AUSTRALASIAN JOURNAL OF COMBINATORICS
Volume **33** (2005), Pages 217–229

A computer study of some 1-error correcting perfect binary codes

Martin Hessler

Linköpings Universitet
Matematiska institutionen
581 83 Linkping
SWEDEN

Abstract

A general algorithm for classifying 1-error correcting perfect binary codes of length n, rank $n-\log_2(n+1)+1$ and kernel of dimension $n-\log_2(n+1)-2$ is presented. The algorithm gives for $n=31$ that the number of equivalence classes is 197.

1 Introduction

A *1-error correcting perfect binary code* of length n is a set of words $C \subset Z_2^n$ such that for every $z \in Z_2^n$ we have a unique word $c \in C$ such that $d(c,z) \leq 1$. The integer d is the Hamming function, which is defined as the number of non-zero positions in the word. The set Z_2^n is the direct product of n copies of the finite field with two elements, i.e. $Z_2^n = Z_2 \times Z_2 \times \ldots \times Z_2$. Two perfect codes C and C' are *equivalent* if there exists a permutation δ on the coordinate set and a word $c' \in C'$ such that $C = \delta(C' + c')$. The restriction on c' is not necessary and is only there to keep the zero word in the code.

In this note we will consider 1-error correcting perfect binary codes of length n, rank $n - \log_2(n+1) + 1$ and with a kernel of dimension $n - \log_2(n+1) - 2$. For this set of perfect codes we will present a general algorithm to classify the number of equivalence classes for this set of codes. The classification will be done by using the super dual [2]. The algorithm will first be defined and then refined in order for it to run on a modern personal computer. The presentation is based on the case $n = 31$, for which the number of equivalence classes have been calculated to 197.

The result in this note is one of the first enumerations of codes of length $n = 31$; earlier works dealing with enumeration of 1-error correcting perfect binary codes are almost exclusively concerned with the codes of length $n = 15$. Some previous results for 1-error correcting perfect binary codes are by Hergert [3] who in 1985

showed that there are 19 different codes of length 15 and rank 12. Later, in 2000, Phelps [7] enumerated, by use of a computer, all codes of length $n = 15$ obtained by the doubling construction due to Solov'eva [8] and Phelps [6]. Dejter and Delgado [1] strengthened in 2002 the result by Hergert showing that there exist three codes with kernel of dimension nine, three codes with kernel of dimension eight and 12 codes with kernel of dimension seven for the perfect codes with length 15 and rank 12. Further results are by Malyugin [5], who in 1999 listed many of the codes of length 15 and by Heden [2] who in 2003 showed that there is only one 1-error correcting perfect binary code of length $n = 31$, of rank 30 and with a kernel of dimension 23.

2 Preliminaries and notation

Perfect codes are denoted by C, which is the set of all words in the code. The *kernel* of a perfect code is the set of *periods* of the code, $ker(C) = \{c | c + C = C\}$. The *rank* is the dimension of the linear span $< C >$ of the words in the code C. Throughout this note we will denote the dimension of the kernel with k and the rank of the code with r.

We can make a covering of the code with side classes of the kernel; this covering is given by a choice of coset representatives $c_i \notin ker(C)$, $i = 1, 2, \ldots t$, such that

$$C = ker(C) \cup (c_1 + ker(C)) \cup \ldots \cup (c_t + ker(C))$$

where $t = 2^{n-\log_2(n+1)-k} - 1$.

The product between two words z and s is defined as:

$$z \cdot s = (z_1, \ldots, z_n) \cdot (s_1, \ldots, s_n) = \sum_{i=1}^{n} z_i \cdot s_i \pmod 2.$$

The weight of a word is the Hamming distance from the zero word. In this paper we will always assume that the zero word is in the code. By a *perfect code* we always mean a 1-error correcting perfect binary code.

3 Super dual

To any code C we may define a super dual [2]. The super dual C^* of C is a linear code defined from the rowspan of a matrix pair $(G|H)$, $C^* =< (g_i|h_i) >$ where g_i and h_i are the i:th row in the corresponding matrices, $i = 1, 2, \ldots, n - k$. The rows g_i in the matrix G are a base for the dual space of the kernel of the code C. The matrix G is of size $(n - k) \times n$ with full rank. The matrix H will have size $(n - k) \times (2^{n-\log_2(n+1)-k} - 1)$ and rank $r - k$. The position $h_{i,j}$ in the matrix H is $h_{i,j} = g_i \cdot c_j$ for $i = 1, 2, \ldots, n - k$, where the words c_j are coset representatives, $j = 1, 2, \ldots, 2^{n-\log_2(n+1)-k} - 1$.

