Stedman and Erin Triples encoded as a SAT Problem

Andrew Johnson

December 9, 2018

Stedman and Erin Triples encoded as a SAT Problem

Andrew Johnson[1]

Winchester, England
andrew_johnson@cantab.net

Abstract

A very old quest in campanology is the search for peals, which can be considered as constrained searches for Hamiltonian cycles of a Cayley graph. Two particularly hard problems are finding bobs-only peals of Stedman Triples and Erin Triples. We show how to efficiently reduce them to boolean satisfiability and use a SAT solver to help find bobs-only peals of Stedman Triples, and express the unsolved problem of bobs-only Erin Triples as an unsolved SAT problem. This approach is based on the author's very efficient general reduction of the Hamiltonian Cycle Problem (HCP) to Boolean Satisfiability (SAT) converting any Hamiltonian Cycle problem with n vertices and m directed edges to a SAT problem with approximately n.log2(m) variables and 2m.(log2(n)+1) clauses.

1 Introduction

English style church bell ringing is performed by people each ringing a bell by means of a rope attached to a wheel which is attached to a bell so that the bell rotates 360 degrees first one way then reverses, the bell sounding at the end of each rotation. It is hard to explain concisely so a viewing video clip[25] and animation[4] is helpful. The bells are not rung in tunes, or haphazardly, but in sequences according to mathematically definable rules which lead to some very hard mathematical problems. *Change ringing* is based on the idea of ringing bells in different sequences, and analysis of it involves permutations, group theory, Hamiltonian cycles and in this paper, boolean satisfiability.

Numbering. Each bell is given numeric identifier, from 1 to n where n is the number of bells. [In practice they are numbered from 1, the lightest (highest pitched) to n, the heaviest (lowest pitched).]

Row. Each bell rings exactly once, in some sequence, before any bell rings again. An example of a sequence, or permutation, of the numbers $1 \ldots n$ is known as a *row* as the term permutation also occurs in other contexts.

Rounds. The initial sequence or *row* $1, \ldots, n$ is known as *rounds*.

Change. One *row* is followed by a different *row* as the bells ring in a different order. The permutation from one *row* to the succeeding *row* is a *change*. There is a *change ringing* rule that each bell may only move at most one place in the order in the *row* when comparing two successive *rows*. [This is because of physical constraint of the inertia of the bell (which weigh from 100kg to 4000kg depending on the installation in the tower) means that changing the speed is hard.] For example *row* `54321678` immediately followed by *row* `45312768` is permitted, but *row* `54321678` followed by *row* `43512768` is not (as bell number `5` has jumped from ringing first in the order to third in the order. This means that a valid *change* is one or more separate adjacent transpositions. So `54321678` to `45312768` is the *change* (1 2)(4 5)(6 7).

2 The Methods of Stedman Triples and Erin Triples

There are various *methods* of generating a sequence of different *rows* such that no *row* is repeated before returning to the starting point. Two methods are *Stedman Triples*[30] and *Erin Triples* which operate on 7 bells. [In practice they are rung on eight bells with the eighth bell ringing last in each *row* to complete the octave of a diatonic scale.] Stedman Triples and Erin Triples are the result of applying permutations (or *changes*) to successive *rows* as follows:

$$\begin{aligned} p_1 &= (2\ 3)(4\ 5)(6\ 7) \\ p_3 &= (1\ 2)(4\ 5)(6\ 7) \\ p_5 &= (1\ 2)(3\ 4)(6\ 7) \\ p_7 &= (1\ 2)(3\ 4)(5\ 6) \\ Q &= p_1 \cdot p_3 \cdot p_1 \cdot p_3 \cdot p_1 \cdot \{p_7 | p_5\} \\ S &= p_3 \cdot p_1 \cdot p_3 \cdot p_1 \cdot p_3 \cdot \{p_7 | p_5\} \\ \mathit{Stedman\ Triples} &= [Q \cdot S]^n \\ \mathit{Erin\ Triples} &= S^n \end{aligned} \tag{1}$$

Each application of p_1, p_3, p_5 or p_7 generates a successive *row*. The sequence of permutations is divided into sets of six, where the last permutation of the *six* can be varied. Here p_7 or p_5 can be chosen freely to vary the sequence of *rows*. In practice permutation p_7 is the default and the ringers ring the sequence by learning the pattern, and the sequence of *rows* repeats when $n = 7$, generating 84 different *rows* for Stedman Triples and 42 different *rows* for Erin Triples. To extend the sequence further, a ringer called the *conductor* calls out *bob* at appropriate points which is the signal for the ringers to replace p_7 by p_5. The absence of a *bob* is a *plain*.

Six. The 6 *rows* generated by any first *row* and the first 5 *changes* of Q or S are known as a *six*.

Quick Six. A *six* generated by Q is known as a *quick six*.

Slow Six. A *six* generated by S is known as a *slow six*.

Bob. The application of permutation p_5 as the *change* after the last *row* of a *six*.

Plain. The application of permutation p_7 as the *change* after the last *row* of a *six*. The *change* of p_7 or p_5 can be considered as a transition from one *six* to another.

Call. A variation to the sequence of *changes* of a method, such as a *bob* or *plain*.

3 Peals

Ringers like to ring a *peal* which with 7 bells changing means ringing all $7! = 5040$ possible *rows* without repetition starting and ending with *rounds*. It can be considered as a Hamiltonian cycle of the Cayley graph generated by $\langle p_1, p_3, p_5, p_7 \rangle$ of the symmetric group S7 subject to rules about which edges can be used in which order. A performance of a *peal* takes about 3 hours of non-stop ringing. The first *peal* of Stedman Triples was rung in 1731; the first of Erin Triples in

1908. Those performances required an additional type of *call* such as *single* $p_{567} = (1\ 2)(3\ 4)$. Two of the hardest and oldest questions in the mathematics of bell ringing are whether a full *peal* of Stedman Triples or Erin Triples can be rung just using *bobs* and *plains*. The answer to the former was settled in 1994 by Wyld[37], though the solution was not published until after another solution was discovered, performed and published by Johnson and Saddleton[14] in 1995. Johnson then published additional solutions in 1995[10], 2012 and 2017[13]. Whether a *peal* of Erin Triples can be rung with just *bobs* and *plains* is still an open question — the bobs-only Erin Triples problem.

