \ldots, c^{n-1}a_{n-1}]$.

Proof. Suppose (A, B) is a complex Golay pair and let z be on the unit circle.

E1. Note that the reverse of $A(z)$ is $z^{n-1}A(z^{-1})$ and $|z^{n-1}| = 1$. Then

$$|z^{n-1}A(z^{-1})|^2 + |z^{n-1}B(z^{-1})|^2 = |A(z^{-1})|^2 + |B(z^{-1})|^2 = 2n$$

since (A, B) is a complex Golay pair and z^{-1} is on the unit circle.

E2. The conjugate reverse of $A(z)$ is $z^{n-1}\overline{A}(z^{-1})$ and $|z^{n-1}\overline{A}(z^{-1})| = |\overline{A(z)}| = |A(z)|$.

E3. $|B(z)|^2 + |A(z)|^2 = |A(z)|^2 + |B(z)|^2 = 2n$.

E4. $A(z)$ scaled by i is $iA(z)$ and $|iA(z)| = |A(z)|$.

E5. $|A(iz)|^2 + |B(iz)|^2 = 2n$ since (A, B) is a complex Golay pair and iz is on the unit circle.

□

Definition 9. *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 8.*

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 10 (cf. Fiedler (2013))**.** *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 8. 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 $b_0 = 1$. □

2.3. Filtering properties

We now prove some useful properties that all complex Golay pairs satisfy and that will be exploited by our algorithm for enumerating complex Golay pairs. The properties will be used as filtering criteria: if a sequence does not satisfy them then we know it cannot possibly be part of a complex Golay pair and may therefore be filtered away. The next well-known lemma is one of the most powerful results of this form.

Lemma 11 (cf. Popović (1991))**.** *Let* A *be a complex sequence of length* n*. If* $|A(z)|^2 > 2n$ *for some* z *on the unit circle 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 $|B(z)|^2 \geq 0$, we must have $|A(z)|^2 + |B(z)|^2 > 2n$, in contradiction to Definition 4. □

Example 12. *The sequence* $A := [1, 1, 1]$ *cannot be a member of a complex Golay pair since* $|A(1)|^2 = 9$ *is larger than* $2n = 6$.

Fiedler (2013) derives a variation of Lemma 11 that is useful because it can be applied knowing only around half of the entries of A. It is derived using the following variation of Lemma 6. 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 13 (cf. Fiedler (2013))**.** *Let A be a complex sequence of length n. Then*

$$|A_{\text{even}}(z)|^2 + |A_{\text{odd}}(z)|^2 = N_A(0) + 2\,\text{Re}\Bigg(\sum_{\substack{s=1\\ s\ even}}^{n-1} N_A(s)z^{-s}\Bigg).$$

Proof. The proof proceeds like in the proof of Lemma 6 and we derive

$$|A_{\text{even}}(z)|^2 + |A_{\text{odd}}(z)|^2 = \sum_{\substack{k,l=0\\ k,\,l\ \text{even}}}^{n-1} a_k\overline{a}_l z^{k-l} + \sum_{\substack{k,l=0\\ k,\,l\ \text{odd}}}^{n-1} a_k\overline{a}_l z^{k-l} = \sum_{\substack{k,l=0\\ k-l\ \text{even}}}^{n-1} a_k\overline{a}_l z^{k-l}.$$

From this we see that the coefficient on z^s for odd s is zero and the coefficient on z^s for even s is the same as in Lemma 6 from which the result follows. □

Corollary 14. *Let A be a complex sequence of length n. If $|A_{\text{even}}(z)|^2$ or $|A_{\text{odd}}(z)|^2$ is strictly larger than $2n$ for some z on the unit circle then A is not a member of a complex Golay pair.*

Proof. If (A, B) is a complex Golay pair then by Lemma 13 we have

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

but if either $|A_{\text{even}}(z)|^2 > 2n$ or $|A_{\text{odd}}(z)|^2 > 2n$ then this cannot hold. □

Example 15. *The sequence $A \coloneqq [1, x, 1, y, 1, z, 1]$ cannot be a member of a complex Golay pair (regardless of the values of x, y, and z) since $|A_{\text{even}}(1)|^2 = 16$ is larger than $2n = 14$.*

2.4. Sum-of-squares decomposition types

The next lemma is useful because it allows us to write $2n$ as the sum of four integer squares. It is stated by Holzmann and Kharaghani (1994) using a different notation; we use the notation resum(A) and imsum(A) to represent the real and imaginary parts of the sum of the entries of A. For example, if $A \coloneqq [1, i, -i, i]$ then resum(A) = imsum(A) = 1.

