Computer-aided Constructions of Commafree Codes

Aaron A. Windsor
aaron.windsor@gmail.com

Abstract

We determine the maximum size $W_k(n)$ of a commafree code with codeword length k and alphabet size n for a few previously unknown values of k and n. With the aid of modern SAT solver tooling we prove that $W_4(5) = 139$, $W_6(3) = 113$, and $W_{12}(2) = 334$ and exhibit codes that achieve these bounds.

1 Introduction

A set C of k-letter words over an n-letter alphabet is called a *commafree code* if whenever $a = a_1a_2 \dots a_k$ and $b = b_1b_2 \dots b_k$ are in C, none of the substrings of ab of length k that overlap both a and b ($a_2a_3 \dots a_kb_1,\ a_3a_4 \dots a_kb_1b_2,\ \dots,\ a_kb_1 \dots b_{k-1}$) are in C.

The k-letter words over an n-letter alphabet can be partitioned into equivalence classes defined by cyclic rotations ($[a_1a_2 \dots a_k] = \{a_1a_2 \dots a_k,\ a_2a_3 \dots a_1,\ \dots,\ a_ka_1 \dots a_{k-1}\}$). These equivalence classes contain up to k words each, but any word appearing in an equivalence class with fewer than k distinct rotations cannot appear in a commafree code, since then the self-concatenation ww of any word w in that class would also contain w as an overlap. Furthermore, at most one word w from each equivalence class of size k can appear in a commafree code because all other rotations of w appear in overlaps of ww.

Golumb, Gordon and Welch [6] showed that the number of such equivalence classes of size k for k-letter words over an n-letter alphabet is counted by

$$L_k(n) = \frac{1}{k} \sum_{d|k} \mu(d) \cdot n^{k/d} \tag{1}$$

where $\mu(d)$ is the Möbius function, defined as

$$\mu(d) = \begin{cases} 1 & \text{if } d = 1 \\ 0 & \text{if } d \text{ has any squared prime factor} \\ (-1)^r & \text{otherwise, where } d \text{ has exactly } r \text{ distinct prime factors} \end{cases}$$

Let $W_k(n)$ be the maximum size of a commafree code consisting of k-letter words over an n-letter alphabet. Golumb, Gordon, and Welch [6] proved that $W_k(n) \leqslant L_k(n)$ for

positive n, k and $W_k(n) = L_k(n)$ for odd k up to 15. Eastman [4] extended the latter results constructively to show that $W_k(n) = L_k(n)$ for all odd k.

For even k, much less is known. In [6], it is proven that $W_2(n) = \lfloor \frac{1}{3}n^2 \rfloor$ and $W_4(3) = L_4(3) = 18$. Jiggs [10] reports computer experiments that show $W_8(2) = L_8(2) = 30$ and $W_4(4) = 57 < L_4(4) = 60$. Niho [15] reports a backtracking program that determined $W_{10}(2) = L_{10}(2) = 99$. Knuth [11] presents a backtracking program that can determine $W_4(n)$ easily for $n \leqslant 4$.

We summarize these known results in Table 1, showing differences between $L_k(n)$ and $W_k(n)$ for small values of k and n with blank entries where $W_k(n)$ is unknown. New results from this paper appear with an asterisk.

Table 1: $L_k(n) - W_k(n)$

k \ n	2	3	4	5	6	7	8	9	10	
2	0	0	1	2	3	5	7	9	12	...
3	0	0	0	0	0	0	0	0	0	...
4	0	0	3	11*						
5	0	0	0	0	0	0	0	0	0	...
6	0	3*								
7	0	0	0	0	0	0	0	0	0	...
8	0									
9	0	0	0	0	0	0	0	0	0	...
10	0									
11	0	0	0	0	0	0	0	0	0	...
12	1*									
13	0	0	0	0	0	0	0	0	0	...
14										

