Searching a $(33,16)$ Doubly-Even Code For a $(22,33,12,8,4)$-BIBD

Richard Bilous
Concordia University
Department of Computer Science
e-mail: umbilou1@cs.concordia.ca

Abstract: *It is known that if a $(22,33,12,8,4)$-BIBD exists then its incidence matrix is contained in a $(33,16)$ doubly-even self-orthogonal code (that does not contain a coordinate of zeros). There are 594 such codes, up to equivalence. It has been theoretically proven that 116 of these codes cannot contain the incidence matrix of such a design. For the remaining 478 codes, an exhaustive clique search may be tried, on the weight 12 words of a code, to determine whether or not it contains such an incidence matrix. Thus far, such a search has been used to show 299 of the 478 remaining codes do not contain the incidence matrix of a $(22,33,12,8,4)$-BIBD.*

In this paper, an outline of the method used to search the weight 12 words of these codes is given. The paper also gives estimations on the size of the search space for the remaining 179 codes. Special attention is paid to the toughest cases, namely the 11 codes that contain 0 weight 4 words and the 21 codes that contain one and only one weight 4 word.

1 Outline

Let C be a $(33,16)$ doubly-even self-orthogonal code over $GF(2)$. Our algorithm for determining whether or not C contains an incidence matrix for a $(22,33,12,8,4)$-BIBD works in two stages. In Sections 2, 3, and 4 we discuss the first stage of our algorithm. In this stage, we find a set L of $22 \times n$ matrices P, where $n < 33$, with the property that C contains an incidence matrix for our design if and only if C contains an incidence matrix $[P|Q]$, where $P \in L$. In Section 5, we discuss the second stage of our algorithm. In this stage, for each $P \in L$, we search the weight 12 words of C for an incidence matrix $[P|Q]$. In Section 6, we discuss the pruning we perform during our search. In Section 7, we briefly discuss the performance of our algorithm on the 299 codes we have searched. Finally, in Section 8, we give estimates on the size of the search space for the remaining 179 codes.

JCMCC **46** (2003), pp. 53-64

2 Word Blocks and Left Patterns

We use the term *word block* to refer to any set of codewords that may occur in the orthogonal complement of a $(33, 16)$ doubly-even self-orthogonal code. A *zero coordinate* of W is a coordinate in which every codeword in W has a value of 0. A *left word block* is a word block whose non-zero coordinates all occur to the left of its zero coordinates. In Figure 1, all the left word blocks we consider are given. Unless stated otherwise, whenever the term "word block" is used, we are referring to one of the left word blocks in Figure 1.

Let W be a left word block. We use the notation $n(W)$ to denote the number of non-zero coordinates in W. The notation $W_{(n(W))}$ is used to denote W with its last $33 - n(W)$ coordinates deleted. A *left pattern* of W is a $22 \times n(W)$ matrix P such that: 1) each row in P is orthogonal to every vector in $W_{(n(W))}$, 2) each row in P has weight less than or equal to 12, 3) each pair of rows in P *intersect* in at most 4 positions (i.e. there are at most 4 columns in which both rows have a value of one), and 4) each column in P has weight 8. In Figure 2, examples of a left pattern for the d_1-block and for the f_1-block are given.

Let C be a code whose orthogonal complement contains the word block W. If C contains an incidence matrix A for a $(22, 33, 12, 8, 4)$-BIBD then the $22 \times n(W)$ submatrix in the first $n(W)$ columns of A is a left pattern P of W. We refer to this left pattern P as the left pattern A "begins with".

