New Results on Periodic Golay Pairs

Tyler Lumsden[1], Ilias Kotsireas[2], Curtis Bright[1]

[1]School of Computer Science, University of Windsor, Canada.
[2]CARGO Lab, Wilfrid Laurier University, Canada.

Contributing authors: lumsdent@uwindsor.ca; ikotsireas@wlu.ca; cbright@uwindsor.ca;

Abstract

In this paper, we provide algorithmic methods for conducting exhaustive searches for periodic Golay pairs. Our methods enumerate several lengths beyond the currently known state-of-the-art available searches: we conducted exhaustive searches for periodic Golay pairs of all lengths $\boldsymbol{v \leq 72}$ using our methods, while only lengths $\boldsymbol{v \leq 34}$ had previously been exhaustively enumerated. Our methods are applicable to periodic complementary sequences in general. We utilize sequence compression, a method of sequence generation derived in 2013 by Đoković and Kotsireas. We also introduce and implement a new method of "multi-level" compression, where sequences are uncompressed in several steps. This method allowed us to exhaustively search all lengths $\boldsymbol{v \leq 72}$ using less than 10 CPU years. For cases of complementary sequences where uncompression is not possible, we introduce some new methods of sequence generation inspired by the isomorph-free exhaustive generation algorithm of orderly generation. Finally, we pose a conjecture regarding the structure of periodic Golay pairs and prove it holds in many lengths, including all lengths $\boldsymbol{v < 100}$.

Keywords: Periodic Golay pairs, Compression, Orderly generation, Complementary sequences, Algorithms, Computer Science

MSC Classification: 11B83 , 05B20 , 94A55 , 68W30

Curtis Bright was supported by an NSERC Discovery Grant and Tyler Lumsden was supported by an NSERC undergraduate research assistantship.

1 Introduction

Introduced by Marcel Golay in 1949 [1], Golay pairs are used for their relevance in various communications applications, such as multi-slit spectrometry. Golay pairs consist of two $\{-1, +1\}$-sequences (A, B), each of length v, such that their respective aperiodic autocorrelations sum to zero for all nontrivial shifts. Periodic Golay pairs are a periodic analog of Golay pairs. In particular, periodic Golay pairs consist of two $\{-1, +1\}$-sequences (A, B), each of length v, such that their respective *periodic* autocorrelations sum to zero for all nontrivial shifts.

Periodic Golay pairs are also of independent mathematical interest; for example, periodic Golay pairs of length v can be used to construct Hadamard matrices of order $2v$ [2]. If a Golay pair of length v exists, then v is called a Golay number and similarly if a periodic Golay pair of length v exists, then v is called a periodic Golay number.

A Golay pair is also a periodic Golay pair, but the converse is not necessarily true, as there are periodic Golay numbers that are not Golay numbers. Currently, the only seven known periodic Golay numbers that are definitely not Golay numbers are 34, 50, 58, 68, 72, 74, and 82 [2]. In 1998, Đoković introduced a canonical form for the equivalence of Golay pairs [3]. A description of all Golay pairs for lengths $v \leq 100$ was reported by Borwein and Ferguson in 2003 [4], where five primitive Golay pairs were identified from which all Golay pairs of lengths $v \leq 100$ can be derived. In 1974, Turyn [5] described a method of constructing Golay pairs of length vm from existing pairs of lengths v and m. In 2008, Fiedler, Jedwab, and Parker [6] used Turyn's construction to demonstrate a framework for Golay pairs of length 2^v and also addressed the extra Golay pairs found in exhaustive searches previously conducted by Li and Chu in 2005 [7]. Fiedler, Jedwab, and Parker also created their own approach to exhaustive enumeration of Golay pairs in 2006 [8], where they consider Golay complementary sequences as projections of multi-dimensional Golay arrays. Turyn's construction was also slightly generalized to periodic Golay pairs by Đoković and Kotsireas in 2015 [9] by noticing that Turyn's multiplication formula for Golay pairs can also be used to construct periodic Golay pairs as well.

The length v of the two sequences making up a periodic Golay pair must be an even number that when doubled is a sum of two squares. This implies that there are no periodic Golay pairs for certain lengths such as 14, since 28 is not a sum of two squares. Currently, the smallest undecided periodic Golay number is 90 [10]. Periodic Golay pairs are well-studied, with a number of previous works generating periodic Golay pairs computationally. For example, in 2008, the existence of periodic Golay pairs in lengths $v \leq 50$ was settled by Kotsireas et al. [11]. In 2015, eight inequivalent pairs of length 72 were generated by Đoković and Kotsireas [9], and they also generated a pair of length 74 in 2014 [2]. Our methods not only recreate most of these results but also exhaustively enumerate the lengths, providing complete lists of periodic Golay pairs for lengths $v \leq 72$. Our results confirm and extend the previous known exhaustive enumerations of periodic Golay pairs of lengths $v \leq 32$ by Balonin and Đoković in 2015 [12] and lengths $v \leq 34$ by Crnković, Danilović, Egan, and Švob in 2022 [10].

Our search pipeline performs three sequential steps to exhaustively enumerate periodic Golay pairs (A, B) of length v: candidate generation, candidate matching, and equivalence filtering. First, we generate a list of candidate sequences corresponding to all possible A sequences, and a list of all possible B sequences (see Sections 3.1 to 3.3). Second, we describe an algorithm to efficiently match and test all combinations of (A, B) to construct our final list of periodic Golay pairs (see Section 3.4). Third, the last step is to filter our final list up to the equivalence, removing any pairs that are equivalent under the set of periodic Golay pair equivalence operations (see Section 2.1). This final step reduces the size of the list and provides an experimental verification of results between independent exhaustive searches. We provide several algorithms for equivalence filtering in Section 5.

Lengths $v \leq 50$ are possible to enumerate using just the generate-and-match method, but larger orders quickly become unfeasible. To work around this, we apply sequence compression, introduced by Đoković and Kotsireas [13], as a method of reducing the length of candidate sequences at the cost of increasing the alphabet size (see Section 2.2). This results in a vastly reduced search space and an improvement in computation time by orders of magnitude. The downside is that it requires an efficient uncompression method—we cover our uncompression algorithm in Section 4. The notable previous work of Crnković et al. [10] for enumerating lengths $v \leq 34$ also utilized sequence compression, however they noted that the enumeration for length 40 as out of reach with current methods. In order to go farther, we introduce a new method of "multi-level" sequence compression that dramatically improves cases where the length v is a product of many small primes such as 2 and 3. Additionally, we make use of the methodology of orderly generation, an algorithm used to generate objects in an isomorph-free way. This significantly improves the candidate generation step of our pipeline, and can also benefits cases which cannot utilize sequence compression. We provide precise descriptions of our algorithms throughout the paper, as well as benchmarks of our implementations on various lengths in Section 6.

As a consequence of our extended enumerations, we discover patterns exhibited by our exhaustive data for periodic Golay pairs of larger lengths. We propose a conjecture for constructing periodic Golay pairs for the lengths v in which periodic Golay pairs exist (see Section 7). We prove that the conjecture holds in all lengths $v < 100$ with the possible exception of $v = 50$, and we experimentally verify that the conjecture does hold for $v = 50$. The conjecture implies a formula exists describing all possible $(v/2)$-compressions of periodic Golay pairs. We also propose heuristics for the form of d-compressions of periodic Golay pairs for compression factors d other than $v/2$.

2 Background on Periodic Golay Pairs

Generally, *complementary sequences* are defined as a collection of sequences such that their periodic or aperiodic autocorrelation functions sum to a constant value at all shifts $s \neq 0$. A periodic Golay pair of even length v, denoted by $\mathrm{PG}(v)$, is a pair of two complementary sequences of length v, with $\{\pm 1\}$ entries, such that their periodic autocorrelation function coefficients sum to 0. Precisely, a periodic Golay pair (A, B) of length v is characterized by the two sequences

$$A = [a_0, \ldots, a_{v-1}], \text{ where } a_i \in \{\pm 1\} \text{ for } i = 0, \ldots, v-1, \text{ and}$$

$$B = [b_0, \dots, b_{v-1}], \text{ where } b_i \in \{\pm 1\} \text{ for } i = 0, \dots, v-1,$$

as well as the equation

$$\mathrm{PAF}(A, s) + \mathrm{PAF}(B, s) = 0, \text{ for } s = 1, \dots, v/2,$$

where s denotes the shift variable, and PAF denotes the *periodic autocorrelation function* defined by

$$\mathrm{PAF}(A, s) = \sum_{k=0}^{v-1} a_k a_{k+s}.$$

In this definition, the entries of A are taken to repeat periodically, i.e., $a_{v+i} = a_i$ for all $i \geq 0$. A consequence of the definition is that the following sum-of-squares Diophantine equation must have solutions for a periodic Golay pair of even length v to exist [2]. Precisely, we have the following theorem.