4 Peal Searches and Satisfiability

Composers of *peals* have since 1952[20] used computers for searches; the problem for Stedman and Erin Triples is particularly hard as there are 840 places in a full *peal* of 5040 *rows* where a *bob* might be called, giving a total of $2^{840} = 7.3 \times 10^{252}$ possibilities. Generally these searches are done as depth-first tree searches and many *peals* have been composed using singles[23]. The problem can be considered as finding a Hamiltonian cycle in a restricted Cayley graph of S7. See the papers by White[36] for further details of group theory and *change ringing*. The paper by Haythorpe and Johnson[6] also shows it can be considered as a pure Hamiltonian cycle problem using a special subgraph gadget to represent each *six* and the remainder of the graph links the *sixes*.

4.1 Sixes and SAT Encoding

For the first time the problem has also been fully encoded using boolean satisfiability and the technique of the author shown here has proved a practical method of approaching this problem. For Stedman Triples the 5040 *rows* in a *peal* are divided into *sixes*, each containing 6 consecutive *rows*, and each of the *sixes* has 6 possible forms which differ in the order the same 6 *rows* appear, as shown in Table 1. Those *rows* cannot appear in any other six. For parity reasons *rows* such as `1325476` or `1324567` cannot appear at the end of a *six* when just *bobs* and *plains* are used. Each of the *sixes* can appear once only in a *peal*, otherwise a *row* is repeated, and must appear once in some form for all possible *rows* to appear. Instead of considering the problem as a pure Hamiltonian cycle with 27720 nodes (Stedman) or 13440 nodes (Erin) and converting it to SAT the problem is better considered as a directed Hamiltonian cycle problem between 840 nodes (the *sixes*) subject to rules about which exits from a *six* are permitted given how the *six* is entered.

Quick six	Quick six	Quick six	Slow six	Slow six	Slow six
2135476	3215476	1325476	1325476	2135476	3215476
2314567	3124567	1234567	3124567	1234567	2314567
3215476	1325476	2135476	3215476	1325476	2135476
3124567	1234567	2314567	2314567	3124567	1234567
1325476	2135476	3215476	2135476	3215476	1325476
1234567	2314567	3124567	1234567	2314567	3124567

Table 1: Forms of an example *six* all containing the same rows

See Table 2 for an example of part of a transition table. Each *six* can only be visited once, but the possible exits depend on which six-type the *six* was entered with. The transition table is constructed by generating all the possible *sixes*, then generating all the types of each *six* and numbering them such that all the types of one *six* have unique consecutive identifying integers. For convenience the six-types can also be identified by the last *row* of the *six*. The transitions from one six-type to another with a *bob* or *plain* are then calculated. The precise allocation of identifying integers does not matter as they will not feature when the solution is translated back to the original domain.

Source	Destination with plain	Destination with bob	First *row* of *six*	Identifier (last *row*)	*Six* index	Type index
1	4	7	1325476	1234567SS	1	1
2	10	13	2135476	2314567SS	1	2
3	16	19	3215476	3124567SS	1	3
4	22	25	2143657	2416375SS	2	1
5	28	31	4213657	4126375SS	2	2
6	34	37	1423657	1246375SS	2	3
7	40	43	2143576	2415367SS	3	1
8	46	49	4213576	4125367SS	3	2
9	52	55	1423576	1245367SS	3	3
10	58	61	3241657	3426175SS	4	1
11	64	67	4321657	4236175SS	4	2
12	70	73	2431657	2346175SS	4	3
⋮						
2518	2348	2252	4312567	4135276SS	840	1
2519	2346	2247	1432567	1345276SS	840	2
2520	2370	2277	3142567	3415276SS	840	3

Table 2: six-type transition table for Erin Triples

There are $5040/6 = {}^7P_4 = 840$ possible *sixes*, and for Stedman Triples each can appear in 1 of 6 forms as shown in Table 1. For Erin Triples, only *slow sixes* occur, and so the *sixes* each could appear in 1 of 3 forms. This suggests a simple encoding for each *six*, representing which form of the *six* occurs. Possible encodings of the type index could be binary/log, direct (one-hot), order or twisted-ring[12], as shown in Table 3. Twisted-ring is a new encoding, representing n states with $n/2$ bits, and only requiring the testing of 2 bits to determine a state. *Valid states* shows how to set the variables to represent each of the 6 states. *Test bits* show which bits need to be tested to show that the variables represent that state. *Invalid states* shows combinations of bits which need to be excluded to ensure that at exactly one state is decoded by the *test bits*.

