Application of Coding Theory to the Design of Frequency Hopping Lists

Derek Smith, Alexander Sakhnovich, Stephanie Perkins,
David Knight, Lesley Hughes

Division of Mathematics and Statistics,
School of Technology,
University of Glamorgan,
Pontypridd CF37 1DL,
Wales, UK

Technical Report UG–M–02–1

19th February 2002

Contents

Glossary of notation

Most of the notations used in this report are well-known and are taken from [20].

n	Block length of a code.
k	Dimension of a linear code.
d	Minimum Hamming distance of a code.
$A(n,d)$	Maximum number of codewords in any binary code (linear or nonlinear) of length n and minimum distance d between codewords.
$[n,k,d]$	Linear code of length n, dimension k and minimum distance d.
(n,M,d)	Nonlinear code of length n, minimum distance d and with M codewords.
w	Weight of a codeword of a constant weight code.
$A(n,d,w)$	Maximum number of codewords in any constant weight code of length n, weight w and minimum distance d between codewords.
$GF(q)$	A finite field with q elements.
$A_q(n,d)$	Maximum number of codewords in any q-ary code (linear or nonlinear) of length n and minimum distance d between codewords.
$S(t,k,v)$	A Steiner system with v points, blocks of size k and any t points contained in exactly one block.

Additionally, the following notations are introduced:

$q_1 \mid q_2$	A mixed code, with entries from two fields, with q_1 and q_2 elements respectively.
(q_1,q_2,n_1,n_2,d)	A code of length n_1+n_2, where the first n_1 entries of any codeword take values from 0 to q_1-1, the next n_2 entries take values from 0 to q_2-1, and the minimal Hamming distance between different codewords is no less than d
$A_{q_1,q_2}(n_1,n_2,d)$	The maximal possible cardinality of a (q_1,q_2,n_1,n_2,d) code.
$S_\tau(\mathbf{c})$	A cyclic shift (by τ positions mod n) of a codeword $\mathbf{c}$ in a constant weight code.

1 Introduction

This report is concerned with the construction of constant weight codes and their application in radio networks. Constant weight codes have received considerable theoretical attention (see for example [7]), and many applications have been reported. The application considered here arises when the assignment of fixed frequencies to transmitters is replaced by the technique known as *frequency hopping*.

Most radio frequency assignment work involves the static assignment of frequencies to transmitters. In static assignment the attenuation of an interfering radio signal due to frequency separation is exploited to minimise interference by careful frequency planning. This problem of assigning frequencies to transmitters in the most efficient way is usually referred to as the *frequency assignment problem*, although there are many variations of the problem that can be identified. The earliest author to treat the problem as a generalised graph colouring problem was Metzger [21]. Zoellner and Beall [39] regarded the approach as a breakthrough in frequency assignment technology. Hale [15] presented a variety of transmitter and frequency orderings for use with sequential (greedy) algorithms. A detailed survey of the prospects for frequency assignment algorithms was presented by Lanfear [18]. Some theoretical results on frequency assignments treated as T-colourings were given by Roberts [26]. More up to date information on the frequency assignment problem can be found in [29] and in the forthcoming book [19]. In recent years it has become increasingly recognised that meta-heuristic algorithms are capable of finding better quality assignments than sequential algorithms. The system FASoft developed at the Universities of Glamorgan and Cardiff [17] contains implementations of several effective meta-heuristic algorithms, and in particular has demonstrated the effectiveness of *tabu search* and *simulated annealing*. This has required the use of graph theoretic lower bounds [28, 2, 30], which are now capable of demonstrating that the assignments generated are usually optimal or near optimal in terms of the span of frequencies required. In particular, FASoft is the first system to have demonstrated the capability of solving to optimality the harder variations of the standard benchmark problems known as the *Philadelphia* problems [30, 31]. These problems had remained unsolved since they were first presented in 1973. More recent work at Glamorgan has shown that it is possible to generate good lower bounds for the interference penalty in weighted fixed spectrum problems [22], and has also adapted and further developed the meta-heuristic algorithms for these problems.

The option of moving from static assignment to *frequency hopping* has several advantages. One of these advantages is concerned with the security of communication. Another, usually referred to as *frequency diversity*, is concerned with the reduction of

the effects of frequency specific signal fading. A third advantage is usually referred to as *interference diversity*. When radios hop the interference from a particular radio is spread more equitably through the network. Thus the error control coding employed to combat interference may cope with all of the levels of interference encountered at each receiver, where it may not cope with the peak interference on any receiver when hopping is not employed. The use of error control coding of the information to be transmitted is well developed. The study of the performance of this error control coding in hopping networks is certainly of interest, but is not considered in this report. The aim here will be to consider how the theory and construction of the codes known as *constant weight codes* can aid the generation of *hopping lists*. These are the lists of frequencies over which the individual radios hop. They should be distinguished from the *hopping sequences*, which define the order in which radios hop over the frequencies in their individual hopping list. Coding theory also has a role in the construction of hopping sequences, see for example [38], but hopping sequences are not the focus of this report.

Frequency hopping is used in some current systems (including the GSM mobile telephone systems of some operators) and is one of the techniques that will increasingly replace static assignment in the future. As already stated, instead of assigning a single frequency to a transmitter, the transmitter is assigned a list of frequencies, or hopping list, which is a subset of the set of available frequencies. Transmitters change frequency periodically (i.e they hop) in a predetermined pattern that is known to other members of the network. Hopping can achieve frequency and interference diversity [11]. The frequency assignment problem can now be viewed in three stages: generating potential hopping lists, assigning some or all of these hopping lists to transmitters and assigning the hopping sequences in which transmitters hop over the frequencies in the hopping lists [38, 27, 3]. This project is concerned with the generation of sets of hopping lists to assign to the transmitters.

Maximum resistance to interference is apparently obtained if the overlap between hopping lists is minimised. If transmitters are close they should be assigned lists with small overlap. However, it should be noted that any list can be reused if the distance between transmitters is sufficiently large. Larger reuse distances (leading to lower overall interference) are then achievable if the number of lists (with a given maximum overlap) is maximised. Thus lists with a defined small maximum overlap may be preferred to disjoint lists, which would have to be re-used too frequently. Although it is not necessary that all hopping lists have the same length, this work concentrates principally on this case. The constant length assumption will be discussed in section 2.

2 Formulation of the problem

The problem described in the introduction for constant length hopping lists can be interpreted as a problem about constant weight codes. A binary codeword $(c_1, c_2, \ldots, c_n)$ is used to represent a hopping list $\{f_{i_1}, f_{i_2}, \ldots, f_{i_w}\}$ selected from n potential frequencies $(f_1, f_2, \ldots, f_n)$. Then $c_i = 1$ if frequency f_i is in the list and $c_i = 0$ otherwise. Clearly the codewords have weight w if there are w frequencies in the list. The problem is to maximise the number of codewords in the constant weight code. $A(n, d, w)$ is the maximum possible number of binary codewords of length n, Hamming distance at least d apart and constant weight w. A constant weight code with $A(n, d, w)$ codewords may give very significantly greater numbers of hopping lists with defined maximum overlap than the software currently being investigated by suppliers of software to mobile telephone operators. In a similar way, the problem when the hopping lists need not have constant length can be interpreted as a problem about (not necessarily linear) block codes with constraints on the weight enumerator.

A review of work on constant weight codes up to about 1977 can be found in [20]. Later work can be found in [33, 7]. It is important to use codes generated by uniform constructions as much as possible, as the assignment algorithms may need to modify the parameters of the lists and regenerate them as they proceed. Uniformity of constructions may sometimes have to take priority over the use of the very best code known.

The extent to which a constant weight code (representing hopping lists all of the same length) is necessary or desirable will now be discussed. The appropriate choices of parameters, in particular the desirability of long hopping lists, will also be considered.

Standard descriptions of frequency hopping often assume a single list of frequencies is used. In a system without base stations but with different lists for different transmitters there may be no reason to assume that any one list should be shorter than any other. However, the situation in mobile telephone applications is more complex. Björklund et al. [3] generate hopping lists (referred to as *mobile allocation lists* or *MALs*) for GSM networks by an optimization routine. Each list starts as the set of frequencies allocated to a sector by a static frequency assignment algorithm. The size of the list for a sector thus starts as the number of transmitters in the sector, and so varies from sector to sector. The number of frequencies in each list is allowed to change as the optimisation of a chosen measure of interference proceeds. Available frequencies may be added or deleted, but the initial size of the list is taken as a minimum. This approach successfully demonstrates lower interference than is achieved in a statically assigned network. However, it remains unclear whether this

optimization, particularly given its rather small number of iterations, is capable of generating the best hopping lists. It remains quite possible that lists generated by some good combinatorial construction remain a better choice, at least for initial generation of the lists. The possibility of synchronizing certain sets of sectors (perhaps all the sectors with a base station at the same site) may also need to be considered. In this case a hopping list would be generated for the site instead of for a sector or for an individual transmitter. Good methods for generating appropriate constant weight codes are necessary if the merits of these codes for list generation are to be assessed. The authors are aware of at least one mobile telephone company which has started optimisations from constant weight codes, although the codes are generated by a very simple heuristic and are far from best possible.

Both cyclic hopping and random hopping can be used in GSM. Cyclic hopping gives the maximum frequency diversity gain, but no interference gain. Random hopping gives both frequency diversity gain and interference diversity gain. Cyclic hopping should not be used when lists all have the same length, as in the discussion here.

The report will concentrate on constant length lists, and particularly on sets of lists with a defined maximum overlap and with many lists available. However, as a result of the above discussion, if it is possible to generate a good set of lists, for example by a Steiner system construction, the lists (which are not of constant length) obtained by deleting a small number of points (frequencies) may still prove useful.

The basic idea of good hopping list design is to achieve interference resistance by minimising clashes, when two close radios hop onto the same frequency. In systems where all radios hop over all frequencies, the hopping sequences are designed to minimise interference by minimising clashes [38]. It is a particular challenge to design good hopping lists and then to design hopping sequences for these lists which further minimise interference. Normally random hopping sequences are used, although hopping sequences based on Reed-Solomon codes are also possible.

The specification of a maximum overlap between lists only addresses co-channel interference. Planning to minimise adjacent channel interference has to be addressed at the second or third stages of the planning process. Note that if the lists are generated by a constant weight code, the list re-use distance is generally greater than the frequency re-use distance with static planning. However, the re-use distance for an individual frequency in the lists will be smaller. Thus the advantage in combating interference depends on the error control coding and interleaving coping with the errors that may arise as a result of the potential clash of frequency at smaller re-use distances. As a result, it can be specified that two lists can overlap in 1, 2 or certainly

at most 3 frequencies if they are likely to be useful in these applications. Thus

$$1 \leq w - d/2 \leq 3$$

or

$$2w - 2 \leq d \leq 2w - 6.$$

Frequency hopping gives an interference gain with as few as 3 or 4 frequencies, whereas hopping over 2 frequencies can result in degradation when compared to the non-hopping case [35]. On the other hand, it is demonstrated in [27] that the frame erasure rate decreases as the number of frequencies increases, up to a maximum gain at 8 frequencies. Björklund et al. [3] found that a small average hopping list length (below 2) was best with their optimization method, although with randomly generated hopping lists the interference decreases as the number of frequencies increases to 8. These two results appear somewhat contradictory. In any event they support a decision here to consider lists of length w with $3 \leq w \leq 8$. The decision to consider $n \leq 63$ arises partly for practicality, and partly because the number of frequencies in a GSM system using frequency hopping cannot exceed 63.

Thus in summary, it can be stated that the parameters of interest are not completely fixed, but generally:

1. $n \leq 63$ as 63 is the maximum number of frequencies possible in GSM systems when hopping. Of course future systems might not have this restriction.
2. $3 \leq w \leq 8$ as lists of length 2 are known to give no advantages as a result of diversity. 8 may be a typical limit which might be increased.
3. $d = 2w-2$ or $d = 2w-4$, although it is possible that $2w-6$ could be of interest as well.

Other radio systems may require much larger values of n (say 200–300) and w (say 50–100). In such cases search techniques can become completely impractical, and the results of section 6 may be particularly useful.

The general aim of this report is not to show that hopping lists constructed from constant weight codes have significant merit. Rather it is to consider how the necessary constant weight codes can be quickly and efficiently generated in an Engineering environment when the need for them arises. Codes derived from *Steiner systems* will be of interest (when they exist) if they offer particular efficiencies. Tables of constant weight codes are available in [33]. However, they are largely incomplete for

$n > 28$. Additionally, use of many of the best known codes requires a wide variety of constructions and sometimes reference to more than one paper in the literature. In this application the Engineer may prefer a small number of general constructions which, taken together, will always produce a reasonably good code. Hopefully the availability of this report will allow further evaluations of the merit of using constant weight codes, in comparison with approaches such as that in [3].

3 An upper bound for $A(n, d, w)$

$A(n, d, w)$ is the maximum possible number of binary codewords of length n, weight w and pairwise Hamming distance no less than d.

Let X be a set of v points. A *Steiner system* is a collection of distinct sets of k points chosen from X (called *blocks*) with the property that any set of t of the points lies in exactly one block. The blocks of the Steiner system correspond to the codewords of a constant length code. If point i is in a block then $c_i = 1$ in the corresponding codeword, otherwise $c_i = 0$ in that codeword. It is unfortunate that Steiner systems are fairly rare, as when they exist the following theorem (taken from page 528 of [20]) shows that $A(n, d, w)$ takes its maximum possible value. In this report the inequality of theorem 3.1 and the (sometimes) more accurate inequality in the proof will both be referred to as the *Johnson bound*. A fuller account of the Johnson bound can be found in [20].

Theorem 3.1

$$A(n, 2\delta, w) \leq \frac{n(n-1)\dots(n-w+\delta)}{w(w-1)\dots\delta} \tag{3.1}$$

with equality if and only if a Steiner system $S(w-\delta+1, w, n)$ exists.

Proof. Consider a matrix B in which the i'th row of B is the i'th codeword of the constant weight code of length n, minimum distance 2δ and weight w. $\frac{w}{n}A(n, 2\delta, w)$ is the mean of the number of 1's in the columns of B, which is less than or equal to the maximum number of 1's in any column. Choose a column with the maximum number of 1's, delete the column and all codewords with a 0 in that column. A matrix B_1 in which the rows form a constant weight code of length $n-1$, minimum distance 2δ and weight $w-1$ which has no less than $\frac{w}{n}A(n, 2\delta, w)$ codewords is obtained. As $A(n-1, 2\delta, w-1)$ is the maximum number of codewords in such a code,

$$A(n, 2\delta, w) \leq \left\lfloor \frac{n}{w} A(n-1, 2\delta, w-1) \right\rfloor .$$

Applying the same process to B_1 and subsequent reduced matrices $B_2, B_3, \ldots, B_{w-\delta}$ and noting that $A(n-w+\delta-1, 2\delta, \delta-1) = 1$ it follows that

$$A(n, 2\delta, w) \leq \left\lfloor \frac{n}{w} \left\lfloor \frac{n-1}{w-1} \cdots \left\lfloor \frac{n-w+\delta}{\delta} \right\rfloor \right\rfloor \right\rfloor .$$

The inequality of the theorem is immediate.

If the codewords correspond to the blocks of a Steiner system $S(w-\delta+1, w, n)$ then $t = w-\delta+1$, $k = w$ and $v = n$. Applying the formula

$$b = \frac{\lambda \binom{v}{t}}{\binom{k}{t}}$$

from page 60 of [20] (with $\lambda = 1$ for a Steiner system) gives equality in the statement of the theorem.

Now suppose that the inequality in the statement of the theorem holds with equality. Then

$$A(n, 2\delta, w) = \frac{n}{w} A(n-1, 2\delta, w-1)$$

and in general

$$A(n-i, 2\delta, w-i) = \frac{n-i}{w-i} A(n-i-1, 2\delta, w-i-1) \text{ for } i = 0, 1, 2, \ldots, w-\delta.$$

The mean of the number of 1's in the columns of B is equal to the maximum number of 1's and equals $A(n-1, 2\delta, w-1)$. Choose any set $\{j_1, j_2, \ldots, j_{w-\delta+1}\}$ of coordinates. If all codewords (i.e. rows of B) with a 1 in the j_1 coordinate are chosen and this coordinate is deleted from the code, a code of length $n-1$, minimum distance 2δ, constant weight $w-1$ and $A(n-1, 2\delta, w-1)$ codewords is obtained. Repeating this process, codes with $A(n-1, 2\delta, w-1)$, $A(n-2, 2\delta, w-2)$, $\ldots$, $A(n-w+\delta-1, 2\delta, \delta-1)$ codewords are obtained, where rows have been selected from the original matrix B with 1's in coordinates $\{j_1, j_2, \ldots, j_{w-\delta+1}\}$. Since $A(n-w+\delta-1, 2\delta, \delta-1) = 1$, the original code forming the matrix B corresponds to a Steiner system $S(w-\delta+1, w, n)$. □

It is interesting to observe here that a Steiner system construction gives $A(50, 12, 8) = 350$ (i.e. 350 hopping lists). Typical software in current use will generate only around 100 lists. Our motivation is to extend at least some of the benefit of Steiner system constructions to other sets of parameters.

4 Lexicographic codes

In this section lexicographic codes will be evaluated. These have been considered in detail by Conway and Sloane [10]. Here lexicographic codes (for relevant sets of parameters) can be used as a baseline against which other codes can be judged. The lexicographic codes can be compared with the best codes known, and with upper bounds listed in [7, 1, 33, 34].

In this section a binary codeword $(c_1, c_2, \ldots, c_n)$ of weight w will be represented by an integer vector $(v_1, v_2, \ldots, v_w)$ where $1 \leq v_i \leq n$, $v_1 < v_2 < \ldots < v_w$, $c_v = 1$ for $v = v_j$, $j \in \{1, 2, \ldots w\}$ and $c_v = 0$ otherwise.

A vector $(v_1, v_2, \ldots, v_{(j-1)}, v_j, \ldots, v_w)$ *lexicographically precedes* a vector $(u_1, u_2, \ldots, u_{(j-1)}, u_j, \ldots, u_w)$ if (for some j) $v_j < u_j$ and $(v_1, v_2, \ldots, v_{(j-1)}) = (u_1, u_2, \ldots, u_{(j-1)})$. This precedence relation can then be used to construct a *lexicographic ordering* of the vectors.

The parameters under consideration are those discussed in section 2. Thus codes are constructed here with length n (usually $n \leq 63$) and constant weight w ($3 \leq w \leq 8$). The minimum distance d considered here is often $2w - 4$ (so $2 \leq d \leq 12$), but other cases are also considered.

Generate vectors $V_i = (v_{i1}, \ldots, v_{iw})$, $(i = 1, 2, \ldots, \binom{n}{w} = m)$, lexicographically with $V_1 = (1, 2, \ldots, w)$, $V_2 = (1, 2, \ldots, w-1, w+1)$, $\ldots$, $V_m = (n-w+1, n-w+2, \ldots, n-1, n)$. In the integer vector V_i the component v_{ij} is the position in which the j'th 1 appears in the binary codeword represented by V_i. From these lexicographically ordered vectors V_i a subset $L = \{a_1, \ldots, a_A\}$ is sought such that a_i and a_j $(i \neq j)$ have at most $o = w - \frac{d}{2}$ elements in common (or *overlaps*), and A approximates to its maximum value $A(n, d, w)$.

A basic computational method is to select a subset $L_{init} = \{a_1, \ldots, a_N\}$ (meeting the minimum distance condition) randomly. The current subset $L_{current}$ is set to L_{init}. All the vectors V_i are tested in some order, adding V_i to $L_{current}$ if V_i has distance at least d from all previously selected vectors in $L_{current}$. If the chosen order is $V_q, V_{q+1}, \ldots, V_m, V_1, \ldots, V_{q-1}$ the method can be denoted by $M(N, q)$. If the order is $V_j = (s, s+1, \ldots, s+w-1), V_{j+1}, \ldots, , V_m, V_1, \ldots, V_{j-1} = (s-1, n-w+2, \ldots, n-1, n)$ the method is denoted by $M(N, \ 's')$.

4.1 Results

All runs were performed on a Pentium II 350MHz PC using the Gnat Ada95 compiler. Some typical versions of the Ada code used can be found in Appendix 12. Using $M(1,1)$ the computational time T (in hours) to produce one A-value is given very approximately by $T = 2 \times 10^{-9}\binom{n}{w}$. Then, for example, $(n,d,w) = (60,12,8)$ is feasible with $T = 5$ (hours). Results for several runs correspond to several different choices of L_{init}.

n	d	w	$w-\frac{d}{2}$	d	$A(n,d,w)$	Best A	Comments	time
17	10	10	5	$2w-10$	**5**	**5**		0.07s
20	8	7	3	$2w-6$	**80**	43	after several runs	0.6s
						64	16 times in 10^3 starts	5m
20	10	8	3	$2w-6$	**17**	**17**		0.4s
21	10	9	4	$2w-8$	**27**-35	22	1st run, hard to improve	0.02m
						24	3 times in 2×10^3 starts	48m
22	10	9	4	$2w-8$	**35**-51	31	once in 400 starts	16m
22	10	10	5	$2w-10$	**46**-73	36	after about 10 runs	0.08m
24	8	8	4	$2w-8$	**759**	724	after a few runs	10s
						165-**759**	variable. 200 starts	
24	10	8	3	$2w-6$	**38**-60	33	once in 300 starts	17m
24	10	11	6	$2w-12$	**95**-223	76	4 times in 100 starts	60m
24	10	12	7	$2w-14$	**122**-247	78	second run	0.8m
28	6	7	4	$2w-8$	**4680**	1982-2005	40 random starts	44m
28	10	7	2	$2w-4$	**36**	30	an early run	4s
28	10	10	5	$2w-10$	**210**-821	180	an early run	252s
36	10	7	2	$2w-4$	?-180	77	twice in 100 random starts	70m
50	12	8	2	$2w-4$	**350**	104-107	13 random starts	715m
60	8	6	2	$2w-4$	?-1650	982	3 runs each 17m	51m
60	12	8	2	$2w-4$	?-562	203	1 run	5h

Table 1: Values of A for lexicographic codes constructed using $M(1,1)$ (several or many random starts) compared with known bounds

4.2 Comments on results

The results are surprisingly good from the point of view of the A-values achieved, and run times are satisfactory. The result $A = 107$ when $A(50,12,8) = 350$ can

be obtained by a Steiner system construction is very disappointing. This was the observation which originally motivated this work.

4.3 Some further results

$A(10, 4, 4) = 30$ is obtained from a Steiner system. $M(N, 1)$ usually gave A about 20, with $A = 30$ achieved in a very small proportion of runs (0% for $N = 1$, 0.9% for $N = 3$ appears best.)

$A(14, 6, 7) = 42$. Typically $M(1, 1)$ gave $A = 28$ with $T = 1.4 \times 10^{-5}$ (hours). The best $M(N, 1)$ was $M(2, 1)$ ($A = 39$ in 0.03% of runs). $M(2,' 2')$ gave $A = 42$ in 0.6% of runs.

$A(24, 6, 7) \geq 1368$. Typically $M(1, 1)$ gave $A = 815$ with $T = 1.4 \times 10^{-3}$ (hours). The best A obtained was $A = 830$ using $M(3,' 2')$.

$A(28, 6, 7) = 4680$ is obtained from a Steiner system. Typically $M(1, 1)$ gave $A = 1990$ with $T = 1.4 \times 10^{-3}$ (hours).

$A(50, 12, 8) = 350$ is obtained from a Steiner system. $M(2,' 2')$ gave $A = 110$ in 2 of 10 runs and $M(3,' 2')$ gave $A = 111$ in 1 of 48 runs.

4.4 Results based on counting overlaps

Suppose that a set $L_{init} = \{a_1, a_2, \ldots\}$ ($a_i \in \{V_i\}$) of vectors meeting the minimum distance condition is given. Let $o_{i,j}$ represent the number of overlaps between a_i and a_j (i.e. the number of positions where the corresponding binary codewords are both 1). Let $o_i = \sum_{j=1, j \neq i}^{A} o_{i,j}$ (i.e. the total number of overlaps that a_i has with all other elements of $L_{current}$), and let $R = \max_i(o_i) - \min_i(o_i)$. From computer experiment it was found that the greater the A value, the smaller the value of R. For example consider $(n, d, w) = (14, 6, 7)$. For a run with $A = 27$ the code has $R = 12$. For a run with $A = 39$ the code has $R = 7$. For two runs with $A = A(14, 6, 7) = 42$ the codes have $R = 2$ and $R = 0$ respectively.

This suggests a modified approach. Starting from a randomly chosen set L_{init}, this is repeatedly increased by adding another vector a_i. The next a_i is selected by finding eligible V_i for which o_i is a maximum. This method is denoted M' (using the same

notation otherwise as for M). The point of the modification is that fewer of the remaining eligible V_i are precluded from subsequently joining $L_{current}$. The difficulty is that run times are much longer, since for every a_i added to $L_{current}$ it might be necessary to inspect many V_i. Approximately, $T' = \frac{A^2 T}{40}$ (hours) might be expected.