For each of our left words blocks W, we have found a *complete set* $L(W)$ of left patterns of W, up to row and column rearrangement. What we mean by a "complete set" is a set that contains every left pattern P of W for which it has not be determined, by means other than our exhaustive search, that there does not exist an incidence matrix A that begins with P. In Table 1, $|L(W)|$ is given for each of our word blocks W. The seven left patterns of the f_1-block and the three left patterns of the d_1-block can be found in [2]

W	f_1	d_1	d_2	d_2^*	d_3	d_3^*	d_4
$\|L(W)\|$	7	3	11	8	20	2	14

Table 1: The size of the set $L(W)$ for each of the word blocks W in Figure 1.

$$\begin{array}{cccccc} 1 & 1 & 1 & 1 & 1 & 0^* \end{array}$$

the f_1-block

$$\begin{array}{ccccc} 1 & 1 & 1 & 1 & 0^* \end{array}$$

the d_1-block

$$\begin{array}{ccccccc} 1 & 1 & 1 & 1 & 0 & 0 & 0^* \\ 1 & 1 & 0 & 0 & 1 & 1 & 0^* \end{array}$$

the d_2-block

$$\begin{array}{ccccccccc} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0^* \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0^* \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0^* \end{array}$$

the d_2^*-block

$$\begin{array}{ccccccccc} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0^* \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0^* \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0^* \end{array}$$

the d_3-block

$$\begin{array}{cccccccccc} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0^* \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0^* \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0^* \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0^* \end{array}$$

the d_3^*-block

$$\begin{array}{ccccccccccc} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0^* \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0^* \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0^* \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0^* \end{array}$$

the d_4-block

Figure 1: Our left word blocks. The notation 0^* means just enough zeros to extend the codeword to one with 33 components.

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

pattern 4.0

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

pattern 2.0

Figure 2: Examples of left patterns. Pattern 4.0 is a left pattern of the f_1-block. Pattern 2.0 is a left pattern of the d_1-block.

3 The Word Block of a Code

Let L denote our set of 478 codes that we have not eliminated theoretically. The orthogonal complement of each C in L contains several word blocks that can be permuted, with a coordinate permutation, to at least one of the left word blocks in Figure 1. For each C in L, we select one word block in its orthogonal complement, and then permute the coordinates of C until the word block becomes the corresponding left word block in Figure 1. The running time of our search algorithm may be greatly affected by the word block we choose. Therefore, the word block we select is the one that we believe will result in the shortest running time.

We classify each code based on the word block we select for the code. For example, if for a code C the word block we select is permuted to a d_3-block, then we classify C as a d_3-code. For a given word block W, the notation $C(W)$ is used to denote the set of codes C in L that have been classified as W-codes. In Table 2, we list the number of codes in each class of codes. All 478 of these codes can be found at [1].

W	f_1	d_1	d_2	d_2^*	d_3	d_3^*	d_4
$\|C(W)\|$	11	132	122	115	44	17	37

Table 2: Number of codes in each class.

Thus far, we have search all of the d_4, d_3^*, d_3, and d_2^*-codes, and many of the d_2-codes. Among the codes we have not searched, our estimations have shown the toughest cases are the codes that have at most one weight 4 word. There are 11 codes that contain no weight 4 words (i.e. the 11 codes we classify as f_1-codes) and 21 codes that contain exactly one weight 4 word (all of which are d_1-codes).

4 The Left Patterns of a Code

Let W be a word block and let C be a code in $C(W)$. If C contains an incidence matrix A for a $(22, 33, 12, 8, 4)$-BIBD then the $22 \times n(W)$ submatrix in the first $n(W)$ columns of A must be a left pattern P of W.

Our algorithm for determining whether or not C contains an incidence matrix for a $(22, 33, 12, 8, 4)$-BIBD works by first producing a set $L(W, C)$, of left patterns of W, with the property that C contains an incidence matrix for a $(22, 33, 12, 8, 4)$-BIBD if and only if C contains an incidence matrix

that begins with one of the left patterns in $L(W,C)$. For each $P \in L(W,C)$, our algorithm then searches the weight 12 words of C for an incidence matrix that begins with the left pattern P. If for each $P \in L(W,C)$, our algorithm does not find an incidence matrix that begins with P, then we can conclude C does not contain an incidence matrix for a $(22,33,12,8,4)$-BIBD.