Lemma 16 (cf. Holzmann and Kharaghani (1994))**.** *If (A, B) is a complex Golay pair then*

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

Proof. Using Definition 4 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. □

A consequence of Lemma 16 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 16 on these complex Golay pairs one obtains the fact that $2n$ can be decomposed as the sum of four integer squares as

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

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 $\mathrm{resum}(A) = 0$ and $\mathrm{imsum}(A) = 5$. 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 28. 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 10 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 14 to every possibility for A_{even} and A_{odd}. There are an infinite number of z on the unit circle so it is not possible to apply Corollary 14 using all such z. One simple approach is to try a sufficiently large number of points z so that when a point exists with $|A'(z)|^2 > 2n$ (where A' is either A_{even} or A_{odd}) such a point is usually discovered. For example, Bright et al. (2018b) tested Corollary 14 for 2^{14} equally-spaced points z around the unit circle.

Instead, in our implementation we use a nonlinear programming method to estimate the maximum of $|A'(z)|^2$ for z on the unit circle. This can be done with the NLPSOLVE command of the computer algebra system MAPLE though for efficiency we use a custom

C implementation of a variant of the quadratic interpolation method described by Sun and Yuan (2006).

We write $|A'(z)|^2$ in terms of the real variable θ by using the substitution $z = e^{i\theta}$ and define $f(\theta) \coloneqq |A'(e^{i\theta})|^2$. The quadratic interpolation method to estimate the maximum of $f(\theta)$ over $0 \le \theta < 2\pi$ proceeds as follows:

1. Evaluate f at the 2^7 points $\theta_0, \ldots, \theta_{127}$ where $\theta_k \coloneqq \frac{2\pi k}{128}$ and let $f_k \coloneqq f(\theta_k)$. If $f_k > 2n$ for some k we can immediately filter A' by Corollary 14.
2. For every value of f that is larger than its neighbours (i.e., values of f_k that satisfy $f_{k-1} \le f_k$ and $f_k \ge f_{k+1}$) we use interpolation to find the quadratic polynomial that passes through the points (θ_{k-1}, f_{k-1}), (θ_k, f_k), and (θ_{k+1}, f_{k+1}). Let θ^* be the value of θ that maximizes the quadratic polynomial; this is well-known known to be

$$\frac{1}{2} \cdot \frac{f_{k-1}(\theta_k^2 - \theta_{k+1}^2) + f_k(\theta_{k+1}^2 - \theta_{k-1}^2) + f_{k+1}(\theta_{k-1}^2 - \theta_k^2)}{f_{k-1}(\theta_k - \theta_{k+1}) + f_k(\theta_{k+1} - \theta_{k-1}) + f_{k+1}(\theta_{k-1} - \theta_k)}$$

   and let $f^* \coloneqq f(\theta^*)$.
3. Note that f^* is often a better approximation to a local maximum of f than f_k was, and if $f^* > 2n$ then we can filter A' by Corollary 14. Otherwise we can use the point (θ^*, f^*) to derive a tighter interval in which a local maximum of f must lie. For example, if $\theta_k < \theta^* < \theta_{k+1}$ and $f^* > f_k$ then we can repeat the previous step except using the points (θ_k, f_k), (θ^*, f^*), and (θ_{k+1}, f_{k+1}).

One can use this method to derive more and more accurate approximations to the local maxima of f though this is mostly unnecessary for our purposes as we only care about finding a value for θ with $f(\theta) > 2n$. A good approximation to a local maximum was usually found after a single interpolation step so in our implementation we would move on to looking for another local maximum of f after repeating the interpolation step three times. If all values of f_k were examined and no points θ were found with $f(\theta) > 2n$ then we save A' as a sequence which could not be filtered.