With this modification and $(n, d, w) = (14, 6, 7)$, $M'(2,'2')$ gave $A = 42$ in 17 of 50 runs and $M'(2, 2037)$ gave $A = 42$ in 23 of 50 runs, with $T' = 0.02$ mins. For $(n, d, w) = (24, 6, 7)$, $M'(2,'2')$ gave $A = 856$ in just one run, but with $T' = 18$ hours. Some reductions in run time are possible, but on this basis, if $A = 140$ were achievable for $(n, d, w) = (50, 12, 8)$ the run time might be expected to be about 500 hours.

4.5 Randomly constructed codes

When run time is likely to be a problem, reasonably good codes can be constructed randomly instead of lexicographically. Words of weight w are chosen randomly and tested to see whether they meet the distance condition with previously chosen codewords. If they do, they are added to the code. The ultimate number of codewords selected is smaller than can be obtained by lexicographic search, but more codewords can be obtained quickly. Results are included in section 10.

4.6 Summary

Lexicographic codes appear to be reasonably good generally, except when a Steiner system exists when, with a notable exception, they appear to be poor. Some improvements to the algorithm are possible, although run times may increase. In general the run times will be too long for some of the larger parameter sets of interest.

One pragmatic approach is to make a conservative estimate of the run time for lexicographic search (say $T = 3 \times 10^{-9} \binom{n}{w}$ hours on a Pentium II 350MHz PC using the Gnat Ada95 compiler). If the predicted run time is too long, the random search method should be used in place of lexicographic search.

5 Linear fractional transformation and linear transformation schemes

In section 3 it was indicated why Steiner systems are of particular interest. In this section constructions of some Steiner systems using linear fractional transformations are described. The constructions presented will then be independent of the ideas and terminology of finite projective geometry. Hopefully this will make the constructions more accessible to Engineers concerned with the design of frequency hopping lists. It should be noted, however, that use of this construction still requires the ability to carry out arithmetic in finite fields. It is also interesting to consider whether generalisations of the linear fractional transformation construction might construct other constant weight codes which are useful for our purposes.

Steiner systems $S(3,\, q+1,\, q^r+1)$ exist for any prime power q [12] (see also [20] and references therein). These systems can be constructed using linear fractional transformations in finite fields. Finite fields $GF(q)$ with q elements exist whenever q is a power of a prime. An account of the construction of finite fields can be found in [24], where a proof of the following theorem can also be found:

Theorem 5.1 *A finite field $GF(q^r)$ has a subfield $GF(q^s)$ if and only if s is a divisor of r.*

It will now be shown how linear fractional transformations can be used to construct Steiner systems $S(3,\, q+1,\, q^r+1)$ for q a prime power. It can be seen that the blocks of such a system correspond to the codewords of a constant weight code with $A(q^r+1,\ 2q-2,\ q+1) = \frac{q^{r-1}(q^{2r}-1)}{q^2-1}$.

Let $R = GF(q) \cup \infty$ and $\mathcal{R} = GF(q^r) \cup \infty$ where the element with symbol ∞ will have arithmetic properties analogous to ∞ in the extended real numbers. Then define a linear fractional transformation $L : R \to \mathcal{R}$ by

$$L(z) = \frac{k_{11}z + k_{12}}{k_{21}z + k_{22}}, \quad k_{11}k_{22} - k_{12}k_{21} \neq 0, \quad z \in R, \quad k_{11}, k_{12}, k_{21}, k_{22} \in GF(q^r)$$

The nondegeneracy condition $k_{11}k_{22} - k_{12}k_{21} \neq 0$ is necessary to ensure that $L(z)$ is not constant. $L(\infty) = \frac{k_{11}}{k_{21}}$, and $L(-k_{22}k_{21}^{-1}) = \infty$.

Such a transformation can also be written in matrix form:

$$L_K(z) = K \begin{bmatrix} z \\ 1 \end{bmatrix}, L_K(\infty) = K \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{5.1}$$

where $k_{ij} \in GF(q^r)$, $k_{11}k_{22} - k_{12}k_{21} \neq 0$, $z \in R := GF(q) \cup \infty$, and $K = \begin{bmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{bmatrix}$. In this case $L_K(z)$ is represented as a 1-dimensional subspace so that if $L_K(z_1) = \begin{pmatrix} p_1 \\ p_2 \end{pmatrix}$ and $L_K(z_2) = \begin{pmatrix} q_1 \\ q_2 \end{pmatrix}$ then $L_K(z_1) = L_K(z_2)$ if and only if $\begin{pmatrix} p_1 \\ p_2 \end{pmatrix} = c \begin{pmatrix} q_1 \\ q_2 \end{pmatrix}$, $c \neq 0$, $c \in GF(q^r)$.

The mapping L has two important properties:

1. L is a one-to-one mapping so $|L(R)| = |R|$. Indeed, if L is extended to a mapping $L : \mathcal{R} \to \mathcal{R}$ then L is still a one-to-one mapping.
2. If for two linear fractional transformations L_1 and L_2, $L_1(R) \neq L_2(R)$ then $|L_1(R) \cap L_2(R)| < 3$.

The first of these properties will be stated and proved with L in matrix form L_K:

Theorem 5.2 *Let F denote a finite field and L_K denote a linear fractional transformation*

$$L_K(z) = K \begin{bmatrix} z \\ 1 \end{bmatrix}, L_K(\infty) = K \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{5.2}$$

($k_{ij} \in F$, $k_{11}k_{22} - k_{12}k_{21} \neq 0$, $z \in \mathcal{R} := F \cup \infty$), where $K = \begin{bmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{bmatrix}$. Then L_K transforms distinct points of $\mathcal{R}$ into distinct points, i.e. $|Q| = |L_K(Q)|$ for any $Q \subset \mathcal{R}$.

Proof. Suppose $L_K(z_1) = L_K(z_2)$ ($z_1 \neq z_2$). If $z_1, z_2 \neq \infty$, then $K \begin{bmatrix} z_1 \\ 1 \end{bmatrix} = cK \begin{bmatrix} z_2 \\ 1 \end{bmatrix}$ (for some $c \in F$). If one of the z_i is ∞, say $z_2 = \infty$, then $K \begin{bmatrix} z_1 \\ 1 \end{bmatrix} = cK \begin{bmatrix} 1 \\ 0 \end{bmatrix}$. As K is nonsingular, multiplying throughout by K^{-1} in both cases, gives contradictions. □

Two different proofs of the second property will be given. The first makes use of two lemmas.

Lemma 5.3 *Given two sets $\{x_i\}$ and $\{y_i\}$ $(i = 1, 2, 3)$ of three distinct points of R, there exists a linear fractional transformation L such that $L(x_i) = y_i$ for $i = 1, 2, 3$. L maps R onto R. The same result is also true for $\mathcal{R}$.*

Proof. There exists a transformation $L' : R \to R$ which maps x_1, x_2, x_3 to $0, 1, \infty$ respectively. If none of x_1, x_2, x_3 is infinity then this transformation is

$$L'(z) = \frac{\frac{(z-x_1)}{(z-x_3)}}{\frac{(x_2-x_1)}{(x_2-x_3)}}.$$

Otherwise the transformation is $L'(z) = \frac{x_2-x_3}{z-x_3}$ $(x_1 = \infty)$, $L'(z) = \frac{z-x_1}{z-x_3}$ $(x_2 = \infty)$, or $L'(z) = \frac{z-x_1}{x_2-x_1}$ $(x_3 = \infty)$. Similarly, there exists a transformation $L'' : R \to R$ which maps y_1, y_2, y_3 to $0, 1, \infty$ respectively. Then as L'' is invertible, $(L'')^{-1}L'$ is the transformation required in the statement of the lemma. Clearly the same proof works if R is replaced by $\mathcal{R}$. □

Lemma 5.4 *The linear fractional transformation $L : R \to R$ whose existence was proved in lemma 5.3 is unique. Similarly the transformation $L : \mathcal{R} \to \mathcal{R}$ of the same lemma is unique.*

Proof. Note first that any linear fractional transformation $\tilde{L}$ such that $\tilde{L}(0) = 0$, $\tilde{L}(1) = 1$, $\tilde{L}(\infty) = \infty$ is unique. This follows by direct substitution into the general form of the transformation.

Then using the notation introduced in lemma 5.3 it can be seen that

$$L''L(L')^{-1}(0, 1, \infty) = (0, 1, \infty).$$

Thus $L''L(L')^{-1}$ is unique and as the transformations are invertible it follows that L is unique. A similar argument can be used to show that $L : \mathcal{R} \to \mathcal{R}$ is unique. □

Theorem 5.5 *If $L_1, L_2 : R \to \mathcal{R}$ are linear fractional transformations then either $|L_1(R) \cap L_2(R)| \leq 2$ or $L_1(R) = L_2(R)$. Given three points of $\mathcal{R}$ there is a linear fractional transformation L_i for which $L_i(R)$ contains these three points.*

Proof. If $L_1(x_i) = L_2(y_i)$ for two sets $\{x_i\}, \{y_i\}$ of three points from R then $L_1 = L_2L$ (where $L : R \to R$ is the first transformation of lemma 5.3). Thus $L_1(R) = L_2(R)$.

The existence of the mapping L_i stated follows from the final statement of lemma 5.3. □

Thus each linear fractional transformation is uniquely defined by its mapping at 3 points. Hence there exist precisely $3!\begin{pmatrix} |\mathcal{R}| \\ 3 \end{pmatrix}$ different linear fractional transformations and precisely $3!\begin{pmatrix} |R| \\ 3 \end{pmatrix}$ of them map R onto R. From theorem 5.5 $L_1(R) = L_2(R)$ if and only if $L_1 = L_2 L$, where L maps R onto R. Hence there are $3!\begin{pmatrix} |R| \\ 3 \end{pmatrix}$ different transformations that map R onto the same subset. Finally it can be seen that the number of blocks $L(R)$ in the Steiner system is given by the formula

$$M = \frac{\begin{pmatrix} |\mathcal{R}| \\ 3 \end{pmatrix}}{\begin{pmatrix} |R| \\ 3 \end{pmatrix}}. \tag{5.3}$$

Example. The Steiner system $S(3, 8, 50)$ that generates a constant weight code of length $n = 50$, weight $w = 8$ and distance $d = 2(w - 2) = 12$ is constructed via linear fractional transformations after putting $q = 7$ and $r = 2$ in the definitions of R and $\mathcal{R}$. In this case the number M of the blocks in the Steiner system and, correspondingly, words in the code equals 350.

5.1 A proposed procedure for constructing blocks

When a Steiner system exists the $A(n, d, w)$ vectors can be found very quickly and easily. It is convenient to carry out the Galois field calculations in Maple, although run times can be one or two orders of magnitude larger than when using Ada. Using the linear fractional transformation, a maximum of just under n^3 transformations need testing to produce the final set. In practice the set is generated by far fewer transformations. Thus, for example, $A(50, 12, 8) = 350$ vectors can be generated in about 8 minutes using about 2% of the transformations. Considering all transformations would take about 8 hours

It is interesting to attempt to apply the method when no Steiner system exists. Here a computer search to generate blocks of w elements from a set of n elements via linear fractional transformations is proposed.

Procedure 5.6 *Step 1. Choose the maximal value v from the set $\{s : s \leq n,\ s = q^r + 1,\ q \quad \text{is} \quad \text{prime}\}$, let $F := GF(v-1)$ and put $\mathcal{R} := F \cup \infty$.*

Step 2. Choose a subset $Q \in \mathcal{R}$ such that $|Q| = w$.

Step 3. Apply all possible linear fractional transformations defined in (5.1) to the elements of Q. For this purpose fix, for example, $k_{22} = 1$ and consider $(v-1)^2(v-2)$ transformations with different values of k_{11}, k_{12} and k_{21} with $\det K \neq 0$. Then fix $k_{21} = 1$, $k_{22} = 0$ and consider the remaining $(v-1)(v-2)$ transformations. Several (say 4) sets of blocks should be stored. The first (most important one) is denoted by N and the first block Q is included in it automatically. After that each new block $L_K(Q)$ is compared with the blocks already included in N. If it has no more than two overlappings with each of them it is also included in N. If there are exactly three overlappings it is included in N_1. If there are four overlappings the block is included in N_2 etc.

Step 4. After all the transformations L_K are applied consider perturbations of the blocks from N_1, N_2 and so on, starting from the last block included in N_1. Try changing some elements of the blocks by 1 (there can be at most $3^w - 1$ perturbations of each block). Compare each perturbation with the blocks from N. If there are no more than two overlappings include the perturbed block into N and consider perturbations of another block. Consider other perturbations of the same block until all the $3^w - 1$ possibilities are checked, and then consider N_2 etc.

Step 5. If the number of blocks in N is less than any upper bound, choose another subset $\widetilde{Q} \in \mathcal{R}$ such that $|\widetilde{Q}| = w$. Compare the blocks $L_K(\widetilde{Q})$ with those in N and include them into N if there are no more than two overlappings. Repeat for several subsets $\widetilde{Q}$.

Step 6. Repeat the procedure for several different choices of Q.

It may also be reasonable to modify Step 1 of Procedure 5.6 into:

Step $1'$. Choose the minimal value v from the set $\{s : s > n,\ s = q^r + 1,\ q \quad \text{is} \quad \text{prime}\}$ and put $\mathcal{R} := F \cup \infty$, $F := GF(v-1)$. Some blocks will have to be deleted if they contain elements greater than n.

Again the set N can be generated with just under n^3 transformations. $|N|$ (obtained by steps 1 to 3) is generally smaller than can be obtained by lexicographic search. Step 5 sometimes increases the number of blocks slightly, often not at all. Step 4

(and the associated parts of step 3) have not been implemented, as it seems simpler and more useful to use N as a seed for further lexicographic search. The results given here are not encouraging, but the combination of Steiner systems and lexicographic search will be considered further in section 7. Here a small number of illustrative results for the estimates of $A(n, d, w)$ are given.
(28,10,7): Lexicographic search typically gives $A = 30$. Steps 1 to 3 give an initial estimate of 22, increasing to 30 when extended by lexicographic search.
(26,12,8): Lexicographic search typically gives $A = 10$ to $A = 13$. Steps 1 to 3 give an initial estimate of 7 or 8, increasing to 10 when extended by step 5 or lexicographic search.
(30,12,8): Lexicographic search typically gives $A = 17$ to $A = 19$. Steps 1 to 3 give an initial estimate of 12, increasing to 14 when extended by step 5. This increases to 17 when extended by lexicographic search.

5.2 An alternative proof of theorem 5.5 and two generalizations

In this subsection an alternative proof of the first part of theorem 5.5 will be given, i.e. that either $|L_1(R) \cap L_2(R)| \leq 2$ or $L_1(R) = L_2(R)$. This proof is capable of some generalization.

Proof. Suppose that $L_1(x_i) = L_2(y_i)$ for two sets $\{x_i\}$ and $\{y_i\}$ of 3 distinct points from R, i.e. $|L_1(R) \cap L_2(R)| \geq 3$. Then $L_2^{-1}L_1(x_i) = y_i$. If the linear fractional transformation $L = L_2^{-1}L_1$ is written in matrix form as L_K, then the last equalities are equivalent to $KX_i = c_iY_i$, where $X_i = \begin{bmatrix} x_i \\ 1 \end{bmatrix}$ if $x_i \neq \infty$, $X_i = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ if $x_i = \infty$, $Y_i = \begin{bmatrix} y_i \\ 1 \end{bmatrix}$ if $y_i \neq \infty$, $Y_i = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ if $y_i = \infty$ and $c_i \in GF(q^r)$ $(c_i \neq 0)$. If $X = [X_1\ X_2\ X_3]$, $Y = [Y_1\ Y_2\ Y_3]$, $C = \mathrm{diag}\{c_1, c_2, c_3\}$ then $KX_i = c_iY_i$ can be rewritten in matrix form:

$$KX = YC. \tag{5.4}$$

As any three vectors must be dependent there is a vector $f \neq 0$ with entries from $GF(q)$ such that Xf=**0**. Notice that the vectors X_i are pairwise linearly independent. Thus the entries of f are nonzero elements of $GF(q)$ and f is unique (up to a scalar). The same is true about Y and some vector g. On the other hand, it follows from (5.4) that $YCf = KXf = \mathbf{0}$ and so from the uniqueness up to a scalar it follows that $g = cCf$ $(c \in GF(q^r),\ c \neq 0)$. This is possible only if $cc_i \in GF(q)$. Hence the

entries of cYC belong to $GF(q)$. Therefore in view of (5.4) the same is true about the entries of cK. In other words $L_K(R) = R$, i.e. $L_1(R) = L_2(R)$. $\square$

Denote by Q the set of subspaces $\{th\}$, where $t \in GF(q^r)$ and the generating vectors h belong to the set $\{h \,|\, h = \begin{bmatrix} x_i \\ 1 \end{bmatrix}, \quad x_i \in GF(q)\} \cup \{\begin{bmatrix} 1 \\ 0 \end{bmatrix}\}$. It is important for the alternative proof of the first part of theorem 5.5 that if the entries of the 2×2 matrix K ($\det K \neq 0$) belong to $GF(q)$, then $KQ = Q$. Here K maps a subspace th into the subspace tKh ($t \in GF(q^r)$). The equality $KQ = Q$ is closely connected with the fact that Q is a unique maximal set with the generating vectors pairwise linearly independent and having entries from $GF(q)$. Although it seems difficult to obtain an analogue of $KQ = Q$ for blocks overlapping in more than two points, some generalization of the proof is possible.

Theorem 5.7 *Suppose two finite fields F and $F_0 \subset F$ are given. Let a block Q of $w > m$ 1-dimensional subspaces tX_i ($X_i \in F_0^m$, $1 \leq i \leq w$) be chosen so that any m ($m > 1$) generating vectors X_i from the block are linearly independent, and let K_1 and K_2 be two arbitrary nondegenerate $m \times m$ matrices with entries from F. Let L_{K_1}, L_{K_2} be the corresponding linear fractional transformations. Then if the blocks $L_{K_1}(Q)$ and $L_{K_2}(Q)$ overlap in more than m points, it follows that for some scalar $c \neq 0$ all the entries of $cK_2^{-1}K_1$ belong to F_0.*

Proof. Suppose that the blocks $L_{K_1}(Q)$ and $L_{K_2}(Q)$ overlap in more than m points (subspaces). In other words $K_1X = K_2YC$,

$$X = [X_1\ X_2\ \ldots\ X_{m+1}], \quad Y = [Y_1\ Y_2\ \ldots\ Y_{m+1}], \quad C = \operatorname{diag}\{c_1, c_2, \ldots, c_{m+1}\},$$

where the columns X_i and Y_i are the generating vectors of the distinct subspaces of Q with entries from F_0. Putting $K := K_2^{-1}K_1$ the equality $K_1X = K_2YC$ can be rewritten in the form $KX = YC$ and the theorem proved in a similar way to the alternative proof of the first part of theorem 5.5. $\square$

Pursuing this generalization, replace subspaces by vectors and the matrix representations of linear fractional transformations by matrix representations of linear transformations

Theorem 5.8 *Suppose that two finite fields F and $F_0 \subset F$ are given. Let Q be a block of w m-dimensional vectors X_i ($X_i \in F_0^m$, $1 \leq i \leq w$), chosen so that any m vectors from the block are linearly independent, and let K_1 and K_2 be two*

arbitrary nondegenerate $m \times m$ matrices with entries from F. Let M_{K_1} and M_{K_2} be the corresponding linear transformations. Then if the blocks of vectors $M_{K_1}(Q)$ and $M_{K_2}(Q)$ overlap in more than $m-1$ points, it follows that all the entries of $K_2^{-1}K_1$ belong to F_0.

Proof. Chose m vectors in which M_{K_1} and M_{K_2} overlap. Let X and Y be $m \times m$ matrices such that the images of the columns of X under M_{K_1} and the images of the columns of Y under M_{K_2} are these m vectors (in the same order). Then $K_1X = K_2Y$ so $K_2^{-1}K_1 = YX^{-1}$. $\square$

Given cardinality w and fields F and $F_0 \subset F$ theorem 5.8 generates a procedure for constructing blocks with fewer than m overlappings.

Procedure 5.9 *Step 1. Choose (if possible) a block Q of w m-dimensional vectors X_i ($X_i \in F_0^m$, $1 \le i \le w$) so that any m vectors from the block are linearly independent.*

Step 2. Define a set $\mathcal{K}$ of all nondegenerate $m \times m$ matrices K with the entries from F_0 such that $|KQ \cap Q| \ge m$.

Step 3. Determine classes C_i of nondegenerate $m \times m$ matrices with entries from F according to:

$$K_{l+1} \notin \bigcup_{i=1}^{l} C_i, \quad C_i = \{K_iK : \ K \in \mathcal{K}\}.$$

Then construct blocks K_iQ of vectors.

It can be seen that any two of the sets of vectors $\{K_iQ\}$ have no more than $m-1$ overlappings. Suppose that $K_lQ, K_{l'}Q \ \ (l > l')$ overlap in more than $m-1$ vectors. Then from theorem 5.8 $K_{l'}^{-1}K_l \in F_0$. As $|K_{l'}^{-1}K_lQ \cap Q| = |K_lQ \cap K_{l'}Q| \ge m$ it follows that $K_{l'}^{-1}K_l \in \mathcal{K}$. Thus $K_l \in C_{l'}$, which would contradict the definition of the classes.

6 Applications and construction of q-ary and mixed codes

Both binary and q-ary codes have been extensively studied and widely applied. Much less is known about the so called "mixed" codes. A $q_1 \mid q_2$ *mixed code* denotes a set of vectors (words) of length $n = n_1 + n_2$, where the first n_1 entries take values from

0 to $q_1 - 1$ and the remaining n_2 entries take values from 0 to $q_2 - 1$. A theory of mixed codes is developing and some references can be found in [6], where only mixed binary/ternary codes are treated. Nevertheless some important theorems in [6] admit generalizations for the case of the arbitrary mixed codes.

Here the application of $q-ary$ and mixed codes to the construction of binary constant weight codes will be considered. Starting from his important work [16] on incomplete block designs, Hanani has used q-ary codes ($q > 2$) in the form of the m-tuples of T-systems for solving incomplete block design and related constant weight binary code problems. (See [20] and references therein on the generalization of the notion of the T-system and connections between T-systems and orthogonal arrays.) In [14] cyclic q-ary codes were applied for the construction of the cyclic constant weight binary codes.

The use of q-ary and mixed codes to construct binary constant weight codes is based on the following theorem:

Theorem 6.1 *Suppose that C is a (q_1, q_2, n_1, n_2, d) code. Then there exists a constant weight binary code $\widetilde{C}$ with $|C|$ words of length $\widetilde{n} = q_1 n_1 + q_2 n_2$, minimum distance $2d$, and weight $w = n_1 + n_2$. This code is constructed in the following way: take each word $X = [x_1\ x_2\ \ldots\ x_n]$ of the initial $q_1 \,|\, q_2$ code and substitute each x_i in it by a row of q_1 entries if $i \leq n_1$ and by a row of q_2 entries if $i > n_1$. Of those entries (numbering the entries starting from 1) the entry with number $x_i + 1$ is equal to 1 and all other entries are 0.*

Proof. To show that the distance in $\widetilde{C}$ equals $2d$ notice that for any pair of codewords of C the number of positions in which the two codewords differ is at least d. This gives at least $2d$ positions, where one of the codewords equals '1' and another equals '0'. □

The case $n_2 = 0$ corresponds to q-ary codes. Theorem 6.1 admits also easy generalizations for the case of $q_1 \,|\, q_2 \,|\, \ldots \,|\, q_n$ mixed codes.

This theorem gives a general approach, allows significant improvement of some of the existing results on the number of words for constant weight binary codes and can achieve quite good results in many other cases. It is important also that computer search of the nonbinary codes usually takes much less time than search of the binary codes of the corresponding length. Sometimes it proves practically useful that the 1's in the codewords constructed in this way appear regularly (once in each interval).

Results using Theorem 6.1 will be seen in section 10. For these results the underlying $q-$ary or mixed code is obtained (faily easily) by lexicographic search. The results are not as good as other methods described in this report. However, much better results can be obtained if the underlying q-ary code meets the Greismer bound. These codes may also be combined with other codes to give good results. These techniques may prove important in some radio systems where n and w are much larger than the (GSM relevant) values considerd here. Search methods would fail in such circumstances.

Let (q_1, q_2, n_1, n_2, d) denote a code, of length $n = n_1 + n_2$, where the first n_1 entries of any codeword take values from 0 to $q_1 - 1$, the next n_2 entries take values from 0 to $q_2 - 1$, and the minimal Hamming distance between different codewords is no less than d. Denote by $A_{q_1,q_2}(n_1, n_2, d)$ the maximal possible cardinality of (q_1, q_2, n_1, n_2, d). ($A_q(n, d)$ denotes the maximal possible cardinality of the pure q-ary codes of length n and minimal distance d, and q is omitted in the case $q = 2$.) For the codewords v and y denote by $d(v, y)$ the Hamming distance between v and y. Putting $C = (q_1, q_2, n_1, n_2, d)$, define $S_{k,j}(C)$ (or simply $S_{k,j}$) for nonnegative integers k, j by:

$$S_{k,j} := (q_1 - 1)^k (q_2 - 1)^j \left(\begin{array}{c} n_1 \\ k \end{array} \right) \left(\begin{array}{c} n_2 \\ j \end{array} \right).$$

Finally, denote the number $|C|$ of codewords by M.