In this paper, we describe the encoding of the search for $W_k(n)$ values into propositional formulas and report on new results obtained by applying a state-of-the-art satisfiability (SAT) solver to help prove those formulas satisfiable or unsatisfiable. The general technique of encoding existence proofs for combinatorial objects into propositional formulas and using SAT solvers to determine satisfiability of such formulas has been applied with great success to many longstanding open problems in mathematics in recent years, including the van der Waerden number $W(2, 6)$ by Kouril and Paul [13], special cases of the Erdős discrepancy conjecture by Konev and Lisitsa [12], the boolean Pythagorean triples problem by Heule, Kullman, and Marek [8], and Schur number five by Heule [7].

SAT solvers have also facilitated the exploration and verification of coding theory results that previously could only be generated with complicated backtracking programs. Recent examples include Zinovik, Kroening, and Chebiryak's generation of various Combinatorial Gray Codes [16] and Bright, Cheung, Stevens, Kotsireas, and Ganesh's verification of the non-existence of weight 16 codewords in Lam's Problem [2].

2 Preliminaries

Our encodings use propositional formulas in conjunctive normal form (CNF): conjunctions of disjunctive clauses. We will sometimes refer to formulas as clauses when there's a straightforward transformation to a disjunction. When we describe a formula with a set of clauses, it's implied that the clauses in the set are AND-ed together to create a CNF formula.

When s and t are two strings, st indicates the concatenation of s and t. We say $s \sqsubset t$ when s is a substring of t but *not* a prefix or suffix of t.

We will use $[n]$, which represents the set $\{0, 1, \ldots, n-1\}$, as an alphabet. Raising an alphabet to the kth power yields the set of all strings of length k that can be formed from that alphabet.

Finally, in addition to the notation $L_k(n)$ and $W_k(n)$ already introduced, we will use $P_k(n)$ to represent the set of all non-periodic k-letter words over $[n]$. Since $L_k(n)$ counts the number of equivalence classes of non-periodic k-letter words over $[n]$ under rotation, $|P_k(n)| = k \cdot L_k(n)$.

3 SAT Encoding

We encode a series of parameterized propositional formulas $\phi(k, n, m)$ in conjunctive normal form, each of which is satisfiable iff a commafree code of size $L_k(n) - m$ exists with codewords of length k over an alphabet of size n. With such formulas, we can incrementally use a SAT solver to find the smallest $m' \geqslant 0$ such that $\phi(k, n, m')$ is satisfiable, from which we can conclude $W_k(n) = L_k(n) - m'$.

The formulas ϕ have variables X_w for each word $w \in P_k(n)$ as well as variables $X_{[w]}$ for each equivalence class $[w]$ of size k. For each equivalence class $[w_1 w_2 \ldots w_k]$, these variables are connected via the clause $X_{[w_1 w_2 \ldots w_k]} \implies (X_{w_1 w_2 \ldots w_k} \vee X_{w_2 \ldots w_k w_1} \vee \cdots \vee X_{w_k w_1 \ldots w_{k-1}})$.

The bulk of the clauses in each formula encode the commafree property. A straightforward encoding of the commafree property would take each ordered pair of words $x = x_1 x_2 \ldots x_k$ and $y = y_1 y_2 \ldots y_k$ from $P_k(n) \times P_k(n)$ and generate the $k - 1$ clauses $\neg(X_x \wedge X_y \wedge X_{x_2 \ldots x_k y_1})$, $\neg(X_x \wedge X_y \wedge X_{x_3 \ldots y_1 y_2})$, $\ldots$, $\neg(X_x \wedge X_y \wedge X_{x_k \ldots y_{k-2} y_{k-1}})$. Instead of this simple encoding, which can result in huge formulas for the values of k and n we were interested in, we use a special compressed encoding of the commafree property that is discussed in detail in the next subsection.