At the conclusion of this step we 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 all possibilities for A_{even} with all possibilities for A_{odd}. The most straightforward way of doing this would be to simply try all $A_1 \in L_{\text{odd}}$ and $A_2 \in L_{\text{even}}$; this was done by Bright et al. (2018b). However, because both L_{even} and L_{odd} can contain millions of sequences it can be inefficient to try all possible pairings (A_1, A_2). However, many pairings can be eliminated by using the conditions implied by Lemma 16; we now show how to find all possible pairings that satisfy these conditions without needing to try all $A_1 \in L_{\text{odd}}$ and $A_2 \in L_{\text{even}}$.

Let (u_0, u_1, u_2, u_3) be an arbitrary quadruple of integers. We will describe how to efficiently find all A whose entries of odd index are in L_{odd}, whose entries of even index are in L_{even}, and that satisfy

$$\text{resum}(A) = u_0, \quad \text{imsum}(A) = u_1, \quad \text{resum}(i \star A) = u_2, \quad \text{imsum}(i \star A) = u_3. \quad (1)$$

For each $A_1 \in L_{\text{odd}}$ we form the vector

$$V_1 \coloneqq (\text{resum}(A_1), \text{imsum}(A_1), \text{resum}(i \star A_1), \text{imsum}(i \star A_1))$$

and for each $A_2 \in L_{\text{even}}$ we form the vector

$$V_2 \coloneqq (u_0 - \text{resum}(A_2), u_1 - \text{imsum}(A_2), u_2 - \text{resum}(i \star A_2), u_3 - \text{imsum}(i \star A_2)).$$

We now show that the A that satisfy relationship (1) are exactly those formed by the A_1 and A_2 with $V_1 = V_2$.

Lemma 17. *Let $A_1 \in L_{odd}$ and $A_2 \in L_{even}$ and let A be the sequence that satisfies the relationship $A(z) = A_1(z) + A_2(z)$, i.e., A is a sequence of length n with $\{\pm 1, \pm i\}$ entries. Then A satisfies* (1) *if and only if the vectors V_1 and V_2 defined as above satisfy $V_1 = V_2$.*

Proof. Consider the first entry of V_1 and V_2. Since they are equal, we have

$$\text{resum}(A_1) = u_0 - \text{resum}(A_2).$$

Which implies that $\text{resum}(A) = u_0$ since $\text{resum}(A) = \text{resum}(A_1) + \text{resum}(A_2)$. Similarly, we derive $\text{imsum}(A) = u_1$, $\text{resum}(i \star A) = u_2$, and $\text{imsum}(i \star A) = u_3$, as required. □

Therefore we have translated the problem of finding the sequences A that satisfy (1) into the problem of finding $A_1 \in L_{\text{odd}}$ and $A_2 \in L_{\text{even}}$ that have matching vectors V_1 and V_2. We can efficiently solve this problem using a string sorting algorithm as described by (for example) Kotsireas, Koukouvinos, and Seberry (2009). It works as follows: first, we sort the list L_{odd} so that the vectors V_1 formed above appear in lexicographically increasing order when iterating through $L_1 \in L_{\text{odd}}$. Similarly, we sort L_{even} in the same way so that the V_2 appear in lexicographically increasing order.

We can then find all V_1 that match with V_2 by a linear scan through the lists L_1 and L_2 and this requires only one pass through each list. For example, if at some point we find that V_1 is lexicographically greater than V_2 then we try the next L_2 in the list L_{even} and if we instead find that V_2 is lexicographically greater than V_1 then we try the next L_1 in the list L_{odd}. If instead we find $V_1 = V_2$ we iterate through the L_1 and L_2 that share the same value of V_1 and V_2 and save the A formed by joining all such A_1 and A_2 in a new list L_{A}.

It remains to determine the appropriate values of (u_0, u_1, u_2, u_3) to use in (1). To do this we use a quadratic Diophantine equation solver and find all solutions of the Diophantine equation

$$u^2 + v^2 + x^2 + y^2 = 2n \qquad \text{in integers } u, v, x, y.$$

Let U be the set of all pairs (u, v) for which this equation is solvable. By Lemma 16 and operation E5 we know that if A is a complex Golay pair then $(\text{resum}(A), \text{imsum}(A))$ and $(\text{resum}(i \star A), \text{imsum}(i \star A))$ must be members of U. Therefore, we apply the above joining procedure for all $((u_0, u_1), (u_2, u_3)) \in U^2$.