In [2] the following theorem was proved, with changed notation.

Theorem 1 (Heden 2003) *A linear code C^* consisting of words $(g|h)$, where the words g have length $n = 2^m - 1$ and the words h have length κ, is a super dual of a perfect code C of length n if and only if the following three conditions are satisfied:*

(i) for any word $(g|h)$ of C^*, $w(g) \neq (n+1)/2 \Rightarrow w(h) = (\kappa+1)/2$;

(ii) $\kappa = 2^{\dim(C^*)}/(n+1) - 1$;

(ii) there is a matrix $(G|H)$ such that the rows of $(G|H)$ generates C^*, the κ columns of H are distinct and $w(h) \neq (\kappa+1)/2$ for any row h of H.

We will only consider the case of length $n = 31$, rank $r = 27$ and a kernel of dimension $k = 24$. With these parameters, G will be a 7×31 matrix and H a 7×3 matrix. The explicit form of these matrices will be discussed in the next section.

4 Application

The goal in this section is to construct a method to enumerate some perfect codes. This method should not only be correct but also be useful on a typical computer. The first important step is to generalise the individual words to equivalence classes. Note that Lemma 1, below, introduces much of the notation used in the following subsection.

Lemma 1. *Let Δ be a group of automorphisms of a linear subspace S of Z_2^n. For any pair of words (a, b), the partitioning of b:s into equivalence classes of cosets to S, for those δ which fixes the coset $a + S$, will commute with the set of automorphisms Δ.*

Proof. Let (a, b) be any pair of words, $a, b \in Z_2^n$ such that $a \neq b$. Take any group of automorphisms

$$\Delta = \{\delta \mid \delta : S \to S\}.$$

Denote the subset which fixes the coset $x + S$ by

$$\Delta_x = \{\, \delta \mid \delta : x + S \to x + S,\ x \in Z_2^n,\ \delta \in \Delta\}.$$

Define the family of sets $A_\Omega^x = \{\, \delta(x + S) \mid x \in Z_2^n,\ \delta \in \Omega\}$, where Ω denotes any set of bijections.

With this notation, the lemma is equivalent to the following,

$$\delta(A^b_{\Delta_a}) = A^{\delta(b)}_{\Delta_{\delta(a)}},$$

for any $\delta \in \Delta$. It follows from the fact that Δ is a set of automorphisms on the set S, that it is enough to view a single element in the set $A^b_{\Delta_a}$. To finalise the proof of the lemma, it is enough to show that every element $\delta(\gamma(b)) \in A^b_{\Delta_a}$, for any $\gamma \in \Delta_a$, is

equal to an element $\mu(\delta(b)) \in A^{\delta(b)}_{\Delta_{\delta(a)}}$, for some $\mu \in \Delta_{\delta(a)}$, as δ is a bijective mapping. Take $\mu = \delta\gamma\delta^{-1}$ which belong to $\Delta_{\delta(a)}$ as

$$\mu : \delta(a + S) \mapsto \delta(a + S).$$

This concludes the proof.

In the following subsection we will present the general algorithm for classifying 1-error correcting perfect binary codes of length n, rank $n - \log_2(n+1) + 1$ and kernel of dimension $n - \log_2(n+1) - 2$. We will apply the algorithm to the special case $n = 31$.

4.1 Perfect codes of length 31, rank 27 and kernel of dimension 24

Theorem 2. *The number of equivalence classes of perfect codes of length* 31*, rank* 27 *and with a kernel of dimension* 24 *is* 197*.*

The theorem is implicitly proved in the rest of the current subsection, which explicitly proves the validity of the general algorithm used to perform the classification.