Theorem 1. *Suppose* (A, B) *is a* $\mathrm{PG}(v)$*. If* $a = a_0 + \dots + a_{v-1}$ *and* $b = b_0 + \dots + b_{v-1}$*, then*

$$a^2 + b^2 = 2v.$$

As a consequence, many lengths for periodic Golay pairs can be immediately ruled out as impossible. For example, there are only 24 lengths $v \leq 100$, namely 2, 4, 8, 10, 16, 18, 20, 26, 32, 34, 36, 40, 50, 52, 58, 64, 68, 72, 74, 80, 82, 90, 98, and 100, whose double can be written as a sum of two squares and therefore are the only integers which can be a periodic Golay number.

Because periodic Golay pairs have entries that have two possibilities, the total size of the search space for a given length v is at most $2^v \cdot 2^v = 4^v$. This complexity can be refined by considering only those pairs satisfying the sum-of-squares equation. A revised search space complexity is calculated as

$$\binom{v}{(v+a)/2} \times \binom{v}{(v+b)/2},$$

where a and b are solutions to the Diophantine equation in Theorem 1, since a length-v vector with rowsum a and $\{\pm 1\}$ entries contains $(v+a)/2$ ones. If there are multiple ways of writing $2v$ as a sum of two squares then the size of the search space needs to account for each decomposition, i.e., the entire space will be $\sum_{a,b \geq 0: a^2+b^2=2v} \binom{v}{(v+a)/2}\binom{v}{(v+b)/2}$.

Using brute force to enumerate this search space is numerically infeasible, and if we were to instead search by caching the PAF values of all 2^v sequences to avoid recomputation, we would quickly run out of storage space. Committing to a search of size 4^v is computationally unfeasible, so we require the introduction of some strong filtering methods to reduce the running time. One such filtering method is based on a well-known connection between the *discrete Fourier transform* (DFT) of a sequence and its PAF values. The DFT of a sequence $A = [a_0, \dots, a_{v-1}]$ at shift s is defined as

$$\mathrm{DFT}(A, s) \coloneqq \sum_{k=0}^{v-1} a_k \omega^{ks},$$

where $\omega = e^{2\pi \mathbf{i}/v}$ is a primitive vth root of unity. The *power spectral density* (PSD) of a sequence is defined as the squared magnitude of its DFT values, i.e.,

$$\mathrm{PSD}(A, s) \coloneqq |\mathrm{DFT}(A, s)|^2.$$

When $\mathrm{PSD}(A)$ is written without a shift variable, we mean the vector consisting of all PSD values, i.e., $[\mathrm{PSD}(A, 0), \dots, \mathrm{PSD}(A, v-1)]$ (and similarly for $\mathrm{PAF}(A)$).

Fletcher, Gysin, and Seberry [14, Thm. 2] show that the PAF values of two sequences sum to a constant if and only if their PSD-values sum to a constant. Đoković and Kotsireas [13, Thm. 2] prove the following theorem providing a formula relating the PSD and PAF constants of complementary sequences.

Theorem 2. *Let $A_1, \dots, A_t$ be complex sequences of length v. These sequences are complementary, i.e.,*

$$\sum_{i=1}^{t} \mathrm{PAF}(A_i) = [\alpha_0, \alpha, \dots, \alpha]$$

if and only if

$$\sum_{i=1}^{t} \mathrm{PSD}(A_i) = [\beta_0, \beta, \dots, \beta],$$

where the PAF *constants α_0 and α and the* PSD*-constants β_0 and β are related by*

$$\beta_0 = \alpha_0 + (v-1)\alpha, \quad \textit{and} \quad \beta = \alpha_0 - \alpha.$$

For the case of periodic Golay pairs specifically, where the PAF constants sum to 0, this reduces to the following.

Corollary 1. *For any two $\{\pm 1\}$-sequences (A, B) of even length v,*

$$\mathrm{PAF}(A, s) + \mathrm{PAF}(B, s) = 0, \ \textit{for } s = 1, \dots, v/2,$$

if and only if

$$\mathrm{PSD}(A, s) + \mathrm{PSD}(B, s) = 2v, \ \textit{for } s = 0, \dots, v/2.$$

This enables us to leverage the $O(v \log v)$ complexity of the fast Fourier transform rather than the $O(v^2)$ complexity of computing the vector of PAF values naively. Furthermore, the PSD values of a sequence are nonnegative by definition. This allows us to use the following filtering bound on any individual binary sequence A.

Corollary 2. *Suppose A is a member of a periodic Golay pair of length v. Then*

$$\mathrm{PSD}(A, s) \leq 2v.$$

In practice, this filters a large portion of our search space of 2^v sequences, so storing all sequences passing this filtering test is feasible. In certain cases, additional restrictions on permissable PSD values of complementary sequences can be derived. For example, Kotsireas and Koutschan [15] prove that if (A, B) is a Legendre pair (a complementary sequence pair with a PAF constant of $\alpha = -2$) then $\mathrm{PSD}(A, v/3)$ and $\mathrm{PSD}(B, v/3)$ are perfect integer squares when $v/3$ is an integer. The following lemma is an analogue of their result for periodic Golay pairs.

Lemma 1. *If A is a sequence in a* $\mathrm{PG}(v)$*, then* $\mathrm{PSD}(A, v/2)$ *is a perfect integer square, and* $\mathrm{PSD}(A, v/4)$ *is an integer when $v/4$ is an integer.*

Proof. Consider a primitive vth root of unity, $\omega = \exp(2\pi\mathbf{i}/v)$, and note that $\omega^{v/2} = -1$. Thus, $\mathrm{PSD}(A, v/2) = \left|\sum_{k=0}^{v-1}(-1)^k a_k\right|^2 = \left(\sum_{k=0}^{v/2-1}(a_{2k} - a_{2k+1})\right)^2$, the square of an integer.

Note that $\omega^{v/4} = \mathbf{i}$ and $\mathrm{PSD}(A, v/4) = \mathrm{DFT}(A, v/4) \cdot \overline{\mathrm{DFT}(A, v/4)}$ where the overline denotes complex conjugation. Since

$$\mathrm{DFT}(A, v/4) = \sum_{k=0}^{v/4-1} \left(a_{4k} - a_{4k+2} + \mathbf{i}(a_{4k+1} - a_{4k+3})\right) = A_0 + \mathbf{i}A_1$$

where $A_0 \coloneqq \sum_{k=0}^{v/4-1}(a_{4k} - a_{4k+2})$ and $A_1 \coloneqq \sum_{k=0}^{v/4-1}(a_{4k+1} - a_{4k+3})$ are integers, we have $\mathrm{PSD}(A, v/4) = (A_0 + \mathbf{i}A_1)(A_0 - \mathbf{i}A_1) = A_0^2 + A_1^2$ is an integer.

□

Since $\mathrm{PSD}(A, s)+\mathrm{PSD}(B, s) = 2v$ for all s including $s = v/2$, the values $\mathrm{PSD}(A, v/2)$ and $\mathrm{PSD}(B, v/2)$ give a sum-of-squares decomposition of $2v$—and all ways of writing $2v$ as a sum of two integer squares can be determined in advance. For example, for $v = 68$, up to order and signs the only sum-of-squares decomposition of $2v$ is $6^2 + 10^2$. Thus, if (A, B) is a PG(68), then $\mathrm{PSD}(A, 34) = 6^2$ and $\mathrm{PSD}(B, 34) = 10^2$ or vice versa.