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 10) of length n will

be in L_{A}. To decrease the size of L_{A} even further we perform additional filtering before adding sequences into L_{A}, for example, we ensure that

$$(\mathrm{resum}(-1 \star A), \mathrm{imsum}(-1 \star A)) \quad \text{and} \quad (\mathrm{resum}(-i \star A), \mathrm{imsum}(-i \star A))$$

are both in U for each A added to L_{A}. We also filter those A with $|A(z)|^2 > 2n$ for some z on the unit circle (see Section 3.5 for optimization details).

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 $A \in L_{\mathrm{A}}$ 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 by Ganesh et al. (2012) 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 and good matrices by Bright et al. (2018a,c). 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. Increased expressiveness is also a feature of SMT (SAT modulo theories) solvers, though SMT solvers typically require additional overhead and only support a fixed number of theories such as those specified in the SMT library (Barrett et al., 2016). 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
$$\neg v_0^{\text{cur}} \vee \neg v_1^{\text{cur}} \vee \neg v_{2n-2}^{\text{cur}} \vee \neg v_{2n-1}^{\text{cur}}$$
that says that at least one of $\{v_0, v_1, v_{2n-2}, v_{2n-1}\}$ must have their value changed.
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 10. In fact, since Lemma 10 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 unit 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 2.2 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 by Gibson and Jedwab (2011) 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 the preprocessing step and stage 1 it is necessary to evaluate a polynomial at points on the unit circle and determine its squared absolute value. The fastest way we found to do this used the discrete Fourier transform (DFT). 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 the jth entry of $\mathrm{DFT}(A')$ is exactly $A'(z_j)$ where $z_j \coloneqq \exp(2\pi i j/N)$. Thus, we determine the values of $|A'(z_j)|^2$ by taking the squared absolute values of the entries of $\mathrm{DFT}(A')$. If $|A'(z)|^2 > 2n$ for some z then by Lemma 11 or Corollary 14 we can discard A' from consideration. To guard against potential inaccuracies introduced by the algorithms used to compute the DFT we actually ensure that $|A'(z)|^2 > 2n + \epsilon$ for some tolerance ϵ which is small but larger than the accuracy of the DFT (e.g., $\epsilon = 10^{-3}$).

In stage 1 we solve the quadratic Diophantine equation $u^2 + v^2 + x^2 + y^2 = 2n$ in integers u, v, x, y. In fact, we can also add the constraints

$$u + v \equiv n \pmod 2 \quad \text{and} \quad x + y \equiv n \pmod 2$$

(the first implies the second) because of the following lemma.

Lemma 18. *Suppose u and v are the* resum *and* imsum *of a sequence $A \in \{\pm 1, \pm i\}^n$. Then $u + v \equiv n \pmod 2$.*

Proof. Let $\#_c$ denote the number of entries in A 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 A. □

In the final filtering step of stage 1 we ensure that Diophantine equations of the form $R^2 + I^2 + x^2 + y^2 = 2n$ are solvable in integers (x, y) where R and I are given. This can be efficiently done by precomputing a Boolean two dimensional array D such that $D_{|R|,|I|}$ is true if and only if this equation has a solution, making the check for solvability a fast lookup. At this point we also check if we can find a z on the unit circle with $|A(z)|^2 > 2n$. Because the joining process is the bottleneck of our algorithm it is important to make this filtering step as efficient as possible; we did this by computing the values of $|A(z)|^2$ via the expression $|A_1(z) + A_2(z)|^2$ and precomputing the values of $A_1(z)$ and $A_2(z)$ for $A_1 \in L_{\text{odd}}$ and $A_2 \in L_{\text{even}}$ on points of the form $z = \exp(2\pi i j/32)$ with $j = 0, \ldots, 31$.

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 checked the condition for z of the form $\exp(2\pi i j/N)$ with j odd and $N = 8$, 16, and 32 in that order. (This ignores checking the condition when $z \in \{\pm 1, \pm i\}$ but that is desirable since $|A(i^k)|^2 = \mathrm{resum}(i^k \star A)^2 + \mathrm{imsum}(i^k \star A)^2$ and the sums-of-squares condition is a strictly stronger filtering method.)