The set of left patterns $L(W)$ for the word block W is used to find the set $L(W,C)$. By definition of $L(W)$, if C contains an incidence matrix that begins with a left pattern P_1, then $L(W)$ must contain one and only one left pattern P_2 such that the rows and columns of P_1 are a permutation of the rows and columns of P_2. If P_1 is simply a permutation of the rows of P_2 then C must also contain an incidence matrix that begins with P_2. However, if we must also apply a column permutation π_c to P_1, in order to produce P_2, then it is not necessarily true that C also contains an incidence matrix that begins with P_2. (The reason for this is, of course, that when we apply π_c to P_1 we are also permuting the coordinates of C, and thus all we can say, in the most general sense, is that there must exist a code *equivalent* to C that contains an incidence matrix that begins with P_2.) Therefore, for each $P \in L(W)$, we may have to include, in $L(W,C)$, all left patterns $\pi_c P$, where π_c is a column permutation, that are distinct up to row rearrangement. Note that, due to the automorphism group of the code, we may not have to include all such patterns in $L(W,C)$.

5 Searching for an Incidence Matrix

Let W be a word block and let $C \in C(W)$. For each left pattern $P \in L(W,C)$, we determine whether or not C contains an incidence matrix $[P|Q]$ by performing an exhaustive search on the weight 12 words in C.

In describing how our algorithm works, we will make use of the following terminology and notation: The term *left word of* C will be used to refer to any $n(W)$–vector l that occurs in the first $n(W)$ components of a codeword in C. The term *right word of* C will refer to any $(33-n(W))$–vector r that occurs in the last $33-n(W)$ components of a codeword in C. For the left word l, we will use $R(C,l)$ to denote the set of all right words r of C in which lr forms a weight 12 word of C that does not intersect any of the weight 4 words in C (if any) in 4 positions (the notation lr means concatenation). Note that the reason we do not include weight 12 words that intersect a weight 4 word in 4 positions is that, as shown in [2], any such weight 12 word cannot be a row in an incidence matrix for a $(22,33,12,8,4)$-BIBD.

For a given code C, the size of the set $R(C,l)$ is basically a function of

the weight of the left word l. For the f_1-codes and left words l with weight 0, 2, and 4, the average size of $R(C,l)$ is 1851, 800, and 200, respectively For the d_1-codes with one weight 4 word and left words l with weight 0 and 2, the average size of $R(C,l)$ is 3192 and 1235, respectively.

Our algorithm for determining whether or not C contains an incidence matrix $[P|Q]$ works by recursively filling, row by row, the rows of Q until either an incidence matrix has been produced or all possibilities have been exhausted. Given a Q whose first $i-1$ rows have been filled, the possibilities for row i of Q are the right words r in the set $R(C,l_i)$, where l_i is the left word in row i of P, in which $l_i r$ intersects each of the first $i-1$ rows of $[P|Q]$ in four positions. Note that since the weight of the columns of Q do not typically come into play until the last few rows of Q are being filled, we do not consider the weight of the columns of Q in our algorithm. (Furthermore, it is known that if Q can be completed, then its columns will all have weight 8.)

The running time of our algorithm is greatly affected by the arrangement of the rows of P. Therefore, before we begin our search, we attempt to sort the rows of P in a manner that will result in the shortest running time. Factors taken into consideration when sorting the rows of P include the theoretical number of possibilities for row i of Q, the amount of pruning that may be done, and empirical results from testing.

6 Pruning the Search Space

Suppose the first $i-1$ rows of Q have been filled. Let R_i denote the set of right words that are candidates for row i of Q. That is, R_i is the set of all right words $r \in R(C,l_i)$, where l_i is the left word in row i of P, such that $l_i r$ intersects each of the first $i-1$ rows of $[P|Q]$ in four positions. We have several methods for determining when inserting a given $r \in R_i$ into row i of Q will not lead to a solution, each of which we will briefly describe next.