Despite the large number of sequences that can be filtered by these methods, a search space of size 2^v is infeasible for lengths $v > 40$. Thus, a key component of improving the search for a $\mathrm{PG}(v)$ is to reduce the search space even farther. We now introduce two tools that allowed us to reduce the search space size of our searches.

2.1 Equivalence

There are several equivalence operations that can be defined on periodic Golay pairs. If a pair of sequences (A, B) is a $\mathrm{PG}(v)$, then the following operations can be applied to generate another $\mathrm{PG}(v)$ which is said to be *equivalent* to the original pair. In the following, v is the length of each sequence in the original sequence pair (A, B), and i is bounded as $0 \leq i < v$.

1. [Swap] (B, A), swap sequences A and B.
2. [Shift A] $([a_{i+1 \bmod v}], B)$, rotate the elements of sequence A by one shift.
3. [Reverse A] $([a_{(v-1)-i}], B)$, reverse the elements of sequence A.
4. [Decimation] $([a_{ki \bmod v}], [b_{ki \bmod v}])$, for all $k < v$ with k coprime to v, replace a_i with $a_{ki \bmod v}$ and b_i with $b_{ki \bmod v}$.
5. [Alternating Negation] $([(-1)^i a_i], [(-1)^i b_i])$, negate every odd indexed element of sequence A and B.

2.2 Compression

Considering the exponential 2^v size of the search space for sequences in a $\mathrm{PG}(v)$, it is useful to reduce the exponent. Fortunately, one can compress a sequence to a smaller length while maintaining its complementarity [13]. Specifically, we can create a new sequence with length $d = v/m$, where v is the length of the original sequence $A = [a_0, \dots, a_{v-1}]$, and m is any divisor of v. We call this new sequence $A^{(m)} = [a_0^{(m)}, \dots, a_{d-1}^{(m)}]$ the m-compression of A, with entries that are defined by $a_i^{(m)} = a_i + a_{i+d} + \cdots + a_{i+(m-1)d}$. This compression method reduces the size of the search space to $(m+1)^d$. For example, if we compress a length 10 sequence over a $\{\pm 1\}$ alphabet (i.e., a search space of size $2^{10} = 1024$) by a factor of 2, the compressed search is over sequences of length 5 and a $\{\pm 2, 0\}$ alphabet (i.e., a search space of size $3^5 = 243$). We use the fact that compressions of a $\mathrm{PG}(v)$ will also be complementary sequences (over a larger alphabet) in order to search for smaller-length sequences in a reduced search space. After filtering as many smaller-length sequences as possible, we apply an uncompression method to recover periodic Golay pairs of length v.

3 Candidate Generation

Candidate generation is a necessary step in constructing our compressed pairs and in practice is the hardest step of our pipeline due to the aforementioned exponential search space. Thus, in creating algorithms for generating individual candidate sequences, we strive to consistently reduce the search space by as large a factor as possible to mediate this problem.

3.1 Integer Partitions

There are many characteristics that periodic Golay pairs exhibit that can be used to limit the search space. For example, as described in Theorem 1, the sum of all elements in a given sequence must be a particular value for the sequence to be part of a periodic Golay pair. Since it is simple to enumerate all decompositions of $2v$ as a sum of squares, with this information we can limit the search space to only sequences whose rowsums correspond to either a or b where $a^2 + b^2 = 2v$.

To generate only those sequences which sum to a or b, the process is straightforward for the case where the given alphabet is $\{-1, +1\}$. An integer can be partitioned into a sequence of v elements from an alphabet of $\{-1, +1\}$ in exactly one way (up to ordering). Thus, we can directly count the number of -1s and $+1$s that must occur; there are exactly $(a+v)/2$ positives and $(a-v)/2$ negatives in a $\{\pm 1\}$-sequence of length v with rowsum a. However, the problem is more complicated when considering an arbitrary alphabet, as there may be more than one combination of the alphabet that sum to a. After calculating all partitions of a or b into sums of terms from the necessary alphabet (e.g., for 2-compression the alphabet is $\{0, \pm 2\}$), we only consider sequences that are permutations of our partitions, and this gives us our search space.

3.2 Orderly Generation

Introduced independently by Faradžev [16] and Read [17] in 1978, "orderly" generation is a form of isomorph rejection that filters equivalent nodes in a search tree. Other methods of isomorph rejection include the "canonical augmentation" method of McKay [18]. By considering just equivalent shifts of a sequence, orderly generation on sequences of length v can reduce the search space size by approximately a factor of v.

Length	Compression Factor	Integer Partition	Orderly	Hybrid
26	N/A	4s	1s	0s
32	N/A	264s	76s	34s
34	N/A	1064s	285s	120s
40	2	118s	85s	38s

Table 1: Running time comparison of candidate generation methods. Times are given in seconds.

We can theoretically reduce our search space to only inequivalent sequences by limiting our search to only a canonical representative of each equivalence class in our search space. For periodic Golay pairs, we take the lexicographically minimal sequence of each equivalence class as our canonical representative.

Example 1. *Consider the sequence* $A = [1, *, *, *]$*, where* $* \in \{-1, 1\}$ *denotes undefined entries. If any of the undefined entries are* -1*, then* A *cannot be a lexicographically minimal representative of its equivalence class, since otherwise shifting its entries would produce an equivalent sequence with a first entry of* -1*.*

This implies that we can always reduce our search space from 2^n to $2^{n-1} + 1$, essentially half of the original size. But this only removes one branch of the search tree. To further remove entire sections of our search space during generation, we recursively filter out partially defined subsequences (rather than waiting to filter full sequences of length v). To do this, we generate our sequences using backtracking, starting from the 0th element, and working down to the $(v - 1)$th element. If it can be shown that any subsequence cannot be lex minimal no matter the value of its remaining elements, then we throw away that branch of the tree and skip to the next. In practice, this method reduces our search tree to its canonical representatives. Still, it introduces some computational overhead in calculating whether a subsequence can be lexicographically minimal, typically by generating its equivalence class and determining if a lexicographically smaller subsequence exists. Furthermore, the number of sequences in a removed branch may be inconsequential. In practice, we used orderly generation on partially defined sequences with up to $v/2$ defined entries, as past that point the number of sequences removed is inconsequential.

3.3 Hybrid Generation

As shown in Table 1, the orderly method outpaces the integer partition method described in Section 3.1. However, there is inefficiency when considering that the further the subsequences are completed, the smaller the removed branches become. To optimize the stretches of computation where the lower end of a sequence is being completed, we devised a hybrid approach. Our incomplete sequences are filtered to the length $v/2$, but the remaining length is instead constructed using the integer partitions method to consider only the sequences that can sum to a or b. Specifically, our sequences use orderly generation for subsequences up to the length $v/2$. To complete our subsequences, we want to ignore those sequences that cannot sum to a or b, so we compute the current sum of elements of each of our subsequences, and only fill the rest of our sequence with integer partitions that can complete the sum to a or b. This method takes advantage of both the orderly and integer partitions methods and combines them to complement each other. As demonstrated in Table 1, this “hybrid” method outpaces both individual methods.

3.4 Matching

By this point in our pipeline, we are left with a list of A sequences and a list of B sequences which all pass the PSD filtering and Diophantine equation tests. To complete our pipeline, we will use these lists to “match” all pairs that satisfy the PSD equation in Corollary 1. This matching step gives us an exhaustive list of periodic Golay pairs (or, if compression is used, a list of all compressions of periodic Golay pairs). The naive solution would be to try all possible combinations of list A and list B, but this results in a time complexity quadratic in the list lengths. We instead use the matching method originally described by Kotsireas, Koukouvinos, and Seberry [19], and later used by Bright, Kotsireas, and Ganesh to exhaustively generate Williamson sequences [20]. This method effectively reduces our matching problem to a string sorting problem. From Corollary 1 we have

$$\mathrm{PSD}(A,s) = 2v - \mathrm{PSD}(B,s), \text{ for } s = 0,\dots,v/2.$$

We sort the A sequences so that the values of $\mathrm{PSD}(A,s)$ occur in lexicographic order and sort the B sequences so the values of $2v - \mathrm{PSD}(B,s)$ occur in lexicographic order. Then a “match”, a sequence with $\mathrm{PSD}(A,s) = 2v - \mathrm{PSD}(B,s)$, corresponds to a periodic Golay pair. In practice, we round the PSD values to the nearest integer. This could introduce spurious solutions, but when a match occurs it can easily be checked that it is indeed a periodic Golay pair. The matching process requires only one scan through both lists, making it a linear procedure once the lists have been sorted. The worst case of this algorithm occurs when every sequence in list A is compatible with every sequence in list B. This case is necessarily quadratic, since the output size is quadratic. However, this case does not occur in practice for our exhaustive searches. The precise implementation of this algorithm is given in Algorithm 1.