After the end of each *six* a *bob call* can be made. A simple single bit encoding is sufficient to record whether a *bob* or a *plain* occurs. With these encodings the linkage between the *sixes* can be encoded using clauses quite simply. The SAT clauses are generally of a support encoding style; when a certain condition is present then other variables are forced to particular values. Each six-type leads to another six-type depending on whether the *six* is followed with a *plain* or a *bob*. For example, for Erin Triples with direct encoding of the six-type as in Table 4 encodes *six* 1, type 1 followed by a bob going to *six* 3, type 1 as this DIMACs encoded CNF clause:

```
-841 -1 847 0
```

where variables 1 to 840 represent the *call* type of each *six*, variables 841 to 843 represent the six-type of *six* 1 and variables 847 to 849 represent the six-type of *six* 3. Other clauses ensure

Encoding	Valid states	Test bits	Invalid states	Number of exclusion clauses	Number of bits
Log	000	000	110	$\leq \lfloor log_2(n-1) \rfloor$	$\lceil log_2 n \rceil$
	001	001	111		
	010	010			
	011	011			
	100	1x0			
	101	1x1			
Direct (or one-hot)	000001	xxxxx1	xxxx11	$n(n-1)/2+1$	n
	000010	xxxx1x	xxx1x1		
	000100	xxx1xx	xx1xx1		
	001000	xx1xxx	...		
	010000	x1xxxx	11xxxx		
	100000	1xxxxx	000000		
Order	00000	xxxx0	xxx10	$n-1$	$n-1$
	00001	xxx01	xx10x		
	00010	xx01x	xx10x		
	00100	x01xx	x10xx		
	01000	01xxx	10xxx		
	10000	1xxxx			
Twisted-ring	000	0x0	010	$n-4$	$\lceil n/2 \rceil$
	001	x01	101		
	011	01x			
	111	1x1			
	110	x10			
	100	10x			

Table 3: 1 of n encoding showing ways of encoding 6 possible states

that exactly one of variables 841, 842 and 843 is true, and there are similar clauses for every other six.

Six 1 followed by a plain encoded as variable 1 false.
Six 1 followed by a bob encoded as variable 1 true.
Six 2 followed by a bob encoded as variable 2 true.
⋮
Six 840 followed by a bob encoded as variable 840 true.
Six 1, type 1 encoded as variable 841 true.
Six 1, type 2 encoded as variable 842 true.
Six 1, type 3 encoded as variable 843 true.
Six 2, type 1 encoded as variable 844 true.
⋮
Six 3, type 1 encoded as variable 847 true.

Table 4: variable allocation for Erin Triples

4.2 Previous Hamiltonian Cycle SAT Encodings

With just those constraints the graph of *sixes* would be linked into closed loops. The big problem with encoding Hamiltonian cycle type problems into SAT is enforcing the only one loop requirement. There have been a variety of approaches to the problem, including Iwama[8], Creignou[1] who saw HCP as SAT-hard but not SAT-easy, Plottikov[17], Nasu[16], Prestwich[18], Soh[26], Velev and Gao[31][33][32].

Velev and Gao describe these as 'complete occupancy constraints, enforcing that each position in the permutation is occupied by a vertex;' and 'exclusivity positional constraints, ensuring that only one vertex can appear at a given position in the permutation'.

Previously this was achieved by some of the following:

- Direct encoding of node position in the order: n bits per node, n nodes, n^2 variables.

- Log encoding, where the node position is encoded as a binary number.

There can then be of order n^2 variables and n^3 clauses which is a very large number of clauses to ensure no two nodes have the same node position. Velev and Gao's later improvements[32] note 'that half of the ordering variables and two-thirds of the transitivity constraints can be eliminated.', but that still leaves a lot of variables and clauses. Soh et al. [26] note that with Velev's approach 'the size of the encoded clauses which explodes to over 100 million even when the input graph size is 500'. Soh's approach requires an incremental SAT solver or native boolean cardinality handling, which restricts the choice of solver.

4.3 New Hamiltonian Cycle SAT Encoding

The author's invention[11] is to give each node a compact, encoded sequence number and to define rules such that successor nodes receive a sequence number based on the predecessor sequence number according to simple rules. Each node passes on the next sequence number to the following node. By enforcing the rules that the first node has the first number and the node which becomes the last and links to the first has the last possible sequence number the ordering is established. Each node has to have a successor. This means there is either a long chain to the final node, or a loop back to earlier in the chain. There cannot be a loop back to an earlier node otherwise the sequence numbers from the two paths do not agree. Therefore all the nodes must be in a chain, and the last node links to the first.

As a Hamiltonian cycle can be started at any point we can designate an arbitrary node as the start without a sequence number; any successor of the start receives the first sequence number. The last node, the predecessor of the start, must have the last possible sequence number.

Numbering of the nodes for the sequence number could be done using binary arithmetic but a better scheme is to use the well known electronic engineering circuit of a linear-feedback shift register(LFSR)[3][35][15] which generates a LFSR sequence of $2^b - 1$ states as this uses only $2(b+1)$ or $2(b+2)$ clauses for b bits, where $b = \lceil log_2 n \rceil$.

For example this shows how to calculate a successor sequence number $B_1, B_2, \ldots, B_{10}$ from a predecessor $A_1, A_2, \ldots, A_{10}$ using a LFSR sequence of length 1023 generated by a shift register of 10 stages with taps at (10, 7) feeding an exclusive-or gate $\oplus$.

$$\begin{aligned}
B_1 &= A_{10} \\
B_2 &= A_1 \\
B_3 &= A_2 \\
B_4 &= A_3 \\
B_5 &= A_4 \\
B_6 &= A_5 \\
B_7 &= A_6 \\
B_8 &= A_7 \oplus A_{10} \\
B_9 &= A_8 \\
B_{10} &= A_9
\end{aligned} \tag{2}$$

This can be encoded in a similar fashion, so if *six* 1 has sequence number variables 2521 to 2530 and *six* 3 has sequence number variables 2541 to 2550 then the following clauses show the linkage, using direct encoding of six-type and *call* as in Table 4. The `-841 -1` are gating literal terms which when the variables are true force the literals to be false and so at least one of the literals in the remainder of the clause must be true.

```
c wrap
-841 -1 -2530 2541 0
-841 -1 2530 -2541 0
c copy up
-841 -1 -2521 2542 0
-841 -1 2521 -2542 0
-841 -1 -2522 2543 0
-841 -1 2522 -2543 0
-841 -1 -2523 2544 0
-841 -1 2523 -2544 0
-841 -1 -2524 2545 0
-841 -1 2524 -2545 0
-841 -1 -2525 2546 0
-841 -1 2525 -2546 0
-841 -1 -2526 2547 0
-841 -1 2526 -2547 0
c XOR clauses
-841 -1 -2527 -2530 2548 0
-841 -1 2527 2530 2548 0
-841 -1 -2527 2530 -2548 0
-841 -1 2527 -2530 -2548 0
c copy up
-841 -1 -2528 2549 0
-841 -1 2528 -2549 0
-841 -1 -2529 2550 0
-841 -1 2529 -2550 0
```