To encode the cardinality constraint m, we introduce additional variables and clauses that apply a comparator network to the $X_{[w]}$, selecting the lowest m values. In the resulting reordered list of $X_{[w]}$, we use one or two unit clauses (depending on whether $m = 0$) to assert that exactly m of the $X_{[w]}$ are false. We experimented with optimal constructions of sorting and selection networks and with BDD-based cardinality constraints, but in the end, found not much difference in the running time of the solver between any of these techniques. As a result, we used the simplest possible encoding of a selection network consisting of $m + 1$ layers of comparators, with layer i consisting of comparators that simulate bubble sort to place the minimum of the $m - i$ remaining elements into position

i. For a discussion of options for efficiently encoding cardinality constraints into boolean formulas, see [5].

To break symmetry in hopes of speeding up unsatisfiable instances, we follow Knuth [11] and add unit clauses that prohibit the lexicographically smallest $\lfloor \frac{k}{2} \rfloor$ words in the single equivalence class $[0 \ldots 01]$ from being chosen. This is a valid restriction because $[0 \ldots 01]$ is closed under reversal and any commafree code is also commafree when all words are reversed. In experiments with smaller values of k and n, we found this symmetry-breaking technique could roughly halve the solving time of unsatisfiable instances.

3.1 An Improved Encoding of the Commafree Property

Since $|P_k(n)| = k \cdot L_k(n)$ and since $L_k(n) = \mathcal{O}(\frac{1}{k} n^k)$ by (1), a straightforward encoding of the commafree property using ternary clauses $\overline{X}_s \vee \overline{X}_t \vee \overline{X}_u$ for every $t \sqsubset su$ requires $\mathcal{O}(k^3 L_k(n)^2) = \mathcal{O}(kn^{2k})$ clauses for words of length k over an alphabet of size n. This can generate billions of clauses for even small values of k and n. In this section, we demonstrate a compressed encoding of the commafree property, proving the following:

Proposition 1. *For $k, n > 0$, the commafree property*

$$\bigwedge_{\substack{s,t,u \in P_k(n) \\ t \sqsubset su}} \overline{X}_s \vee \overline{X}_t \vee \overline{X}_u$$

can be encoded in $\mathcal{O}(n^{\frac{3}{2}k})$ clauses using $\mathcal{O}(n^{\frac{k}{2}})$ additional variables.

Our compressed encoding is achieved by factoring the original clauses using bounded variable addition (BVA) [14]. BVA is a preprocessing technique that can reduce the number of clauses in an encoding by adding variables. BVA relies on the Davis–Putnam clause distribution rule [3] to create an equisatisfiable formula by replacing a set of clauses with a new variable and a smaller set of clauses.

Given clauses $C_1 = x \vee a_1 \vee \cdots \vee a_n$ and $C_2 = \bar{x} \vee b_1 \vee \cdots \vee b_m$, resolution allows us to infer the clause $C_1 \otimes C_2 = a_1 \vee \cdots \vee a_n \vee b_1 \vee \cdots \vee b_m$. For sets of clauses S_x and $S_{\bar{x}}$ where every clause in S_x contains the literal x and every clause in $S_{\bar{x}}$ contains the literal $\bar{x}$, $S_x \otimes S_{\bar{x}}$ can be defined naturally as $\{C_1 \otimes C_2 \mid C_1 \in S_x \text{ and } C_2 \in S_{\bar{x}}\}$. The Davis–Putnam clause distribution rule allows the clauses $S_x \cup S_{\bar{x}}$ to be replaced by $S_x \otimes S_{\bar{x}}$ to create an equisatisfiable CNF formula when S_x and $S_{\bar{x}}$ contain all occurrences of the variable x from the formula. BVA applies this transformation in reverse when $S_x \otimes S_{\bar{x}}$ is larger than $S_x \cup S_{\bar{x}}$, adding a single variable to the formula while reducing the number of clauses.