4 Uncompression

As described in Section 2.2, the compression method efficiently reduces our initial search space but requires an extra uncompression step. Uncompression requires constructing all possible sequences of length v that compress to a given sequence of length $d = v/m$. This can be an extremely arduous task given a larger compression factor of m, so the choice of divisor is very important for efficient generation. We propose a method of optimizing some cases of large-divisor compression to significantly improve cases where the length of a sequence has many small divisors.

4.1 Multi-level Compression

Generating compressed sequences can be difficult since it must be accomplished using a candidate generation step, which can be slow. Thus, we implemented a new method

Algorithm 1 MATCH(v, Ω_A, Ω_B), a sequence matching algorithm.

Input: v, the length of the sequences to match.
Ω_A, a list of sequences which satisfy the PSD filtering test with constant rowsums.
Ω_B, a list of sequences which satisfy the PSD filtering test with constant rowsums.
Output: Ω, the list of compatible periodic Golay pairs found from Ω_A and Ω_B.

```
1. [Sorting]
  // Below $\mathrm{PSD}(A)$ and $2v - \mathrm{PSD}(B)$ are defined as the sequences constructed from
  the values $\mathrm{PSD}(A, s)$ and $2v - \mathrm{PSD}(B, s)$, respectively, for $s = 0, \ldots, v/2$.
  Sort list $\Omega_A$ lexicographically by the values of $\mathrm{PSD}(A)$
  Sort list $\Omega_B$ lexicographically by the values of $2v - \mathrm{PSD}(B)$

2. [Initialization]
  $A :=$ the first sequence in $\Omega_A$
  $B :=$ the first sequence in $\Omega_B$
  $\Omega := \emptyset$

3. [Matching]
  while $A$ is not the end of $\Omega_A$ and $B$ is not the end of $\Omega_B$ do
      if $\mathrm{PSD}(A) = 2v - \mathrm{PSD}(B)$ then    // Match found
          // Find all sequences with the same PSDs as $A$ and $B$
          $\mathrm{temp}_A := \mathrm{PSD}(A)$
          $\mathrm{temp}_B := 2v - \mathrm{PSD}(B)$
          $\Gamma_A := \emptyset$
          $\Gamma_B := \emptyset$
          while $A$ is not the end of $\Omega_A$ do
              if $\mathrm{PSD}(A) = \mathrm{temp}_A$ then
                  $\Gamma_A := \Gamma_A \cup A$
                  $A :=$ the next sequence in $\Omega_A$
              end if
          end while
          while $B$ is not the end of $\Omega_B$ do
              if $\mathrm{PSD}(B) = \mathrm{temp}_B$ then
                  $\Gamma_B := \Gamma_B \cup A$
                  $B :=$ the next sequence in $\Omega_B$
              end if
          end while
          $\Omega := \Omega \cup (\Gamma_A \times \Gamma_B)$    // Add all pairs of combinations from $\Gamma_A$ and $\Gamma_B$
      end if
      if $\mathrm{PSD}(A) < 2v - \mathrm{PSD}(B)$ then    // Lexicographical comparison
          $A :=$ the next sequence in $\Omega_A$
      else   // Thus $\mathrm{PSD}(A) \geq 2v - \mathrm{PSD}(B)$
          $B :=$ the next sequence in $\Omega_B$
      end if
  end while
```

where we generate candidate d-compressed pairs by uncompressing candidate $(d \cdot e)$-compressed pairs for some uncompression factor $e > 1$. Essentially, we utilize the observation that uncompression is significantly faster with smaller divisors and the fact that compression is also possible with several small divisors, rather than one large divisor.

Theorem 3. *Suppose a sequence A has length v. For all positive integers d, e where de divides v, the e-compression of the d-compression of A, $(A^{(d)})^{(e)}$, equals $A^{(de)}$, the $(d \cdot e)$-compression of A.*

Proof. Let $A^{(d)}$ denote the d-compression of A and with entries given by $a_k^{(d)} = \sum_{i=0}^{d-1} a_{k+iv/d}$ for $k = 0, \ldots, v/d - 1$. The kth entry of the e-compression of the d-compression of A is

$$(A^{(d)})_k^{(e)} = \sum_{i=0}^{e-1} a_{k+i(v/d)/e}^{(d)} = \sum_{i=0}^{e-1}\sum_{j=0}^{d-1} a_{k+(i+ej)v/(de)} = \sum_{i=0}^{de-1} a_{k+iv/(de)} = a_k^{(de)}$$

where the third equality rewrites the double sum as a single sum using the identity $\bigcup_{i=0}^{e-1}\bigcup_{j=0}^{d-1}\{i+ej\} = \{0, \ldots, de-1\}$. □

This allows us to perform uncompression by a large compression factor by splitting it into several rounds of uncompression by intermediate factors.

4.2 Description of Uncompression

An uncompression algorithm must generate a list of uncompressions with the requirement that if a sequence compresses to a given candidate compression, then that sequence must be present in the list of uncompressions. In other words, we need to compute the preimage of a given sequence under compression. A single entry in a compression involves a fixed-length sum, and this implies that to revert this process and uncompress a sequence, we need to calculate all possible fixed-length sums for each element and construct all possible combinations of these for our final list of sequences.

Example 2. *If we are to uncompress the sequence $A = [0, 2]$, with compression factor 2, the uncompressed list must be $[-1, 1, 1, 1]$ or $[1, 1, -1, 1]$, by observing that $0 = -1 + 1$ and $2 = 1 + 1$ are the only possible length-2 $\{\pm 1\}$-sums of 0 and 2.*

Since all possible permutations of these fixed-length partitions must be considered, we utilize backtracking to implement the algorithm. Utilizing recursion to implement backtracking, the detailed algorithm is described in Algorithms 2 and 3.

5 Equivalence Filtering

It is convenient to consider only the pairs that are inequivalent to one another. This is useful in optimizing various steps of our algorithms, as we can allow ourselves to only process those inequivalent pairs, reducing our computational costs by a factor up to $32v^2\varphi(v)$ for length v [10], where $\varphi(v)$ denotes Euler's totient function. Furthermore, it is difficult to verify the correctness of the results of different methods of search if the results are not standardized in some form.

Algorithm 2 UNCOMPRESS(ω, e, d), uncompress a sequence ω by a factor of e (a wrapper for Algorithm 3).

```
Input: $\omega$, the original compressed sequence with length $v$.
$d$, the compression factor of the sequences which $e$-compress to $\omega$.
$e$, the uncompression factor to be applied to $\omega$.
Output: The set of all length $ve$ sequences that $e$-compress to $\omega$.

1. [Initialization]
   $\gamma :=$ A sequence of length $m := ve$ with all undefined elements.
   return UNCOMPRESS($\gamma, \omega, 0, d, e$)    // All sequences that $e$-compress to $\omega$
```