6.1 Right Representative Pruning

Let π_c be a permutation of the columns of Q for which there exists a permutation π_r of the rows of Q such that: 1) the first $i-1$ rows of Q and $\pi_r \pi_c Q$ are equal, 2) π_r does not move row i of Q, and 3) C contains an incidence matrix $[P|Q]$ if and only if C contains an incidence matrix $[P|\pi_r \pi_c Q]$. Then for any two right words $r_1, r_2 \in R_i$, if $r_1 = \pi_c r_2$ then inserting r_1 into row i of Q will lead to a solution if and only if inserting r_2 into row i leads to a solution. Thus, our algorithm only needs to try one of

r_1 and r_2. This observation leads us to a method of pruning we refer to as *right representative pruning.*

Before our algorithm begins trying the different possibilities for row i of Q, it first finds a set Π_i of columns permutations π_c for which there exists a π_r with the three properties described above. For each right word $r \in R_i$, our algorithm then uses Π_i to find the unique $r' \in R_i$, that will be processed first by the algorithm, for which there exists a $\pi_c \in \Pi_i$ such that $\pi_c r' = r$. We refer to r' as the *right representative* of r and denote it by $rep_i(r)$. (Of course, r' may be r itself.) Our right representatives are such that $rep_i(r_1) = rep_i(r_2)$ whenever there exists a $\pi_c \in \Pi_i$ such that $\pi_c r_1 = r_2$, where $r_1, r_2 \in R_i$. Furthermore, if r is the right representative of some r_1 then r is its own representative (i.e. $rep_i(r) = r$).

Inserting a right word r into row i of Q will lead to a solution if and only if inserting $rep_i(r)$ into row i leads to a solution. Therefore, the only right words $r \in R_i$, that our algorithm tries as row i of Q, are those that are their own representative. This gives us our right representative pruning.

6.2 Must Not Come Before Pruning

For this pruning, before we begin our search for an incidence matrix in C, that begins with a particular left pattern P, we first find a set S of row permutations π_r with the property that C contains an incidence matrix $[P|Q]$ if and only if C contains an incidence matrix $[P|\pi_r Q]$.

Let $\pi_r \in S$. Suppose π_r has the form:

$$\begin{pmatrix} 1 & 2 & \cdots & i-1 & i\cdots \\ 1 & 2 & \cdots & i-1 & j\cdots \end{pmatrix}$$

where $j > i$. Suppose further that the first $j-1$ rows of Q have been filled. Then since C contains an incidence matrix $[P|Q]$ if and only if C contains an incidence matrix $[P|\pi_r Q]$, inserting a right word $r \in R_j$ into row j of Q will lead us to a solution if and only if C contains an incidence matrix $[P|Q']$, where the first $i-1$ rows of Q and Q' are equal, and row i of Q' contains the right word r. Therefore, if the algorithm has already determined there does not exist such an incidence matrix $[P|Q']$, then we know inserting r into row j of Q will not lead us to a solution.

6.3 Pruning Using Patterns Under A Weight 5 Word

For each of our codes C, $C^\perp$ contains at least six weight 5 words. Let $\vec{w}$ be a weight 5 word in $C^\perp$ and let S denote the set of five coordinates of

C in which $\vec{w}$ has a value of one. Then if C contains an incidence matrix A for a $(22, 33, 12, 8, 4)$-BIBD, the five columns of A, that correspond to the coordinates in S, must form a left pattern of the f_1-block. Therefore, whenever our algorithm produces a $[P|Q]$ in which the five columns that correspond to S cannot be completed to a left pattern of the f_1-block, then our algorithm can backtrack. For example, suppose the first i rows of $[P|Q]$ contain three rows that have weight 4 in the five columns that correspond to S. Then $[P|Q]$ cannot be completed to a left pattern of the f_1-block as it is known that any left pattern of the f_1-block has at most two rows with weight 4.