6.1 Bounds

Consider a mixed code $C = (q_1, q_2, n_1, n_2, d)$. It can easily be seen that the spheres with centres in C and radius $\left[\frac{d-1}{2}\right]$ do not intersect. Each sphere contains $\sum_{k+j \le \left[\frac{d-1}{2}\right]} S_{k,j}$ different words, only one of which belongs to C, and the total number of possible words equals $q_1^{n_1} q_2^{n_2}$. This gives the simplest estimate

$$|C| \le \frac{q_1^{n_1} q_2^{n_2}}{\sum_{k+j \le \left[\frac{d-1}{2}\right]} S_{k,j}}. \tag{6.1}$$

A somewhat stronger estimate can be obtained as a generalization of the proof of Proposition 3.1 of [6], where the case $q_1 = 2$, $q_2 = 3$, and $d = 3$ was considered, (compare also [36]). Let $t = \left[\frac{d-1}{2}\right]$ and define:

$$\mathcal{U} = \{(k, j) : k + j = t, \quad 0 \le k \le n_1, \quad 0 \le j \le n_2\};$$

where the pairs of integers (k, j) satisfy the condition:

$$((q_1 - 1)(n_1 - k) + (q_2 - 1)(n_2 - j)) \bmod (t + 1) \neq 0. \tag{6.2}$$

Theorem 6.2 *Let $C = (q_1, q_2, n_1, n_2, d)$, where $n_1 > 0$, $n_2 > 0$ and d is an odd integer with $d = 2t+1 > 1$ for positive integer t. Suppose $\mathcal{U} \neq \emptyset$ (with condition (6.2) satisfied) and choose integers t_1 and t_2 such that $(t_1, t_2) \in \mathcal{U}$. Then an upper bound for $|C|$ is given by the inequality*

$$|C| \leq \frac{q_1^{n_1} q_2^{n_2}}{(\sum_{k+j \leq t} S_{k,j} + \varepsilon)}, \tag{6.3}$$

where

$$\varepsilon = \max(1,\ ((q_1 - 1)n_1 + (q_2 - 1)n_2)^{-1} S_{t_1,t_2}) \tag{6.4}$$

The theorem is proved in [23].

Example 6.3 *Consider the codes $(3, 4, 5, 5, 5)$. Then $t = 2$, $S_{2,0} = 40$, $S_{0,2} = 90$ and $S_{1,1} = 150$. In particular, $\max_{t_1+t_2=2} S_{t_1,t_2} = S_{1,1}$. As $(q_1 - 1)(n_1 - 1) + (q_2 - 1)(n_2 - 1) = 20 \neq 0 \mod 3$, the pair $t_1 = t_2 = 1$ satisfies (6.2). Substituting this pair into (6.4) gives $\varepsilon = 6$. Additionally, $S_{0,0} = 1$, $S_{1,0} = 10$, $S_{0,1} = 15$, and so from (6.3), $|(3, 4, 5, 5, 5)| \leq \frac{(12)^5}{312} < 798$. Simple lexicographic search gives a code with 183 words, giving a first lower bound which may be capable of significant improvement.*

Remark 6.4 *The existence of u such that $d(u, C) = t + 1$ in the proof of Theorem 6.2 given in [23] means that the corresponding mixed code is not perfect. In other words if $\mathcal{U} \neq \emptyset$ the code is not perfect, and $\mathcal{U} = \emptyset$ is a necessary condition for the existence of the perfect mixed (q_1, q_2, n_1, n_2, d) code. The necessity of the condition $((q_1 - 1)(n_1 - t_1) + (q_2 - 1)(n_2 - t_2)) \bmod (t + 1) = 0$ for any $t_1 \geq 0$ and $t_2 \geq 0$ $(t_1 + t_2 = t)$ follows also from Theorem 1 (case $r = 1$) of [37].*

The whole set of necessary conditions for the existence of perfect codes from Theorem 1 of [37] can be obtained from the proof (given in [23]) of the following modification of Theorem 6.2. Following [37], consider vectors in the Cartesian product $Q_1 \times Q_2 \times \cdots \times Q_n$, where Q_i contains elements from 0 to $q_i - 1$, i.e. consider a mixed $q_1 \,|\, q_2 \,|\, \cdots \,|\, q_n$ code ($n_1 = n_2 = \cdots = n_n = 1$). Denote by V the set $\{1, 2, \ldots, n\}$. If $U \subseteq V$ put $\overline{U} = V \setminus U$.

Theorem 6.5 *Let C be a code in $Q_1 \times Q_2 \times \cdots \times Q_n$ with minimum distance $d = 2t+1$ $(t > 0)$. Suppose that for some integer $0 < r \leq t$ there exists a subset $U \subset V$ such that*

$$|U| = t - r + 1, \qquad \sum_{R \subset \overline{U},\, |R| = r} \prod_{j \in R} (q_j - 1) \neq 0 \bmod \begin{pmatrix} t + r \\ r \end{pmatrix}. \tag{6.5}$$

Then for any nonempty family $\mathcal{U}$ of subsets U that satisfy (6.5):

$$M \le \left((\sum_{U\in\mathcal{U}} (\min_{j\in U} q_j - 1)) \prod_{j\in V} q_j \right) \times \tag{6.6}$$

$$\left((\sum_{U\in\mathcal{U}} (\min_{j\in U} q_j - 1))(\sum_{T\subset V, |T|\le t} \prod_{j\in T} (q_j - 1)) + \sum_{U\in\mathcal{U}} \prod_{j\in U} (q_j - 1) \right)^{-1}.$$

Consider now a matrix G, the M rows of which are the codewords of C. By analogy with the proof of the Plotkin bound for the binary case (see [20])

$$2d \begin{pmatrix} M \\ 2 \end{pmatrix} \le \sum_{u\in C} \sum_{v\in C} d(u,v) \le n_1 K_1 + n_2 K_2, \tag{6.7}$$

where K_1 is an upper bound for the contribution of each of the first n_1 columns of G to the sum of distances $d(u,v)$, and K_2 is an upper bound for the contribution of each of the remaining n_2 columns of G. To estimate the sum $K(X)$ of distances between the entries of a column X with the entries taking values from 0 to $q-1$, denote by x_j the number of entries j in the column. Then, taking into account that $\sum_{j=0}^{q-1} x_j = M$, it follows that

$$K(X) = \sum_{j=0}^{q-1} x_j (M - x_j) = M^2 - \sum_{j=0}^{q-1} x_j^2. \tag{6.8}$$

Notice that

$$\min_{\sum_{j=0}^{q-1} x_j = M} \sum_{j=0}^{q-1} x_j^2 \ge q \left(\frac{M}{q} \right)^2 = \frac{M^2}{q}. \tag{6.9}$$

In view of (6.8) and (6.9), the inequality $K_p \le \frac{q_p - 1}{q_p} M^2$ is obtained. Substitute this estimate into the right hand side of (6.7) to derive

$$(M-1)d \le \left(n_1 \frac{q_1 - 1}{q_1} + n_2 \frac{q_2 - 1}{q_2} \right) M. \tag{6.10}$$

Inequality (6.10) can be rearranged to give:

Proposition 6.6 *Let $C = (q_1, q_2, n_1, n_2, d)$, where $d > \left(n_1 \frac{q_1-1}{q_1} + n_2 \frac{q_2-1}{q_2} \right)$, and let $M = |C|$. Then*

$$M \le d \left(d - n_1 \frac{q_1 - 1}{q_1} - n_2 \frac{q_2 - 1}{q_2} \right)^{-1}. \tag{6.11}$$

The estimate (6.11) can be improved by taking into account that all the values x_j are natural numbers. In this way the Plotkin bounds for the binary [20] and binary/ternary [6] codes can be obtained.

Example 6.7 *Notice that the Plotkin bound cannot be applied to the code* $(3,4,5,5,5)$ *in Example 6.3 because for these parameters* $n_1\frac{q_1-1}{q_1}+n_2\frac{q_2-1}{q_2}=7\frac{1}{12}>d$. *On the other hand Theorem 6.2 is inapplicable for the code* $(3,4,5,5,8)$ *as* d *is even, but the Plotkin bound yields* $|(3,4,5,5,8)|\le 8(8-7\frac{1}{12})^{-1}$, *i.e.,* $|(3,4,5,5,8)|\le 8$. *(The lower bound obtained by lexicographic search is* $|(3,4,5,5,8)|\ge 6$.*) An attempt to obtain an estimate of* $|(3,4,5,5,8)|$ *from the estimate of* $|(3,4,5,5,7)|$ *gives:*

$$S_{0,0}=1,\ S_{0,1}=15,\ S_{0,2}=90,\ S_{0,3}=270,\ S_{1,0}=10,\ S_{1,1}=150,\ S_{1,2}=900,$$

$$S_{2,0}=40,\ S_{2,1}=600,\ S_{3,0}=80,\ \varepsilon=(2\times 5+3\times 5)^{-1}S_{1,2}=36.$$

It follows from (6.3) that $|(3,4,5,5,8)|\le|(3,4,5,5,7)|<114$, *but the estimate is too large to be useful. Finally, in the case of the code* $(3,4,6,6,9)$ *the Plotkin bound works much better than Theorem 6.2. Since* $n_1\frac{q_1-1}{q_1}+n_2\frac{q_2-1}{q_2}=8\frac{1}{2}$, *the Plotkin bound yields* $|(3,4,6,6,9)|\le 18$ *(the lower bound by lexicographic search is* $|(3,4,6,6,9)|\ge 12$*). At the same time, in this case* $t=4$, $S_{0,4}=3^4\times 15$, $S_{1,3}=2\times 3^3\times 120$, $S_{2,2}=4\times 3^2\times 15^2$, $S_{3,1}=2^3\times 3\times 120$, *and* $S_{4,0}=2^4\times 15$. *Inequality (6.2) is* $(2(6-t_1)+3(6-t_2))\bmod 5\ne 0$, *and* $t_1=1$, $t_2=3$ *can be chosen to satisfy it. (If* $t_1=t_2=2$, *then* $(2(6-t_1)+3(6-t_2))=20$.*) By (6.4),* $\varepsilon=S_{1,3}/30=216$. *Additionally*

$$S_{0,0}=1,\ S_{0,1}=18,\ S_{0,2}=135,\ S_{0,3}=540,\ S_{1,0}=12,$$

$$S_{1,1}=216,\ S_{1,2}=1620,\ S_{2,0}=60,\ S_{2,1}=1080,\ S_{3,0}=160,$$

and (6.3) yields the estimate $|(3,4,6,6,9)|\le 129$, *which is many times less accurate than the Plotkin bound.*

6.2 Constructions of q-ary and mixed codes

Analogously to the binary and binary/ternary cases, the codes of the form $(c,c+\widetilde{c})$ (with $c\in C$ and $\widetilde{c}\in\widetilde{C}$) can be constructed.

Proposition 6.8 *Given two codes* $C=(q_1,q_2,n_1,n_2,d)$ *and* $\widetilde{C}=(\widetilde{q}_1,\widetilde{q}_2,\widetilde{n}_1,\widetilde{n}_2,\widetilde{d})$ $(n_1+n_2=\widetilde{n}_1+\widetilde{n}_2)$ *the* $(c,c+\widetilde{c})$ *code* $\mathcal{C}$, *where* $c\in C$ *and* $\widetilde{c}\in\widetilde{C}$, *is a* $(q_1,q_2,q_3,q_4,q_5,n_1,n_2,n_3,n_4,n_5,D)$ *code such that* $|\mathcal{C}|=|C||\widetilde{C}|$, $D=\min(2d,\widetilde{d})$, $q_3=\max(q_1,\widetilde{q}_1)$, $n_3=\min(n_1,\widetilde{n}_1)$, $q_5=\max(q_2,\widetilde{q}_2)$, $n_5=\min(n_2,\widetilde{n}_2)$. *The values of* q_4 *and* n_4 *are defined in the following way:* $q_4=\max(q_1,\widetilde{q}_2)$, $n_4=n_1-\widetilde{n}_1$ *if* $n_1\ge\widetilde{n}_1$, *and* $q_4=\max(q_2,\widetilde{q}_1)$, $n_4=\widetilde{n}_1-n_1$ *if* $n_1\le\widetilde{n}_1$. *Moreover* q_1,q_2,q_3,q_4 *and* q_5 *can take at most 4 different values. If* $q_1=\widetilde{q}_1$ *and* $q_2=\widetilde{q}_2$ *it can be assumed without loss of generality that* $q_1\le q_2$. *Thus* $C{=}(q_1,q_2,n_1+\widetilde{n}_1,n_2+\widetilde{n}_2,D)$ *if* $n_1\ge\widetilde{n}_1$, *and* $C{=}(q_1,q_2,2n_1,2n_2,D)$ *if* $n_1\le\widetilde{n}_1$.

The proof is analogous to the one given on page 76 of [20].

In particular, this proposition proves helpful if one needs to construct a mixed code from existing pure q_1 and q_2 codes.

Corollary 6.9 *Given two codes $C = (q_1, n, d)$ and $\widetilde{C} = (q_2, n, \widetilde{d})$ $(q_1 < q_2)$ the $(c, c+\widetilde{c})$ code $\mathcal{C}$, where $c \in C$ and $\widetilde{c} \in \widetilde{C}$, is a (q_1, q_2, n, n, D) code such that $|\mathcal{C}| = |C||\widetilde{C}|$ and $D = \min(2d, \widetilde{d})$.*

Proof. Put $n_1 = \widetilde{n}_2 = n$, $\widetilde{n}_1 = n_2 = 0$ in the last statement of Proposition 6.8. □

Example 6.10 *A code $C = (4, 8, 3)$, $|C| = 2048$ was constructed in [4] and a code $\widetilde{C} = (5, 8, 6)$, $|\widetilde{C}| = 45$ was constructed in [5]. Thus the corresponding $(c, c + \widetilde{c})$ code $\mathcal{C}$ is a $(4, 5, 8, 8, 6)$ code such that $|\mathcal{C}| = 2048 \times 45$.*

Linear q-ary codes that attain the Griesmer bound also prove useful for the construction of mixed codes. The Griesmer or Solomon-Stiffler bound was initially introduced and studied by Griesmer for $q = 2$ and Solomon and Stiffler for $q > 2$ [32]. Let $n_q(k, d)$ be the minimal value of n, for which a q-ary $[n, k, d]$ code exists, and $\lceil x \rceil$ denote the smallest integer greater than or equal to x. Then the bound is:

$$n_q(k, d) \geq g_q(k, d) := \sum_{i=0}^{k-1} \lceil \frac{d}{q^i} \rceil. \tag{6.12}$$

In [13], Section 5, general results have been obtained for $k = 3$:

Theorem 6.11 *If q is even, $(q \geq 4)$:*

$$n_q(3, d) = g_q(3, d) \quad \text{for} \quad q = 2^m \quad \text{and} \quad d \leq q + 2. \tag{6.13}$$

If q is an odd prime power:

$$n_q(3, d) = g_q(3, d) \quad \text{if} \quad d < q \quad \text{or} \quad d = q + 1. \tag{6.14}$$

Example 6.12 *Given $k = 3$, $d = 4$, $q = 4, 5, 7, \ldots$ it follows from Theorem 6.11 that $n_q = g_q = 6$. Using Corollary 6.9, $(q_1, q_2, 6, 6, 4)$ codes can be constructed from the $(q_1, 6, 2)$ codes $(q_1 < q_2)$ obtained by simple lexicographic search, and $(q_2, 6, 4)$ codes*

($|(q_2, 6, 4)| = q_2^3$) that attain the Griesmer bound. For the $(q_1, 6, 2)$ codes estimates: $|(3, 6, 2)| \geq 183$, $|(4, 6, 2)| \geq 1024$, $|(5, 6, 2)| \geq 2045$, $|(6, 6, 2)| \geq 5856$, $|(7, 6, 2)| \geq 14707$, $|(8, 6, 2)| \geq 32768$ can be obtained. In particular, a $(3, 4, 6, 6, 4)$ code $\mathcal{C}$ with $M = 183 \times 64 = 11712$ can be constructed in this way (which can be compared with 6540 codewords obtained for these parameters by direct lexicographic search).

If the Griesmer bound is attained for $k = 3$ and $d = n - 2$ then $M = q^3$. This is the best possible result for all (linear and non-linear) codes.

The matrix construction of the code V in Theorem 6.13 below (taken from [25] and [4]) can be used for the construction of non-linear codes. Let $A = [a_1\ a_2\ \ldots\ a_n]$ be an $r \times n$ matrix with column vectors a_i from $(GF(q))^r$. For the vectors x and y $(x, y \in (GF(q))^r)$ the *distance between x and y using A* is defined by the equality $d_A(x, y) = \min\{w(h) \,|\, hA^T = x - y,\ h \in (GF(q))^n\}$. If $hA^T = x - y$ has no solution, then $d_A(x, y) = r$. The distance $d_A(S)$, where $S \subseteq (GF(q))^r$, is now defined by the formula $d_A(S) = \min_{s,s' \in S,\ s \neq s'} d_A(s, s')$.

Theorem 6.13 *[25] Let A be a parity check matrix for a linear code with minimum distance d'. Then the code $V = \{v \in (GF(q))^n \,|\, vA^T \in S\}$ has minimum distance $\min\{d_A(S), d'\}$.*

This construction was used in [4] in combination with tabu search.

Proposition 6.1 from [6] modifies the last theorem for the case of mixed binary/ternary codes. It is immediate that this modification holds for any mixed $q_1 \mid q_2$ code:

Theorem 6.14 *Let A be a $r_1 \times n_1$ parity check matrix for a q_1-ary linear code with minimum distance d_1, let B be a $r_2 \times n_2$ parity check matrix for a q_2-ary linear code with minimum distance d_2, and let S be a subset of $(GF(q_1))^{r_1}(GF(q_2))^{r_2}$. Then the code*

$$V = \{(v_1, v_2) \in (GF(q_1))^{n_1}(GF(q_2))^{n_2} \,|\, (v_1A^T, v_2B^T) \in S\}$$

has minimum distance $\min\{d_{A,B}(S), d_1, d_2\}$, where

$$d_{A,B}(s, s') = \min\{w(h) \,|\, h = (h_1, h_2),\ (h_1A^T, h_2B^T) = s - s',$$

$$h_k \in (GF(q_k))^{n_k}\ (k = 1, 2)\}\ \textit{and}\ d_{A,B}(S) = \min_{s,s' \in S,\ s \neq s'} d_{A,B}(s, s').$$

A modification of Theorem 6.14 can prove useful for the construction of codes with larger distances between the codewords.

Theorem 6.15 *Let A be a $r_1 \times n_1$ parity check matrix for a q_1-ary linear code, let B be a $r_2 \times n_2$ parity check matrix for a q_2-ary linear code, and let Φ be a subset of the juxtaposition of the corresponding linear codes: $\Phi \subset \{\phi = (\phi_1, \phi_2) \,|\, \phi_1 A^T = 0, \quad \phi_2 B^T = 0\}$ such that $\min_{\phi,\phi' \in \Phi,\ \phi \neq \phi'} d(\phi, \phi') \geq d = d(\Phi)$. Let S be a subset of $(GF(q_1))^{r_1}(GF(q_2))^{r_2}$ and let $\Psi \subset (GF(q_1))^{n_1}(GF(q_2))^{n_2}$ be chosen so that the mapping $\psi = (\psi_1, \psi_2) \to (\psi_1 A^T, \psi_2 B^T)$ is a one-to-one mapping from Ψ onto S. Then the code V generated by the union of cosets: $V = \Phi + \Psi$ has minimum distance $d(V) \geq \min\{d(\Phi), d_{A,B}(S)\}$ and cardinality $|V| = |\Phi| \cdot |\Psi|$.*

The proofs of Theorem 6.14 and Theorem 6.15 are analogous to the proof of Theorem 6.13 (see [25] and [4]).

Example 6.16 *Theorem 6.15 gives, for example, $A_{2,3}(10, 4, 8) \geq 16$ with $|\Phi| = 2$, the number of cosets $|\Psi| = 8$, $A = [I_9 \ A_2]$, and $B = I_4$, where I_k denotes the identity matrix of order k, the first 7 entries of the column A_2 equal 1 and the last 2 equal zero. (Notice that $A_{2,3}(9, 4, 8) = 9$ and $A_{2,3}(10, 3, 8) = 6$ [6].) Analogously $A_4(7, 4) \geq 128$; the same estimate was obtained in [4] using quasicyclic codes. The $(7, 6, 4)$ code with $M = A_7(6, 4) = 343 = 7^3$, $|\Phi| = 7$, and $|\Psi| = 49$ can be constructed in this way also. (Another way to obtain $M = 343$ is to construct the $(7, 6, 4)$ code attaining Griesmer bound with $k = 3$, which is possible according to Theorem 6.11.) Finally, the lower bound in Example 6.3 can be improved to $M = 189$.*

6.3 Applications to the construction of constant weight codes

It was shown in [14] that q-ary codes with odd prime values of q and lengths $n = q - 1$ generate constant weight binary codes asymptotically meeting the Johnson upper bound. Another result of this type can be derived from Theorems 6.11 and 6.1.

Theorem 6.17 *Suppose that a sequence of prime powers q_n $(n \to \infty)$ is given, such that either the q_n's are powers of 2 and $n - 2 \leq q_n$, or the q_n's are odd and $n - 2 < q_n$. Then there exists a sequence of linear q_n-ary codes C_n of lengths n with $k = 3$ and $d = n - 2$ such that the corresponding binary constant weight codes $\widetilde{C}_n$ (obtained by the procedure of Theorem 6.1), asymptotically meet the Johnson upper bound.*

Proof. It follows from Theorem 6.11 that $n_{q_n}(3, n-2) = g_{q_n}(3, n-2)$ (the Griesmer bound is attained). From (6.12) and the restrictions on q_n it can be seen that $g_{q_n}(3, n-2) = n$ and so $n_{q_n}(3, n-2) = n$. In other words, there exist q_n-ary linear codes

$C_n = [n, 3, n-2]$. Applying Theorem 6.1 to construct the binary codes $\widetilde{C}_n$ with $|C_n|$ words of length $\widetilde{n} = q_n n$, minimum distance $\widetilde{d} = 2d$, and weight $w = n$. In its simpler form, the Johnson upper bound $J(\widetilde{C}_n)$ for $\widetilde{C}_n$ (see [20]) is equal to

$$J(\widetilde{C}_n) = \prod_{i=0}^{w-d} \frac{\widetilde{n} - i}{w - i} = q_n^3(1 + o(1)). \tag{6.15}$$

As $|C_n| = q_n^3$ the theorem is proved. □

It seems probable that sequences of constant weight binary codes which asymptotically meet the Johnson bound could be obtained (using Theorem 6.1) from mixed codes which are not pure q-ary codes. This is illustrated by the following example:

Example 6.18 *Simple lexicographic search gives codes with* $|(7, 8, 1, 7, 7)| = 56$ *and* $|(8, 8, 7)| = 64$*; the corresponding Johnson bounds being 63 and 72, respectively. In the code with* $(q, n, d) = (7, 6, 4)$ *and* $M = 343$ *discussed in Example 6.16, the Johnson bound (J.b.) equals* $\left[\frac{42}{6}\left[\frac{41}{5}\left[\frac{40}{4}\right]\right]\right] = 574$*. For the codes* $(7, 7, 5)$*,* $(8, 8, 6)$*, and* $(9, 9, 7)$ *we have J.b.* $= 7 \times 72$*, J.b.* $= 8 \times 91$*, and J.b.* $= 9 \times 111$*. The ratios J.b.*$/M$ *decrease from* 1.469 *to* 1.422 *and* 1.37*, respectively.*

Apart from being good asymptotically the construction of the binary code via the mixed one produces some good results even for smaller values of n.