Algorithm 3 UNCOMPRESS(γ, ω, i, d, e), uncompression algorithm with recursive backtracking.

```
Input: $\omega$, the original compressed sequence of length $v$.
$\gamma$, a sequence of length $m := ve$ with $ie$ defined entries and $(v-i)e$ undefined entries.
$i$, the index of $\omega$ at which to start uncompressing.
$d$, the compression factor of the sequences which $e$-compress to $\omega$.
$e$, the uncompression factor to be applied to $\omega$.
Output: $\Omega$, the set of all length $m$ sequences whose entries match the defined entries
of $\gamma$ and whose other entries $e$-compress to $[\omega_i, \ldots, \omega_{v-1}]$.

1. [Initialization]
   $\Omega :=$ an empty set of length $m$ sequences.
   // We are to construct permutations that account for all possibilities of sequences
   that could compress to $\omega_i$. These permutations consist of elements in the set $A :=$
   $\{ x \in \mathbb{Z} : -d \leq x \leq d \text{ and } x \equiv d \pmod 2 \}$
   $\Omega_{\text{subseq}} :=$ the set of all length-$e$ integer permutations of the $i$th element of $\omega$, $\omega_i$.

2. [Base case, nothing to uncompress]
   if $i == v$ then
       return $\{\gamma\}$
   end if

3. [Recursive step, uncompress all possible permutations of $\omega_i$ into $\gamma$]
   // Loop to construct subsequences using all possible length-$e$ integer permutations
   of $\omega_i$
   for $\delta$ in $\Omega_{\text{subseq}}$ do
       // Assign the elements of the current permutation into $\gamma$
       for $j$ in $\{0, \ldots, e-1\}$ do
           $\gamma_{i+jv} := \delta_j$
       end for
       // Recursively call the algorithm to construct all combinations
       $\Omega := \Omega \cup$ UNCOMPRESS($\gamma, \omega, i+1, d, e$)
   end for
```

The general filtering algorithm described by Bright et al. [21, Alg. 3.1] for the postprocessing of complex Golay pairs (Golay pairs over the alphabet $\{\pm 1, \pm \mathbf{i}\}$) accomplishes this task. This algorithm was used for filtering complex Golay pairs, and for periodic Golay pairs it quickly reaches unfeasibility for larger lengths due to its computation and memory requirements. The method filters sequences by iterating through the list of sequences, computing and caching the equivalence class of every sequence that has not previously been seen in any previous equivalence class. If a sequence has been found in a previous equivalence class, we remove it. This method essentially requires storing the set of all periodic Golay pairs in memory, which is unfeasible in our case. We provide modifications that address these problems. In what follows, Ω will refer to a set of periodic Golay pairs, and ω will refer to a pair of sequences (typically implemented by two arrays).

5.1 Equivalence Class Generation

The most straightforward algorithm to generate the equivalence class of a given pair is described in Algorithm 4, where we apply our equivalence operations to our given base pair and continuously apply the equivalence operations to all newly generated pairs until no new pairs are found in an iteration. Our equivalence filtering methods require the generation of equivalence classes of all given pairs, so an efficient method of equivalence class generation is necessary.

Algorithm 4 EQUIV(ω_{base}), the first algorithm to generate the equivalence class of a pair ω_{base}.

```
Input: $\omega_{\text{base}}$, the original sequence pair of length $v$.
Output: $\Omega_{\text{class}}$, the equivalence class of $\omega_{\text{base}}$.
// Note that $\Omega_{\text{class}}$ changes each iteration
$\Omega_{\text{class}} \coloneqq \{\omega_{\text{base}}\}$
for $\omega \in \Omega_{\text{class}}$ do
    // $e_1, \ldots, e_n$ is a list of generators of the PG($v$) symmetry group
    $\Omega_{\text{class}} \coloneqq \Omega_{\text{class}} \cup \{e_1(\omega), \ldots, e_n(\omega)\}$
end for
```

Algorithm 4 is far too slow for larger lengths v, where an equivalence class has a size of up to $32v^2\varphi(v)$ [10] because each equivalent sequence is generated several times. For example, if the equivalence operations $e_1, \ldots, e_n$ are performed on a sequence ω, it is evident that ω will be redundantly generated again from the set obtained in $\{e_1(\omega), \ldots, e_n(\omega)\}$ directly in the next iteration. Thus, we develop a faster algorithm to generate equivalence classes.

Let ω be our original sequence, and let Ω_{sym} be the group of symmetries generated from applying Algorithm 4 on ω_{sym}, where

$$\omega_{\text{sym}} = ([1, 2, \ldots, v], [v+1, v+2, \ldots, 2v]).$$

Essentially, we construct a sequence ω_{sym} with elements consisting of $2v$ variables, with each variable $\{1, \ldots, 2v\}$ representing a distinct index in a pair of length-v sequences. Note ω_{sym} is not a $\mathrm{PG}(v)$ itself; it is a symbolic representation of the identity equivalence operation where the entry at each position gets mapped to itself. In this way, the set Ω_{sym} will have size $32v^2\varphi(v)$ and encode each symmetry in the symmetry group of periodic Golay pairs of length v. Algorithm 5 describes the procedure for generating an equivalence class from the $\mathrm{PG}(v)$ symmetry group. For each symmetry ω_{sym} in Ω_{sym}, we create an equivalent sequence ω_{e} from ω.

Algorithm 5 $\textsc{Equiv}(\omega_{\text{base}})$, an alternate algorithm to generate the equivalence class of a pair ω_{base}.

```
Input: $\omega_{\text{base}}$, the original sequence pair of length $v$.
Output: $\Omega_{\text{class}}$, the equivalence class of $\omega_{\text{base}}$.
$\Omega_{\text{class}} \coloneqq \{\omega_{\text{base}}\}$
// $\Omega_{\text{sym}}$ is the PG($v$) symmetry group
for $\omega_{\text{sym}} \in \Omega_{\text{sym}}$ do
    $\omega_{\text{e}} \coloneqq \textsc{Apply}(\omega_{\text{sym}}, \omega)$, the symmetry $\omega_{\text{sym}}$ applied to $\omega$
    $\Omega_{\text{class}} \coloneqq \Omega_{\text{class}} \cup \{\omega_{\text{e}}\}$
end for
```

To apply a symmetry ω_{sym} to a sequence ω, we effectively rearrange the indexes of our base pair from the sequences in our symmetry group. Using our particular encoding of the $\mathrm{PG}(v)$ symmetry group, symmetries are applied using the following procedure:

1. If the ith element of ω_{sym} is k, then set it to the $(k-1)$th element of ω.
2. If the ith element of ω_{sym} is negative, then negate the $(k-1)$th element of ω.
3. If the elements of sequence A of ω_{sym} refer to indexes larger than v, then swap the sequences.

We repeat this procedure for each sequence in the symmetry group of periodic Golay pairs, thus directly generating our equivalence class from the symmetry group. This method avoids the repetition where an equivalent pair is regenerated several times; the method only needs a single iteration over the $32v^2\varphi(v)$ elements of Ω_{sym}.

5.2 List filtering

The output of an equivalence filtering algorithm is a list containing exactly one representative of each equivalence class present in the original list. The naive method requires storing the equivalence class of every pair in our original list, which is not feasible for lengths $v > 32$. To filter equivalent pairs from our list of generated pairs, we loop through our list, storing the equivalence class of our current pair, and removing all other pairs from the list that are in this pair's equivalence class. Our modified algorithm avoids the overbearing memory costs of the original by only storing one equivalence class at a time, rather than caching the entire set of all equivalence classes throughout the computation. Note that this modified algorithm will have a small but noticeable slowdown due to the added set subtraction step.

Algorithm 6 Filter(Ω_{pairs}), an equivalence filtering algorithm for a set of pairs Ω_{pairs}.

```
Input: $\Omega_{\text{pairs}}$, the original list of periodic Golay pairs.
Output: $\Omega_{\text{inequiv}}$, a set containing exactly one representative of each equivalence class present in $\Omega_{\text{pairs}}$.
$\Omega_{\text{inequiv}} \coloneqq \Omega_{\text{pairs}}$
for $\omega \in \Omega_{\text{inequiv}}$ do
    // Generate the equivalence class of $\omega$, removing $\omega$ itself
    $\Omega_{\text{class}} \coloneqq \textsc{Equiv}(\omega) \setminus \{\omega\}$
    // Remove all sequences equivalent to $\omega$ from $\Omega_{\text{inequiv}}$, excluding $\omega$
    $\Omega_{\text{inequiv}} \coloneqq \Omega_{\text{inequiv}} \setminus \Omega_{\text{class}}$
end for
```

Algorithm 6 filters a list by iterating through the original list from its beginning to its end. On each iteration, the current sequence pair ω is used as a representative of an equivalence class and the algorithm removes all pairs equivalent to ω from the list. It follows that we will be left with exactly one representative of each equivalence class. This algorithm requires the generation of each equivalence class exactly once, but it still isn't ideal in all cases due to the amount of memory used by the set operations. This makes the algorithm difficult to parallelize, and thus unfeasible for larger computations.