Proof. We first show, by using Theorem 1, that we without loss of generality can choose to represent the super dual of any perfect code of length 31, rank 27 and with a kernel of dimension 24 in the following way:

$$(G \mid H) = \begin{pmatrix} 00000000000000011111111111111111 & 000 \\ 00000001111111100000000011111111 & 000 \\ 00011110000111100001111000011111 & 000 \\ 01100110011001100110011001100110 & 000 \\ 10101010101010101010101010101010 & 100 \\ f(d_1) & 010 \\ f(d_2) & 001 \end{pmatrix},$$

where f is a function from Z_{15} to Z_{31} defined below. We will consider the possibilities for the function f. Thereby, we must keep in mind which properties made it possible to fix the matrix H and the first five rows in the matrix G. These properties are: (a) the super dual is a linear code and (b) H is for this simple case a full rank matrix. As any permutation of the coordinate set gives an equivalent code and as, from criteria (i) of Theorem 1, any word in the linear span of the first five rows of G must have weight 16, we can assume the first five rows of G to be as above. By the same criteria we get that the same conclusions will hold for the linear span of the rows number $1, 2, 3, 4$ and 6 as well as the rows number $1, 2, 3, 4$ and 7. This implies that they must differ from the fifth word by the addition of a word $(0, a_1, a_1, a_2, a_2, \ldots, a_{15}, a_{15}) \in Z_{31}$. Let d denote the fifth row of the matrix G. The function f will consequently be defined by:

$$f(a_1, a_2, \ldots, a_{15}) = (0, a_1, a_1, a_2, a_2, \ldots, a_{15}, a_{15}) + d.$$

(A similar construction was used in [2] to prove that there are three equivalence classes of the perfect codes of length 15, rank 12 and with a kernel of dimension 9.)

A simplex code S is a linear code of length n, for which all words have weight $(n+1)/2$ or 0. We will consider the simplex code defined from the row span of the following matrix S:

$$S = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

We note that all matrices which give the same rowspan as S are equivalent to S and that the simplex code S will contain all the symmetry properties of the five first rows in the super dual of the original code except the permutations of the 01:s in the word d. These additional permutations are of no importance for the classification of the number of non-equivalent perfect codes, as such a permutation only changes the d word and not the words d_1, d_2, as the definition of the function f includes the word d. Hence the number of non-equivalent triplets $0 = f^{-1}(d)$ and d_1, d_2 representing side classes of the simplex code S of length 15 in Z_2^{15} will be the number of equivalence classes of perfect codes. This is a consequence of the fact that the function f is bijective for the set of odd row additions for the last three rows and that the even additions do not give a super dual on the chosen form.

We will for clarity denote the triplets $(0 + S, a + S, b + S)$ with (a, b). By considering row additions in the matrix pair $(G|H)$ and addition in the corresponding perfect code, we see that this gives the following equivalence relation for our triplets. The set Δ is the group of automorphisms of S, the simplex code defined above. If $(a, b) \simeq (d, e)$ then we know that (d, e) is equal to one of the following six triplets for $\delta \in \Delta$, $((\delta(a), \delta(b))$, $(\delta(b), \delta(a))$, $(\delta(a), \delta(a + b))$, $(\delta(a + b), \delta(a))$, $(\delta(a + b), \delta(b))$ or $(\delta(b), \delta(a + b))$. Note that $\Delta \simeq GL(\, Z_2, 4)$. This can be proved by considering matrix multiplication from the left by the matrix representation of the general linear group. The matrix multiplication will induce a permutation on the above matrix representation of S.

In order to facilitate work for the reader, we will make a *preliminary proof* of the fact that the equivalence classes of the perfect codes with a triplet representation with the two elements in a set $Z \subset Z_2^n$, can be ordered for any ordering of the set Z. This is the first step when constructing the algorithm below.

Consider any ordering of the set Z and an equivalence class represented by a triplet (z_i, z_j). We remind of the meaning of the triplet by using the notation from the proof of Lemma 1. The triplet (z_i, z_j) represents two cosets to the simplex code; these two are chosen from the three sets $A_\Delta^{z_i}$, $A_\Delta^{z_j}$ and $A_\Delta^{z_i+z_j}$. The equivalence class is also determined by the relation between these sets defined by the relation between the two words z_i, z_j. Consider for example the relation between the support of these two words, and hence also the weight of the word $z_i + z_j$, which will be constant under any permutation of the coordinate set.

Consider the three distinct possibilities, which can occur for the sets $A_\Delta^{z_i}$, $A_\Delta^{z_j}$ and $A_\Delta^{z_i+z_j}$:

type 1: all the three sets are disjoint.
type 2: two of the sets are equal.
type 3: all the sets are equal.