These techniques were useful in the Flinders Hamiltonian Cycle Project[2] Challenge[5], which had 1001 Hamiltonian graphs from 66 vertices up to 9528 vertices. The author gained a comfortable second place with 614 solved problems out of 1001 (compared to the winner with 985). Interestingly, all the unsolved problems of the winner were based on Hamiltonian Stedman graphs, so Stedman Triples provides particularly tricky mathematical problems.

4.4 Variable and Clause Counts

We can now calculate the number of variables and clauses required to encode the problems in SAT.

4.4.1 Erin Triples

For Erin Triples there are only *slow sixes*, so 3 types per *six*. This can be encoded in 2 bits. There are 840 *sixes*; the first does not need to be numbered, so 839 *sixes* requiring sequence numbers. At least 10 bits are required for 839 different numbers. The *call* after each *six* (*plain* or *bob*) needs to be encoded, requiring a bit each time. This requires $840 \times 2 + 839 \times 10 + 840 \times 1 = 10910$ variables.

For the clauses, enforcing the six-type restriction for binary encoding requires $840 \times 1 = 840$ clauses to exclude the invalid type $\{11\}$. For the 6 possible successor six-types (3 after a *plain*, 3 after a *bob*) for a *six*, 4 need two variables to be set/cleared for the *six* type, and 2 need just one as six-type 3 is fully determined by $\{1x\}$. This means that 10 clauses are needed per *six* to set the successor six-type, so $840 \times 10 = 8400$ clauses.

For the sequence number, the first *six* needs to set 6 possible successor *six* sequence numbers to the initial value depending on the six-type and *call* of the first *six*. This needs $6 \times 10 = 60$ clauses. The 6 possible final six-types need to check that sequence number is the final number if the six-type links back to the start, for a further $6 \times 10 = 60$ clauses. There are then 839 sixes each with 3 six-types each going to two (*plain* or *bob*) possible other six-types, excluding the final six-types above. These require 22 clauses, so need $(839 \times 2 \times 3 - 6) \times 22 = 110616$. This means that $840 + 8400 + 60 + 60 + 110616 = 119976$ clauses.

4.4.2 Stedman Triples

For Stedman Triples, there are 840 *sixes*, each with a possible *call* following, so $840 \times 1 = 840$ variables for *calls*. For the six-type, there are 6 possible values, so with binary encoding, 3 variables per *six* are needed for $840 \times 3 = 2520$ variables.

There is a strict alternation between *quick sixes* and *slow sixes* according to the rules for Stedman Triples, so to save variables we can have the rule that *slow sixes* have the same sequence number as the preceding *quick six*. This halves the range of the sequence number and so removes a variable per *six*. Also the first *six* does not need a number — we define special rules that any of the successor *sixes* have the first sequence number if they are chosen as the second *six*, and the last *six* must have the last possible *six* number when it links back to the first *six*. We therefore need 420 different sequence numbers, requiring 9 bits. So we need $(840-1) \times 9 = 7551$ variables for the sequence number. The total is then $840 + 2520 + 7551 = 10911$ variables.

For clauses for Stedman Triples, each possible *six* has one of 6 types. With binary encoding, the two unused values out of eight must be excluded; this can be done with one clause per *six*. Each *six* has 6 types, each of which has two possible destinations depending on the *call*, (*plain* or *bob*). For a particular *six*, 4 of the six-types require 3 of 3 variables to be set/cleared to fully define the six-type, and because of restrictions in coding 6 out of 8 in binary the other two only require 2 variables to be set/cleared. With 12 destinations, 8 require 3 variables and so 3 clauses and 4 require 2 variables and so 2 clauses, so we need $840 \times (8 \times 3 + 4 \times 2) = 26880$ clauses. Similarly to Erin setting the initial sequence number for 12 successor *sixes* to the first *six* requires $12 \times 9 = 108$ clauses, and checking the final sequence number also requires $12 \times 9 = 108$ clauses. The *quick six* to *slow six* transition requires $9 \times 2 = 18$ clauses to copy the sequence number; this applies to 839 *sixes* each with 3 *quick* six-types and 2 *calls* except for the 6 *quick* six-types which with the appropriate *call* precede the first *six*, so for a total of $(839 \times 3 \times 2 - 12) \times 18 = 90504$ clauses. For the *slow six* to *quick six* transition with the LFSR transition involving exclusive-or, 20 clauses instead of 18 are required per sequence number, for a total of $(839 \times 3 \times 2 - 12) \times 20 = 100560$ clauses. The total is then $840 + 26880 + 108 + 108 + 90504 + 100560 = 219000$ clauses.

4.4.3 Efficiency

These variable and clause counts are much smaller than the counts with Velev's absolute or relative encodings or even the counts for Soh's incremental coding of a 900 node graph. Generating these SAT clauses is also a quick process as no complex calculations are required, so can easily be done in seconds.