Algorithm 7 Filter(Ω_{pairs}), a parallelizable equivalence filtering algorithm for a set of pairs Ω_{pairs}.

```
Input: $\Omega_{\text{pairs}}$, the original list of periodic Golay pairs.
Output: $\Omega_{\text{inequiv}}$, a set containing exactly one representative of each equivalence class present in $\Omega_{\text{pairs}}$.
$\Omega_{\text{inequiv}} \coloneqq \emptyset$
for $\omega \in \Omega_{\text{pairs}}$ do
    // Generate the equivalence class of $\omega$
    $\Omega_{\text{class}} \coloneqq \textsc{Equiv}(\omega)$
    // Take the lexicographically minimal sequence from $\Omega_{\text{class}}$ as our representative
    $\Omega_{\text{inequiv}} \coloneqq \Omega_{\text{inequiv}} \cup \{\min(\Omega_{\text{class}})\}$
end for
```

Instead, we modify Algorithm 6 to generate the equivalence class of all sequences in our original list and take exactly one canonical representative from each to store in our final list. Algorithm 7 converts each pair in the original list to its lexicographically minimal form. Thus, if two pairs are equivalent, they get the same minimal canonical representative. We are left with a set containing a canonical representative of each equivalence class. This algorithm is better for larger lists because it avoids the complexity of subtracting two sets. The generation of each equivalence class can be done independently, so this algorithm can be trivially parallelized. The only weakness of this method compared to the first is that if two pairs are equivalent, both of their

Order	Candidate Generation	Uncompression	Total Time	Compression Factor
20	0s	N/A	0.0d	N/A
26	1s	N/A	0.0d	N/A
32	34s	N/A	0.0d	N/A
34	9s	4s	0.0d	2
40	38s	211s	0.0d	2
50	121621s	3264s	1.4d	2
52	58s	64810s	0.8d	$4 \hookrightarrow 2$
58	1555206s	180172s	20.1d	2
64	5s	126187201s	1460.5d	$8 \hookrightarrow 4 \hookrightarrow 2$
68	9292s	315361452s	3650.1d	$4 \hookrightarrow 2$
72	20s	175812s	2.0d	$8 \hookrightarrow 4 \hookrightarrow 2$

Table 2: Timings recorded for the exhaustive searches of lengths $v \leq 72$. Times for candidate generation and uncompression are given in seconds, and the total time used is given in days.

equivalence classes will be generated. This problem can be remedied by filtering "partially" up to only a few select equivalences, which can be performed very quickly, and then fully filtering this reduced list. Algorithm 7 was necessary instead of Algorithm 6 for computing the cases where our original lists were too large to compute without parallelization—in particular, the lengths 64 and 68. With these modifications, and by selecting our equivalences strategically, candidate compression sequences can be postprocessed very quickly in comparison to their generation time. Furthermore, the exhaustive enumeration of periodic Golay pairs up to all equivalences is made feasible for all lengths $v \leq 72$.

6 Results

All of our algorithms are implemented in C++, and we make use of the FFTW (Fastest Fourier Transform in the West) C library [22] to quickly calculate the PSD values. Our results are archived online,[1] and to reproduce our results we provide an online repository containing our code as well as instructions for installing and executing our code.[2] Our computations were performed on the Compute Canada [23] cluster Cedar. The computations were all performed using no more than 4 GiB of memory and were run on Intel CPUs of type Broadwell, Cascade Lake, or Skylake.

The timings of our algorithms in practice are provided in Tables 2 and 3. Table 3 also provides the number of equivalence classes found by our searches in each length. The matching step is omitted due to the inconsequential computation time relative to the rest of the search. Our computations exhaustively searched all lengths $v \leq 72$. All lengths 52 and smaller are easily reproducible, as Table 2 indicates that they require less than a day of computation time using the multi-level compression method.

The length 58 search could not be done using multi-level compression, as the prime factorization of 58 is 2×29, meaning only compression by a factor of 2 or 29 is possible. For $v = 58$, the generation of its candidate 2-compressions took ≈ 18 days,

[1] Archived results available at https://zenodo.org/records/12792345

[2] Code available at https://github.com/tylerlumsden/GolayPair

Length	Equivalence Classes	Filtering Time	Filtering Time
2	1	0s	0.0d
4	1	0s	0.0d
8	2	0s	0.0d
10	1	0s	0.0d
16	11	1s	0.0d
20	34	7s	0.0d
26	53	35s	0.0d
32	838	989s	0.0d
34	373	655s	0.0d
40	9281	14732s	0.2d
50	8753	22427s	0.3d
52	14354	194653s	2.2d
58	13386	313832s	3.6d
64	1112383	518413046s	6000.1d
68	90240	18041532s	208.8d
72	1753	41400s	0.5d

Table 3: Counts for the number of equivalence classes found in each search and filtering time given in seconds and days.

whereas the 2-compression generation in length 64 took only 12 hours, showing the extreme improvement in efficiency achieved by using multi-level compression. The uncompression step for $v = 58$ took about 2 days to compute.

The computations for length 64 were done in parallel, with the final uncompression step taking roughly 4 CPU years. The length 64 search was completed by first generating all 4-compressions of length $64/4 = 16$, then uncompressing those to an intermediate length resulting in 2-compressions of length 32. These resulted in roughly ≈ 160 million candidate 2-compressed pairs. The final step was to uncompress these 2-compressions, which required 4 CPU years, as each uncompression took less than one second. Every step before the final 2-uncompression took only 12 hours.

The exhaustive search for length 68 required 9292 seconds of candidate generation of 4-compressions of length $68/4 = 17$, and 6.25 CPU days to uncompress these to 2-compressed pairs. Afterward, the final uncompression step took 10 CPU years.

As noted in Table 3, the equivalence class count for $v = 72$ is significantly smaller than the preceding lengths. This fact contributed to the short computation time in length 72, as in comparison the length 64 search which had ≈ 160 million candidate 2-compression pairs, the length 72 search had only 64,601 candidate 2-compression pairs. The likely reason for the small equivalence class counts is because its direct divisors, 18 and 36, are not periodic Golay numbers.

Our algorithms reached an exhaustive classification for lengths $v \leq 72$, significantly extending the recent classification of periodic Golay pairs for lengths $v \leq 34$ by Crnković, Danilović, Egan, and Švob [10]. For the lengths $v \leq 34$, our results are identical to their reported results. Balonin and Đoković [12] give results for exhaustive searches that they performed in all lengths $v \leq 26$, and also report some counts for the number of equivalence classes they found in the lengths $v \in \{32, 34, 40\}$. By personal communication, we were informed that the searches with $v \leq 32$ were exhaustive and the searches with $v > 32$ were not exhaustive. Indeed, our counts match their counts

v	(a,b)	$\big([\frac{a+b}{2},\frac{a-b}{2}],[\frac{a+b}{2},\frac{b-a}{2}]\big)$	$([0,a],\,[0,b])$
2	$(0,2)$	1	0
4	$(2,2)$	1	1
8	$(0,4)$	1	1
10	$(2,4)$	1	0
16	$(4,4)$	11	11
18	$(0,6)$	0	0
20	$(2,6)$	21	13
26	$(4,6)$	53	0
32	$(0,8)$	400	438
34	$(2,8)$	373	0
36	$(6,6)$	0	0
40	$(4,8)$	6012	3269
50	$(6,8)$	5293	0
50	$(0,10)$	3460	0
52	$(2,10)$	8619	5735
58	$(4,10)$	13386	0
64	$(8,8)$	1112383	1112383
68	$(6,10)$	57095	33145
72	$(0,12)$	863	890

Table 4: Counts of $(v/2)$-compressions for all PG(v) with $v \le 72$.

for all lengths $v \le 32$. For PG(34), their non-exhaustive search found 256 equivalence classes (117 fewer than our search), but surprisingly for PG(40) they report finding 9301 equivalence classes (20 classes more than our search). We contacted the authors, but were unable to discover the cause of the discrepancy. An independent verification of the counts would be desirable.