We will now show how we can make an ordered choice from the set of non-equivalent perfect codes. Define the *least member* function

$$lm(A) = min(\{i \mid z_i \in A\}),$$

on any subset A of the ordered set Z. We note that, by following the ordering of Z when choosing z_i and z_j, the following relation will always be fulfilled:

$$i = lm(A_\Delta^{z_i}) \leq lm(A_\Delta^{z_j}) \leq lm(A_\Delta^{z_i+z_j}).$$

By Lemma 1, we can now fix z_i and get two new distinct cases for the sets $A_{\Delta_{z_i}}^{z_j}$ and $A_{\Delta_{z_i}}^{z_i+z_j}$:

case a: they are disjoint
case b: they are equal.

Hence in total there are five possibilities 1a, 2a, 2b, 3a and 3b. For each of these five cases, we can assign a number to the corresponding equivalence class by using the induced ordering from the ordering of Z. This concludes the preliminary proof.

The following algorithm is essentially the ordering of the non-equivalent perfect codes, refined to run on a normal computer.

The algorithm: We construct an algorithm which places every triplet (a, b) in a set L which corresponds to a class of equivalent side classes. Hence L will be the ordering of the equivalence classes in the preliminary proof. The algorithm will go through every pair $(a, b) \simeq (b, a)$. We will prove that every triplet (a, b) placed in L is non-equivalent to any other triplet in L and that every triplet non-equivalent to all triplets contained in L is added. In the algorithm, assignment will be done to the left, thus $A = B$ will assign the value of B to A. The sets A, B will be used to store already considered and/or added triplets, A for the outer loop and B for the inner loop.

Sets of words A, B, L
Ordered Set $Z_2^{15} \setminus \{0\} = \{z_0, z_1 \ldots z_N\}$
Δ, Δ_x, S as in Lemma 1.

$A = S$
for$_1$ $(i = 0, 1, \ldots, N-1)$
  while$_1$ ($z_i \in A$)
    $i = i + 1$
  end while$_1$

$B = S$
for$_2$ ($j = i+1, i+2, \ldots, N$)
  while$_2$ ($z_j \in B$)
    $j = j + 1$
  end while$_2$
  if ($z_j \notin \{S + z_i\} \cup A$ and $z_i + z_j \notin A$)
    $L = L \cup \{(z_i, z_j)\}$
    $B = B \cup \{\, \delta_k((S + z_j) \cup (S + z_i + z_j)) \mid \delta_k \in \Delta_{z_i}\,,\ k = 0, 1, \ldots\}$
  else
    $B = B \cup \{z_j\} \cup \{z_i + z_j\}$
  end if
end for$_2$
$A = A \cup \{\, \delta_k(S + z_i) \mid \delta_k \in \Delta\,,\ k = 0, 1, \ldots\}$
end for$_1$

We will now prove that any two perfect codes, enumerated by the above algorithm, are non-equivalent. This will be done by induction.

The first triplet (z_i, z_j) is new and we also assure that $S \neq S + z_i$ and $S \neq S + z_j$ and $S + z_i \neq S + z_j$. If we add the triplet (z_i, z_j), then it is a new equivalence class, if no equivalent triplet (a, b) already exists in the non-empty L. The six possible cases of equivalent triplets are as stated above. We will prove that no such equivalent triplet $(a, b) \in L$ can exist. Two cases when adding a triplet to a non-empty L, either $z_i = a$ or $z_i \neq a$.

The case $z_i \neq a$: In this case we have the following relation

$$A \supset \{\, \delta_k(S + a) \mid \delta_k \in \Delta\,,\ k = 0, 1, \ldots\}.$$

We only add (z_i, z_j) if $z_i, z_j, z_i + z_j \notin A$, which is impossible if $(a, b) \simeq (z_i, z_j)$. To prove this we consider the six possible equivalence relations. If $(a, b) \in L$, then the inclusion, as stated above, is valid. The while$_1$-test insures from this inclusion that $z_i \not\simeq a$, thus if the triplets are equivalent, then either $z_i \simeq a + b$ or $z_i \simeq b$. The while$_2$-test insures that $z_j \not\simeq a$, thus we only have two possibilities left: either $(z_i, z_j) \simeq (b, a+b)$ or $(z_i, z_j) \simeq (a+b, b)$. The while$_2$-test also insures that $z_i + z_j \notin A$, but for the only remaining cases we see that $z_i + z_j \simeq a$. Hence if we add (z_i, z_j) it will not be equivalent to any equivalence class in L.