5 Variations Using Groups

Restricted versions of the problem can be considered by searching for cycles in the Schreier coset graph[36]. Only certain groups are suitable, and the identifier [0.01] etc. is from Price[19]. This maps multiple *sixes* into each node of the new graph, as cosets of the chosen group[21], so the new graph is smaller and is faster to search. A Hamiltonian cycle in the coset graph translates to one or more loops in the original graph, so once they have been expanded other techniques can be used to attempt to link the cycles into one big cycle. The following graphs in Table 5 and Table 6 have the possibility of inducing an odd number of loops in the original Sted1 and Erin1 graphs. An odd number of loops is advantageous as there is the possibility of replacing three suitably chosen *bobs* (p_5) with three *plains* (p_7) or vice versa, linking three loops into one big loop via a 3-way shuffle. With some of the groups there is even the chance of a single loop in Sted1, and the resultant *peal* is in repeated parts, a great aid to the conductor. Other groups which also divide the Sted1 and Erin1 graphs but always induce an even number of loops in the original graph are shown in Table 7 and Table 8.

Graph	Sixes	LFSR bits	Variables	Clauses	Satisfiability (Solutions)	Solve time
Sted1 [0.01]	840	9	10911	219000	SAT	
Sted2 [4.07]	420	8	5032	99324	SAT	
Sted3 [6.33]	280	8	3352	66144	SAT	
Sted4 [6.26]	210	7	2303	44538	SAT	
Sted5 [5.05]	168	7	1841	35226	SAT (4)	≈1 week on Colossus
Sted6 [6.32]	140	7	1533	29628	SAT (132)	
Sted7 [7.07]	120	6	1194	22338	UNSAT	
Sted10 [5.04]	84	6	834	15562	SAT (4)	≈3s
Sted20 [7.12]	42	5	373	6734	SAT (6)	<1s
Sted21 [7.05]	40	5	355	6408	UNSAT	<1s

Table 5: Stedman graphs

If a mixed-parity group is used then the *sixes* can appear in 12 forms (for Stedman) and 6 forms (for Erin). These groups also induce an even number of loops in the Sted1 and Erin1 graphs, but provide further hard SAT problems. They are shown in Table 9 and Table 10 As the *sixes* appear in more forms additional variables and clauses are needed over the even parity

Graph	Sixes	LFSR bits	Variables	Clauses	Satisfiability	Solve time
Erin1 [0.01]	840	10	10911	119976	UNKNOWN	
Erin2 [4.07]	420	9	5031	54888	SAT	
Erin3 [6.33]	280	9	3351	36548	UNSAT	≈2w Colossus
Erin4 [6.26]	210	8	2302	24870	UNSAT	≈1d
Erin5 [5.05]	168	8	1840	19872	UNSAT	≈10m
Erin6 [6.32]	140	8	1532	16540	UNSAT	
Erin7 [7.07]	120	7	1193	12732	UNSAT	≈1m
Erin10 [5.04]	84	7	833	8880	UNSAT	<1s
Erin20 [7.12]	42	6	372	3894	UNSAT	<1s
Erin21 [7.05]	40	6	354	3704	UNSAT	<1s

Table 6: Erin graphs

Graph	Sixes	LFSR bits	Variables	Clauses	Satisfiability (Solutions)
Sted4 [4.04]	210	7	2303	44538	UNKNOWN
Sted4 [6.35]	210	7	2303	44538	SAT
Sted8 [6.23]	105	6	1044	19677	UNSAT
Sted12 [6.14]	70	6	694	13062	SAT (248)
Sted12 [7.33]	70	6	694	13062	UNSAT
Sted24 [6.09]	35	5	310	5631	UNSAT
Sted24 [7.28]	35	5	310	5631	UNSAT
Sted60 [6.05]	14	3	95	1542	SAT (20)
Sted168 [7.03]	5	2	28	393	UNSAT

Table 7: additional Stedman graphs

group graphs. These Erin graphs require 3 variables per *six* for the *six* type and the Stedman graphs require 4 variables, with binary encoding. So for Erin2m [2.01m], there are 420 *sixes*, 420 variables for the *call* type, $420 \times 3 = 1260$ variables for the six-type, and $419 \times 9 = 3771$ variables for the sequence number.

There is further scope for reducing the clauses when there are 6 six-types per *six* by using a twisted-ring encoding. Then only 2 clauses are required to set any type, but 2 clauses are required per *six* to exclude invalid types. This reduces Sted1 down to 213120 clauses, and Erin2m to 111456 clauses.

Note that Sted8, Sted24 and Sted168 are trivially non-satisfiable by inspection because the length of each part is not a multiple of 12, so the alternation of Q and S cannot be maintained around a loop. This may not be obvious to a SAT solver, rather like the pigeon-hole problem.

6 Solving the SAT Problems

Once a solution to the SAT problem is found it is then translated back into the original domain as a sequence of *bobs* and *plains*. One way is to find the six-type for the first *six* (the one without a sequence number) and the *call* type from the state of the variables. That *call* (*bob* or *plain*) is then recorded. The next *six* and six-type is then found by reference to the transition table using the six-type and *call* type. The *call* following that *six* is then recorded. This process is repeated until the first *six* is reached again. The result is a sequence of *bobs* and *plains* which

Graph	Sixes	LFSR bits	Variables	Clauses	Satisfiability
Erin4 [4.04]	210	8	2302	24870	UNSAT
Erin4 [6.35]	210	8	2302	24870	UNSAT
Erin8 [6.23]	105	7	1043	11127	UNSAT
Erin12 [6.14]	70	7	693	7382	UNSAT
Erin12 [7.33]	70	7	693	7382	UNSAT
Erin24 [6.09]	35	6	309	3329	UNSAT
Erin24 [7.28]	35	6	309	3329	UNSAT
Erin60 [6.05]	14	4	373	6734	UNSAT
Erin168 [7.03]	5	3	94	922	UNSAT

Table 8: additional Erin graphs

Graph	Sixes	LFSR bits	Variables	Clauses	Satisfiability (Solutions)
Sted2m [2.01m]	420	8	5452	208308	UNKNOWN
Sted4m [4.05m]	210	7	2513	93906	SAT
Sted4m [4.06m]	210	7	2513	93906	UNKNOWN
Sted8m [4.03m]	105	6	1149	41769	UNSAT
Sted10m [7.14m]	84	6	918	33348	UNSAT
Sted20m [5.03m]	42	5	415	14358	SAT (24)

Table 9: mixed parity group additional Stedman graphs

could be given to a *conductor* in order to ring a *peal*.