7 The $(v/2)$-uncompression conjecture for PG(v)

An examination of the exhaustive search results obtained in Section 6 reveals some structure holding for all periodic Golay pairs of lengths $v \le 72$. It also holds for all additional sporadic examples of periodic Golay pairs we found in the literature [2, 9, 10]. To describe the structure, suppose (A, B) is the $(v/2)$-decomposition of a PG(v) with rowsums a and b, i.e., $a^2 + b^2 = 2v$. All $(v/2)$-compressions of PG(v) that we considered were equivalent to one of two forms, either $([0,a],[0,b])$ or $\big([\frac{a+b}{2},\frac{a-b}{2}],[\frac{a+b}{2},\frac{b-a}{2}]\big)$ and counts for the number of $(v/2)$-compressions in each of the two forms are presented in Table 4. Note that the case $([0,a],[0,b])$ never occurs when $v/2$ is odd, a fact we prove in Lemma 2 below. We also prove that when $v/2$ is an odd prime that there is only one possible $(v/2)$-compression up to signs and order—see Theorem 4.

Lemma 2. *Suppose v is a periodic Golay number with $v/2$ odd and let (a,b) be a solution of $a^2 + b^2 = 2v$. Then $([0,a],[0,b])$ cannot be the $(v/2)$-compression of any* PG(v).

Proof. When $v/2$ is odd, the $(v/2)$-compression of any $\{\pm1\}$-sequence only consists of odd entries, as each element is constructed from an odd number of ±1s. Since 0, a, and b are all even integers, they cannot be entries of a $(v/2)$-compression of a PG(v). □

Theorem 4. *Suppose (A,B) is the $(v/2)$-decomposition of a $\mathrm{PG}(v)$ with rowsums a and b, i.e., $a^2+b^2=2v$. If $v/2$ is an odd prime, then (A,B) is equivalent to* $\big([\frac{a+b}{2},\frac{a-b}{2}],[\frac{a+b}{2},\frac{b-a}{2}]\big)$.

Proof. Denote the entries of (A,B) by $([a_0,a_1],[b_0,b_1])$ and note that $a_0+a_1=a$ and $b_0+b_1=b$. Because compression preserves complementarity, we have

$$\mathrm{PAF}(A,1)+\mathrm{PAF}(B,1)=2a_0a_1+2b_0b_1=0.$$

Dividing by 2 and using $a_1=a-a_0$ and $b_1=b-b_0$, we have $a_0(a-a_0)+b_0(b-b_0)=0$ or equivalently $a_0^2-aa_0+b_0^2-bb_0=0$. Completing the square twice yields

$$(a_0-a/2)^2-a^2/4+(b_0-b/2)^2-b^2/4=0,$$

which, as $a^2+b^2=2v$, gives

$$(a_0-a/2)^2+(b_0-b/2)^2=v/2.$$

Since a, b, and v are all even, this is a decomposition of the odd prime $v/2$ into a sum of two integer squares. It is well known (e.g., see Section 16.9 of [24]) that $v/2$ must be of the form $4m+1$ for such a decomposition to exist and in total there are eight distinct decompositions. Up to order and signs all decompositions are equivalent to $(a/2)^2+(b/2)^2=v/2$; the first four decompositions are given by all choices of signs in $a_0=a/2\pm a/2$ and $b_0=b/2\pm b/2$, and the last four are given by all choices of signs in $a_0=a/2\pm b/2$ and $b_0=b/2\pm a/2$.

The first four cases for (a_0,b_0) result in (A,B) being equivalent to $([0,a],[0,b])$. Since $v/2$ is odd, by Lemma 2 these are not possible $(v/2)$-compressions. The remaining four cases for (a_0,b_0) all result in (A,B) being equivalent to $\big([\frac{a+b}{2},\frac{a-b}{2}],[\frac{a+b}{2},\frac{b-a}{2}]\big)$. □

Note that the proof of Theorem 4 easily generalizes whenever $v/2$ is either not divisible by a prime of the form $4m+1$, or in its prime factorization has a prime of the form $4m+1$ appearing exactly once. In that case (and only that case), there is at most a single decomposition of $v/2$ up to order and signs.

Theorem 5. *Suppose (A,B) is the $(v/2)$-decomposition of a $\mathrm{PG}(v)$ with rowsums a and b. If the prime factorization of $v/2$ contains no prime of the form $4m+1$, or contains a single prime of the form $4m+1$ appearing exactly once, then (A,B) is equivalent to either $([0,a],[0,b])$ or* $\big([\frac{a+b}{2},\frac{a-b}{2}],[\frac{a+b}{2},\frac{b-a}{2}]\big)$.

Theorem 5 explains the results of every case presented in Table 4 with the exception of $v=50$, because the prime factorization of $v/2=5^2$ contains two copies of the prime 5. In that case, there are two inequivalent decompositions of $v/2$ into a sum of two integer squares, $(\alpha_1,\beta_1)=(0,5)$, and $(\alpha_2,\beta_2)=(3,4)$. Multiple inequivalent decompositions raises the possibility that there might exist $(v/2)$-compressions of other forms, such as $([\alpha_2+\alpha_1,\alpha_2-\alpha_1],[\beta_2+\beta_1,\beta_2-\beta_1])$. In the case $v=50$ and $(a,b)=(6,8)$, this would result in the $(v/2)$-compression $([3,3],[9,-1])$, but interestingly this was not the 25-compression of any PG(50) found by our exhaustive search. The fact that $([3,3],[9,-1])$ didn't occur in practice could potentially indicate there is a yet-undiscovered theoretical reason why such a $(v/2)$-compression is not possible. This

v	d	sequences of length v/d with zero PAF that d-uncompress to a PG(v)
16	2	$[-2,0,2,0,2,0,2,0]$, $[-2,0,2,0,2,0,2,0]$ $\star$
20	5	$[1,1,1,-1]$, $[3,3,3,-3]$
32	8	$[0,0,0,0]$, $[4,4,4,-4]$
40	4	$[0,0,0,0,0,0,0,4,0,0]$, $[0,-4,0,0,4,0,4,0,0,4]$
40	8	$[0,0,0,0,4]$, $[0,0,0,0,8]$
50	10	$[0,0,0,0,0]$, $[0,0,0,0,10]$
50	10	$[0,0,0,0,6]$, $[0,0,0,0,8]$
52	13	$[1,1,1,-1]$, $[5,-5,5,5]$
64	4	$[-4,0,0,0,0,0,4,0,4,0,0,0,0,0,4,0]$, $[-4,0,0,0,0,0,4,0,4,0,0,0,0,0,4,0]$ $\star$
64	8	$[-4,0,0,4,4,0,0,4]$, $[-4,0,0,4,4,0,0,4]$ $\star$
64	16	$[0,0,0,8]$, $[0,0,0,8]$ $\star$
68	17	$[5,-5,-5,-5]$, $[3,-3,-3,-3]$
72	8	$[0,0,0,0,0,0,0,0,0]$, $[0,-8,4,0,4,4,0,4,4]$
72	12	$[0,0,0,0,0,0]$, $[0,0,0,0,0,12]$
72	24	$[0,0,0]$, $[0,0,12]$

Table 5: Some select zero-PAF cases of order $v \neq 2 \cdot p$. The symbol $\star$ indicates that the two sequences of the pair are equal, which seems to occur when v is a perfect square.

makes it tempting to conjecture that the condition on the prime factorization of $v/2$ in Theorem 5 might be dropped.

Conjecture 1. *Suppose (A, B) is the $(v/2)$-decomposition of a* PG(v) *with rowsums a and b. Then (A, B) is equivalent to either* $([0, a], [0, b])$ *or* $\big([\frac{a+b}{2}, \frac{a-b}{2}], [\frac{a+b}{2}, \frac{b-a}{2}]\big)$.

Given we have experimentally tested Conjecture 1 in only a single length that Theorem 5 does not apply to, the evidence for this conjecture is currently rather weak. However, even if the conjecture is false it would be interesting to determine for which other lengths v it holds for. The first length for which the correctness of Conjecture 1 is uncertain is $v = 100$.