The downside of precomputing the values of $A_1(z)$ and $A_2(z)$ for $A_1 \in L_{\text{odd}}$ and $A_2 \in L_{\text{even}}$ is that when the lists L_{odd} and L_{even} are large this requires a significant amount of memory. In our implementation storing the values of $A_2(z)$ for all $A_2 \in L_{\text{even}}$ in the lengths 27 and 28 each required about 8GB of RAM, the searches in lengths 25 and 26 each required about 2GB of RAM, and the searches in lengths 23 and 24 each required about 0.5GB of RAM. However, the amount of precomputation could be increased or decreased depending on the amount of memory available.

In stage 1 we make a pass through the lists L_{odd} and L_{even} for each valid possibility of the values (u_0, u_1, u_2, u_3). However, it is possible to reuse some work between passes.

For example, it is only necessary to sort the lists L_{odd} and L_{even} once. Even though the values of V_2 depend on the values (u_0, u_1, u_2, u_3) the relative ordering of L_{even} is unaffected since all V_2 are shifted by the same amount in each pass. It is even possible to do some parts of the passes simultaneously: for example, if the first coordinate of V_1 is greater than the first coordinate of V_2 for some u_0 then V_1 is lexicographically greater than V_2 in all passes with that value of u_0 and so we can iterate to the next L_2 in all such passes.

Furthermore, at the conclusion of stage 1 we add an additional round of filtering to the sequences in L_{A}. Since this step is not a bottleneck we use a more involved filtering process here; in our implementation we ensured that $|A(z)|^2 \leq 2n$ for the 2^{10} points of the form $z_j = \exp(2\pi i j/2^{10})$. Additionally, we keep track of the local maxima found by this process and use the quadratic interpolation method described in Section 3.1 to find more accurate approximations to the local maxima of $|A(z)|^2$.

In stage 2 one can also include properties that complex Golay sequences must satisfy in the SAT instances. As an example of this, we state the following proposition that was published by Bright et al. (2018b) at ISSAC 2018.

Proposition 19. *Let (A, B) be a complex Golay pair. Then*

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

To prove this, we use the following simple lemma.

Lemma 20. *Let $c_k \in \mathbb{Z}_4$ for $k = 0, \ldots, n-1$. Then*

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

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 $\sum_{k=0}^{n-1} c_k = \#_1 + 2\#_2 + 3\#_3 = 4\#_1 + 2\#_2 \equiv 0 \pmod 2$. □

We now prove Proposition 19.

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 19 becomes the additive congruence

$$c_k + c_{n-k-1} + d_k + d_{n-k-1} \equiv 0 \pmod 2. \tag{2}$$

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

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

for $s = 1, \ldots, n-1$. Using Lemma 20 and the fact that $-1 \equiv 1 \pmod 2$ gives

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

for $s = 1, \ldots, n-1$. With $s = n-1$ (or $s = 1$) one immediately derives (2) for $k = 0$. Adding together these congruences for $s = n-1$ and $s = n-2$ derives (2) for $k = 1$. The congruences for $s = n-2$ and $s = n-3$ give (2) for $k = 2$ and proceeding in this manner one derives (2) for all k. □

In short, Proposition 19 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 add the two binary clauses

$$v_{2k} \vee v_{2(n-k-1)} \qquad \text{and} \qquad \neg v_{2k} \vee \neg v_{2(n-k-1)}$$

to our SAT instance. These clauses say that exactly one of v_{2k} and $v_{2(n-k-1)}$ is true. Conversely, if an even number of a_k and a_{n-k-1} are real then an even number of b_k and b_{n-k-1} must also be real. In this case we add the two binary clauses

$$v_{2k} \vee \neg v_{2(n-k-1)} \qquad \text{and} \qquad \neg v_{2k} \vee v_{2(n-k-1)}$$

to our SAT instances.