In practice solving of these problems is improved by choosing appropriate encodings. Generally direct encoding of the six-type works best as it leads to simpler and fewer clauses expressing the presence or absence of a certain six-type. Adding redundant clauses based on higher level restrictions of the problem can also help. These include:

- Disallow certain sequence numbers (such as all zeroes, or numbers beyond the last sequence number or before the first).
- Disallow two simultaneous inbounds (both source six-type and source *call*) to a particular six-type.
- Disallow two simultaneous inbounds (both source six-type and source *call*) to a particular *six*.
- Disallow an inbound (both source six-type and source *call*) to a particular *six* if the *six* is already of a different six-type.
- Disallow a six-type if it does not have at least one input six-type with the right *call* available. This is a very important optimisation, without it sted20 takes 2209 seconds, with it takes 1.54 seconds using Cryptominisat5[27][28].
- Disallow a six-type if both exits (with a *plain* or a *bob*) do not have an available destination six-type.
- Check that according to some complex reasoning bobs come in sets of three.

Graph	Sixes	LFSR bits	Variables	Clauses	Satisfiability (Solutions)
Erin2m [2.01m]	420	9	5451	114396	UNKNOWN
Erin4m [4.05m]	210	8	2512	52050	SAT
Erin4m [4.06m]	210	8	2512	52050	UNKNOWN
Erin8m [4.03m]	105	7	1148	23409	UNSAT
Erin10m [7.14m]	84	7	917	18684	UNSAT
Erin20m [5.03m]	42	6	414	8250	SAT (4)

Table 10: mixed parity group additional Erin graphs

- If a *six* is of a particular type and we do not yet know which of the source *sixes* will be the source, but both source *sixes* have a common variable value for a sequence number bit then we can set the associated sequence number bit variable of this *six*.
- If a *six* is of a particular type and we do not yet know which of the destination *sixes* will be the destination, but both destination *sixes* have a common variable value for a sequence number bit then we can set the associated sequence number bit variable of this *six*.
- Force a *call* to be a bob if all the destination six-types from a *six* with a *plain* are unavailable and vice versa.
- If all the destination *sixes* from the *quick* six-types of this *six* are unavailable then this *six* must be of one of the *slow* six-types, and vice versa.
- If all the destination *sixes* from the *quick* six-types of this *six* with a bob are unavailable then this *six* must be of one of the *slow* six-types or the *call* must be a *plain*, and vice versa.
- At least one source six-type for any one of the six-types of this *six* must be available.
- At least one destination six-type for any one of the six-types of this *six* must be available.
- When generating a long list of related clauses, use subsumption to eliminate redundant clauses.
- Choose different LFSR taps and modes.

The largest problems can be hard to solve directly (Sted1, Sted2, Sted3, Sted4, Erin1, Erin2); some of the graphs have been solved through various other techniques. The solution can then be used to set some of the SAT variables, making the SAT problem quicker to solve. The number of set variables can easily be adjusted to vary the difficulty of the problem.

Although many of the Erin graphs are unsatisfiable there are modified problems which are satisfiable. For example the Hamiltonian cycle restriction can be removed allowing multiple loops in the graph. Also, one of those solutions can then be selected and the longest loop chosen (length m) and a partial Hamiltonian cycle restriction imposed for a loop of length m starting from one of the *sixes* in that loop. The remaining *sixes* link freely in loops, aided by the LFSR sequence number characteristic that the state of all zeroes is followed by the state of all zeroes, so any length of loop can link to the start. A restriction of only one inbound link per *six* does need to be imposed however.

Other encodings can be used. For the six-type the options considered were: direct (one-hot) encoding, order encoding, twisted-ring encoding and binary/log encoding. The twisted-ring encoding looked promising on paper with only two variables to be tested, but it did not work out that way.

LFSR sequences for 8 bits normally require a 4-tap register, e.g. (8,6,5,4). Sometimes a short-cycle counter such as (8,5) with a period of 217 states is acceptable such as for Erin4 with 210 *sixes*, and so $210 - 1 = 209$ required states. XORs can also be avoided by combining counters with twisted-ring counters. For example a 10-bit sequence can be made by combining an 8-bit LFSR (8,5) with a 2-bit twisted-ring counter giving a sequence of $LCM(217, 2 \times 2) = 217 \times 2 \times 2 = 868$ states and a 8-bit sequence by combining a 6-bit LFSR and a 2-bit twisted-ring counter using a logical-not $\neg$ giving $63 \times 4 = 252$ states.

$$\begin{aligned} B_1 &= A_8 & B_2 &= A_1 \\ B_3 &= A_2 & B_4 &= A_3 \\ B_5 &= A_4 & B_6 &= A_5 \oplus A_8 \\ B_7 &= A_6 & B_8 &= A_7 \\ B_9 &= \neg A_{10} & B_{10} &= A_9 \end{aligned} \tag{3}$$

A top-bottom[34] hybrid LFSR with a cycle length of 255 can also be implemented as follows:

$$\begin{aligned} B_1 &= A_6 \oplus A_8 & B_2 &= A_1 \\ B_3 &= A_2 & B_4 &= A_3 \\ B_5 &= A_4 & B_6 &= A_5 \\ B_7 &= A_6 & B_8 &= A_7 \oplus A_8 \end{aligned} \tag{4}$$

Bits	Taps	Length	Notes
2	(2,1)	3	
3	(3,2)	7	
4	(4,3)	15	
5	(5,3)	31	
6	(6,5)	63	
7	(7,6)	127	
8	(8,5)	217	non-maximal
8	(8,7,6)	255	top-bottom hybrid
8	(8,6,5,4)	255	
9	(9,5)	511	
10	(10,9) (8,5)	868	non-maximal LFSR with twisted-ring counter
10	(10,7)	1023	

Table 11: LFSR Taps

These combined LFSR and twisted-ring counters sometimes had an advantage in SAT solve time for other counter lengths. Example counters are shown in Table 11.