Example 6.19 *a) It is easy to see that the condition in (6.13) is fulfilled for* $q = 8$*,* $k = 3$ *and* $d = 6$*. Thus by Theorem 6.11 we get* $n_q = g_q$*. According to (6.12) we get:* $g_8(3, 6) = 6 + 1 + 1 = 8$*, and so there exists a code with* $n = 8$*,* $q = 8$*,* $k = 3$ *and* $d = 6$*. Hence Theorem 6.1 yields an important estimate* $A(64, 12, 8) \geq 8^3$*. Analogously,* $A(72, 14, 9) \geq 8^3$ *and* $A(80, 16, 10) \geq 8^3$ *can be obtained.*

b) By (6.12) and (6.14) the equality $n_7(3, n-2) = g_7(3, n-2) = n$ *is valid for* $n \leq 8$*. Thus* $A_7(n, n-2) \geq 7^3 = 343$*. In view of Theorem 6.1 it follows that* $A(7n, 2(n-2), n) \geq 343$ *(*$n \leq 8$*). In particular,* $A(56, 12, 8) \geq 343$*,* $A(49, 10, 7) \geq 343$*,* $A(42, 8, 6) \geq 343$ *and* $A(35, 6, 5) \geq 343$*. The best known estimates from [33] give* $A(42, 8, 6) \geq 307$ *and* $A(35, 6, 5) \geq 367$*.*

Tables of the bounds of $A_3(n, d)$ are given in [6]. The codes for $q = 4$ and $q = 5$ are discussed (and tables for the lower bounds of $A_q(n, d)$ are given) in the recent papers [4], [5] and references therein. Here Theorem 6.1 can be applied to some of these estimates. Lower bounds for $A(qn, 2n-4, n)$ $(q = 4, 5)$ can be obtained from the

q	n=5	n=6	n=7	n=8	n=9	n=10	n=11
4	64	64	32	32	18	16	12
5	125	125	53	45	41	25	25

Table 2: Lower bounds for $A(qn, 2n-4, n)$ $(q = 4, 5)$.

lower bounds for $A_4(n, n-2)$ [4] and for $A_5(n, n-2)$ [5]. The results are displayed in Table 2.

These results can be compared with the best known estimates in Table 3 from [33] (where ? denotes a previously unknown value).

q	n=5	n=6	n=7	n=8	n=9	n=10	n=11
4	84	78	37	20	?	?	?
5	210	130	?	?	?	?	?

Table 3: Best known lower bounds for $A(qn, 2n-4, n)$ $(q = 4, 5)$.

The results of Example 6.12 together with the results from Table 2 and Proposition 6.8 yield $A(42, 8, 12) \geq 183 \times 64$, $A(54, 8, 12) \geq 1024 \times 125$ and $A(72, 8, 12) \geq 2045 \times 343$, in particular. Consider, finally, table 4.

$n_1 + n_2$	$n_1 = 0$	$n_1 = 1$	$n_1 = 2$	$n_1 = 3$
11	243	162	108	72
12	729	486	324	216

Table 4: Lower bounds for $A(2n_1 + 3n_2, 6, n_1 + n_2)$ by Theorem 6.1.

Its estimates are obtained from the tables of the binary/ternary codes from [6] and improve several existing lower bounds for $A(2n_1 + 3n_2, d, n_1 + n_2)$ (compare Table 4 with Table 5).

$n_1 + n_2$	$n_1 = 0$	$n_1 = 1$	$n_1 = 2$	$n_1 = 3$
11	106	87	72	58
12	465	375	300	239

Table 5: Best known lower bounds for $A(2n_1 + 3n_2, 6, n_1 + n_2)$ from [33].

Using [6], $A(33, 14, 11) \geq |(3, 11, 7)| \geq 36$, $A(32, 14, 11) \geq |(2, 3, 1, 10, 7)| \geq 24$ are obtained; the corresponding bounds in [33] being $A(33, 14, 11) \geq ?$, $A(32, 14, 11) \geq 20$.

6.4 Nonbinary and mixed lexicographic search

As was shown in section 4, lexicographic search is a general and fruitful method. Modifications that may be used to enlarge the number of codewords obtained or speed up the calculations are of particular interest. Lexicographic search can be conducted for q-ary and mixed codewords also. Such a search is much faster than the search among the binary codewords corresponding to the same problem. If one does not need too many codewords it is better to use lexicographic search for q-ary (or mixed) codewords. For example, it took 220 min. to search lexicographically for the binary words of length $n = 30$, distance $d = 10$ and weight $w = 9$ and only 3 seconds for the corresponding mixed code search. Moreover, it sometimes proves fruitful to use the q-ary and mixed codewords when looking for seeds for further binary lexicographic search. A good seed can intrinsically improve lexicographic search. For instance, one gets 16 4-ary codewords of length $n = 11$ and distance $d = 8$ by simple lexicographic search and 32 after the choice of 1 proper seed [4]. When looking for the seed the speed of the search is very important and chosing the best q-ary seed often improves somewhat the next binary search. Another example shows that though lexicographic q-ary search may help, other ways of constructing q-ary codes often bring much better results. In the previous subsection using the best 4-ary code we have improved the lower estimate of $A(32, 12, 8)$ from 20 to 32. Lexicographic binary search gives us here $M = 17$, lexicographic 4-ary search with the best seed 0, 0, 0, 2, 2, 3, 0, 1 gives $M = 20$, and lexicographic 4-ary search with this seed, completed afterwards by lexicographic binary search, gives $M = 24$. The last estimate is worse then 32 but better then the one from [33]. Finally we have tried this approach on a small section of the table of estimates with $d = 10$ [33]. The results from [33] state that:

$$A(29, 10, 7) \geq 36, \quad A(29, 10, 8) \geq ?, \quad A(29, 10, 9) \geq 96,$$

$$A(30, 10, 7) \geq 39, \quad A(30, 10, 8) \geq ?, \quad A(30, 10, 9) \geq 65.$$

Now the table should be

$$A(29, 10, 7) \geq 36, \quad A(29, 10, 8) \geq 75, \quad A(29, 10, 9) \geq 137,$$

$$A(30, 10, 7) \geq 40, \quad A(30, 10, 8) \geq 91, \quad A(30, 10, 9) \geq 168.$$

The estimates for $A(29, 10, 8)$ and $A(29, 10, 9)$ were obtained by lexicographic binary search. The same lexicographic binary search gave the estimate $A(30, 10, 9) \geq 167$ and $A(30, 10, 8) \geq 89$. Using the seed with "1" in the positions

$$4, 8, 10, 14, 17, 19, 22, 25, 29$$

the estimate $M = 168$ for $A(30, 10, 9)$ is obtained. Using the seed with "1" in the positions 1, 6, 11, 14, 20, 21, 27, 28 gives the estimate $M = 91$ for $A(30, 10, 8)$. (The

seeds were obtained from the best mixed code seed.) It was necessary to change by one one of the positions in the seeds obtained from the mixed code codewords to improve the estimates for $A(29, 10, 7)$ and $A(30, 10, 7)$. In the case $n = 29$ and $w = 7$ lexicographic search with a seed with "1" in the positions 2, 7, 11, 15, 20, 23, 26 gave $M = 35$, and in the case $n = 30$ and $w = 7$ lexicographic search with a seed with "1" in the positions 1, 6, 11, 16, 21, 26, 27 gave $M = 40$. In the same way the lower bound of $A(30, 10, 10)$ was improved from 287 to 291 (the seed here was 2, 5, 8, 11, 13, 17, 20, 23, 25, 28). Overall, all the estimates were improved except one.

6.5 Summary

Nonbinary and mixed and codes can prove useful for the generation of good constant weight codes, although their impact on the parameter sets relevant to GSM systems is fairly minor. However, when other radio systems are considered (for which the values of n and w are much larger and search methods will fail), the use of the methods described in this section becomes essential.

7 Lexicographic search using Steiner system seeds

In section 6.4 some modifications of lexicographic search have already been discussed. Here a group of modifications to binary lexicographic search related to the use of Steiner system seeds and other seeds will be considered.

In section 3 a correspondence was established between a Steiner system $S(w - \delta + 1, w, n)$ and an $A(n, 2\delta, w) \times n$ matrix $B = B(n, 2\delta, w)$. The rows of B are codewords of weight w of a maximal constant weight code with minimum distance 2δ. The next group of modifications make use of a theorem that follows immediately from the proof of Theorem 3.1.

Theorem 7.1 *Suppose for some $w \geq \delta$ and $p > 0$ there exists a Steiner system $S(w + p - \delta + 1, w + p, n + p)$ and $B(n + p, 2\delta, w + p)$ is the matrix whose rows are the codewords of the corresponding constant weight code. Then this B can be obtained from a certain $A(n, 2\delta, w) \times n$ matrix $B_n = B(n, 2\delta, w)$ by a sequence of transformations $B_r \rightarrow B_{r+1}$, each transformation consisting of adding 1 at the right*

end of each row of B_r and adding rows with zeros in the last position to the resulting matrix.

This theorem can form the basis of a heuristic algorithm. Choose a good code of length n, weight w and distance 2δ and form the matrix $B_n = B(n, 2\delta, w)$. Then construct the matrices B_{r+1} $(r \geq n)$ in the way prescribed by the theorem. The additional rows of the matrices B_{r+1} are found by lexicographic search.

As an example, use as a seed the codewords corresponding to the Steiner system $S(3, 4, 10)$ (constructed using the methods of section 5). In this case $A(10, 4, 4) = 30$, so there are 30 codewords forming the seed. Results for $A(11+c, 4, 5+c)$ $(0 \leq c \leq 9)$ are shown in the second column of table 6. At the first step the codewords correspond-

n	lexicographic search from Steiner system seed	lexicographic search with seed from previous search	simple lexicographic search
11	66	34	34
12	132	68	68
13	132	127	116
14	171	210	203
15	253	322	315
16	404	488	504
17	619	699	588
18	906	978	756
19	1261	1332	988
20	1728	1791	1336

Table 6: Results for $A(11 + c, 4, 5 + c)$ - lexicographic search with and without seed.

ing to the Steiner system $S(4, 5, 11)$ $(A(11, 4, 5) = 66)$ are obtained, and at the next step the codewords corresponding to the Steiner system $S(5, 6, 12)$ $(A(12, 4, 6) = 132)$. Simple lexicographic search gives much worse estimates (see column 4 of table 6). Notice that at the next four steps the gains are quickly lost.

More stable gains are obtained in this case when the results of the previous lexicographic search are used as the seed for the next step. Compare the estimates obtained for $A(11+c, 4, 5+c)$ $(0 \leq c \leq 9)$ by simple lexicographic search (column 4 of table 6) with lexicographic searches that use the seed from the previous search with "1" added on the right-hand end of each codeword (column 3 of table 6).

Lexicographic search already improves some lower bounds in the tables of the best constant weight codes in [33]. For example, results for $A(31 + c, 6, 3 + c)$ and $A(32 +$

$c, 6, 3 + c$) are given in tables 7 and 8. It can be seen that results in the second columns can improve the best known results in [33], for example $A(34, 6, 6) = 1710$ (quoted in [33] as by simple lexicographic search), $A(35, 6, 6) = 1945$ (quoted in [33] as by simple lexicographic search), $A(33, 6, 5) = 302$ and $A(34, 6, 5) = 334$ (given in [33] by sections of the previous two codes). The codes in the second column of the tables (obtained using the seed from the previous search) significantly improve these best known results. In order to obtain good results it is essential that both the length and the weight increase in successive searches. However, it should be noted that the best results obtained by simple lexicographic search (see, for example, column 4 in tables 7 and 8) also improve these results. Lexicographic search can be very variable over a number of runs (see section 4) and results for a single run do not necessarily give a fair comparison of two methods. Thus it is difficult to give a definite answer as to whether "improved" lexicographic methods necessarily improve on simple lexicographic search.

n	lexicographic search with seed from previous search	best known	simple lexicographic search
31	10	10	10
32	64	80	64
33	372	302	379
34	1725	1710	1740

Table 7: Results for $A(31 + c, 6, 3 + c)$.

n	lexicographic search with seed from previous search	best known	simple lexicographic search
32	10	10	10
33	68	82	68
34	407	334	413
35	1973	1945	1973

Table 8: Results for $A(32 + c, 6, 3 + c)$.

8 Cyclic constant weight codes

A cyclic shift of a codeword $\mathbf{c}$ in a constant weight code is a codeword $S_\tau(\mathbf{c})$ such that $S_\tau(\mathbf{c})(i) = \mathbf{c}((i + \tau) \bmod n)$. A constant weight code is *cyclic* if any cyclic shift

of a codeword is also a codeword. (Unlike cyclic codes, cyclic constant weight codes with more than one codeword cannot be linear. This is easily seen by noting that they do not contain a zero codeword.) Many of the best constant weight codes are cyclic. For example, some of the relevant Steiner systems are cyclic. It turns out that it is quite practical to generate cyclic constant weight codes. For most of the relevant parameter sets given in section 2, a cyclic constant weight code with the maximum number of codewords can be found. For others it may still be possible to find a code with a number of codewords that is close to maximum, or simply not guaranteed to be maximum. The basic procedure is given in [7]. First all cyclic sets (consisting of a codeword and all its distinct cyclic shifts) are determined. These are represented by the vertices of a weighted graph, with the vertex labelled by the number of codewords in the set. Two vertices are joined if the pair does not conflict with the minimum distance condition. A maximum clique algorithm is then used to find the maximum weighted clique in this graph which, from the cyclic sets represented by its vertices, gives the maximum cyclic constant weight code.

Clearly all of the distinct cyclic shifts of a vector of weight 0 can be determined (there is only one!) Given representatives of the cyclic sets for the set of vectors of weight i, these can be extended by a single 1 in all possible ways and the lexicographically maximal representatives for the cyclic sets of weight $i+1$ can be found. This process is continued until representatives for all the weight i cyclic sets with $i \leq w$ have been found.

Note that this process can be speeded if it is guaranteed that a weight $i+1$ cyclic set generated has not been generated previously. It will now be demonstrated how this can be done:

Algorithm 1 *Consider a complete set of binary vectors of length n and weight i, each of which is a lexicographically maximal representative of the set of all its cyclic shifts. Suppose also that these are arranged in decreasing lexicographic order, e.g. $n = 6$, $i = 2$:*
110000
101000
100100.

From each vector, new vectors of weight $i+1$ are generated by converting a single 0 to a 1 (starting at the left hand 0 and working to the right hand 0). Convert each vector generated to its lexicographically maximal form over all cyclic shifts of itself.

Record the vector if it is not previously in the list. *In the above example this gives:*
111000
110100
110010
101010
101001.

Theorem 8.1 *The vectors of weight $i+1$ are written down in decreasing lexicographical order.*

Proof. Suppose that a single 0 is converted to a 1 in a lexicographically maximal vector of weight i. If it is added before the last 1, it will already have been generated earlier, by a lexicographically previous vector of length i, and thus is not recorded. For example, if 101000 becomes 111000, it will have been generated previously from 110000.

Now suppose a single 0 is converted to a 1 in a vector of weight i after the last 1 in that vector. If the vector is already lexicographically maximal over all cyclic shifts of itself it appears in decreasing lexicographical order in the list of such generated vectors added to the list. In fact, it can be placed in decreasing lexicographical order in the list consisting of vectors of weight i, each followed by the new vectors of weight $i+1$ generated from it, followed by the next vector of weight i, etc..

Thus it is only necessary to consider the case where the vector obtained by converting a 1 to a 0 is not lexicographically maximal over all cycles of itself. Let $\mathbf{c}^{(i)}$ be the vector of weight i and $\mathbf{c}^{(i+1)}$ be the vector of weight $i+1$ obtained from it. Let the change take place in position $p+1$ of $\mathbf{c}^{(i)}$. Let $\mathbf{c}^{(i+1)}_{1\ldots p}$ be the vector consisting of the first p components of $\mathbf{c}^{(i+1)}$. Let $\mathbf{d}^{(i+1)}$ be the lexicographically maximal cyclic shift of $\mathbf{c}^{(i+1)}$, and let $\mathbf{d}^{(i+1)}_{1\ldots p}$ be the vector consisting of the first p components of $\mathbf{d}^{(i+1)}$.
(i) If $\mathbf{d}^{(i+1)}_{1\ldots p}$ is lexicographically greater than $\mathbf{c}^{(i+1)}_{1\ldots p}$ then $\mathbf{d}^{(i+1)}$ would already have been generated earlier, by a lexicographically previous vector of length i.
(ii) $\mathbf{d}^{(i+1)}_{1\ldots p}$ is lexicographically less than $\mathbf{c}^{(i+1)}_{1\ldots p}$ is impossible with our assumption that $\mathbf{c}^{(i+1)}$ is not lexicographically maximal and $\mathbf{d}^{(i+1)}$ is. Thus it only remains to consider the case: (iii) $\mathbf{d}^{(i+1)}_{1\ldots p} = \mathbf{c}^{(i+1)}_{1\ldots p}$ (and therefore both have weight i). As $\mathbf{c}^{(i+1)}$ is 1 in position $p+1$ our assumption implies that $\mathbf{d}^{(i+1)}$ is 1 in position $p+1$ and so the two vectors $\mathbf{c}^{(i+1)}$ and $\mathbf{d}^{(i+1)}$ are equal, contradicting our assumption. □

The proof of the above theorem shows that *it is possible to avoid searching the cyclic sets* to check whether the new representative has already been generated. Thus the

algorithm can be simplified:

Algorithm 2 *Consider a complete set of binary vectors of length n and weight i, each of which is a lexicographically maximal representative of the set of all its cyclic shifts. Suppose also that these are arranged in decreasing lexicographic order. From each vector new vectors of weight $i+1$ are generated by converting a single 0 to a 1* after the rightmost 1 in the vector, *(starting at the left hand 0 of the tail of* 0*'s and working to the right hand 0). For each vector generated determine whether it is lexicographically maximal over all cyclic shifts of itself. If it is, record the vector, otherwise discard it.*

For $t = w - d/2 + 1$, a matrix B with columns indexed by cyclic sets of weight w and with rows indexed by cyclic sets of weight t can be formed. B specifies how often a representative vector of weight t is covered by the vectors in a given cyclic set of weight w. Cyclic sets of weight w for which the corresponding row of B contains an entry greater than 1 can be discarded (as two elements of the cyclic constant weight code will have overlap greater than $w - d/2$).

The remaining cyclic sets of weight w are represented by the vertices of a weighted graph G, with the weights of the vertices giving the number of vectors in the cyclic set. Two vertices are adjacent if the cyclic sets do not cover the same t-set (so that the two cyclic sets, taken together, do not contradict the distance condition). A maximum weighted clique can then be found, for example using a simple modification of the algorithm of Carraghan and Pardalos [8].

In the results presented in section 10 the algorithm is run to completion when possible. In some cases an unweighted version of the algorithm has been used due to software limitations, with the weight summation carried out afterwards, and no guarantee of optimality. Sometimes the algorithm does not complete, but the largest (weighted) clique obtained by ordering the transmitters smallest degree last, or the reverse of this ordering, has been taken.

9 Steiner system constructions

In this section, constant weight codes which correspond to Steiner systems (see section 3) will be considered. Such codes are of particular interest when they have significantly more codewords than the best code that can be obtained by one of the basic methods of this report (cyclic, binary lexicographic, non-binary/mixed lexicographic

or random). As indicated previously, constructions specific to individual cases will not be studied if the value of A is only slightly increased. The case $A(50, 12, 8) = 350$ has already been mentioned, where at most 105 codewords are achievable by lexicographic search (or other basic method). It is necessary to consider whether this situation is common or rare.

It is also necessary to consider whether other codes obtained from the Steiner systems (for example by deleting a co-ordinate) are useful.

9.1 The cases $S(2, 3, v)$, $S(2, 4, v)$, $S(2, 5, v)$.

Steiner triple systems $S(2, 3, v)$ exist if and only if $v \equiv 1$ or 3 mod 6. Consider ordered pairs (i, k) where $i \in Z_{2t+1}$ and $k \in Z_3$. For $v = 6t + 3$ the triples are:

$$\{(i, 0), (i, 1), (i, 2)\} \quad \text{for all} \quad i \in Z_{2t+1}$$

$$\{(i, k), (j, k), ((t + 1)(i + j), k + 1)\} \quad \text{for all} \quad i, j \in Z_{2t+1}, i \neq j, k \in Z_3.$$

For $v = 6t+1$ it is necessary to start from a Latin square $L_1(x, y) = x+y$ over Z_{2t}. This is symmetric and its main diagonal entries are $2, 4, \ldots, 2t, 2, 4, \ldots, 2t$. Relabelling the entries, a symmetric Latin square $L(x, y)$ over Z_{2t} with main diagonal entries in the order $1, 2, \ldots, t, 1, 2, \ldots, t$ is obtained. Consider ordered pairs (i, k) where $i \in Z_{2t}$ and $k \in Z_3$, together with the symbol ∞. The triples are:

$$\{(i, 0), (i, 1), (i, 2)\} \quad \text{for all} \quad i, \quad 1 \leq i \leq t$$

$$\{(i, k), (i - t, k + 1), \infty\} \quad \text{for all} \quad i, \quad t + 1 \leq i \leq 2t, \ k \in Z_3$$

$$\{(i, k), (j, k), (L(i, j), k + 1)\} \quad \text{for all} \quad i, j \in Z_{2t}, i \neq j, k \in Z_3.$$

Note that more generally, the value of $A(n, 4, 3)$ is known for all n. References can be found in Theorem 4 of [7].

Steiner quadruple systems $S(2, 4, v)$ exist if and only if $v \geq 4$ and $v \equiv 1$ or 4 mod 12. References can be found in Theorem 12 of [7]. More generally, the value of $A(n, 6, 4)$ is known for all n. Details and references can be found in Theorem 6 of [7].

Steiner systems $S(2, 5, v)$ exist if and only if $v \geq 5$ and $v \equiv 1$ or 5 mod 20. References can be found in Theorem 12 of [7].

In will be seen in section 10 that the best result of one on the basic methods in this report is generally close to the upper bound. Thus a detailed study of these constructions may prove unnecessary for the application addressed in this report.

9.2 The cases $S(3, 4, v)$

Steiner quadruple systems $S(3, 4, v)$ exist if and only if $v \equiv 2$ or 4 mod 6. A reference can be found in [7]. More generally, the value of $A(n, 4, 4)$ is known unless $n \equiv 5(\text{mod}6)$ and $n > 11$. Details and references can be found in Theorem 5 of [7].

Again the best result of one on the basic methods in this report is generally close to the upper bound. Thus a detailed study of these constructions may prove unnecessary.

9.3 Codes for linear fractional transformations

As already noted in section 5, Steiner systems $S(3,\ q+1,\ q^r+1)$ exist for any prime power q. This gives the following cases for the relevant parameters:

- $q = 3$, $A(10, 4, 4) = 30$; (also achievable by lexicographic search or the basic cyclic construction).
- $q = 3$, $A(28, 4, 4) = 819$; ($A = 770$ achievable by the cyclic construction).
- $q = 4$, $A(17, 6, 5) = 68$; (also achievable by the cyclic construction).
- $q = 5$, $A(26, 8, 6) = 130$; (also achievable by the cyclic construction).
- $q = 7$, $A(50, 12, 8) = 350$; ($A = 105$ best achievable by lexicographic search).

It can be seen that $A(50, 12, 8) = 350$ is the unique case where a Steiner system constructed by the linear fractional transformation method gives a very significant increase in the number of codewords. Note that $S(2, q, q^r)$ can be obtained by choosing all blocks of $S(3,\ q+1,\ q^r+1)$ containing a specific point and deleting that point. This gives values for $A(9, 4, 3)$, $A(27, 4, 3)$,$A(16, 6, 4)$,$A(25, 8, 5)$, $A(49, 12, 7)$ of 12, 117, 20, 30, 56 respectively, (improving values of 9, 117, 16, 25, 49 obtained by the best of the basic methods).

9.4 Relevant Steiner systems

Other relevant Steiner systems are listed in [7]. The general cases are:

- $S(2, q+1, q^3+1)$ gives $A(28, 6, 4) = 63$ (improving the best result $A = 42$ achieved by lexicographic search). For the reader familiar with projective geometry, the Steiner system $S(2, q+1, q^3+1)$ is the (hermitian curve) unital in the desarguesian projective plane $PG(2, q^2)$ of order q^2. Without knowledge of projective geometry, the construction proceeds as follows. Let $F = GF(q^2)$ be a finite field (where q is a prime power). Let (a, b, c) be a triple with $a, b, c \in F$, where at least one of a, b c is not zero. Denote by $(a : b : c)$ the set of all non-zero multiples of (a, b, c). Then $(a : b : c) = \{(fa, fb, fc) | f \in F \setminus \{0\}\}$. Similarly, define $[x : y : z] = \{[fx, fy, fz] | f \in F \setminus \{0\}\}$ for $x, y, z \in F$, not all zero. Treating the symbols $(a : b : c)$ as points and the symbols $[x : y : z]$ as lines, a point $(a : b : c)$ lies on a line $[x : y : z]$ if and only if $ax + by + cz = 0$. This gives a projective plane $PG(2, q^2)$. The points of the unital are the $q^2 + 1$ points $(a : b : c)$ satisfying $a^{q+1} + b^{q+1} + c^{q+1} = 0$. It can be shown that the lines of $PG(2, q^2)$ contain either 1 or $q + 1$ points of the unital. The lines of the unital are the the lines of $PG(2, q^2)$ containing exactly $q + 1$ points of the unital. These lines are the blocks of the Steiner system $S(2, q+1, q^3+1)$.
- $S(2, 2^a, 2^{a+b} + 2^a - 2^b)$ for $a \leq b$ gives no new cases.