The case $z_i = a$: If an equivalent triplet $(a, b) \simeq (z_i, z_j) = (a, z_j)$ exists in L, then from Lemma 1 we know that either $S + b = \delta(S + z_j)$ or $S + b = \delta(S + a + z_j)$, $\delta \in \Delta_a$. This is impossible if $(a, b) \in L$ as

$$B \supset \{\, \delta_k((S + b) \cup (S + a + b)) \mid \delta_k \in \Delta_a\,,\ k = 0, 1, \ldots\}.$$

Hence the triplets cannot be equivalent as the while$_2$ insures that $z_j \notin B$.

We will now prove that the algorithm makes a complete ordering of all possible perfect codes.

It is sufficient to show that any triplet (z_i, z_j), which is non-equivalent to all triplets (a, b) in L, is added to L, as all triplets are considered by the algorithm. Observe that A contains all words in the same orbit as all earlier z_i. Thus if $z_k \in A_\Delta^{z_i}$ (defined in Lemma 1) then $k \geq i$. By Lemma 1 we can find all equivalence classes by fixing any member in the orbit of z_i, thus for this algorithm, also the particular case for z_i. This explains why the completeness is not affected by the fact that all triplets $(a, b) \in L$ will be such that $a \not\simeq z_i$ or $a = z_i$, as i is least member in it's equivalence class and hence will give the maximal possible z_j's.

The while$_2$ will not affect the completeness of the algorithm as we in the for$_2$-loop have a constant i and we only add equivalent words to B, under the restricted permutation group Δ_{z_i}, which is exactly the criteria induced by Lemma 1. The two cases for the if-test will be as noted, thus if $z_i \simeq z_j$ then $A_\Delta^{z_i} \cap A = \emptyset$ and all the cases (z_i, z_j) such that $z_i \simeq z_j$ will be added and hence insuring completeness for this case. The only problem remaining is if $z_j \in A$ but if this happens, then the least member j^* of the equivalence class to z_j is such that $j^* = lm(A_\Delta^{z_j}) < i = lm(A_\Delta^{z_i})$ (lm defined in the preliminary proof) and hence the triplet (z_{j^*}, z_i) has already been added. This concludes the proof of the completeness of the algorithm.

References

[1] I. J. Dejter and A. A. Delgado, STS-Graphs of perfect codes mod kernel, submitted.

[2] O. Heden, Perfect codes from the dual point of view I, submitted to *Discrete Mathematics*.

[3] F. Hergert, *Algebraische methoden für nichtlineare codes*, Thesis Darmstadt, 1985.

[4] M. Hessler, Perfect codes considered as isomorphic spaces, submitted to *Discrete Mathematics*.

[5] S. A. Malyugin, On enumeration of perfect binary codes of length 15, *Discrete Analysis and Operation Research* 1(6)2 (1999), 48–73.

[6] K. T. Phelps, A combinatorial construction of perfect codes, *SIAM J. Alg. Disc. Meth.* 5 (1983), 398–403.

[7] K. T. Phelps, An enumeration of 1-perfect binary codes of length 15, *Australas. J. Combin.* 21 (2000), 287–298.

[8] F. I. Solov'eva, On binary nongroup codes, *Methody Diskr. Analiza* 37 (1981), 65–76.

A Appendix

The table below contains a triplet representation for all the equivalence classes of perfect codes found in the computer search described above. These triplets can easily be used to reconstruct the corresponding perfect codes using methods in [4].

The types are as defined above and the integer $|Sym|$ is the cardinality of the set of permutations δ on the coordinate set, which fulfil that $\delta(S) = S$ and that $\delta(d_i+S) = d_j + S$, $\delta(d_i + S) = d_i + S$ or $\delta(d_i + S) = d_i + d_j + S$, $i, j \in \{1, 2\}$, $i \neq j$. The linear code S is the simplex code of length $n = 15$, see above. *Note that $|Sym|$ is not equal to the cardinality of the symmetry group of the corresponding perfect code.* Viewing the table below we notice some peculiarities, for example, the code with the largest symmetry group is of type 1. This is surprising as we would think that the codes of type 3 and 2 would be better candidates for a big symmetry group, as we for the codes of type 3 and 2 can get additional members in the symmetry group from the orbits between the cosets to S. Also it is surprising to see the large difference between the most restricting triplets with the elementary symmetry group to the least restricting with 1344 permutations in their symmetry group. Note that the types a and b are given by fixing the coset $d1 + S$.