6.4 Equivalent Case Processed Pruning

This form of pruning is only performed on the nine f_1-codes C that contain more than one f_1-block that is isomorphic to the chosen f_1-block W. It is based on the following observation: Let W' be an f_1-block in $C^{\perp}$ (other than W) that is isomorphic to W. (That is, W' is an f_1-block for which there exists an automorphism of C that permutes the five non-zero coordinates of W' to the first five coordinates of $C^{\perp}$.) Let S denote the five non-zero coordinates of W'. Suppose C contains an incidence matrix $[P|Q]$. Then the five columns in $[P|Q]$ that correspond to the coordinates in S form a left pattern P' of the f_1-block. Furthermore, since there exists an automorphism of C that permutes S to the first five coordinates of C, C must also contain an incidence matrix that begins with the left pattern P'. Therefore, if inserting a right word into row i of Q results in a $[P|Q]$ in which the five columns of $[P|Q]$, that correspond to S, can only be completed to a left pattern P' that has already been processed by the algorithm, then the algorithm can backtrack. For example, suppose our algorithm has already (unsuccessfully) searched for incidence matrices that begin with each of the left patterns P' that contain at least one weight 4 row. Now, suppose our algorithm is searching for an incidence matrix that begins with a left pattern P that does not contain any weight 4 rows (note that there is one left pattern of the f_1-block that does not contain any weight 4 rows). Then, whenever insertion of a right word r into row i of Q results in a $[P|Q]$ that has weight 4 in the five columns of row i that correspond to S, we know $[P|Q]$ cannot be completed to an incidence matrix.

6.5 Look Ahead Right Word Pruning

For this pruning, we maintain sets of all right words that may potentially be inserted into row m of Q, for $m = i, i+1, \ldots, 22$. Let $R_{i,m}$ denote

the set of right words $r \in R(C, l_m)$, where l_m is the left word in row m of P, such that rl_m intersects each of the first $i-1$ rows of $[P|Q]$ in 4 positions. Then if $[P|Q]$ can be completed to an incidence matrix, row m of Q must be an element of $R_{i,m}$. Now, many of the sets $R_{i,m}$ may be equal. For example, if rows m_1 and m_2 are equal then $R_{i,m_1} = R_{i,m_2}$. Let $N(i,m)$ denote the number sets $R_{i,m'}$ in which $R_{i,m'} = R_{i,m}$. Then in order to complete $[P|Q]$ to an incidence matrix, our algorithm must eventually select $N(i,m)$ right words from the set $R_{i,m}$. Furthermore, for the word blocks we use, these right words must be distinct. Therefore, whenever our algorithm produces a set $R_{i,m}$ in which $|R_{i,m}| < N(i,m)$, then we know $[P|Q]$ cannot be completed to an incidence matrix.

7 Results

Thus far, we have used our algorithm to search all 37 d_4-codes, 17 d_3^*-codes, 44 d_3-codes, and 115 d_2^*-codes. We have also searched 86 of the 122 d_2-codes. Thus far, the code with the largest search tree contained $1.48e+11$ nodes and took 9.5 days to search. On average, our algorithm searched $1.8e+10$ nodes per day. However, we believe with further optimization of both our algorithm and our implementation, we can sufficiently increase the number of nodes searched per day to allow us to search the remaining codes.

8 Estimations

We conclude this paper with estimations on the size of the search space for the 11 f_1-codes and the 21 d_1-codes that contain only one weight 4 word. Also included is an estimation on the size of the search space the $478 - 32 = 147$ remaining codes we have not searched. Additional details on the estimations for the f_1-code C_2 are also given.

Comparisons between our estimations and the actual size of the search space for the codes we have searched have shown that our estimations are quite good. For example, for the code searched with $1.48e+11$ nodes, our estimation for the number of nodes was $1.54e+11$.