Recall that the commafree property can be expressed by ternary clauses $\overline{X}_s \vee \overline{X}_t \vee \overline{X}_u$ where $t \sqsubset su$. Since s always has a non-empty overlap with t and t always has a non-empty overlap with u, we can introduce a new variable and use resolution to define a subset of the ternary clauses, either by parameterizing the subset by the overlap p between t and u:

$$\{\overline{X}_{ab} \vee \overline{X}_{bp} \vee v_p \mid ab \in P_k(n), bp \in P_k(n)\} \otimes \{\bar{v}_p \vee \overline{X}_{pc} \mid pc \in P_k(n)\}$$

or by parameterizing the subset by the overlap q between s and t:

$$\{\overline{X}_{aq} \vee v'_q \mid aq \in P_k(n)\} \otimes \{\bar{v}'_q \vee \overline{X}_{qb} \vee \overline{X}_{bc} \mid qb \in P_k(n), bc \in P_k(n)\}$$

Any ternary clause $\overline{X}_s \vee \overline{X}_t \vee \overline{X}_u$ describing the commafree property can be generated by either construction above by choosing an appropriate p or q. By always choosing an appropriately small subset to generate any particular ternary clause, we can prove Proposition 1:

Proof of Proposition 1. We claim that the conjunction of:

$$\bigwedge_{\substack{p \in [n]^i \\ 0 < i \leqslant \lceil \frac{k}{2} \rceil}} \{\overline{X}_{ab} \vee \overline{X}_{bp} \vee v_p \mid ab \in P_k(n), bp \in P_k(n)\} \cup \{\bar{v}_p \vee \overline{X}_{pc} \mid pc \in P_k(n)\} \tag{2}$$

and

$$\bigwedge_{\substack{q \in [n]^i \\ 0 < i \leqslant \lfloor \frac{k}{2} \rfloor}} \{\overline{X}_{aq} \vee v'_q \mid aq \in P_k(n)\} \cup \{\bar{v}'_q \vee \overline{X}_{qb} \vee \overline{X}_{bc} \mid qb \in P_k(n), bc \in P_k(n)\} \tag{3}$$

has the desired properties.

If $k = 1$, the commafree property is empty and both (2) and (3) are empty. Otherwise, for $k > 1$ and any $s, t, u \in P_k(n)$ with $t \sqsubset su$, t either has a suffix of length at most $\lceil \frac{k}{2} \rceil$ that overlaps u or a prefix of length at most $\lfloor \frac{k}{2} \rfloor$ that overlaps s. In the former case, $\overline{X}_s \vee \overline{X}_t \vee \overline{X}_u$ can be generated by resolving clauses from (2) and in the latter case it can be generated by resolving clauses from (3).

Fix a p and consider the two sets of clauses defined by p in each term of (2). Those containing $\overline{v}_p$ clearly have size at most $n^{k-|p|}$. Those containing v_p are completely defined by the choice of $ab \in P_k(n)$, hence there are at most $|P_k(n)| = k \cdot L_k(n)$ of these. So the total number of clauses in the encoding (2) is bounded from above by $\sum_{i=1}^{\lceil \frac{k}{2} \rceil} n^i (k \cdot L_k(n) + n^{k-i})$. Since by (1), $L_k(n) = \mathcal{O}(\frac{1}{k} n^k)$, and since by a similar argument as above, there are no more clauses in (3) than in (2), the total number of clauses is $\mathcal{O}(n^{\frac{3}{2}k})$. □

The savings from this compressed encoding for formulas on the frontier between known and unknown values of $W_k(n)$ are substantial. A straightforward encoding of $\phi(14, 2, 0)$ using all ternary clauses $\overline{X}_s \vee \overline{X}_t \vee \overline{X}_u$ produces a 70 GB input file with about 19,000 variables and over 3 billion clauses, but the compressed encoding is only 130 MB with about 20,000 variables and a little over 6 million clauses.