Other specific systems (excluding the linear fractional cases above, but including those of types $S(2, 3, v)$, $S(2, 4, v)$, $S(2, 5, v)$ and $S(3, 4, v)$) are:

- $S(2, 3, 7)$: This is cyclic and gives $A(7, 4, 3) = 7$. The code is obtainable by the basic cyclic construction or simply as a cycle of (1011000).
- $S(3, 4, 8)$: This gives $A(8, 4, 4) = 14$. The code is an extended cyclic code. Take all cycles of (1011000) followed by a 1, together with all cycles of (0100111) followed by a 0.
- $S(2, 3, 13)$: This gives $A(13, 4, 3) = 26$. The code is obtainable by the basic cyclic construction.
- $S(2, 4, 13)$: This gives $A(13, 6, 4) = 13$. The code is obtainable by the basic cyclic construction.
- $S(3, 4, 14)$: This gives $A(14, 4, 4) = 91$. See [7] for a construction. The basic cyclic method gives $A = 77$.
- $S(2, 3, 15)$: This gives $A(15, 4, 3) = 35$. The code is obtainable by the basic cyclic construction.

- $S(2,4,16)$: This gives $A(16,6,4) = 20$. A construction is given in [7]. The basic cyclic method (or the non-binary lexicographic method) give $A = 16$.

- $S(3,4,16)$ This gives $A(16,4,4) = 140$. The code can be obtained by simple lexicographic search

- $S(3,5,17)$: This gives $A(17,6,5) = 68$. The code is obtainable by the basic cyclic construction.

- $S(2,3,19)$ This gives $A(19,4,3) = 57$. The code is obtained from $A(20,4,4) = 285$ below by choosing all codewords with a 1 in the last position and deleting that position. An alternative construction is given in section 9.1 or the basic cyclic construction can be used.

- $S(3,4,20)$: This gives $A(20,4,4) = 285$. The code is obtainable by the basic cyclic construction.

- $S(2,3,21)$ This gives $A(21,4,3) = 70$. The code is obtained from $A(22,4,4) = 385$ below by choosing all codewords with a 1 in the last position and deleting that position. An alternative construction is given in section 9.1 or the basic cyclic construction can be used.

- $S(2,5,21)$ This gives $A(21,8,5) = 21$. The code is obtained from $A(22,8,6) = 77$ below by choosing all codewords with a 1 in the last position and deleting that position. Alternatively, the basic cyclic construction can be used

- $S(3,4,22)$: This gives $A(22,4,4) = 385$. The code is obtainable by the basic cyclic construction.

- $S(3,6,22)$: This gives $A(22,8,6) = 77$. The code can be obtained from the weight 8 words in the $[24,12,8]$ binary Golay code. A generator matrix for this code is given in table 9 [20]: Choose all codewords with a 1 in the last two co-ordinates and delete those co-ordinates. The best basic method is binary lexicographic search, giving $A = 44$.

- $S(4,7,23)$: This gives $A(23,8,7) = 253$. The code can be obtained from the weight 8 words in the $[24,12,8]$ binary Golay code with generator matrix given in table 9 [20]. Choose all codewords with a 1 in the last co-ordinate and delete that co-ordinate. Binary lexicographic search gives $A = 110$.

- $S(2,3,25)$ This gives $A(25,4,3) = 100$. The code is obtained from $A(26,4,4) = 650$ below by choosing all codewords with a 1 in the last position and deleting that position. An alternative construction is given in section 9.1 or the basic cyclic construction can be used.

110000000000011011100010
101000000000001101110001
100100000000010110111000
100010000000001011011100
100001000000000101101110
100000100000000010110111
100000010000010001011011
100000001000011000101101
100000000100011100010110
100000000010001110001011
100000000001010111000101
000000000000111111111111

Table 9: A generator matrix for the $[24, 12, 8]$ binary Golay code

- $S(2, 4, 25)$: This gives $A(25, 6, 4) = 50$. $A = 31$ is obtainable by Lexicographic search. $A = 50$ can be obtained from the code given below for $S(3, 5, 26)$. Choose all codewords with a 1 in the last co-ordinate and delete that co-ordinate.

- $S(2, 5, 25)$: This gives $A(25, 8, 5) = 30$. This is obtained using the construction described for $S(2, q, q^2)$ in section 9.3 . The basic cyclic method gives $A = 25$.

- $S(3, 4, 26)$: This gives $A(26, 4, 4) = 650$. $A = 637$ is obtainable by the basic cyclic construction. See [7] for a construction of the Steiner system.

- $S(3, 5, 26)$; This gives $A(26, 6, 5) = 260$. The code is obtainable by the basic cyclic construction.

- $S(3, 6, 26)$: This gives $A(26, 8, 6) = 130$. The code is obtainable by the basic cyclic construction.

- $S(2, 3, 27)$ This gives $A(27, 4, 3) = 117$. The code is obtained from $A(28, 4, 4) = 819$ below by choosing all codewords with a 1 in the last position and deleting that position. Alternative constructions are given in section 5, section 9.1, or the basic cyclic construction can be used.

- $S(2, 4, 28)$: This gives $A(28, 6, 4) = 63$. $A = 56$ is obtainable by the basic cyclic construction.

- $S(3, 4, 28)$: This gives $A(28, 4, 4) = 819$. $A = 770$ is obtainable by the basic cyclic construction.

- $S(2,3,31)$ This gives $A(31,4,3) = 155$. The code is obtained from $A(32,4,4) = 1240$ (see [7] Theorem 5) by choosing all codewords with a 1 in the last position and deleting that position. The code is obtainable by the basic cyclic construction. An alternative construction is given in section 9.1.

- $S(2,6,31)$: This gives $A(31,10,6) = 31$. The code is obtainable by the basic cyclic construction.

- $S(2,3,v)$ $v \in \{33,37,39,43,45,49,51,55,57,61,63\}$. These all exist and can be obtained by the construction of section 9.1. The corresponding codes give $A(n,4,3) = \frac{n(n-1)}{6}$ for $n \in \{33,37,39,43,45,49,51,55,57,61,63\}$. The codes can all be obtained by the basic cyclic construction.

- $S(2,4,v)$ $v \in \{37,40,49,52,61\}$. These all exist and references to constructions can be found in theorem 12 of [7]. The corresponding codes give $A(n,6,4) = \frac{n(n-1)}{12}$ for $n \in \{37,40,49,52,61\}$. The codes can all be obtained by the basic cyclic construction.

- $S(2,5,v)$ $v \in \{41,45,61\}$. These all exist and references to constructions can be found in theorem 12 of [7]. The corresponding codes give $A(n,8,5) = \frac{n(n-1)}{20}$ for $n \in \{41,45,61\}$. The code $A(41,8,5)$ can be obtained by the basic cyclic construction. For $n = 45$ and $n = 61$ the best result from a basic method was given by lexicographic search, giving $A = 73$ and $A = 131$ respectively.

- $S(3,4,v)$ $v \in \{32,34,38,40,44,46,50,52,56,58,62\}$. These all exist and references to constructions can be found in theorem 12 of [7]. The corresponding codes give $A(n,4,4) = \frac{n(n-1)(n-2)}{24}$ for $n \in \{32,34,38,40,44,46,50,52,56,58,62\}$. None of the codes can be obtained by our basic constructions. For $n \in \{44,38,40,44,46,50\}$ the basic cyclic construction gave A values of $1394, 1881, 2190, 3025, 3427, 4300$ respectively. For $n \in \{32,52,56,58,62\}$ the best A values of $1232, 4437, 6006$ and 6921 were obtained by lexicographic search. $S(3,4,46)$ can also be obtained fron $S(4,5,47)$ below by choosing all codewords with a 1 in the last position and deleting that position.

- $S(4,5,47)$. This can be obtained from $S(5,6,48)$ below by choosing all codewords with a 1 in the last position and deleting that position. It gives $A(47,4,5) = 35673$. The best basic method is lexicographic search, giving $A = 24488$.

- $S(5,6,48)$. This does not correspond to a code in the range of parameters considered here, but is given to construct $S(4,5,47)$. A construction can be found in [9].

- $S(2,8,57)$: This gives $A(57,14,8) = 57$. The Steiner system can be obtained from the lines in the projective geometry $PG(2,7)$. Let F be the finite field

$GF(7)$ and let (a, b, c) be a triple with $a, b, c \in F$, where at least one of a, b c is not zero. Denote by $(a : b : c)$ the set of all 57 non-zero multiples of (a, b, c). Then $(a : b : c) = \{(fa, fb, fc) | f \in F \setminus \{0\}\}\}$. Similarly, define $[x : y : z] = \{[fx, fy, fz] | f \in F \setminus \{0\}\}\}$ for $x, y, z \in F$, not all zero. Treating the symbols $(a : b : c)$ as points and the 57 symbols $[x : y : z]$ as lines, a point $(a : b : c)$ lies on a line $[x : y : z]$ if and only if $ax + by + cz = 0$. The 57 lines are the blocks of the Steiner system $S(2, 8, 57)$. The best basic method gives $A = 24$.

9.5 Summary of improvements by using Steiner systems

The number of cases where a significant improvement on the best basic method is obtained by a construction in this report is small. These are $A(22, 8, 6)$ and $A(23, 8, 7)$ (from the Golay code), $A(49, 12, 7)$ and $A(50, 12, 8)$ (from the linear fractional construction) and $(57, 14, 8)$ (from the projective geometry $PG(2, 7)$ described above).

Other significant improvements can be obtained for the cases $A(45, 8, 5)$ $A(47, 4, 5)$ and $A(61, 8, 5)$, Constructions, which in some cases may be more complex, are simply referenced in this report.

9.6 Other good codes derived from Steiner systems

Given a code derived from a Steiner system $S(t, w, n)$, all codewords containing a '1' in a particular position can be chosen and that position deleted. The codes obtained are those corresponding to the Steiner system $S(t - 1, w - 1, n - 1)$ and have already been listed under the corresponding Steiner system.

In a similar way, given a constant weight code with parameters n, d, w, all codewords containing a '0' in a particular position can be chosen and that position deleted. The codes obtained are constant weight codes with parameters $n - 1$, d, w. Starting from a code obtained from a Steiner system and applying this procedure a new code is obtained. If it is better than any of the codes obtained by a basic method, it has been inserted in the tables in section 10.

If a good code with parameters n, d, w exists, it can be regarded as a code with parameters $n + i$, d, w for $i = 1, \ldots$ by inserting i additional columns of '0's. If it is better than any of the codes obtained by a basic method, it has been inserted in

the tables in section 10. This result means that the number of codewords is non-decreasing with n. For practical purposes it may be desirable to "split" one or more columns to avoid unused frequencies. If $i \geq \frac{d}{2}$ an additional codeword (at least) is clearly possible, and this has also been inserted in the tables.

10 Results

This section collects together sets of results for the methods described earlier. The results presented will allow a comparison of the methods described in this report (for the relevant ranges of parameters).

The methods generate the following types of codes: lexicographic and randomly generated codes (section 4), codes derived from q-ary or mixed codes obtained by lexicographic search (section 6) and cyclic constant weight codes (section 8). Also presented are isolated codes obtained from Steiner systems and other codes derived from them (section 9).

Tables 10 to 24 present results for the methods according to the parameters defined in section 2, i.e. $n \leq 63$, $3 \leq w \leq 8$ and $d = 2w - 2$. Note that the result quoted is always the first result obtained using the method. For the search techniques better results can often be obtained by carrying out multiple runs. Some indications of the merit of multiple runs for binary lexicographic search were given in section 4.

In the tables the following abbreviations have been used:
CC: Cyclic construction;
NB-Mix: Code derived from non-binary or mixed code obtained by lexicographic search;
B-Lex: Binary lexicographic;
Rand: Random construction;
SS: Steiner system codes;
SS-D: Derivations from Steiner system codes;
UB: Upper bound given by the Johnson bound (section 3);
Previous Best: Best known results prior to the results presented here, as given in [33].

The results presented for binary lexicographic search use the method denoted by $M(1, 1)$ in section 4.

The results for the non-binary/mixed search are based on the use of theorem 6.1

in section 6. Parameters for a suitable non-binary or mixed code are determined and the code is found by lexicographic search

The results for the cyclic code construction (described in section 8) have not been calculated when the number of vertices input into the maximum clique finding software exceeds 10,000 or the number of cliques required to compare with the upper bound exceeds 100. In such cases the memory requirements for the current maximum clique finding software become excessive.

The random method (described in section 4.5) finds application when the lexicographic method proves too time consuming. As such, a complete list of random results is not presented, the random results are provided in cases where the run time of the binary lexocographic method is in excess of approximately two hours.

n	CC	$NB-Mix$	$B-Lex$	$Rand$	SS	$SS-D$	UB	$PreviousBest$
9	9		9		12		12	12
10	10	9	12				13	13
11	11	12	15				18	17
12	16	16	19			20	20	20
13	26	16	23		26		26	26
14	14	16	27			28	28	28
15	35	17	32		35		35	35
16	32	21	35				37	37
17	34	24	40				45	44
18	42	28	43			48	48	48
19	57	32	47		57		57	57
20	40	37	53			60	60	60
21	70	43	58		70		70	70
22	66	49	64			70	73	73
23	69	56	71			77	84	83
24	80	64	78			88	88	88
25	100	64	86		100		100	100
26	104	65	94				104	104
27	117	67	103		117		117	117
28	112	69	112			117	121	121
29	116	72	121			126	135	134
30	130	76	130			140	140	140
31	155	80	140		155		155	155
32	160	85	149				160	160
33	176	91	155		176		176	176
34	170	97	164			176	181	181
35	175	104	172			187	198	197
36	192	112	181			204	204	204
37	222	120	190		222		222	222
38	190	129	199			228	228	228
39	247	139	209		247		247	247
40	240	149	224			247	253	253
41	246	160	236			260	273	272
42	266	172	248			280	280	280
43	301	184	260		301		301	301
44	264	197	273			308	308	308
45	330	211	286		330		330	330
46	322	225	299			330	337	337
47	329	240	313			345	360	359
48	352	256	327			368	368	368
49	392	256	342		392		392	392
50	400	257	354				400	400
51	425	259	371		425		425	425
52	416	261	389			425	433	433
53	424	264	407			442	459	458
54	450	268	425			468	468	468
55	495	272	443		495		495	495
56	504	277	459				504	504
57	532	283	478		532		532	532
58	522	289	497			532	541	541
59	531	296	515			551	570	569
60	560	304	527			580	580	580
61	610	312	547		610		610	610
62	558	321	567			620	620	620
63	651	331	586		651		651	651

Table 10: Comparison of Results $d = 4, w = 3$.

n	CC	$NB-Mix$	$B-Lex$	$Rand$	SS	$SS-D$	UB	$PreviousBest$
9	9		14				18	18
10	30		18		30		30	30
11	33		26				35	35
12	39	21	37			45	54	51
13	52	27	54			65	65	65
14	77	36	75		91		91	91
15	105	48	105				105	105
16	124	64	140		140		140	140
17	153	64	140				157	156
18	189	68	149				202	198
19	228	76	164				228	228
20	285	89	189		285		285	285
21	315	101	217				315	315
22	385	118	259		385		385	385
23	414	140	305				419	416
24	486	168	378				504	498
25	525	192	442				550	550
26	637	222	518		650		650	650
27	675	258	593			702	702	702
28	770	301	702		819		819	819
29	841	343	819				877	875
30	975	392	940				1012	1005
31	1023	448	1078				1085	1085
32	1176	512	1232		1240		1240	1240
33	1254	512	1240			1320	1320	1320
34	1394	520	1259		1496		1496	1496
35	1540	536	1293				1583	1576
36	1692	561	1342				1782	1773
37	1813	585	1403				1887	1887
38	1881	618	1484		2109		2109	2109
39	2145	660	1577			2223	2223	2223
40	2190	712	1687		2470		2470	2470
41	2378	760	1812			2470	2593	2584
42	2499	818	1978			2730	2866	2856
43	2709	886	2122			3010	3010	3010
44	3025	965	2291		3311		3311	3311
45	3195	1043	2483			3465	3465	3465
46	3427	1132	2744		3795		3795	3795
47	3525	1232	2965			3795	3959	3949
48	3828	1344	3209			4140	4320	4308
49	4116	1440	3531			4508	4508	4508
50	4300	1548	3803		4900		4900	4900
51		1668	4140			5100	5100	5100
52		1801	4437		5525		5525	5525
53		1933	4775			5525	5737	5725
54		2078	5199			5967	6196	6183
55		2236	5558			6435	6435	6435
56		2408	6006		6930		6930	6930
57		2576	6463			7182	7182	7182
58		2758	6921		7714		7714	7714
59		2954	7416			7714	7979	7966
60		3165	7957			8265	8550	8535
61		3375	8555			8845	8845	8845
62		3600	9087		9455		9455	9455
63		3840	9765				9765	9765

Table 11: Comparison of Results $d = 4, w = 4$.

n	CC	$NB-Mix$	$B-Lex$	$Rand$	SS	$SS-D$	UB	$PreviousBest$
9	9		14				25	18
10	30		22				36	36
11	66		36				66	66
12	72		56				84	80
13	117		87				140	123
14	154		145				182	169
15	228	61	189				273	237
16	304	81	269				336	312
17	391	108	318				476	424
18	486	144	365				565	518
19	608	192	448				767	684
20	744	256	562				912	874
21	945	256	723				1197	1071
22	1122	272	911				1386	1386
23	1380	304	1140				1771	1771
24		353	1379				2011	1895
25		421	1736				2520	2334
26		489	2130				2860	2670
27		576	2555				3510	3276
28		684	3114				3931	3718
29		816	3687				4750	4095
30		976	4407				5262	4751
31		1136	5141				6274	5481
32		1324	6285				6944	6293
33		1544	6503				8184	
34		1801	7051				8976	
35		2101	7159			7552	10472	
36		2401	7881			8769	11397	
37		2744	8349			10139	13186	
38		3136	9259			11675	14341	
39		3584	10168			13391	16450	
40		4096	11334			15303	17784	
41		4096	12598			17428	20254	
42		4160	14156			19783	21781	
43		4288	15831			22386	24647	
44		4481	17635			25256	26488	
45		4741	19657			28413	29799	
46		5001	21940			31878	31878	
47		5328	24488		35673		35673	35673
48		5724	27231			35673	38006	
49		6192	30324			35674	42336	
50		6736	33667			35674	45080	
51		7280	37018				49980	
52		7900	40882				53040	
53		8600	44928				58565	
54		9385	49597				61959	
55		10000		29452			68156	
56		10000		33006			72072	
57		10000		36461			79002	
58		10000		36703			83311	
59		10000		39689			91025	
60		10000	83280	40586			95748	
61		10000		42482			104310	
62		10000		43778			109678	
63		10000		44836			119133	109368

Table 12: Comparison of Results $d = 4, w = 5$.

n	CC	$NB-Mix$	$B-Lex$	$Rand$	SS	$SS-D$	UB	$PreviousBest$
12	3	9	7				9	9
13	13	7	10		13		13	13
14	14	9	13				14	14
15	15	12	13				15	15
16	16	16	14		20		20	20
17	17	16	15			20	21	20
18	18	16	17			20	22	22
19	19	17	18			21	28	25
20	25	17	20				30	30
21	21	18	23			24	31	31
22	22	20	25			29	38	37
23	23	20	27			35	40	40
24	30	22	30			42	42	42
25	25	24	31		50		50	50
26	52	25	34				52	52
27	54	25	36				54	54
28	56	25	42		63		63	63
29	58	30	44			63	65	65
30	60	32	43			63	67	67
31	62	32	47			64	77	76
32	72	32	52				80	80
33	66	36	60			69	82	82
34	68	40	70			78	93	92
35	70	45	76			88	96	96
36	81	48	85			99	99	99
37	111	53	92		111		111	111
38	114	59	97				114	114
39	117	64	106				117	117
40	130	70	112		130		130	130
41	123	78	116			130	133	133
42	126	86	122			130	136	136
43	129	95	125			131	150	149
44	143	101	131				154	154
45	135	110	136				157	157
46	138	121	145				172	171
47	141	132	142			160	176	176
48	156	144	149			177	180	180
49	196	134	146		196		196	196
50	200	135	159				200	200
51	204	140	164				204	204
52	221	144	172		221		221	221
53	212	143	179			221	225	225
54	216	149	187			221	229	229
55	220	156	198			222	247	246
56	238	164	201				252	252
57	228	171	211			231	256	256
58	232	175	219			248	275	274
59	236	181	225			266	280	280
60	255	187	223			285	285	285
61	305	199	229		305		305	305
62	310	215	238				310	310
63	315	240	247				315	315

Table 13: Comparison of Results $d = 6, w = 4$.

n	CC	$NB-Mix$	$B-Lex$	$Rand$	SS	$SS-D$	UB	$PreviousBest$
12	12		12				19	12
13	13		13				23	18
14	14		16				36	28
15	33	9	34				42	42
16	32	16	33			48	48	48
17	68	20	42		68		68	68
18	54	36	51			68	75	68
19	76	48	59				83	76
20	84	64	72				112	84
21	105	64	83				126	105
22	132	64	98				136	132
23	138	65	112				174	147
24	144	68	132			168	192	168
25	180	74	149			210	210	210
26	260	77	170		260		260	260
27	216	81	192			260	280	260
28	252	90	218			260	302	280
29	290	100	244				365	234
30	306	112	268				390	272
31	341	127	304				415	311
32	384	137	340				492	337
33	429	150	375				528	302
34	476	162	413				557	334
35		182	450				651	367
36		202	496				691	402
37		219	542				732	441
38		200	589				843	480
39		224	638				889	523
40		256	691				936	569
41		288	749				1066	616
42		288	817				1117	666
43		321	874				1169	720
44		346	937				1320	777
45		372	1009				1386	836
46		405	1086				1444	898
47		430	1172				1616	962
48		464	1246				1689	1030
49		504	1330				1764	1100
50		532	1429				1960	1173
51		619	1512				2040	1250
52		668	1617				2121	1332
53		731	1719				2342	1414
54		776	1822				2430	1501
55		824	1918				2519	1591
56		736	2036				2766	1686
57		674	2162				2872	1782
58		576	2276				2969	1884
59		576	2397				3245	1992
60		576	2531				3360	2100
61		898	2665				3477	2215
62		1017	2799				3782	2333
63		1122	2937				3906	3906

Table 14: Comparison of Results $d = 6, w = 5$.

n	CC	$NB-Mix$	$B-Lex$	$Rand$	SS	$SS-D$	UB	$PreviousBest$
12	14		17				30	22
13	26		19				41	26
14	42		26				53	42
15	60		34				90	70
16	64		84				112	112
17	85		73				136	112
18	117	24	98				204	132
19	171	37	125				237	172
20	190	50	160				276	232
21	224	56	200				392	269
22	275	64	247				462	319
23		64	294				521	399
24		64	366				696	532
25		94	440				800	700
26		127	521				910	910
27		159	622				1170	1170
28		189	734				1306	1170
29		226	847				1459	1170
30		265	992				1825	1170
31		303	1156				2015	1189
32		353	1320				2213	1353
33		412	1515				2706	1525
34		468	1740				2992	1710
35		538	1958				3249	1945
36		618	2240				3906	2200
37		676	2526				4261	2478
38		762		2190			4636	2788
39		842		2433			5479	3118
40		944	3545	2696			5926	3483
41		1049		2982			6396	3882
42		1175		3268			7462	4311
43		1284		3552			8005	4773
44		1402		3876			8572	5277
45		1368		4233			9900	5823
46		1568		4543			10626	6409
47		1792		4906			11311	7027
48		2048		5327			12928	7689
49		2190		5704			13793	8406
50		2366	9342	6081			14700	9162
51		2577		6522			16660	9968
52		2807		6796			17680	10828
53		3055		7459			18735	11761
54		3346		7848			21078	12726
55		3605		8379			22275	13752
56		3881		8935			23510	14843
57		4210		9394			26277	16009
58		4532		9572			27762	17220
59		4854		10532			29195	18522
60		5258	20431	11014			32450	19915
61		5638		11630			34160	21349
62		6026		12027			35929	22883
63		6473		12839			39711	37758

Table 15: Comparison of Results $d = 6, w = 6$.

n	CC	$NB-Mix$	$B-Lex$	$Rand$	SS	$SS-D$	UB	$PreviousBest$
15	3	3	4				9	6
16	0	5	6				9	6
17	0	7	7				13	7
18	0	9	8				14	9
19	0	12	12				15	12
20	4	16	12				16	16
21	21	16	14		21		21	21
22	0	16	15			21	22	21
23	23	16	20				23	23
24	24	17	20				24	24
25	25	17	21		30		30	30
26	26	17	23			30	31	30
27	27	18	26			30	32	30
28	28	18	26			30	33	33
29	29	18	27			31	40	34
30	36	18	29				42	36
31	31	20	32				43	43
32	32	23	34				44	43
33	33	24	36				52	44
34	34	25	38				54	47
35	42	27	41				56	56
36	36	31	44				57	57
37	37	31	47			48	66	65
38	38	32	50			55	68	65
39	39	32	52			63	70	65
40	48	32	57			72	72	72
41	82	32	60		82		82	82
42	42	33	62			82	84	84
43	86	35	67				86	86
44	88	33	68			88	88	88
45	54	33	73		99		99	99
46	92	36	76			99	101	
47	94	40	80			99	103	
48	96	40	81			99	105	
49	98	46	85			100	117	
50	110	46	89				120	120
51		52	91				122	
52		54	96				124	
53		57	100				137	133
54		65	105				140	
55		68	107			108	143	
56		65	112			118	145	
57		60	117			129	159	
58		56	119			141	162	
59		48	123			154	165	
60		48	122			168	168	168
61		56	131		183		183	183
62		63	136			183	186	186
63		71	137			183	189	189