4. Results

In order to provide a verification of the counts given by Fiedler (2013), Gibson and Jedwab (2011), and Craigen, Holzmann, and Kharaghani (2002) we implemented the enumeration method described in Section 3. The preprocessing step was performed by a C program and used the mathematical library FFTW by Frigo and Johnson (2005) for computing the values of $f_0, \ldots, f_{127}$ as described in Section 3.5. Stage 1 was performed by a C++ program, used FFTW for computing the values of $A_1(z)$ and $A_2(z)$ and a MAPLE script by Riel (2006) for determining the solvability of the Diophantine equations given in Section 3.3. Stage 2 was performed by the programmatic SAT solver MAPLESAT by Liang et al. (2017). 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 8.5GB of RAM. The lengths up to 22 were run on a single core, the lengths 23 and 24 were run on 10 cores, the lengths 25 and 26 were run on 100 cores, and the lengths 27 and 28 were run on 1000 cores. The work was parallelized across N cores by partitioning the list L_{odd} into N sublists of approximately equal size. Everything in the stages proceeded exactly as before except that in stage 1 the list L_{odd} was about N times shorter than it would have been otherwise. In practice, this allowed us to complete the search nearly N times faster; for example, our search in length 28 required about 3.5 months of CPU time but completed in about 3 hours when distributed across 1000 cores. 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 the total amount of CPU time used across all cores. Our code is available online as a part of the MATHCHECK project (see `uwaterloo.ca/mathcheck`) and the resulting

	Total CPU Time in hours		
n	Preproc.	Stage 1	Stage 2
17	0.00	0.00	0.00
18	0.00	0.01	0.01
19	0.00	0.01	0.01
20	0.00	0.08	0.04
21	0.00	0.71	0.13
22	0.00	0.88	0.15
23	0.01	4.67	0.19
24	0.01	7.09	1.24
25	0.03	97.86	1.37
26	0.06	234.43	4.72
27	0.12	3255.56	49.56
28	0.22	2543.34	25.73

Table 1: The time used to run the various stages of our algorithm in lengths $17 \leq n \leq 28$.

enumeration of complex Golay pairs is available from `https://doi.org/10.5281/zenodo.1246337`.

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 $n \leq 28$. 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; for example in length 28 less than $10^{-8}\%$ of the possible sequences A were added to 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 19), and $n - 1$ programmatic constraints (encoding Definition 2).

Finally, we provide counts of the total number of complex Golay pairs of length $n \leq 28$ in Table 3. The sizes of Ω_{seqs} match those given by Fiedler (2013) in all cases, the sizes of Ω_{all} match those given by Gibson and Jedwab (2011) for all $n \leq 26$ (the largest length they included) and the sizes of Ω_{inequiv} match those given by Craigen et al. (2002) for all $n \leq 19$ (the largest length they exhaustively solved).

Because Fiedler (2013), Gibson and Jedwab (2011), and Craigen et al. (2002) 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, in Table 4 we compare our implementation's timings to the timings we previously presented (Bright et al., 2018b). Table 4 shows that the improved version of our algorithm performs about an order of magnitude faster in general.

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

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	118
11	561	252	99
12	645	860	445
13	2121	884	279
14	2463	3284	294
15	8340	3572	1650
16	9087	12116	829
17	31275	12824	3233
18	34560	46080	11159
19	117597	50944	10918
20	130215	173620	26876
21	446052	194004	81941
22	500478	667304	90163
23	1694871	732232	118747
24	1886562	2515416	200138
25	6447250	2727452	709584
26	7183879	9578506	737891
27	24426370	10591928	7618474
28	27265578	36354113	3687209

Table 2: The number of sequences A_{even}, A_{odd}, and A that passed the filtering conditions of our algorithm in lengths up to 28.

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
26	1280	10240	16
27	0	0	0
28	0	0	0

Table 3: The number complex Golay pairs in lengths up to 28. The table counts the number of individual sequences, the number of pairs, and the number of pairs up to equivalence.

	Total CPU Time in hours	
n	ISSAC'18	This paper
17	0.07	0.00
18	0.27	0.02
19	0.26	0.02
20	0.8	0.12
21	3.86	0.84
22	10.77	1.03
23	45.18	4.87
24	86.97	8.34
25	702.39	99.26
26	–	239.21
27	–	3305.24
28	–	2569.29

Table 4: A comparison between the method presented at ISSAC 2018 (Bright et al., 2018b) and the method used in this paper. The times measure the total amount of computation time that was used to run a complete search in the lengths $17 \leq n \leq 28$ when run on the same hardware.