In Table 3, we give, in detail, our estimations on the size of the search space, over all left patterns, for code C_2. The column labeled "calls" gives our estimation of the number of times our algorithm will be called to fill row *level* of Q. The column labeled "nodes" gives our estimation of the number of times the algorithm will insert a right word into row *level* of Q.

level	calls	nodes	right reps	mncb	wt 5 words	processed	look ahead
1	82	16011	8378	0	0	11	0
2	7606	782418	26294	33850	0	2150	0
3	721075	$1.24e+08$	139007	16966316	110901	1544293	0
4	$1.05e+08$	$6.29e+09$	173413	$2.13e+09$	7324192	81520255	0
5	$4.08e+09$	$2.02e+11$	738866	$2.56e+10$	$2.29e+09$	$3.60e+09$	$1.16e+08$
6	$1.71e+11$	$3.52e+12$	2208311	$1.15e+12$	$3.72e+10$	$7.58e+10$	$1.46e+08$
7	$2.26e+12$	$4.44e+13$	6136210	$6.41e+12$	$6.57e+11$	$5.32e+11$	$6.41e+11$
8	$3.64e+13$	$2.88e+14$	2986220	$1.02e+14$	$5.37e+12$	$1.45e+13$	$2.28e+13$
9	$1.45e+14$	$9.59e+14$	3472672	$8.52e+13$	$2.22e+13$	$2.11e+13$	$3.92e+14$
10	$4.46e+14$	$1.08e+15$	1432825	$4.98e+14$	$1.85e+13$	$5.90e+12$	$4.36e+14$
11	$1.24e+14$	$3.73e+14$	379007	$1.40e+13$	$2.22e+13$	$2.69e+12$	$3.16e+14$
12	$1.84e+13$	$2.41e+13$	305085	$1.15e+13$	$1.07e+12$	$2.32e+11$	$1.08e+13$
13	$4.64e+11$	$1.01e+12$	163029	$2.69e+10$	$1.16e+11$	$6.46e+09$	$8.27e+11$
14	$3.86e+10$	$4.37e+10$	0	$2.08e+10$	$3.07e+09$	$5.95e+08$	$1.60e+10$
15	$3.25e+09$	$4.04e+09$	0	62766167	$4.61e+08$	9088728	$2.64e+09$
16	$8.60e+08$	$8.87e+08$	0	50353412	74832341	9226714	$3.65e+08$
17	$3.87e+08$	$3.94e+08$	0	0	26112985	691877	43499882
18	$3.24e+08$	$3.24e+08$	0	2561315	59995028	0	$2.61e+08$
19	749594	2998377	0	0	0	0	2998377
Totals:	$7.73e+14$	$2.77e+15$	18169318	$7.18e+14$	$7.02e+13$	$4.50e+13$	$1.18e+15$

Table 3: Estimation of the search space size over all patterns for code 2.

The last five columns give the estimated number of times the insertion will be rejected due to one of our five methods of pruning.

In Table 4, we give our estimations on the size of the search space for the 32 codes with at most one weight 4 word. Included for each code C_i is the left word block W used in the search, the number of left patterns in $L(W, C_i)$, the estimated size of the search space over all left patterns, the worst case left pattern, and the estimated size of the search space over the worst case left pattern. Estimations for the remaining 147 codes we have not searched can be obtained from the author of this paper. The estimated size of the search space, over all 147 of these codes, is $1.80e+15$.