Table 16: Comparison of Results $d = 8, w = 5$.

n	CC	$NB-Mix$	$B-Lex$	$Rand$	SS	$SS-D$	UB	$PreviousBest$
15	0		10				20	10
16	8		9			10	24	16
17	0		12			14	25	17
18	18	10	16			20	39	21
19	19	12	22			28	44	28
20	20	21	21			40	50	40
21	21	27	36			56	56	56
22	33	36	44		77		77	77
23	23	48	77			77	84	77
24	76	64	75			77	92	78
25	50	64	66			100	100	100
26	130	64	75		130		130	130
27	81	64	85			130	139	130
28	98	65	88			130	149	130
29		65	99			130	159	130
30		67	107			131	200	130
31		69	119			131	217	130
32		73	127			131	229	137
33		76	138				242	150
34		76	149				294	163
35		80	168				315	178
36		88	183				336	196
37		95	199				351	213
38		109	222				418	231
39		118	240				442	252
40		128	266				466	275
41		137	282				492	285
42		147	307				574	307
43		160	332				602	330
44		164	354				630	355
45		178	381				660	380
46		200	406				759	411
47		224	439				791	
48		256	471				824	
49		256	501				857	
50		264	542				975	
51		253	576				1020	
52		271	605				1057	
53		289	650				1095	
54		314	682				1233	
55		334	729				1283	
56		347	766				1334	
57		376	830				1377	
58		402	872				1537	
59		420	935				1593	
60		446	982				1650	
61		471	1028				1708	
62		505	1079				1891	
63		531	1143				1953	

Table 17: Comparison of Results $d = 8, w = 6$.

n	CC	$NB-Mix$	$B-Lex$	$Rand$	SS	$SS-D$	UB	$PreviousBest$
15	15		15				34	15
16	16		13				45	16
17	17		18			21	58	24
18	18		24			33	64	33
19	19		32			52	105	52
20	40		41			80	125	80
21	45	24	50			120	150	120
22		29	63			176	176	176
23		39	110		253		253	253
24		42	97			253	288	253
25		56	253				328	254
26		64	255				371	257
27		64	262				501	278
28		64	277				556	296
29		64	300				617	300
30		76	325				681	327
31		86	363				885	362
32		104	400				992	403
33		122	444				1079	442
34		150	498				1175	494
35		165	554				1470	555
36		187	620				1620	622
37		214		516			1776	696
38		241		569			1905	785
39		278		647			2328	869
40		310	977	710			2525	965
41		350		793			2729	1095
42		390		867			2952	1206
43		425		944			3526	1344
44		474		1051			3784	1471
45		522		1129			4050	1632
46		578		1223			4337	1795
47		631		1317			5096	1976
48		699		1431			5424	
49		766		1549			5768	
50		835	2600	1668			6121	
51		896		1807			7103	
52		970		1939			7577	
53		1072		2040			8003	
54		1376		2195			8447	
55		1792		2344			9687	
56		2048	4270	2522			10264	
57		2048		2678			10862	
58		1557		2838			11409	
59		1675		3014			12954	
60		1780		3158			13654	
61		1910		3357			14378	
62		2078	6693	3605			15128	
63		2227		3797			17019	7182

Table 18: Comparison of Results $d = 8, w = 7$.

n	CC	$NB-Mix$	$B-Lex$	$Rand$	SS	$SS-D$	UB	$PreviousBest$
18	3	3	4				9	4
19	0	3	4				9	4
20	0	4	4				10	5
21	0	5	5				14	7
22	0	6	5				14	7
23	0	4	5				15	8
24	4	4	7				16	9
25	0	7	9				16	10
26	0	8	10			11	21	13
27	0	10	10			13	22	14
28	0	11	15			16	23	16
29	0	12	15			20	24	20
30	5	13	16			25	25	25
31	31	14	19		31		31	31
32	0	15	23			31	32	31
33	0	17	23			31	33	
34	0	19	26			31	34	
35	35	22	26				35	35
36	36	22	27				42	
37	37	22	30				43	
38	38	25	31				44	
39	39	26	32				45	
40	40	28	35				46	
41	41	31	34				54	
42	49	32	38				56	
43	43	34	39				57	
44	44	29	43				58	
45	45	31	44				60	
46	46	33	46				69	
47	47	32	48				70	
48	56	32	47				72	
49	49	36	50				73	
50	50	32	50				75	
51	51	36	55				85	
52		40	59				86	
53		45	63				88	
54		48	65				90	
55		49	68				91	
56		48	70				102	
57		52	69				104	
58		58	72				106	
59		60	77				108	
60		62	79				110	
61		65	83				122	
62		67	84				124	
63		71	85				126	

Table 19: Comparison of Results $d = 10, w = 6$.

n	CC	$NB-Mix$	$B-Lex$	$Rand$	SS	$SS-D$	UB	$PreviousBest$
18	0		6				20	6
19	0		8				24	8
20	0		9				25	10
21	3	8	11				30	13
22	0	9	14				44	16
23	0	11	15				46	20
24	24	13	18				51	24
25	25	14	22				57	28
26	26	12	25				59	28
27	27	12	27				81	36
28	32	16	30				88	37
29	29	22	34				95	36
30	30	25	37				102	39
31	62	28	43				110	42
32		30	47				141	
33		34	54				150	
34		36	59				160	
35		39	66				170	
36		44	75				180	
37		46	83				222	
38		53	90				233	
39		57	101				245	
40		64	112				257	
41		68	126				269	
42		72	129				324	
43		79	142				344	
44		83	152				358	
45		90	166				372	
46		94		128			394	
47		105		148			463	
48		112		161			480	
49		118	224	171			504	
50		126	234	173			521	
51		137		184			546	
52		146		201			631	
53		154		212			651	
54		192		224			678	
55		224		237			707	
56		256	346	249			728	
57		196		266			830	
58		210		280			861	
59		220		290			893	
60		237		308			925	
61		251		324			958	
62		266	486	336			1080	
63		277		341			1116	

Table 20: Comparison of Results $d = 10, w = 7$.

n	CC	$NB-Mix$	$B-Lex$	$Rand$	SS	$SS-D$	UB	$PreviousBest$
18	0		6				42	9
19	0		8				47	12
20	0		13				60	17
21	21		17				65	21
22	22		20				82	24
23	23		25				126	33
24	27	18	31				138	38
25		20	37				159	48
26		23	45				185	54
27		28	53				199	66
28		31	65				283	78
29		28	74				319	
30		36	88				356	
31		48	104				395	
32		64	119				440	
33		64	134				581	
34		69	156				637	
35		78	174				700	
36		86	197				765	
37		98	223				832	
38		109	353				1054	
39		123		210			1135	
40		136	314	227			1225	
41		150		262			1317	
42		168		287			1412	
43		179		301			1741	
44		198		339			1892	
45		218		371			2013	
46		242		411			2139	
47		268		436			2314	
48		294		471			2778	
49		312		503			2940	
50		338	852	552			3150	
51		370		588			3321	
52		391		644			3549	
53		433		674			4180	
54		469		737			4394	
55		508		785			4661	
56		544	1405	824			4949	
57		579		871			5187	
58		631		943			6017	
59		672		994			6349	
60		724	1908	1057			6697	
61		792		1138			7053	
62		992		1194			7424	
63		1024		1282			8505	

Table 21: Comparison of Results $d = 10, w = 8$.

n	CC	$NB-Mix$	$B-Lex$	$Rand$	SS	$SS-D$	UB	$PreviousBest$
21	3	3	3				9	3
22	0	3	4				9	4
23	0	3	4				9	4
24	0	4	4				10	4
25	0	4	4				14	5
26	0	5	5				14	5
27	0	4	5				15	6
28	4	4	5				16	8
29	0	4	5				16	5
30	0	5	6				17	
31	0	7	6				22	9
32	0	8	9				22	
33	0	10	6				23	
34	0	5	8				24	12
35	5	5	11				25	15
36	0	9	12				25	
37	0	11	15				31	
38	0	13	12				32	
39	0	17	18				33	
40	0	14	18				34	
41	0	12	20				35	
42	6	12	22				36	
43	0	18	19			22	43	
44	0	19	27				44	38
45	0	21	28			30	45	45
46	0	19	30			35	46	
47	0	23	31			41	47	
48	48	25	32			48	48	48
49	49	25	35		56		56	56
50	50	27	35			56	57	
51	51	28	37			56	58	
52	52	31	39			56	59	
53	53	33	39			56	60	
54	54	34	43			57	61	
55	55	32	44			57	70	
56	56	32	45			57	72	
57	57	32	46				73	
58	58	34	47				74	
59	59	32	51	42			75	
60	60	36	53	43			77	
61		40	53	44			87	
62		33	55	46			88	
63		33	57	47			90	

Table 22: Comparison of Results $d = 12, w = 7$.

n	CC	$NB-Mix$	$B-Lex$	$Rand$	SS	$SS-D$	UB	$PreviousBest$
21	0		5				21	5
22	0		6				24	6
23	0		6				25	6
24	3	9	7				27	9
25	0	8	9				31	10
26	0	9	11				45	13
27	0	9	13				47	15
28	0	11	14				52	19
29	0	12	16				58	
30	15	12	17				60	15
31	31	12	20				65	16
32	32	16	23				88	20
33	33	16	23				90	
34	34	18	27				97	
35	35	22	31				105	
36	36	24	36				112	
37	37	27	37				115	
38		26	39				147	
39		29	43				156	
40		33	48			52	165	
41		35	53			64	174	
42		37	58			79	183	
43		40	63			96	193	
44		44	66			117	236	
45		47		58		142	247	
46		52	78	61		171	258	
47		56		67		205	270	
48		58		71		246	282	
49		61		76		294	294	
50		65	105	78	350		350	
51		71		85		350	363	
52		74		91		350	377	
53		81		96		350	390	
54		85		102		350	405	
55		91		106		350	419	
56		95		113		351	490	
57		99		120		351	513	
58		106		128		351	529	
59		110		132		351	545	
60		118	203	140		352	562	
61		123		142		352	587	
62		200	226	152		352	674	
63		224		160		352	693	

Table 23: Comparison of Results $d = 12, w = 8$.

n	CC	$NB-Mix$	$B-Lex$	$Rand$	SS	$SS-D$	UB	$PreviousBest$
24	3	3	3				9	3
25	0	3	3				9	3
26	0	3	4				9	4
27	0	3	4				10	4
28	0	4	4				10	4
29	0	4	4				14	4
30	0	4	4				15	5
31	0	4	5				15	5
32	4	4	5				16	5
33	0	4	5				16	6
34	0	4	5				17	
35	0	5	5				17	7
36	0	5	6				22	9
37	0	6	6				23	
38	0	6	6				23	
39	0	5	6				24	
40	0	5	6				25	
41	0	5	7				25	
42	0	6	7				26	
43	0	8	10				32	
44	0	9	12				33	
45	0	12		11			33	
46	0	13		11			34	
47	0	6		11		13	35	
48	6	6		11		15	36	
49	0	11		13		17	36	21
50	0	14	19	13		20	43	25
51	0	15		13		23	44	
52	0	18		16		27	45	
53	0	19		18		31	46	
54		18		20		36	47	
55		15		18		42	48	
56		15		21		49	49	49
57		24		21	57		57	57
58		23		21		57	58	
59		25		23		57	59	
60		25	34	25		57	60	
61		27		26		57	61	
62		28		26		58	62	
63		56		26			63	63

Table 24: Comparison of Results $d = 14, w = 8$.

The run time for binary lexicographic search has already been discussed in section 4.1. Figure 1 illustrates, for a typical example, the number of codewords produced against run time for the binary lexicographic and the random construction. From the graph it is clear that the random construction finds the majority of codewords relatively quickly, while over time the binary lexicographic method increases the number of codewords in a more uniform manner. It should be noted that the binary lexicographic method will generally result in more codewords than the random construction. However, when run time is a priority the random construction is likely to produce more codewords than the binary lexicographic method over a short period of time. A formula for predicting the run time of binary lexicographic search was given in section 4.6. This can be used for deciding which technique to adopt.

The run times for the non-binary/mixed lexicographic search proves to be the most satisfactory of all the basic methods. In fact, for this method the complete set of results presented here was obtained with a single run of a program. This took approximately one day to complete.

The cyclic code construction is by far the most time consuming method. The method is composed of two parts, i.e. finding cyclic sets of codewords, followed by a maximum clique finding algorithm (section 8). In some of the smaller cases a weighted maximum clique finding algorithm is used, however the current software for this is unable to compute results for cases with large inputs. Simply finding the largest clique with no consideration of the size of the cyclic sets may sometimes lead to slightly smaller codes being found. The maximum clique finding algorithm finds a large clique relatively quickly and only small improvements are made if a longer run is pursued. As such, in some cases the algorithm is terminated early. Table 25 illustrates, in each case, a value of n where the preceeding cases have all been run in under two hours. From the table it can be seen that as w grows in relation to d the corresponding values of n become smaller, indicating that fewer cases are completed in under two hours. In some cases where the specified value of n is exceeded the run time is greater then 24 hours.

d	w	n
4	3	63
4	4	24
4	5	
6	4	55
6	5	25
6	6	18
8	5	43
8	6	23
8	7	
10	6	47
10	7	29
10	8	
12	7	47
12	8	32
14	8	41

Table 25: Largest parameters for which the cyclic construction has been run to completion in under two hours

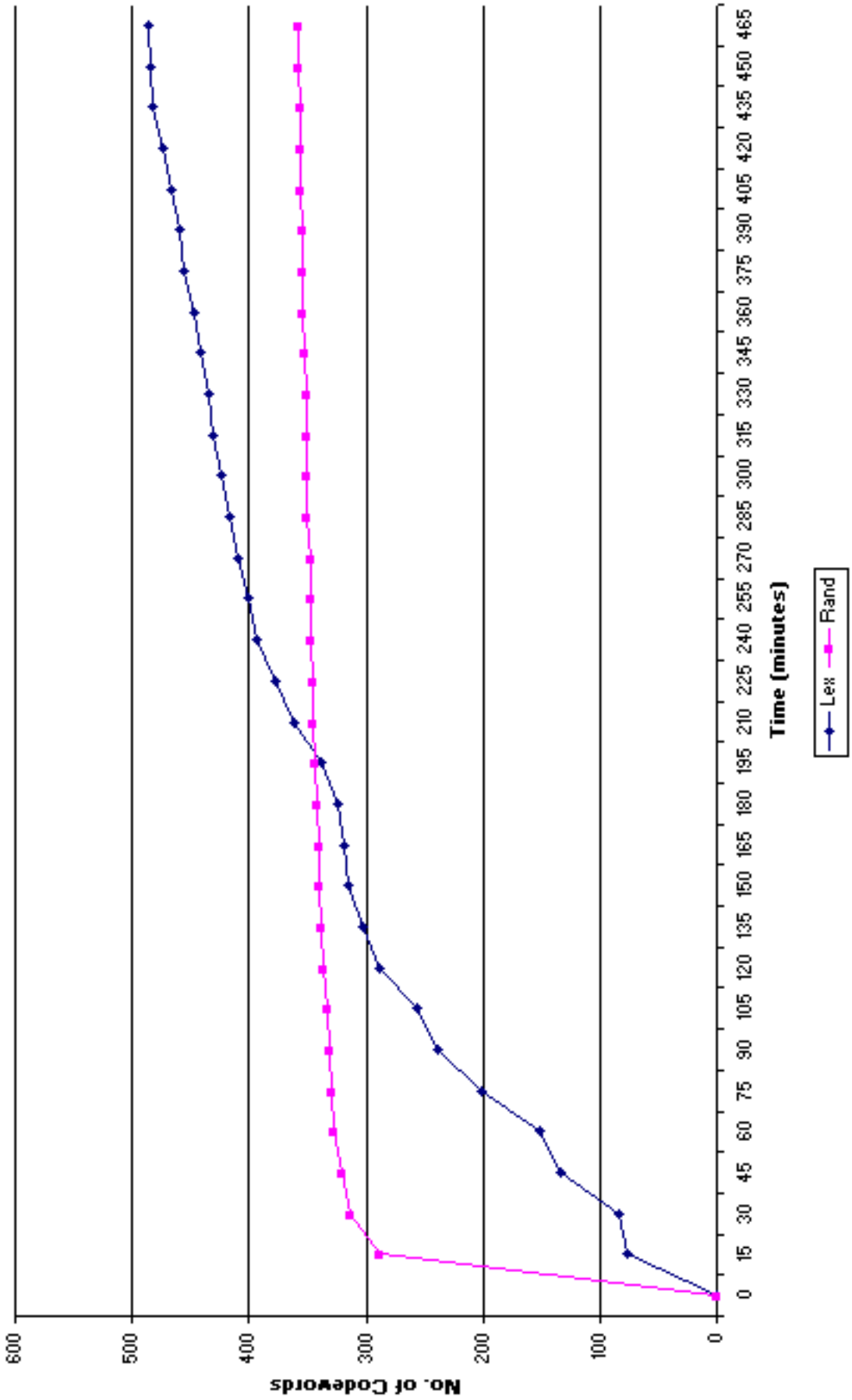

Figure 1: The time taken to find codewords for the lexicographic and random methods.