4 Results

We used the `kissat`[1] SAT solver developed by Biere [1]. Where we could determine $W_k(n)$ exactly by finding an unsatisfiable $\phi(k, n, m-1)$ and a satisfiable $\phi(k, n, m)$, we

[1] http://github.com/arminbiere/kissat

verified unsatisfiable results using both the `drabt`[2] and `drat-trim`[3] proof checkers. All satisfiable results were verified by a simple script that converts the satisfying assignment to a commafree code and verifies its size and the commafree property. Code to generate input files for the formulas $\phi(k, n, m)$ as well as extract and verify the resulting commafree codes is available at `http://github.com/aaw/commafree`.

Using the methods described above and running `kissat` on commodity virtualized hardware[4], we were able to verify all previously known values of $W_k(n)$ for even $k > 2$ ($W_4(2), W_4(3), W_4(4), W_6(2), W_8(2)$, and $W_{10}(2)$) in a few seconds.

We were also able to determine the following new results:

- $W_6(3) = L_6(3) - 3 = 113$
- $W_{12}(2) = L_{12}(2) - 1 = 334$
- $W_4(5) = L_4(5) - 11 = 139$

Codes achieving these three bounds appear in the appendix.

Exact values of $W_k(n)$ for other unknown combinations of k and n could not be obtained within a reasonable amount of time. In many cases, even slightly larger values of k and n could cause `kissat` to run for several weeks without terminating. Solving time for unsatisfiable instances could be significantly improved by running a BVA preprocessor[5] on the formulas $\phi(k, n, m)$ to create formulas $\phi'(k, n, m)$ with 2 to 3 times fewer clauses. The translation to ϕ' using BVA preprocessing did not speed up the solving time on most satisfiable instances.

Using these techniques, we were able to discover the following inequalities for a few previously unexplored values of $W_k(n)$:

- $L_4(6) - 24 \leqslant W_4(6) < L_4(6) - 20$
- $L_6(4) - 33 \leqslant W_6(4) < L_6(4) - 9$
- $L_8(3) - 29 \leqslant W_8(3) < L_8(3) - 6$
- $L_{14}(2) - 10 \leqslant W_{14}(2) < L_{14}(2) - 1$

Formulas solved and `kissat` execution times are summarized in Table 2 below.

5 Conclusion

We have presented new results on the largest achievable commafree codes for several values of k and n. These results were obtained by running a SAT solver on a novel encoding of the problem.

$W_4(6)$ should be the next easiest value to determine exactly, but `kissat` ran for several weeks on $\phi(4, 6, m)$ and $\phi'(4, 6, m)$ for $m \in \{21, 22, 23\}$ without terminating. Attempts

[2] http://fmv.jku.at/drabt/
[3] http://github.com/marijnheule/drat-trim
[4] http://www.linode.com
[5] http://github.com/marijnheule/bva

Table 2: Best Solving Times

(n, k)	ϕ' with largest m found UNSAT	ϕ with smallest m found SAT
$(6, 3)$	$\phi'(6, 3, 2)$ (2 minutes)	$\phi(6, 3, 3)$ (< 1 second)
$(12, 2)$	$\phi'(12, 2, 0)$ (30 minutes)	$\phi(12, 2, 1)$ (7 minutes)
$(4, 5)$	$\phi'(4, 5, 10)$ (1 hour)	$\phi(4, 5, 11)$ (< 1 minute)
$(4, 6)$	$\phi'(4, 6, 20)$ (27 days)	$\phi(4, 6, 24)$ (9 minutes)
$(6, 4)$	$\phi'(6, 4, 9)$ (9 days)	$\phi(6, 4, 33)$ (36 minutes)
$(8, 3)$	$\phi'(8, 3, 6)$ (28 days)	$\phi(8, 3, 29)$ (12 hours)
$(14, 2)$	$\phi'(14, 2, 1)$ (6 days)	$\phi(14, 2, 10)$ (1 day)

at solving the formulas related to $W_4(6)$ and $W_6(4)$ in parallel, both with state-of-the-art cube-and-conquer [9] solvers and with custom cubing, were unsuccessful.