We have also found an interesting phenomenon when analyzing the d-compressions of the results of our exhaustive searches for values of d other than $v/2$. Often, our d-compressed pairs exhibit the characteristic that the PAF values are all zero. That is, $\text{PAF}(A, s) = 0$ and $\text{PAF}(B, s) = 0$ for $s = 1, \ldots, v/2$, and thus trivially satisfying the PAF equation of Theorem 2. We have found examples of this occurring in all lengths v for which $v/2$ is not prime. Table 5 lists the values of v for which we verified this phenomenon. As shown in the table, there are often many divisors d that work for a specific v. In addition, note that there are (up to sign and order) two possible rowsum possibilities for $v = 50$, and there are d-compressions of PG(50)s with all-zero PAFs using $d = 10$ for both rowsum possibilities.

Inspection of Table 5 shows a number of cases where the d-compressions are of the form $([0, \ldots, 0, a], [0, \ldots, 0, b])$ where a and b are the rowsums of the PG(v). If this pattern holds for PG(90) with $d \in \{18, 30\}$, then we would have d-compressions of the form $([0, 0, 6], [0, 0, 12])$ or $([0, 0, 0, 0, 6], [0, 0, 0, 0, 12])$. Exploiting this pattern may make it feasible to find a PG(90), assuming that a d-compression of a PG(90) does in fact exist in one of these two forms. We are currently running searches to determine if such a PG(90) exists, but none has yet been found.

8 Conclusion and Future Work

In this paper, we develop efficient algorithms to search exhaustively for periodic Golay pairs of all lengths v up to and including $v = 72$. This extends significantly the range of lengths that exhaustive searches for periodic Golay pairs have previously been performed. Our algorithms rely primarily on two ingredients, compression and orderly generation. Compression is always applicable because the length of a periodic Golay pair must be an even integer, and we derived a multi-level compression method to improve cases when the length is divisible by a small prime more than once. Orderly generation helps to reduce the search space of our candidate compressed sequences and is also relevant for cases of complementary sequences that cannot utilize sequence compression.

We analyzed the results of our exhaustive searches and discovered patterns pertaining to new structural results on periodic Golay pairs and proved results on the possible forms that the $(v/2)$-compression of a periodic Golay pair of order v must have. We also noticed patterns in d-compressions of periodic Golay pairs for values of d other than $v/2$. In particular, we noticed that in all cases we examined that there were some values of d for which there exist d-compressions of $\mathrm{PG}(v)$ with all-zero PAF vectors. In some of these cases the d-compressions were of the form $([0, \dots, 0, a], [0, \dots, 0, b])$, and we propose that using this pattern might help find periodic Golay pairs for certain larger orders, including the next open case $v = 90$. Overall, we believe that our methods provide strong potential for finding a solution to PG(90). Our upper bound estimation for exhaustively searching the entire search space for length $v = 90$, using a compression factor of 6, is approximately 12,900 CPU years.

References

[1] Golay, M.J.E.: Multi-slit spectrometry. Journal of the Optical Society of America **39**(6), 437 (1949) https://doi.org/10.1364/josa.39.000437

[2] Đoković, D.Ž., Kotsireas, I.S.: Some new periodic Golay pairs. Numerical Algorithms **69**(3), 523–530 (2014) https://doi.org/10.1007/s11075-014-9910-4

[3] Đoković, D.v.Z.: Equivalence classes and representatives of Golay sequences. Discrete Math. **189**(1-3), 79–93 (1998) https://doi.org/10.1016/S0012-365X(98)00034-X

[4] Borwein, P.B., Ferguson, R.: A complete description of Golay pairs for lengths up to 100. Math. Comput. **73**, 967–985 (2004) https://doi.org/10.1090/S0025-5718-03-01576-X

[5] Turyn, R.J.: Hadamard matrices, Baumert-Hall units, four-symbol sequences, pulse compression, and surface wave encodings. Journal of Combinatorial Theory, Series A **16**(3), 313–333 (1974) https://doi.org/10.1016/0097-3165(74)90056-9

[6] Fiedler, F., Jedwab, J., Parker, M.G.: A framework for the construction of Golay sequences. IEEE Trans. Inform. Theory **54**(7), 3114–3129 (2008) https://doi.org/

10.1109/TIT.2008.924667

[7] Fiedler, F., Jedwab, J.: How do more Golay sequences arise? IEEE Trans. Inform. Theory **52**(9), 4261–4266 (2006) https://doi.org/10.1109/TIT.2006.880024

[8] Fiedler, F., Jedwab, J., Parker, M.G.: A multi-dimensional approach to the construction and enumeration of Golay complementary sequences. J. Combin. Theory Ser. A **115**(5), 753–776 (2008) https://doi.org/10.1016/j.jcta.2007.10.001

[9] Đoković, D.Ž., Kotsireas, I.S.: Periodic Golay pairs of length 72. In: Algebraic Design Theory and Hadamard Matrices. Springer Proc. Math. Stat., vol. 133, pp. 83–92. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-17729-8_7

[10] Crnković, D., Danilović, D.D., Egan, R., Švob, A.: Periodic Golay pairs and pairwise balanced designs. Journal of Algebraic Combinatorics **55**(1), 245–257 (2022) https://doi.org/10.1007/s10801-021-01084-0

[11] Kotsireas, I.S., Koukouvinos, C., Pardalos, P.M., Shylo, O.V.: Periodic complementary binary sequences and combinatorial optimization algorithms. Journal of Combinatorial Optimization **20**(1), 63–75 (2008) https://doi.org/10.1007/s10878-008-9194-5

[12] Balonin, N., Đoković, D.Ž.: Symmetry of two-circulant Hadamard matrices and periodic Golay pairs (in Russian). Information and Control Systems (3), 2–16 (2015). https://i-us.ru/index.php/ius/article/view/4442

[13] Đoković, D.Ž., Kotsireas, I.S.: Compression of periodic complementary sequences and applications. Designs, Codes and Cryptography **74**(2), 365–377 (2013) https://doi.org/10.1007/s10623-013-9862-z

[14] Fletcher, R.J., Gysin, M., Seberry, J.: Application of the discrete Fourier transform to the search for generalised Legendre pairs and Hadamard matrices. The Australasian Journal of Combinatorics (23), 75–86 (2001). https://ajc.maths.uq.edu.au/pdf/23/ocr-ajc-v23-p75.pdf

[15] Kotsireas, I., Koutschan, C.: Legendre pairs of lengths $\ell \equiv 0 \pmod 3$. J. Combin. Des. **29**(12), 870–887 (2021) https://doi.org/10.1002/jcd.21806

[16] Faradžev, I.: Constructive enumeration of combinatorial objects. In: Problèmes Combinatoires et Théorie Des graphes, pp. 131–135 (1978)

[17] Read, R.C.: Every one a winner or how to avoid isomorphism search when cataloguing combinatorial configurations, 107–120 (1978) https://doi.org/10.1016/s0167-5060(08)70325-x

[18] McKay, B.D.: Isomorph-free exhaustive generation. Journal of Algorithms **26**(2), 306–324 (1998) https://doi.org/10.1006/jagm.1997.0898

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

[20] Bright, C., Kotsireas, I., Ganesh, V.: Applying computer algebra systems with SAT solvers to the Williamson conjecture. Journal of Symbolic Computation **100**, 187–209 (2020) https://doi.org/10.1016/j.jsc.2019.07.024

[21] Bright, C., Kotsireas, I., Heinle, A., Ganesh, V.: Complex Golay pairs up to length 28: A search via computer algebra and programmatic SAT. Journal of Symbolic Computation **102**, 153–172 (2021) https://doi.org/10.1016/j.jsc.2019.10.013

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

[23] Baldwin, S.: Compute Canada: advancing computational research. In: Journal of Physics: Conference Series, vol. 341. IOP Publishing, Bristol, England (2012). https://doi.org/10.1088/1742-6596/341/1/012001

[24] Hardy, G.H., Wright, E.M., Heath-Brown, D.R., Silverman, J.: An Introduction to the Theory of Numbers. Oxford University Press, Oxford (2008)