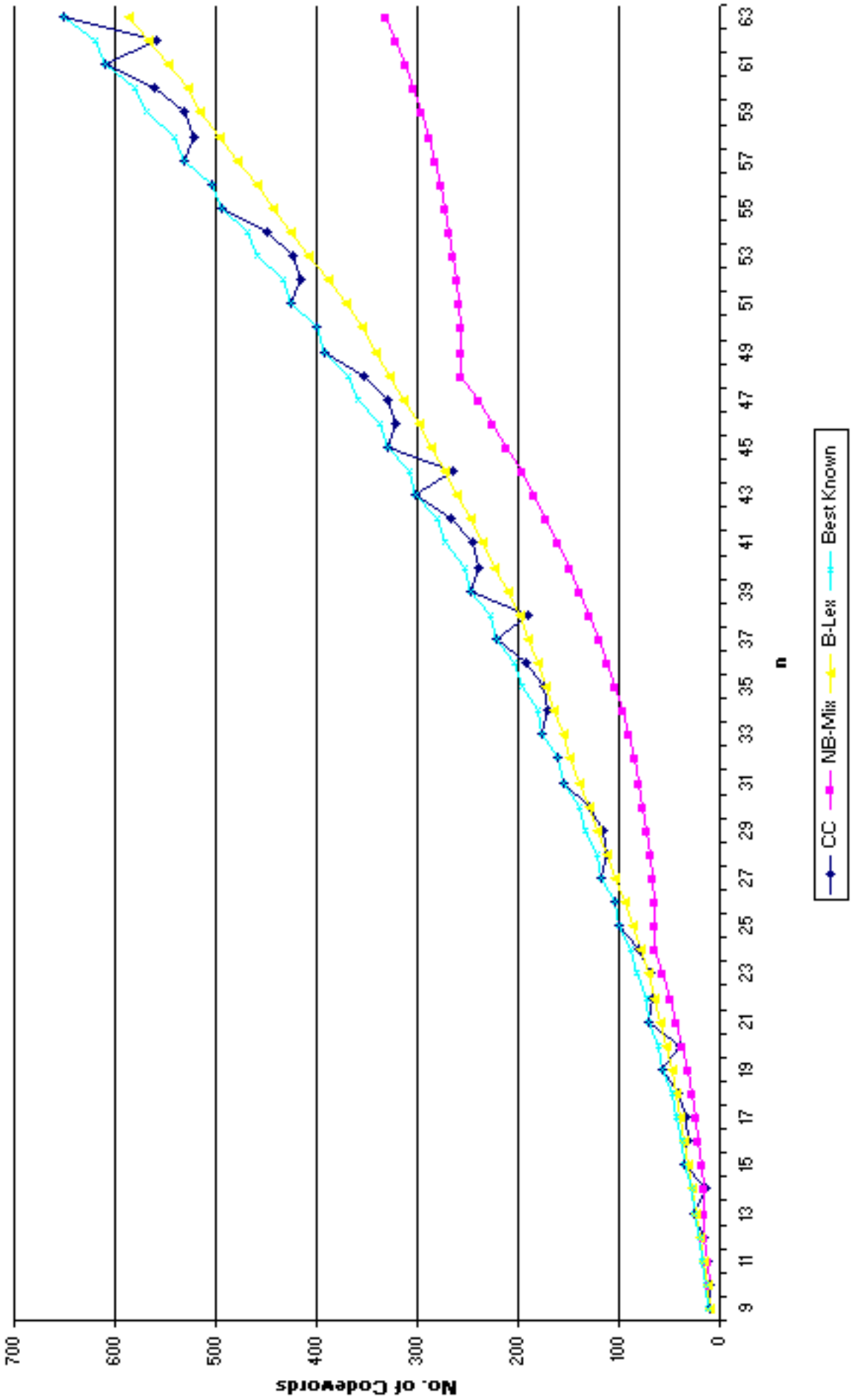

Figure 2: Comparison of Results $d = 4, w = 3$

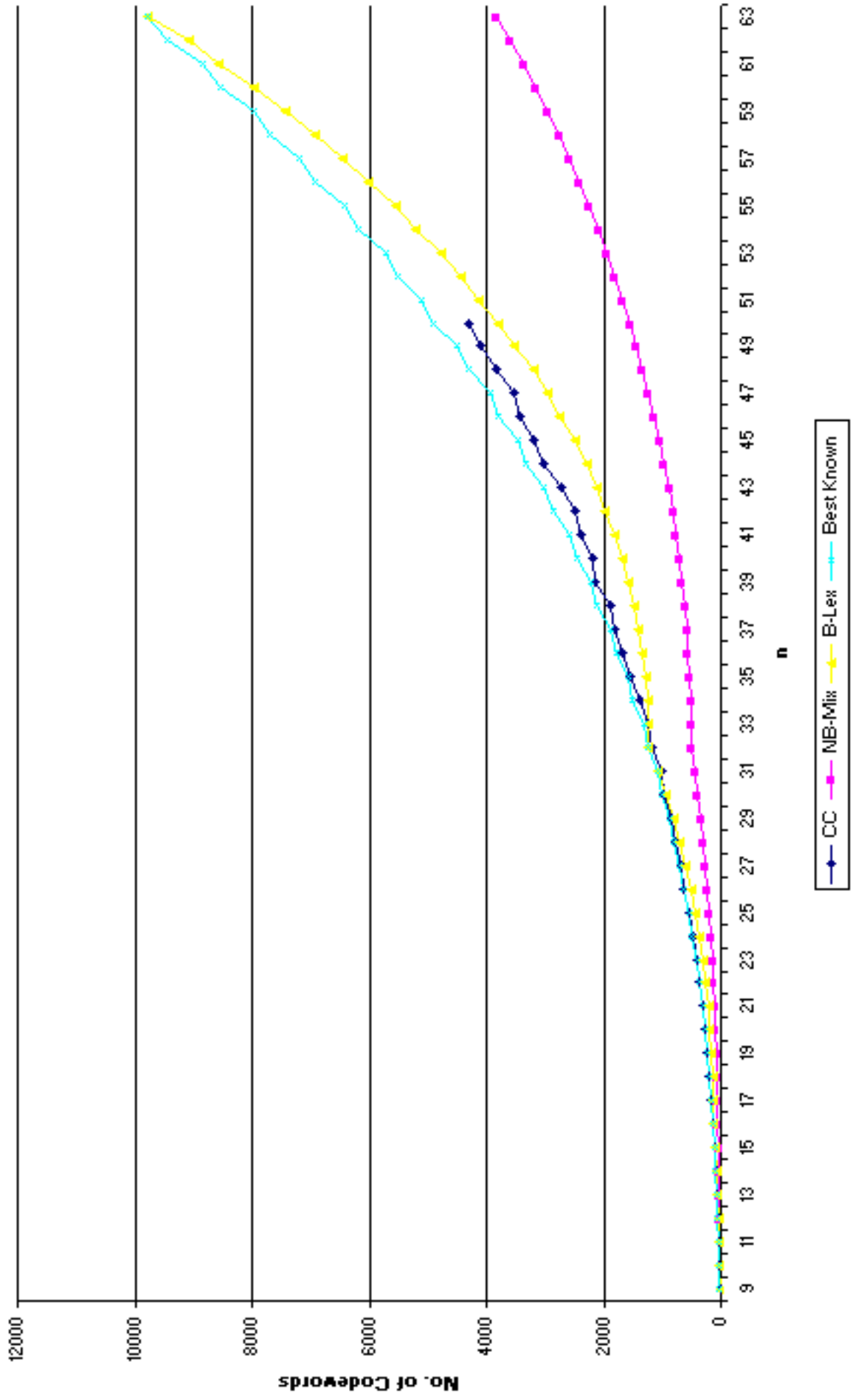

Figure 3: Comparison of Results $d = 4, w = 4$

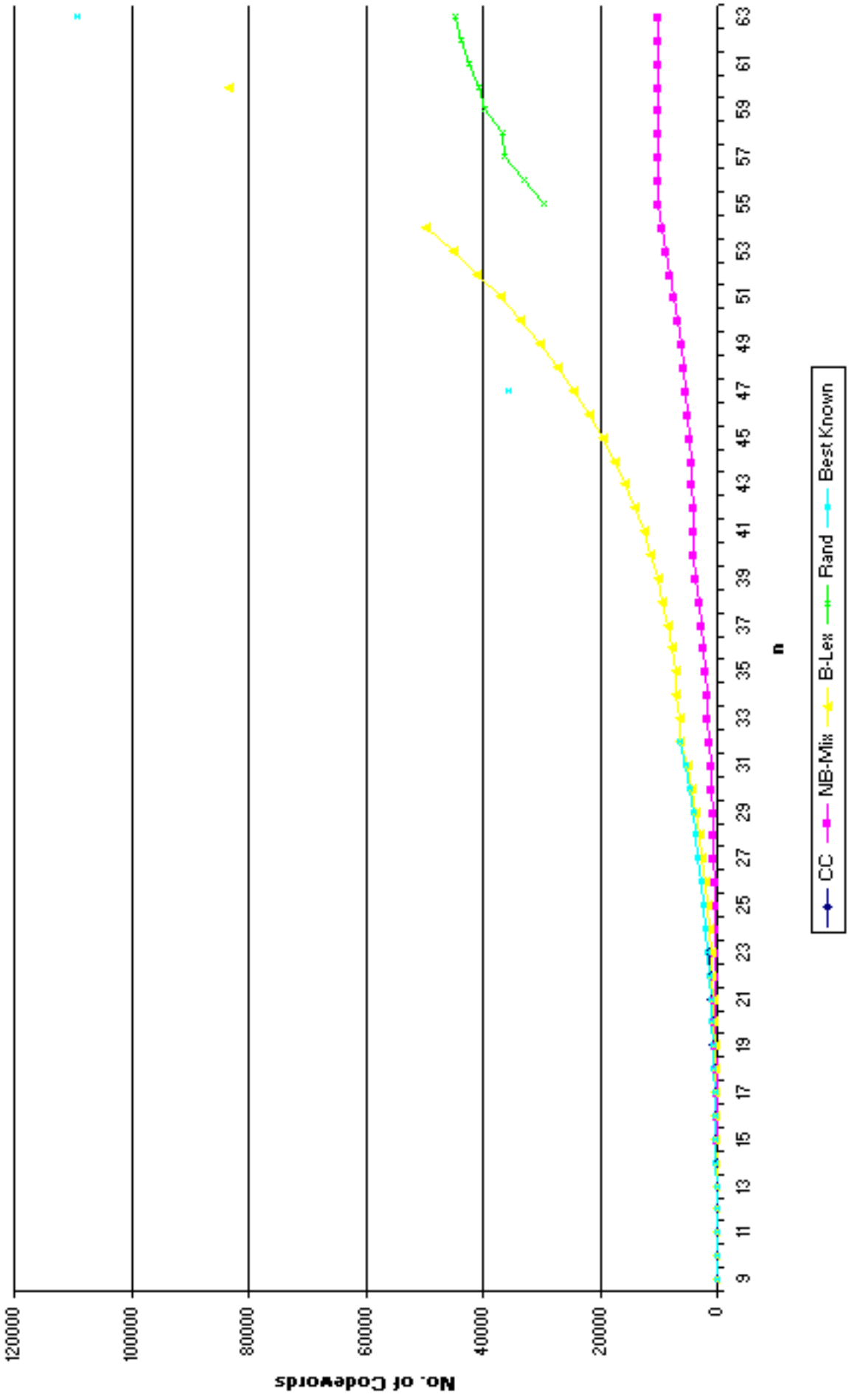

Figure 4: Comparison of Results $d = 4, w = 5$

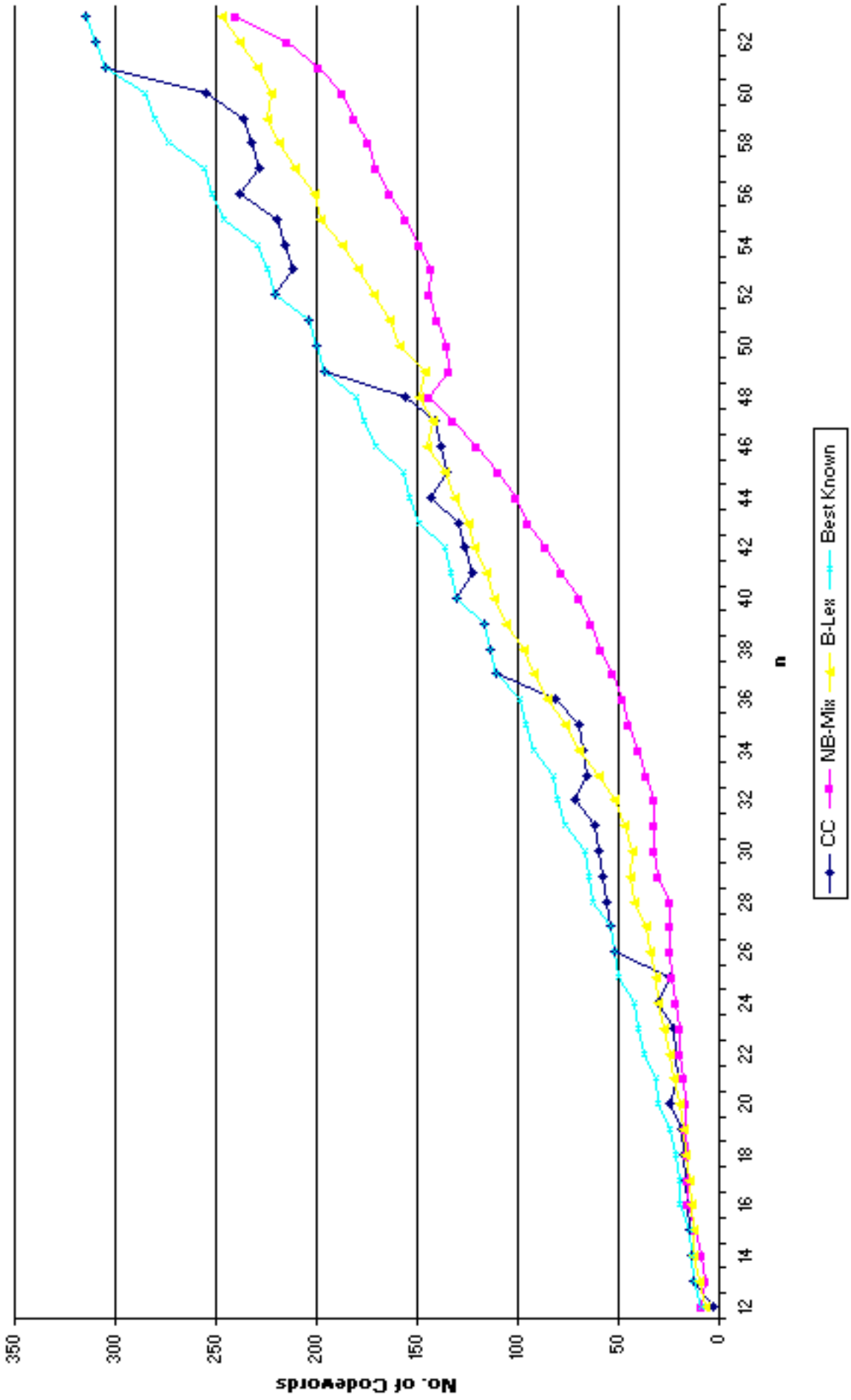

Figure 5: Comparison of Results $d = 6, w = 4$

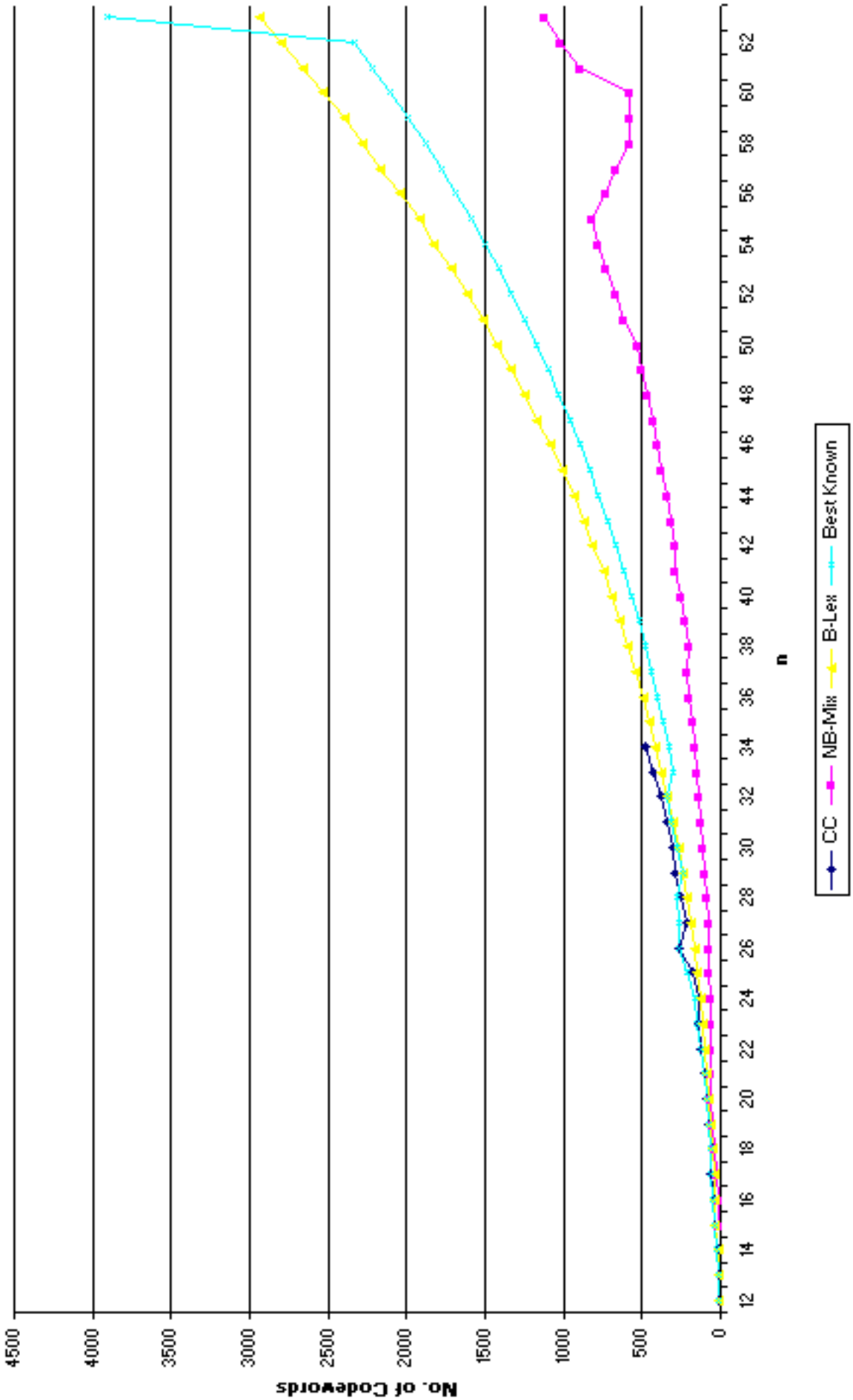

Figure 6: Comparison of Results $d = 6, w = 5$

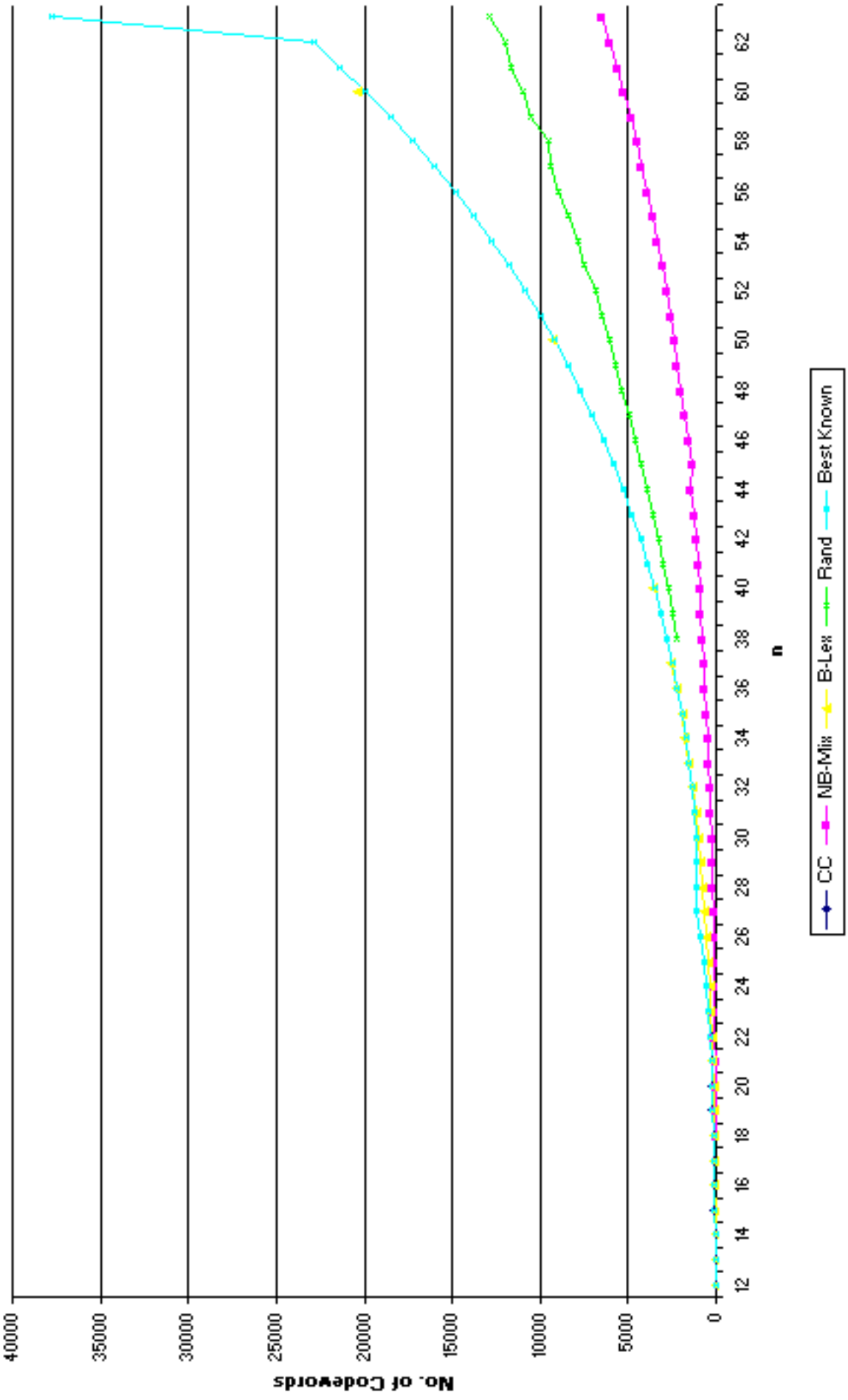

Figure 7: Comparison of Results $d = 6, w = 6$

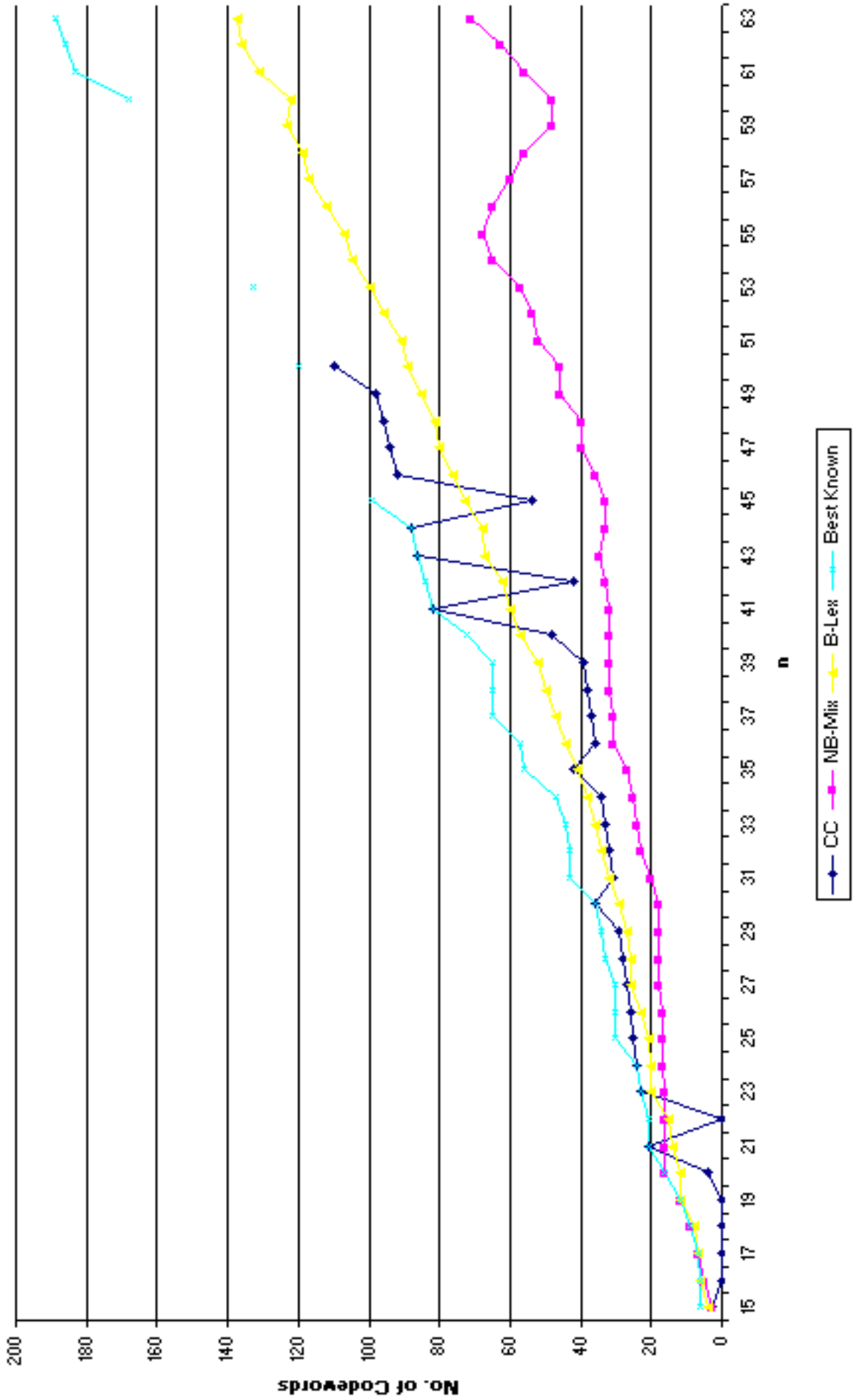

Figure 8: Comparison of Results $d = 8, w = 5$

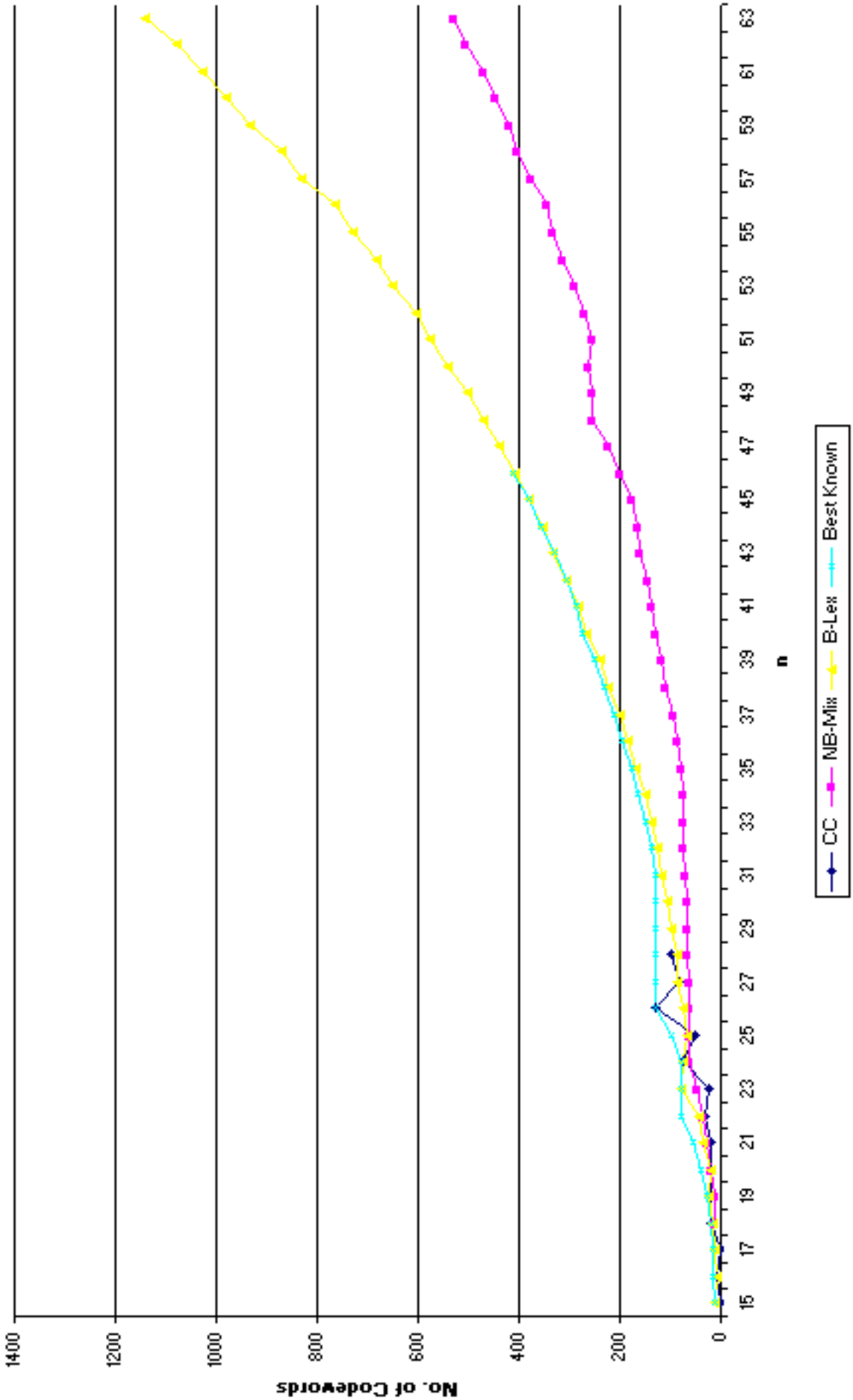