An interesting factor in possible solutions is the total number of *bobs* as minimising this number makes the job of calling the composition from memory easier for the conductor. The number of places where three *bobs* could be replaced by three *plains* thus changing the linkage of loops in the full Sted1 graph is also of interest. Arithmetic is generally hard for SAT solvers, but these are smallish totals (up to 840 *bobs*) and are well handled by a bitonic sorting network which has a major advantage of coping with partial information as the solver progresses towards a solution. These networks can add up to 60,000 variables and 150,000 clauses but performance and solution times can still be acceptable.

With suitable additional constraints Erin7 can be proved UNSAT in less than a minute on a laptop. A search of Sted10 finds a solution takes 2.611 seconds (Glucose) or 21 seconds (MiniSat) depending on options compared to 21 seconds for the first solution from a depth-first search. All four solutions to Sted10 can be found in a few minutes. Erin3 and Sted5 can be searched and proved UNSAT by a supercomputer cluster over a period of a week or two. Restricted versions of Sted3 can be searched, but it appears full versions of Sted3 and Erin2 are too big to search without some hints as to possible solutions. This is a significant advance over direct tree searches, where Sted6 and Erin5 searches required several weeks on a 1000 node supercomputer. A complete search of Sted5 was completed in less than a week (Glucose) on the Flinders University Colossus supercomputer with over 1000 nodes. Unexpectedly [22] no results were found other than the already known Sted10 blocks. A complete search for Erin5 was completed in 10 minutes on a Thinkpad W520 with an Intel Core I7-2720QM 2.2GHz processor running a Windows/Cygwin version of Glucose 4 [24] (single threaded) and MiniSat [29] V1.14. Previously a depth-first search had taken many weeks on Colossus. No solutions were found. A complete search of the Erin4 graphs (3 different groups) takes a few hours. A complete search of Erin3 was perfomed on Colossus in about 2 weeks, but there is no solution. A solution to Erin2 was found by seeding a search with part of a solution to Erin20. These new SAT problems will be good benchmarks for SAT solvers. Some were included in SAT Competition 2018[7] (see Main.zip final/Johnson).

The encoding of Stedman Triples in SAT made possible some restricted Sted3 searches which allowed the author to find some new bobs-only *peals* using just a Thinkpad W550s with an Intel Core i7-5600U 2.60GHz processor. One of these *peals* was rung at the church of St James Garlickhythe, London, by a band of 8 ringers from the Cambridge University Guild of Change Ringers, in a performance[9] lasting 2 hours and 46 minutes of non-stop ringing. Since then the composition has been performed another four times.

This new encoding of Hamiltonian Cycle type problems to Boolean Satisfiability using LFSR sequences will have general applicability — not just for pure Hamiltonian Cycle problems but for many other problems with permutation or ordering type constraints.

7 Acknowledgments

Michael Haythorpe has taken a special interest in the Stedman and Erin Triples problems since I posed them to him. I am very appreciative of his correspondence concerning my approaches to solving the problems and thank him for running many tests using SAT solvers on files generated using techniques in this paper, including searches for the Sted5 and Erin3 graphs using the Colossus supercomputer, and previously tree-searches for Erin6, Erin5 and Sted6.

References

[1] Nadia Creignou. The class of problems that are linearly equivalent to satisfiability or a uniform method for proving NP-completeness. *Theoretical Computer Science*, 145(1):111–145, 1995. Available at http://www.sciencedirect.com/science/article/pii/030439759400182I.

[2] FHCP. Flinders Hamiltonian cycle project. http://www.flinders.edu.au/science_engineering/csem/research/programs/flinders-hamiltonian-cycle-project/.

[3] G. B. Fitzpatrick. Synthesis of binary ring counters of given periods. *Journal of the ACM (JACM)*, 7(3):287–297, July 1960. Available at http://doi.acm.org/10.1145/321033.321042.

[4] Fortran Friends. Animated rounds. http://fortran.orpheusweb.co.uk/Bells/software/rounds.htm.

[5] M. Haythorpe. FHCP challenge set. http://fhcp.edu.au/fhcpcs, 2015.

[6] Michael Haythorpe and Andrew Johnson. Change ringing and Hamiltonian cycles : The search for Erin and Stedman Triples. *ArXiv e-prints*, February 2017. Available at https://arxiv.org/pdf/1702.02623.

[7] Marijn Heule, Matti Jarvisalo, and Martin Suda. SAT competition 2018. http://sat2018.forsyte.tuwien.ac.at/index.php, July 2018.

[8] Kazuo Iwama and Shuichi Miyazaki. SAT-variable complexity of hard combinatorial problems. In *In Proceedings of the World Computer Congress of the IFIP*, pages 253–258. ELSEVIER SCIENCE B.V, August 1994. Available at http://www.lab2.kuis.kyoto-u.ac.jp/~shuichi/research/papers/IFIP94-final-2.ps and http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=3B98A51461E46170920BB24623539A17?doi=10.1.1.45.9473&rep=rep1&type=pdf.

[9] Andrew Johnson. 5040 Stedman Triples rung at St James Garlickhythe, City of London. http://bb.ringingworld.co.uk/view.php?id=1191078.

[10] Andrew Johnson. Bobs-only Stedman Triples made easy. *The Ringing World*, page 841, August 1995. Available at https://web.archive.org/web/20180723083751/http://myweb.tiscali.co.uk/saddleton/stedman/10part.html.