Clever symmetry-breaking techniques that we haven't considered or additional lemmas that account for combinatorial properties of commafree codes added to the formulas ϕ could improve solver performance and allow at least $W_4(6)$ to be determined exactly. Without such additions to the formulas, future improvements in SAT solver technology may allow other values of $W_k(n)$ to be discovered with the techniques in this paper.

6 Acknowledgements

We thank both of the anonymous reviewers for many helpful suggestions and corrections that improved the presentation of this paper. In particular, one reviewer noticed that a BVA preprocessor substantially reduced the original encoding size, which inspired the improved incoding in Section 3.1.

References

[1] Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020.

[2] Curtis Bright, Kevin K.H. Cheung, Brett Stevens, Ilias Kotsireas, and Vijay Ganesh. Unsatisfiability proofs for weight 16 codewords in Lam's problem. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 1460–1466. International Joint Conferences on Artificial Intelligence Organization, 7 2020. Main track.

[3] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201215, July 1960.

[4] W. Eastman. On the construction of comma-free codes. *IEEE Transactions on Information Theory*, 11(2):263–267, 1965.

[5] Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, pages 1–26, 2006.

[6] S. W. Golomb, Basil Gordon, and L. R. Welch. Comma-free codes. *Canadian Journal of Mathematics*, 10:202209, 1958.

[7] Marijn J. H. Heule. Schur number five. In *AAAI*, 2018.

[8] Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and Verifying the boolean Pythagorean Triples problem via Cube-and-Conquer. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing – SAT 2016*, pages 228–245, Cham, 2016. Springer International Publishing.

[9] Marijn J. H. Heule, Oliver Kullmann, Siert Wieringa, and Armin Biere. Cube and Conquer: Guiding CDCL SAT Solvers by Lookaheads. In Kerstin Eder, João Lourenço, and Onn Shehory, editors, *Hardware and Software: Verification and Testing*, pages 50–65, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[10] B. H. Jiggs. Recent results in comma-free codes. *Canadian Journal of Mathematics*, 15:178–187, 1963.

[11] Donald E. Knuth. *Combinatorial Algorithms, Part 2*, volume 4B of *The Art of Computer Programming*. Addison-Wesley Professional, first edition, October 2022.

[12] Boris Konev and Alexei Lisitsa. Computer-Aided Proof of Erdős Discrepancy Properties. *Artificial Intelligence*, 224:103–118, 2015.

[13] Michal Kouril and Jerome L. Paul. The van der Waerden Number $W(2, 6)$ Is 1132. *Experimental Mathematics*, 17(1):53 – 61, 2008.

[14] Norbert Manthey, Marijn J. H. Heule, and Armin Biere. Automated reencoding of boolean formulas. In Armin Biere, Amir Nahir, and Tanja Vos, editors, *Hardware and Software: Verification and Testing*, pages 102–117, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[15] Yoji Niho. On maximal comma-free codes (corresp.). *IEEE Trans. Inf. Theory*, 19:580–581, 1973.

[16] Igor Zinovik, Daniel Kroening, and Yury Chebiryak. Computing binary combinatorial gray codes via exhaustive search with sat solvers. *IEEE Transactions on Information Theory*, 54(4):1819–1823, 2008.

7 Appendix

A code that achieves $W_4(5) = 139$:

$\{0012, 0013, 0014, 0100, 0102, 0103, 0104, 0110, 0111, 0112, 0113,$
$0114, 0204, 0212, 0213, 0214, 0304, 0312, 0313, 0314, 1012, 1013,$
$1014, 2000, 2002, 2003, 2004, 2012, 2013, 2014, 2022, 2023, 2024,$
$2032, 2033, 2034, 2100, 2102, 2103, 2104, 2110, 2111, 2112, 2113,$
$2114, 2204, 2212, 2213, 2214, 2224, 2234, 2304, 2312, 2313, 2314,$
$2324, 2334, 3000, 3002, 3003, 3004, 3012, 3013, 3014, 3022, 3023,$
$3024, 3032, 3033, 3034, 3100, 3102, 3103, 3104, 3110, 3111, 3112,$
$3113, 3114, 3204, 3212, 3213, 3214, 3224, 3234, 3304, 3312, 3313,$
$3314, 3324, 3334, 4000, 4002, 4003, 4004, 4012, 4013, 4014, 4022,$
$4023, 4024, 4032, 4033, 4034, 4042, 4043, 4044, 4100, 4102, 4103,$
$4104, 4110, 4111, 4112, 4113, 4114, 4122, 4123, 4124, 4132, 4133,$
$4134, 4142, 4143, 4144, 4204, 4212, 4213, 4214, 4224, 4234, 4244,$
$4304, 4312, 4313, 4314, 4324, 4334, 4344\}$

The code above omits representatives from the following equivalence classes:

$[0203], [0412], [0413], [0414], [1213], [1214], [1314], [2223], [2233], [2333], [2434]$

A code that achieves $W_6(3) = 113$:

$\{001000, 001100, 001101, 001102, 001200, 001201, 001202, 002000, 002100,$
$002101, 002102, 002200, 002201, 002202, 010102, 011100, 011101, 011102,$
$011110, 011111, 011200, 011201, 011202, 011210, 011211, 012100, 012101,$
$012102, 012110, 012111, 012112, 012200, 012201, 012202, 012210, 012211,$
$012220, 020100, 020102, 020110, 020120, 020200, 020210, 020220, 021100,$
$021101, 021102, 021110, 021111, 021120, 021121, 021200, 021201, 021202,$
$021210, 021211, 022100, 022101, 022102, 022110, 022111, 022112, 022120,$
$022121, 022200, 022201, 022202, 022211, 022212, 101000, 101100, 101200,$
$101201, 101202, 102000, 102100, 102200, 102201, 102202, 111200, 111201,$
$111202, 111210, 111211, 112200, 112201, 112202, 112210, 112211, 112220,$
$121200, 121201, 121202, 121210, 121211, 122120, 122121, 122211, 122212,$
$212200, 212201, 212202, 212210, 212211, 212220, 222100, 222101, 222102,$
$222200, 222201, 222202, 222211, 222212\}$

The code above omits representatives from the following equivalence classes:

$$[001002], [011021], [012022]$$

Finally, a code that achieves $W_{12}(2) = 334$:

$$\{000001000011, 000010000000, 000010000001, 000010000011, 000010000101,$$
$$000100000101, 000100001001, 000101000011, 000101001011, 000110000001,$$
$$000110000011, 000110000101, 000110000111, 000110001001, 000110001011,$$
$$000110001101, 000110001111, 000110010001, 000110010011, 000110010101,$$
$$000110010111, 000110011001, 000110011011, 000110011101, 000110011111,$$
$$000111000011, 000111001011, 001000001001, 001000010001, 001001000011,$$
$$001001001011, 001001010011, 001010000001, 001010000011, 001010000101,$$
$$001010001001, 001010001011, 001010010001, 001010010011, 001010010101,$$
$$001100000001, 001100000101, 001100001001, 001100001101, 001100010001,$$
$$001100010101, 001100011101, 001101000011, 001101001011, 001101010011,$$
$$001110000001, 001110000011, 001110000101, 001110001001, 001110001011,$$
$$001110001101, 001110001111, 001110010001, 001110010011, 001110010101,$$
$$001110011001, 001110011011, 001110011101, 001110011111, 001111000000,$$
$$001111000011, 001111001011, 001111010011, 010000010001, 010001000011,$$
$$010001010011, 010010000001, 010010000011, 010010000101, 010010010001,$$
$$010010010011, 010010010101, 010010100011, 010100000001, 010100000101,$$
$$010100001001, 010100010001, 010100010101, 010101000011, 010101001011,$$
$$010101010011, 010110000001, 010110000011, 010110000101, 010110000111,$$
$$010110001001, 010110001011, 010110001101, 010110001111, 010110010001,$$
$$010110010011, 010110010101, 010110010111, 010110011001, 010110011011,$$
$$010110011101, 010110011111, 010110100011, 010110100111, 010110101011,$$
$$010111000011, 010111001011, 010111010011, 010111011011, 010111100011,$$
$$010111101011, 010111111011, 011000000000, 011000000001, 011000001001,$$
$$011000010001, 011001000011, 011001001011, 011001010011, 011001011011,$$
$$011001101011, 011001111011, 011010000001, 011010000011, 011010000101,$$
$$011010001001, 011010001011, 011010010001, 011010010011, 011010010101,$$
$$011010011001, 011010011011, 011010100011, 011010101011, 011100000000,$$
$$011100000001, 011100000101, 011100001001, 011100010001, 011100010101,$$
$$011100011101, 011101000011, 011101001011, 011101010011, 011101011011,$$
$$011101111011, 011110000011, 011110000101, 011110001001, 011110001011,$$
$$011110010001, 011110010011, 011110010101, 011110011001, 011110011011,$$