Golay pairs revealed by Craigen, Holzmann, and Kharaghani (2002). 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 $A(z)$ 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 $A(z)$ over $\mathbb{F}_p(i)$ to the factorization of $B(z)$ 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 be better than our method in practice. In any case, it would not be possible to apply their theorem in all lengths since n may only be divisible by the primes $p = 2$ or $p \equiv 1 \pmod 4$.

A second possible improvement could be to use the symbolic form of $f(\theta)$ defined in Section 3.1 to help find the maximum of $f(\theta)$. For example, a consequence of Lemma 6 and Euler's identity is that

$$f(\theta) = N_{A'}(0) + 2\sum_{s=1}^{n-1} \mathrm{Re}(N_{A'}(s))\cos(s\theta) + 2\sum_{s=1}^{n-1} \mathrm{Im}(N_{A'}(s))\sin(s\theta)$$

showing that f has a simple form as a sum of sines and cosines. Finding the real roots of the derivative of f would reveal the locations of the local maxima and minima of f. However, again it is unclear if this method would perform better than our approximation method in practice. Note that our method is not guaranteed to find the global maximum of f since we only apply the quadratic interpolation optimization method to ranges that we know contain a local maximum by examining the initial sampling of f. With a more careful branch-and-bound algorithm it would be possible to guarantee that the global maximum was found but the approximations we found were effective enough that we did not pursue this.

Another possible improvement could be obtained by deriving further properties like Proposition 19 that complex Golay pairs must satisfy. For example, consider the following property which could be viewed as a strengthening of Proposition 19:

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

An examination of all complex Golay pairs up to length 28 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

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

and was already singled out by Fiedler, Jedwab, and Parker (2008b) 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 of our ISSAC 2018 submission whose comments improved this article's clarity.

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. ACM, New York, 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

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

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

Bright, C., Ganesh, V., Heinle, A., Kotsireas, I. S., Nejati, S., Czarnecki, K., 2016. MATHCHECK2: A SAT+CAS Verifier for Combinatorial Conjectures. In: Computer Algebra in Scientific Computing - 18th International Workshop, CASC 2016, Bucharest, Romania, September 19–23, 2016, Proceedings. 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: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2–7, 2018. pp. 6573–6580.
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

Bright, C., Đoković, D. Ž., Kotsireas, I., Ganesh, V., 2018c. A SAT+CAS approach to finding good matrices: New examples and counterexamples. To appear at the AAAI Conference on Artificial Intelligence.
URL https://arxiv.org/abs/1811.05094

Craigen, R., 1994. Complex Golay sequences. J. Combin. Math. Combin. Comput. 15, 161–169.

Craigen, R., Holzmann, W., Kharaghani, H., 2002. Complex Golay sequences: structure and applications. Discrete Math. 252 (1–3), 73–89.
URL https://doi.org/10.1016/S0012-365X(01)00162-5

Davis, J. A., Jedwab, J., 1999. Peak-to-mean power control in OFDM, Golay complementary sequences, and Reed–Muller codes. IEEE Transactions on Information Theory 45 (7), 2397–2417.
URL https://doi.org/10.1109/18.796380

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

Fiedler, F., 2013. Small Golay sequences. Advances in Mathematics of Communications 7 (4).
URL http://doi.org/10.3934/amc.2013.7.379

Fiedler, F., Jedwab, J., Parker, M. G., 2008a. A framework for the construction of Golay sequences. IEEE Transactions on Information Theory 54 (7), 3114–3129.
URL https://doi.org/10.1109/TIT.2008.924667

Fiedler, F., Jedwab, J., Parker, M. G., 2008b. A multi-dimensional approach to the construction and enumeration of Golay complementary sequences. Journal of Combinatorial Theory, Series A 115 (5), 753–776.
URL https://doi.org/10.1016/j.jcta.2007.10.001

Frank, R., 1980. Polyphase complementary codes. IEEE Transactions on Information theory 26 (6), 641–647.
URL https://doi.org/10.1109/TIT.1980.1056272

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: International Conference on Theory and Applications of Satisfiability Testing. Springer, pp. 143–156.
URL https://doi.org/10.1007/978-3-642-31612-8_12

Gibson, R. G., Jedwab, J., 2011. Quaternary Golay sequence pairs I: Even length. Designs, Codes and Cryptography 59 (1-3), 131–146.
URL https://doi.org/10.1007/s10623-010-9471-z

Golay, M., 1961. Complementary series. IRE Transactions on Information Theory 7 (2), 82–87.
URL https://doi.org/10.1109/TIT.1961.1057620

Golay, M. J., 1949. Multi-slit spectrometry. JOSA 39 (6), 437–444.
URL https://doi.org/10.1364/JOSA.39.000437

Holzmann, W. H., Kharaghani, H., 1994. A computer search for complex Golay sequences. Australas. J. Combin. 10, 251–258.