Let us also give the final remark, that although the programming used to execute the algorithm above has been tested extensively, a testing which has confirmed know results, it is always possible when using computers that errors may occur.

d1	d2	$\|Sym\|$	Type
100000000000000	010000000000000	192	2a
100000000000000	011000000000000	192	1a
100000000000000	010100000000000	16	1a
100000000000000	011100000000000	16	1a
100000000000000	011110000000000	192	1a
100000000000000	010101000000000	48	1a
100000000000000	001101000000000	48	1a
100000000000000	011101000000000	16	1a
100000000000000	011111000000000	96	1a
100000000000000	011111100000000	1344	1a
100000000000000	010100010000000	6	1a
100000000000000	011100010000000	8	1a
100000000000000	010101010000000	6	1a
100000000000000	010101011000000	48	1a
100000000000000	001101011000000	48	1a
110000000000000	101000000000000	576	3b
110000000000000	100100000000000	48	3b
110000000000000	001100000000000	16	2a
110000000000000	101100000000000	16	2b
110000000000000	000110000000000	16	2a
110000000000000	100110000000000	16	1a

110000000000000	001110000000000	16	1a
110000000000000	101110000000000	16	1a
110000000000000	000011000000000	64	2a
110000000000000	100011000000000	32	2b
110000000000000	001011000000000	32	1a
110000000000000	101011000000000	32	2b
110000000000000	000111000000000	16	1a
110000000000000	100111000000000	16	2b
110000000000000	001111000000000	16	1a
110000000000000	101111000000000	16	2b
110000000000000	000111100000000	192	1a
110000000000000	100111100000000	192	2b
110000000000000	001111100000000	192	1a
110000000000000	101111100000000	192	2b
110000000000000	000100010000000	8	2a
110000000000000	100100010000000	4	2b
110000000000000	001100010000000	4	1a
110000000000000	101100010000000	4	2b
110000000000000	000110010000000	2	1a
110000000000000	100110010000000	2	1a
110000000000000	000011010000000	8	2b
110000000000000	100011010000000	8	2b
110000000000000	001011010000000	8	2b
110000000000000	101011010000000	8	2b
110000000000000	000111010000000	4	1a
110000000000000	100111010000000	4	2b
110000000000000	001111010000000	4	1a
110000000000000	000010000100101	4	2b
110000000000000	000101011000000	16	1a
110000000000000	100101011000000	16	2b
110000000000000	001101011000000	16	1a
110000000000000	000000010011010	16	2b
110000000000000	000100010001000	12	1a
110000000000000	100100010001000	12	2b
110000000000000	001100010001000	12	1a
110000000000000	101100010001000	12	2b
110000000000000	000010010001000	12	2b
110000000000000	100010010001000	12	1a
110000000000000	001010010001000	12	2b
110000000000000	000000100100010	12	1a
111000000000000	100110000000000	64	2a
111000000000000	010110000000000	16	1a
111000000000000	000111000000000	48	1a
111000000000000	100111000000000	16	1a
111000000000000	000111100000000	576	1a

111000000000000	100111100000000	192	1a
111000000000000	100100010000000	4	1a
111000000000000	000110010000000	4	1a
111000000000000	010110010000000	4	1a
111000000000000	000111010000000	12	1a
111000000000000	100111010000000	4	1a
111000000000000	000101011000000	16	1a
111000000000000	110101011000000	48	1a
111000000000000	000100010001000	72	2a
111000000000000	100100010001000	12	1a
111000000000000	000010010001000	12	1a
111000000000000	100010010001000	36	1a
110100000000000	101010000000000	32	2a
110100000000000	011010000000000	8	2a
110100000000000	101011000000000	16	1a
110100000000000	101000100000000	8	2a
110100000000000	001010100000000	32	2a
110100000000000	101010100000000	16	1a
110100000000000	011010100000000	8	1a
110100000000000	001011100000000	48	1a
110100000000000	101011100000000	16	1a
110100000000000	101000010000000	2	2a
110100000000000	111000010000000	2	2a
110100000000000	001010010000000	2	2a
110100000000000	101010010000000	2	2a
110100000000000	011010010000000	1	1a
110100000000000	001011010000000	6	1a
110100000000000	101011010000000	2	1a
110100000000000	100000110000000	2	2a
110100000000000	001000110000000	2	2a
110100000000000	101000110000000	1	1a
110100000000000	111000110000000	2	1a
110100000000000	001010110000000	2	1a
110100000000000	000000000101010	2	2a
110100000000000	011010110000000	1	1a
110100000000000	001011110000000	6	1a
110100000000000	000001000101010	2	1a
110100000000000	010000011000000	2	2a
110100000000000	001000011000000	2	2a
110100000000000	011000011000000	4	1a
110100000000000	111000011000000	4	1a
110100000000000	000001011000000	8	2a
110100000000000	100001011000000	4	1a
110100000000000	010001011000000	2	1a
110100000000000	001001011000000	2	1a