011110011101, 011110011111, 011110100011, 011110101011, 011111000011,
011111001011, 011111010011, 011111011011, 011111100011, 011111101011,
011111111011, 100001000011, 100010000000, 100010000001, 100010000011,
100010000101, 100010100011, 100100000000, 100100000001, 100100000101,
100100001001, 100101000011, 100101001011, 100110000001, 100110000011,
100110000101, 100110000111, 100110001001, 100110001011, 100110001101,
100110001111, 100110100011, 100110100111, 100110101011, 100111000011,
100111001011, 100111100011, 100111101011, 101000000000, 101000000001,
101000001001, 101000010001, 101001000011, 101001001011, 101001010011,
101010000001, 101010000011, 101010000101, 101010001001, 101010001011,
101010010001, 101010010011, 101010010101, 101010100011, 101010101011,
101100000001, 101100000101, 101100001001, 101100001101, 101100010001,
101100010101, 101100011101, 101101000011, 101101001011, 101101010011,
101101011011, 101110000001, 101110000011, 101110000101, 101110001001,
101110001011, 101110001101, 101110001111, 101110010001, 101110010011,
101110010101, 101110011001, 101110011011, 101110011101, 101110011111,
101110100011, 101110101011, 101111000000, 101111000011, 101111001011,
101111010011, 101111100011, 101111101011, 101111111011, 110000010001,
110001000011, 110001010011, 110010000001, 110010000011, 110010000101,
110010010001, 110010010011, 110010010101, 110010100011, 110100000001,
110100000101, 110100001001, 110100010001, 110100010101, 110101000011,
110101001011, 110101010011, 110110000001, 110110000011, 110110000101,
110110000111, 110110001001, 110110001011, 110110001101, 110110001111,
110110010001, 110110010011, 110110010101, 110110010111, 110110011001,
110110011011, 110110011101, 110110011111, 110110100011, 110110100111,
110110101011, 110111000011, 110111001011, 110111010011, 110111010111,
110111011011, 110111100011, 110111101011, 110111111011, 111000001001,
111000010001, 111001000011, 111001001011, 111001010011, 111001101011,
111001111011, 111010000001, 111010000011, 111010000101, 111010001001,
111010001011, 111010010001, 111010010011, 111010010101, 111010100011,
111010101011, 111100000101, 111100001001, 111100010001, 111100010101,
111101000011, 111101001011, 111101010011, 111110000000, 111110000001,
111110000011, 111110000101, 111110001001, 111110001011, 111110010001,
111110010011, 111110010101, 111110011001, 111110011011, 111110011101,
111110011111, 111110100011, 111110101011, 111111000011, 111111001011,
111111010011, 111111100011, 111111101011, 111111111011}

The code above omits a representative from only the equivalence class [000011001011].