[11] Andrew Johnson. Quasi-linear reduction of Hamiltonian cycle problem (HCP) to satisfiability problem (SAT). Disclosure Number IPCOM000237123D, IP.com, Fairport, NY, June 2014. Available at https://priorart.ip.com/IPCOM/000237123.

[12] Andrew Johnson. Minimal decode state machines and counters. Disclosure Number IPCOM000242760D, IP.com, Fairport, NY, August 2015. Available at https://priorart.ip.com/IPCOM/000242760.

[13] Andrew Johnson. Three-part bobs-only peals of Stedman Triples. *The Ringing World*, 5565:1264–1265, December 2017. Available at http://bb.ringingworld.co.uk/issues/2017/1264.

[14] Andrew Johnson and Philip A. B. Saddleton. 5,040 Stedman Triples by Andrew Johnson and Philip A. B. Saddleton. *The Ringing World*, page 227, March 1995. Available at http://www.ringing.org/composition/?id=1742.

[15] Robert Royce Johnson. Binary coded flip-flop counters. US Patent 2,853,238, United States Patent and Trademark Office, Washington, DC, September 1958. Available at https://patents.google.com/patent/US2853238.

[16] M.Nasu. HCP to SAT conversion cubic time algorithm. http://www.aya.or.jp/~babalabo/MMS/HCP2SAT.html, June 2000.

[17] Anatoly D. Plotnikov. Designing SAT for HCP. *CoRR*, cs.LO/9903006, March 1999. Available at https://arxiv.org/pdf/cs/9903006.

[18] Steven Prestwich. SAT problems with chains of dependent variables. *Discrete Applied Mathematics*, 130(2):329–350, 2003. Available at https://www.sciencedirect.com/science/article/pii/S0166218X02004109/pdfft?md5=c555c70931163b164357bec187c07866&pid=1-s2.0-S0166218X02004109-main.pdf.

[19] Brian D. Price. The composition of peals in parts. http://www.ringing.info/bdp/peals-in-parts/parts-0.html.

[20] Brian D. Price. More about q-set laws (letter). *The Ringing World*, page 219, April 1953. Available at http://www.ringing.info/bdp/q-set_parity_law/rw_1953_219.pdf.

[21] Brian D. Price. Mathematical groups in campanology. *The Mathematical Gazette*, 53(384):129–133, May 1969. Available at https://www.jstor.org/stable/3614536.

[22] Philip A. B. Saddleton. Letter. *The Ringing World*, page 219, 1995.

[23] Philip A. B. Saddleton. *A Collection of Compositions of Stedman Triples and Erin Triples.* Central Council of Church Bell Ringers, 1999. Available at http://www.ringing.info/stedman.pdf.

[24] Laurent Simon and Gilles Audemard. The Glucose SAT solver. http://www.labri.fr/perso/lsimon/glucose/.

[25] simonbellringer. Stedman Triples. https://youtu.be/6tIXD9lDjUU.

[26] Takehide Soh, Daniel Le Berre, Stéphanie Roussel, Mutsunori Banbara, and Naoyuki Tamura. Incremental SAT-based method with native boolean cardinality handling for the Hamiltonian cycle problem. In Eduardo Fermé and João Leite, editors, *Logics in Artificial Intelligence*, pages 684–693, Cham, 2014. Springer International Publishing. Available at http://kix.istc.kobe-u.ac.jp/~soh/scarab/jelia2014/soh14jelia-preprint.pdf.

[27] Mate Soos. CryptoMiniSat SAT solver. https://github.com/msoos/cryptominisat.

[28] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2009.

[29] Niklas Sörensson and Niklas Eén. MiniSat. http://minisat.se.

[30] J. Armiger Trollope. *Stedman*. Whitehead and Miller Ltd., Leeds, 1938. Available locations at http://www.worldcat.org/title/stedman/oclc/56925246.

[31] Miroslav Velev and Ping Gao. Efficient SAT techniques for absolute encoding of permutation problems: Application to hamiltonian cycles. In *Eighth Symposium on Abstraction, Reformulation, and Approximation*, Jul 2009. Available at https://aaai.org/ocs/index.php/SARA/SARA09/paper/view/837/1167 and http://www.miroslav-velev.com/Velev_and_Gao_SARA09.pdf.

[32] Miroslav N. Velev and Ping Gao. Efficient SAT techniques for relative encoding of permutations with constraints. In *Proceedings of the 22Nd Australasian Joint Conference on Advances in Artificial Intelligence*, AI '09, pages 517–527, Berlin, Heidelberg, 2009. Springer-Verlag. Available at https://link.springer.com/chapter/10.1007/978-3-642-10439-8_52.

[33] Miroslav N. Velev and Ping Gao. Design of parallel portfolios for SAT-based solving of Hamiltonian cycle problems. In *ISAIM*, 2010. Available at https://pdfs.semanticscholar.org/a73e/66bdaf1b837b2db4f16b41adc54d4f44b62e.pdf?_ga=2.269048497.1353865287.1523865728-1582468662.1523865728.

[34] Laung-Terng Wang and Edward J. McCluskey. Hybrid designs generating maximum-length sequences. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 7(1):91–99, January 1988. Available at https://ieeexplore.ieee.org/document/3134/.

[35] Roy Ward and Timothy C. A. Molteno. Table of linear feedback shift registers. Electronics Technical Report 2012-1, University of Otago, Dunedin, New Zealand, 2012. Available at http://www.physics.otago.ac.nz/reports/electronics/ETR2012-1.pdf.

[36] A. T. White. Ringing the cosets. *Amer. Math. Month*, 94:721–746, October 1987. Available at https://www.jstor.org/stable/2323414.

[37] Colin J. E. Wyld. a 250-year-old problem solved. *The Ringing World*, pages 197–198, 1995.