Kotsireas, I. S., 2013. Algorithms and metaheuristics for combinatorial matrices. In: Handbook of Combinatorial Optimization. Springer, pp. 283–309.
URL https://doi.org/10.1007/978-1-4419-7997-1_13

Kotsireas, I. S., Koukouvinos, C., Seberry, J., 2009. Weighing matrices and string sorting. Annals of Combinatorics 13 (3), 305–313.
URL https://doi.org/10.1007/s00026-009-0027-8

Krone, S., Sarwate, D., 1984. Quadriphase sequences for spread-spectrum multiple-access communication. IEEE Transactions on Information Theory 30 (3), 520–529.
URL https://doi.org/10.1109/TIT.1984.1056913

Li, S., Chen, J., Zhang, L., Zhou, Y., 2008. Construction of quadri-phase complete complementary pairs applied in MIMO radar systems. In: ICSP 2008: 9th International Conference on Signal Processing. IEEE, pp. 2298–2301.
URL https://doi.org/10.1109/ICOSP.2008.4697608

Li, Y., Chu, W. B., 2005. More golay sequences. IEEE Transactions on Information Theory 51 (3), 1141–1145.
URL https://doi.org/10.1109/TIT.2004.842775

Liang, J. H., Poupart, P., Czarnecki, K., Ganesh, V., 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, pp. 119–135.
URL https://doi.org/10.1007/978-3-319-66263-3_8

Lomayev, A., Gagiev, Y., Maltsev, A., Kasher, A., Genossar, M., Cordeiro, C., Nov. 9 2017. Golay sequences for wireless networks. US Patent App. 15/280,635.
URL https://www.google.com/patents/US20170324461

Monagan, M. B., Geddes, K. O., Heal, K. M., Labahn, G., Vorkoetter, S. M., McCarron, J., DeMarco, P., 2005. Maple 10 Programming Guide. Maplesoft, Waterloo ON, Canada.

Nazarathy, M., Newton, S. A., Giffard, R., Moberly, D., Sischka, F., Trutna, W., Foster, S., 1989. Real-time long range complementary correlation optical time domain reflectometer. Journal of Lightwave Technology 7 (1), 24–38.
URL https://doi.org/10.1109/50.17729

Nowicki, A., Secomski, W., Litniewski, J., Trots, I., Lewin, P., 2003. On the application of signal compression using Golay's codes sequences in ultrasound diagnostic. Archives of Acoustics 28 (4).
URL http://acoustics.ippt.gov.pl/index.php/aa/article/view/460

Popović, B. M., 1991. Synthesis of power efficient multitone signals with flat amplitude spectrum. IEEE transactions on Communications 39 (7), 1031–1033.
URL https://doi.org/10.1109/26.87205

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

Seberry, J. R., Wysocki, B. J., Wysocki, T. A., 2002. On a use of Golay sequences for asynchronous DS CDMA applications. In: Advanced Signal Processing for Communication Systems. Springer, pp. 183–196.
URL https://doi.org/10.1007/0-306-47791-2_14

Sivaswamy, R., 1978. Multiphase complementary codes. IEEE Transactions on Information theory 24 (5), 546–552.
URL https://doi.org/10.1109/TIT.1978.1055936

Sun, W., Yuan, Y., 2006. Optimization theory and methods: nonlinear programming. Springer Science & Business Media.
URL https://doi.org/10.1007/b106451

Tseng, C. C., 1971. Signal multiplexing in surface-wave delay lines using orthogonal pairs of Golay's complementary sequences. IEEE Transactions on Sonics and Ultrasonics 18 (2), 103–107.
URL https://doi.org/10.1109/T-SU.1971.29599

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