code	word block	num left patterns	node count	worst pattern	worst count
C_0	f_1	83	$1.66e+15$	6.0	$9.67e+13$
C_1	f_1	83	$1.44e+15$	4.3	$8.55e+13$
C_2	f_1	82	$2.77e+15$	4.0	$8.31e+13$
C_3	f_1	50	$1.08e+15$	6.0	$1.49e+14$
C_4	f_1	35	$1.19e+14$	6.0	$1.75e+13$
C_5	f_1	51	$3.52e+14$	3.9	$1.69e+13$
C_6	f_1	51	$1.29e+14$	5.0	$6.36e+12$
C_7	f_1	31	$3.19e+14$	4.1	$4.28e+13$
C_8	f_1	8	$1.71e+14$	3.1	$4.30e+13$
C_9	f_1	31	$7.96e+13$	4.1	$1.05e+13$
C_{10}	f_1	7	$2.27e+13$	3.0	$7.49e+12$
C_{11}	d_1	4	$4.34e+14$	2.0	$4.32e+14$
C_{12}	d_1	7	$1.71e+14$	2.0	$8.87e+13$
C_{13}	d_1	7	$1.45e+14$	2.0	$8.71e+13$
C_{14}	d_1	7	$1.09e+14$	2.0	$7.14e+13$
C_{15}	d_1	7	$1.17e+14$	2.1	$5.97e+13$
C_{16}	d_1	7	$9.81e+13$	2.0	$6.10e+13$
C_{17}	d_1	7	$1.31e+14$	2.0	$6.61e+13$
C_{18}	d_1	7	$7.33e+13$	2.1	$3.74e+13$
C_{19}	d_1	3	$1.33e+14$	2.0	$1.33e+14$
C_{20}	d_1	3	$1.07e+14$	2.0	$1.07e+14$
C_{21}	d_1	3	$7.05e+13$	2.0	$7.04e+13$
C_{22}	d_1	12	$4.09e+13$	2.2	$1.51e+13$
C_{23}	d_1	12	$2.16e+13$	2.0	$8.40e+12$
C_{24}	d_1	7	$4.02e+13$	2.0	$2.02e+13$
C_{25}	d_1	7	$2.05e+13$	2.0	$1.06e+13$
C_{26}	d_1	3	$1.35e+13$	2.0	$1.34e+13$
C_{27}	d_1	7	$2.14e+12$	2.0	$1.49e+12$
C_{28}	d_1	7	$9.69e+11$	2.1	$5.22e+11$
C_{29}	d_1	4	$8.14e+11$	2.0	$8.11e+11$
C_{30}	d_1	3	$6.15e+11$	2.0	$6.15e+11$
C_{31}	d_1	3	$1.71e+11$	2.0	$1.71e+11$

Table 4: Estimations of the search space size for the codes with at most one weight 4 word.

References

[1] R. T. Bilous. *The* $(33, 16)$ *Doubly-Even Self-Orthogonal Codes.* (Internet site), http://www.cs.umanitoba.ca/~umbiloul/DoublyEvenCodes/, August 2000.

[2] R. T. Bilous. *The Point Code of a* $(22, 33, 12, 8, 4)$- *Balanced Incomplete Block Design.* PhD thesis, University of Manitoba, 2001.

[3] R. A. Fisher and F. Yates. *Statistical Tables for Biological, Agricultural and Medical Research.* Longman, London, 1^{st} edition, 1938.

[4] N. Hamada and Y. Kobayshi. On the block structure of bib designs with parameters $v = 22$, $b = 33$, $r = 12$, $k = 8$, and $\lambda = 4$. *J. Combin. Theory, Ser. A 24*, pages 75–83, 1978.

[5] M. Hall Jr., R. Roth, G. H. J. van Rees, and S. A. Vanstone. On designs $(22, 33, 12, 8, 4)$. *J. Combin. Theory 47*, pages 157–175, 1988.

[6] B. D. McKay and S.P. Radziszowski. Towards deciding the existence of 2-$(22, 8, 4)$ designs. *J. Combin. Theory 22*, pages 211–222, 1996.

[7] G. H. J. van Rees. $(22, 33, 12, 8, 4)$-BIBD, an update. In *Computational and Constructive Design Theory*, pages 337–357. W.D. Wallis, Kluwer Academic Publ., 1996.