1101000000000000	0000000100001100	4	2a
1101000000000000	1000000100001100	4	1a
1101000000000000	1000000001100000	2	2a
1101000000000000	1100000001100000	8	2a
1101000000000000	1010000001100000	2	1a
1101000000000000	1110000001100000	4	1a
1101000000000000	0000010001100000	2	2a
1101000000000000	1000010001100000	4	1a
1101000000000000	0100010001100000	4	1a
1101000000000000	0010010001100000	2	1a
1101000000000000	1010010001100000	4	1a
1101000000000000	0110010001100000	4	1a
1101000000000000	1000000011101000	4	1a
1101000000000000	0010000011101000	4	1a
1101000000000000	1000011001001000	12	1a
1101000000000000	0010100010000010	8	2a
1101000000000000	0100010001001000	4	2a
1101000000000000	1000010001001000	12	1a
1111000000000000	1100110000000000	32	2a
1111000000000000	1010110000000000	24	3a
1111000000000000	1000111000000000	32	2a
1111000000000000	1100111000000000	16	2a
1111000000000000	1110111000000000	48	1a
1111000000000000	1100100100000000	1	2a
1111000000000000	0110100100000000	2	2a
1111000000000000	1000110100000000	2	2a
1111000000000000	0010110100000000	2	3a
1111000000000000	1010110100000000	1	2a
1111000000000000	0010000000011001	2	2a
1111000000000000	0000111100000000	6	1a
1111000000000000	1000111100000000	2	1a
1111000000000000	0000001000011001	2	3a
1111000000000000	1100000110000000	2	2a
1111000000000000	0110000110000000	8	2a
1111000000000000	1000010110000000	2	2a
1111000000000000	0100010110000000	4	2a
1111000000000000	1100010110000000	4	1a
1111000000000000	0010010110000000	4	1a
1111000000000000	1010010110000000	4	2a
1111000000000000	1100000100010000	8	2a
1111000000000000	1000100100010000	12	1a
1111000000000000	0100100100010000	4	2a
1111000000000000	0110100100010000	12	1a
1111000000000000	1100000010010000	2	2a
1111000000000000	0110000010010000	4	1a

111100000000000	100010001001000	4	2a
111100000000000	010010001001000	2	3a
111100000000000	110010001001000	2	2a
111100000000000	011010001001000	4	1a
111100000000000	111010001001000	8	2a
011110000000000	101101000000000	192	3b
011110000000000	111001100000000	64	2b
011110000000000	110100010000000	8	2b
011110000000000	111001010000000	8	2b
011110000000000	110001110000000	12	1a
011110000000000	010100011000000	32	2b
011110000000000	110100011000000	32	2b
011110000000000	110101011000000	96	2b
011110000000000	001101011000000	96	2b
011110000000000	101000010100000	72	2a
111110000000000	110001010000000	2	1a
111110000000000	011001010000000	4	1a
111110000000000	010001110000000	8	2a
111110000000000	010100011000000	8	1a
111110000000000	010101011000000	16	1a
111110000000000	001101011000000	16	1a
111110000000000	100100010100000	6	1a
111111000000000	101100010000000	16	2b
111111000000000	101100110000000	16	2b
111111000000000	010100011000000	48	1a
111111000000000	101100011000000	48	1a
111111100000000	110100010000000	48	1a
111111100000000	110101011000000	720	1a
110100010000000	110001001000000	24	3b
110100010000000	101001001000000	2	3a
110100010000000	111001001000000	4	1a
110100010000000	011101001000000	2	2b
110100010000000	011011001000000	1	2a
110100010000000	101000101000000	2	3b
110100010000000	011000101000000	8	2a
110100010000000	011010101000000	8	2b
110100010000000	100000110010000	24	3b
110100010000000	101000101100000	8	2b
111100010000000	010111001000000	8	2a
110101011000000	100011010100000	120	2b
110101011000000	011011010100000	360	3b

(Received 15 Jan 2004; revised 2 June 2004)