Figure 9: Comparison of Results $d = 8, w = 6$

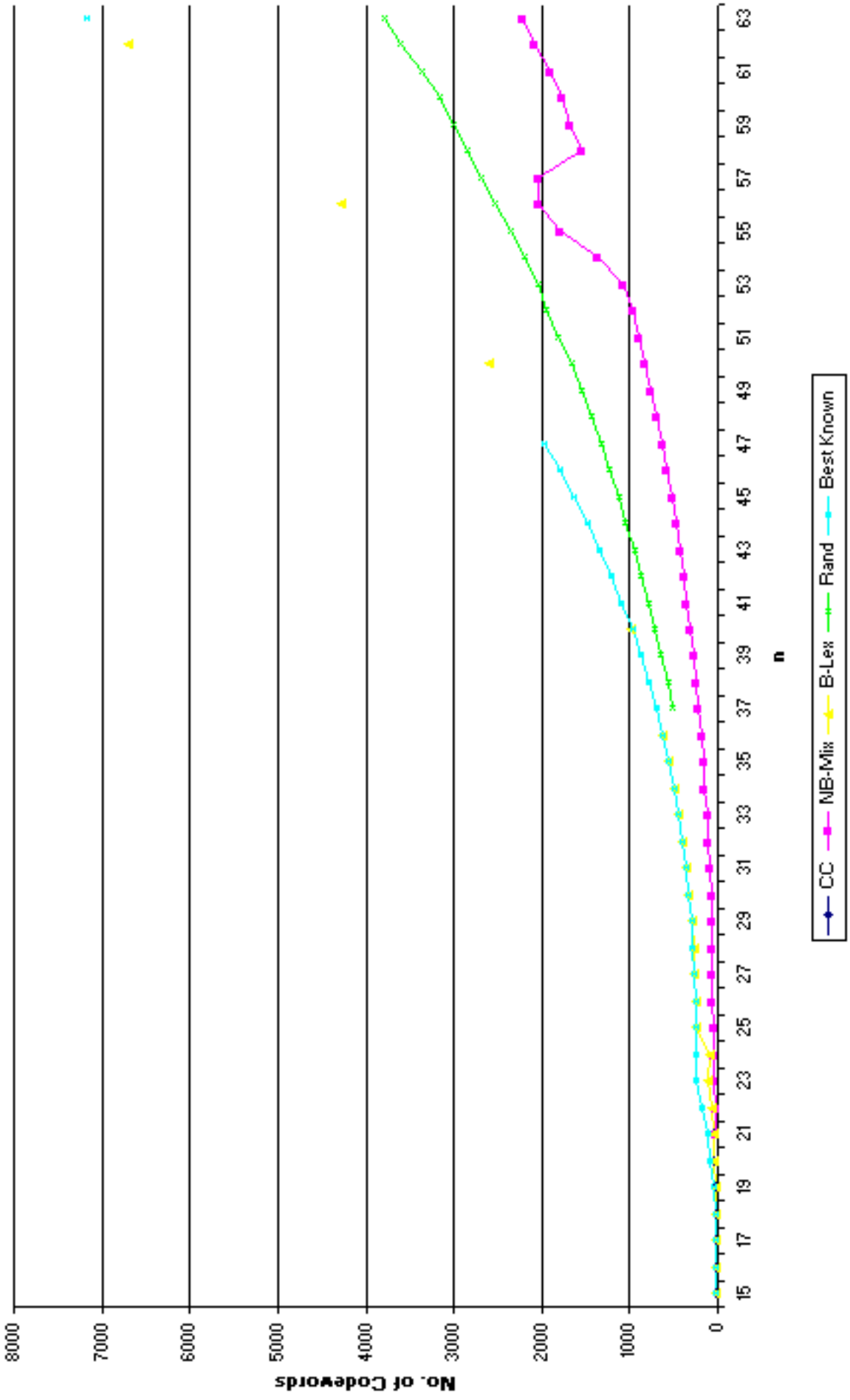

Figure 10: Comparison of Results $d = 8, w = 7$

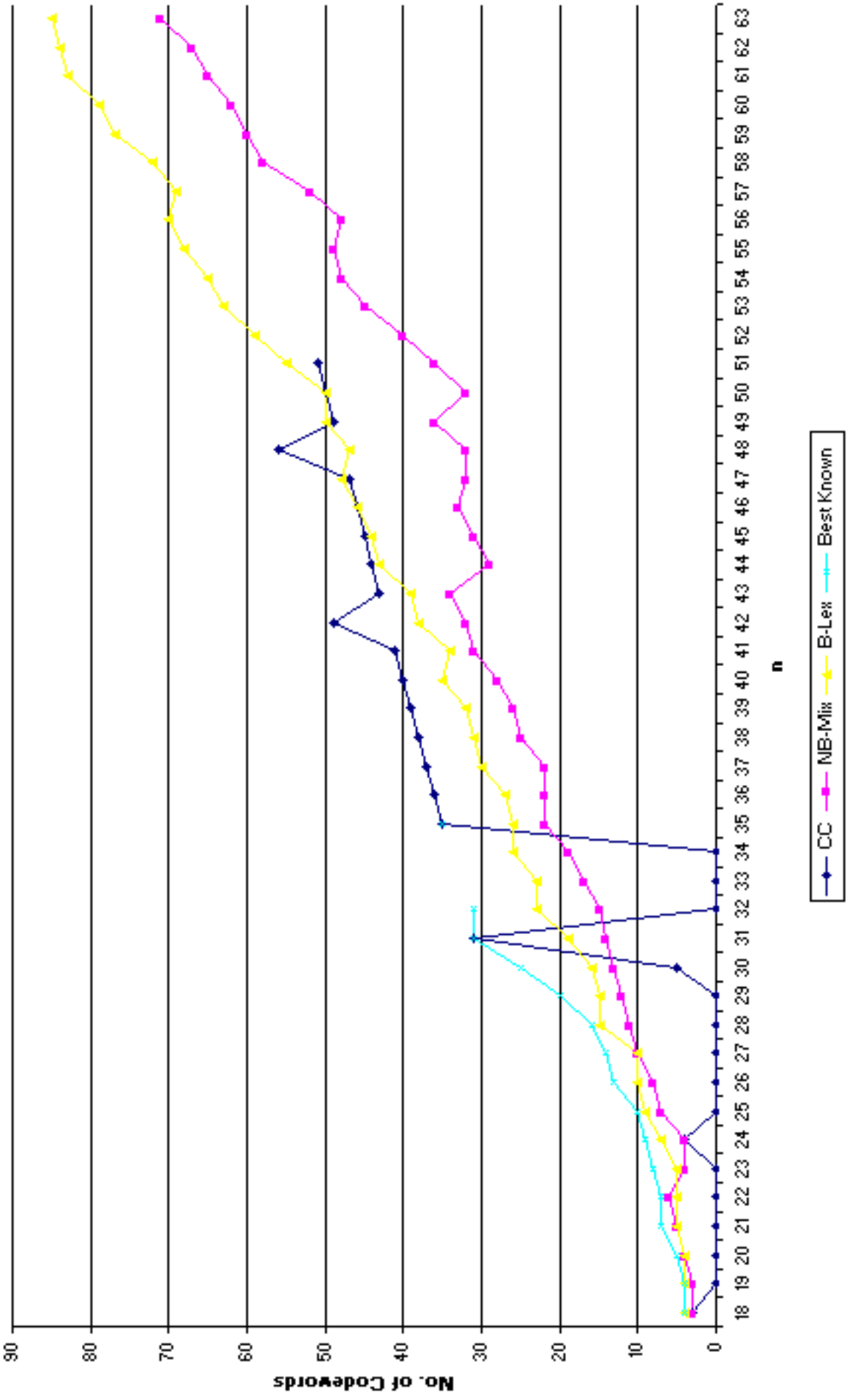

Figure 11: Comparison of Results $d = 10, w = 6$

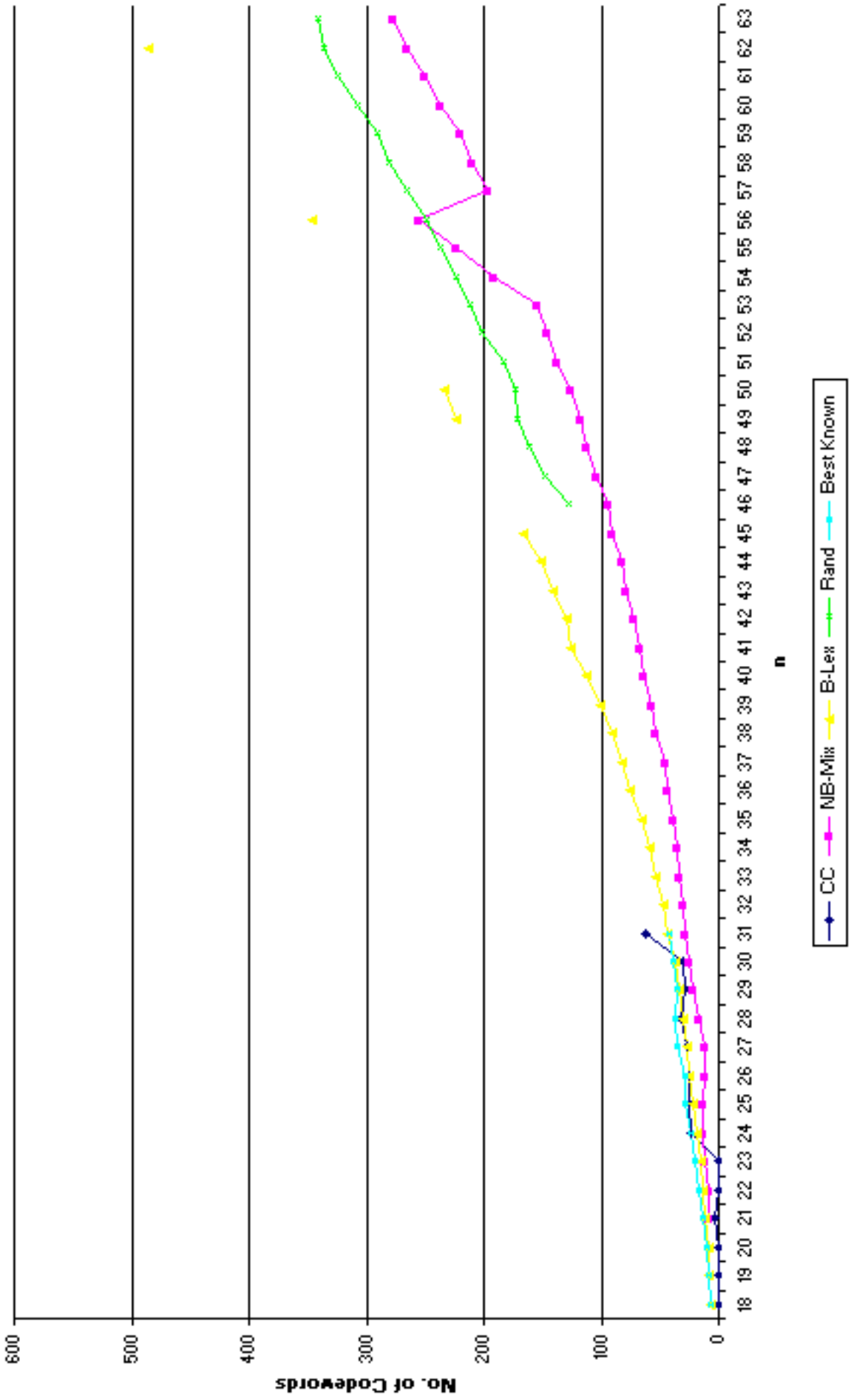

Figure 12: Comparison of Results $d = 10, w = 7$

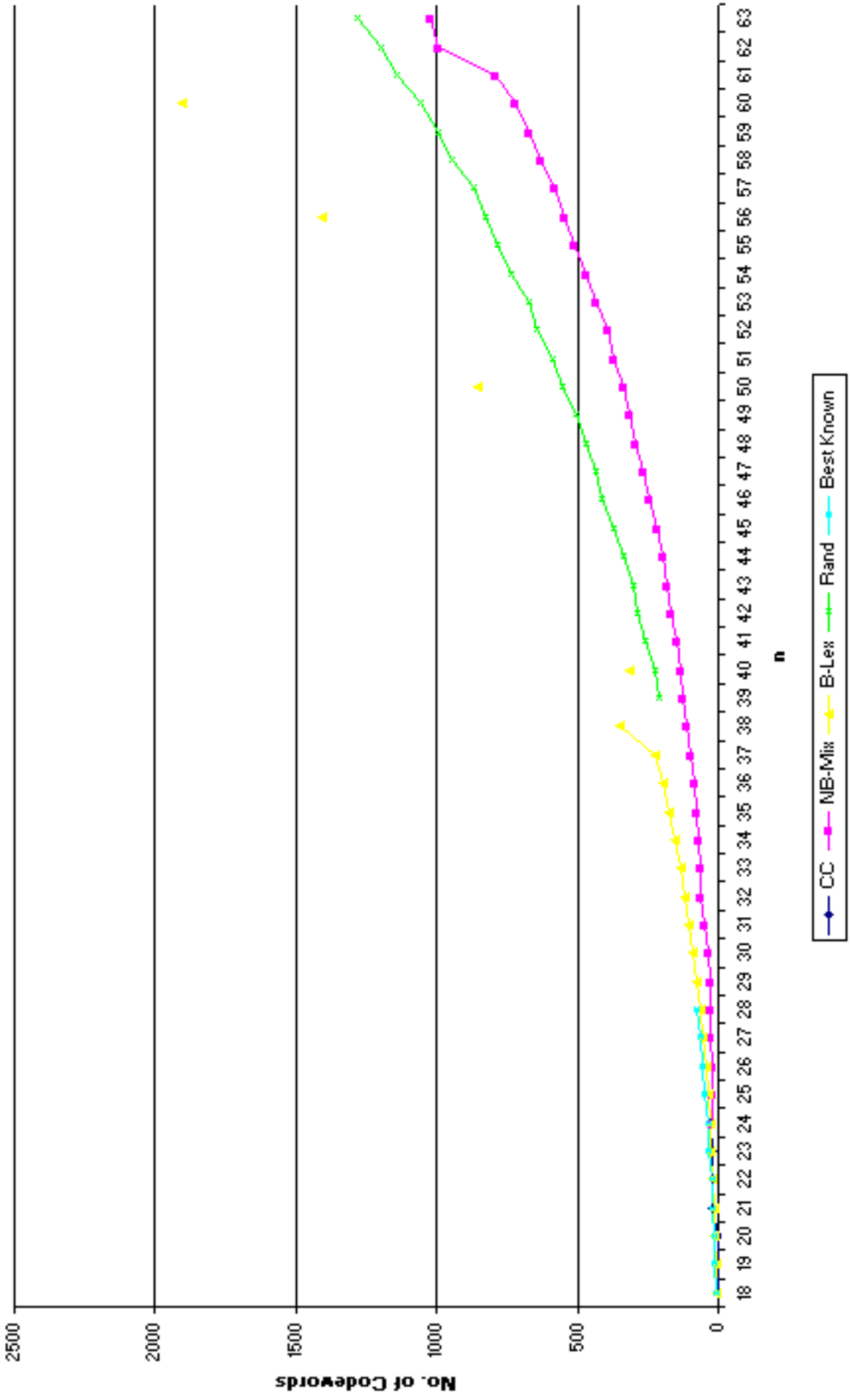

Figure 13: Comparison of Results $d = 10, w = 8$

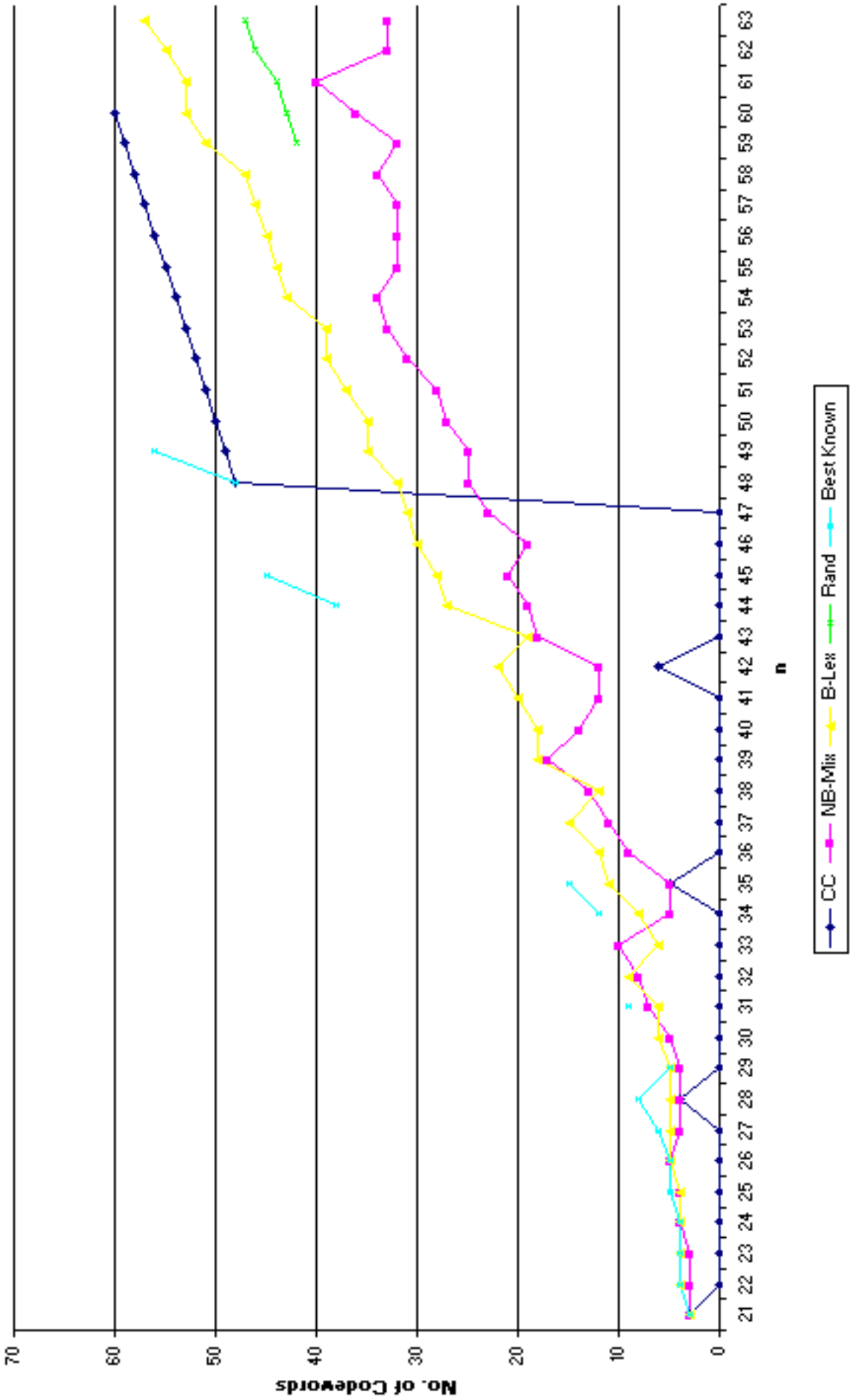

Figure 14: Comparison of Results $d = 12, w = 7$

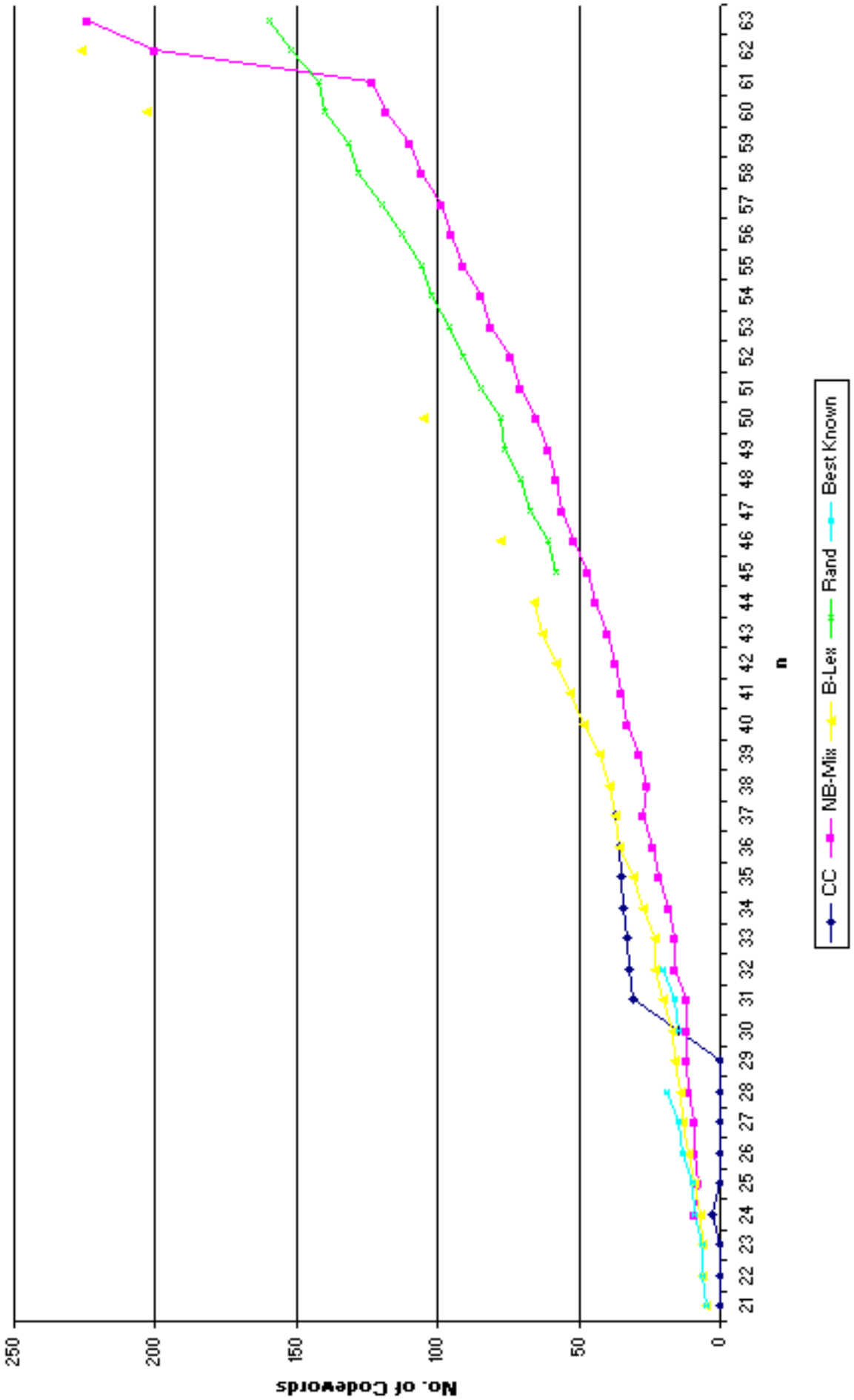

Figure 15: Comparison of Results $d = 12, w = 8$

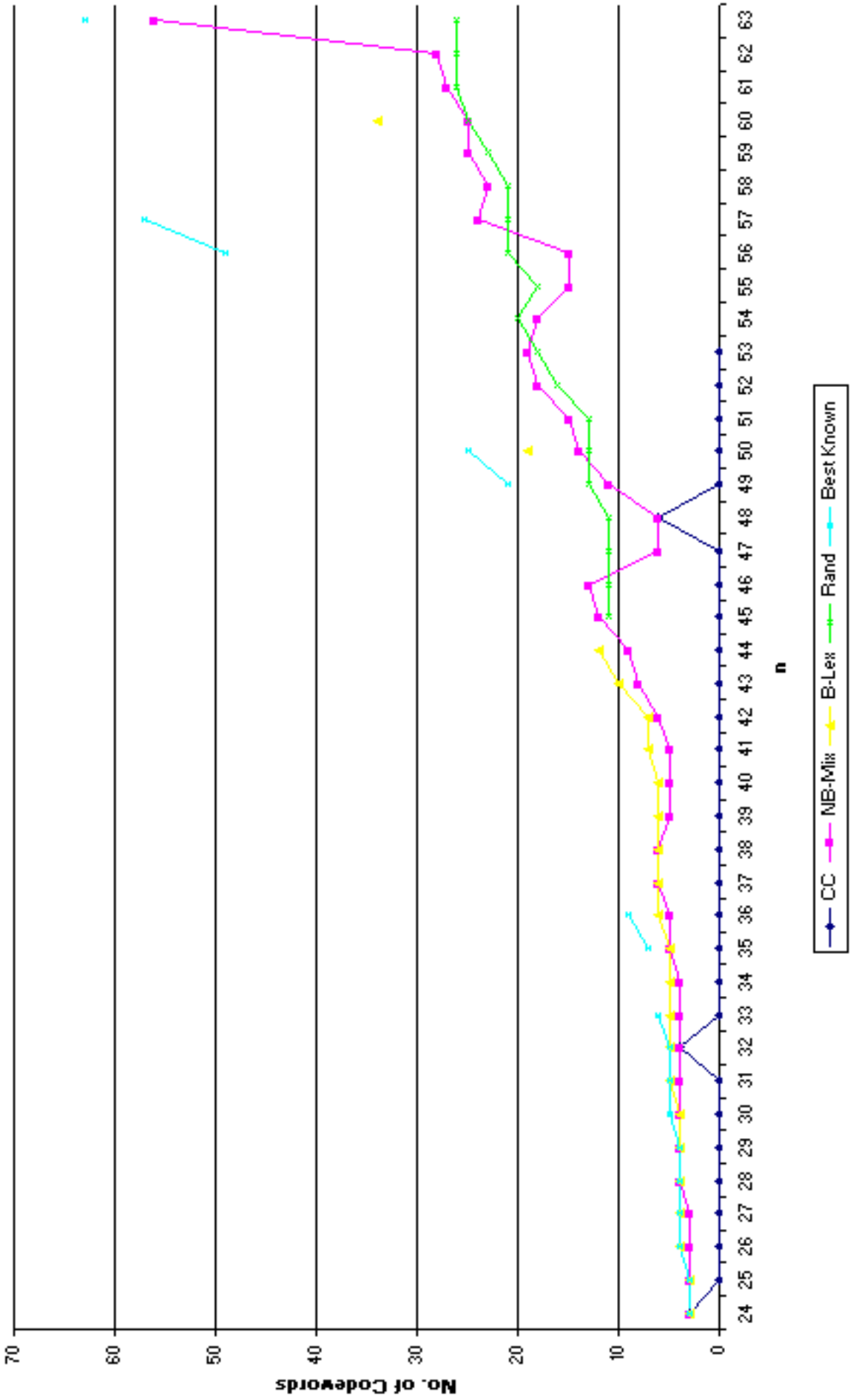

Figure 16: Comparison of Results $d = 14, w = 8$

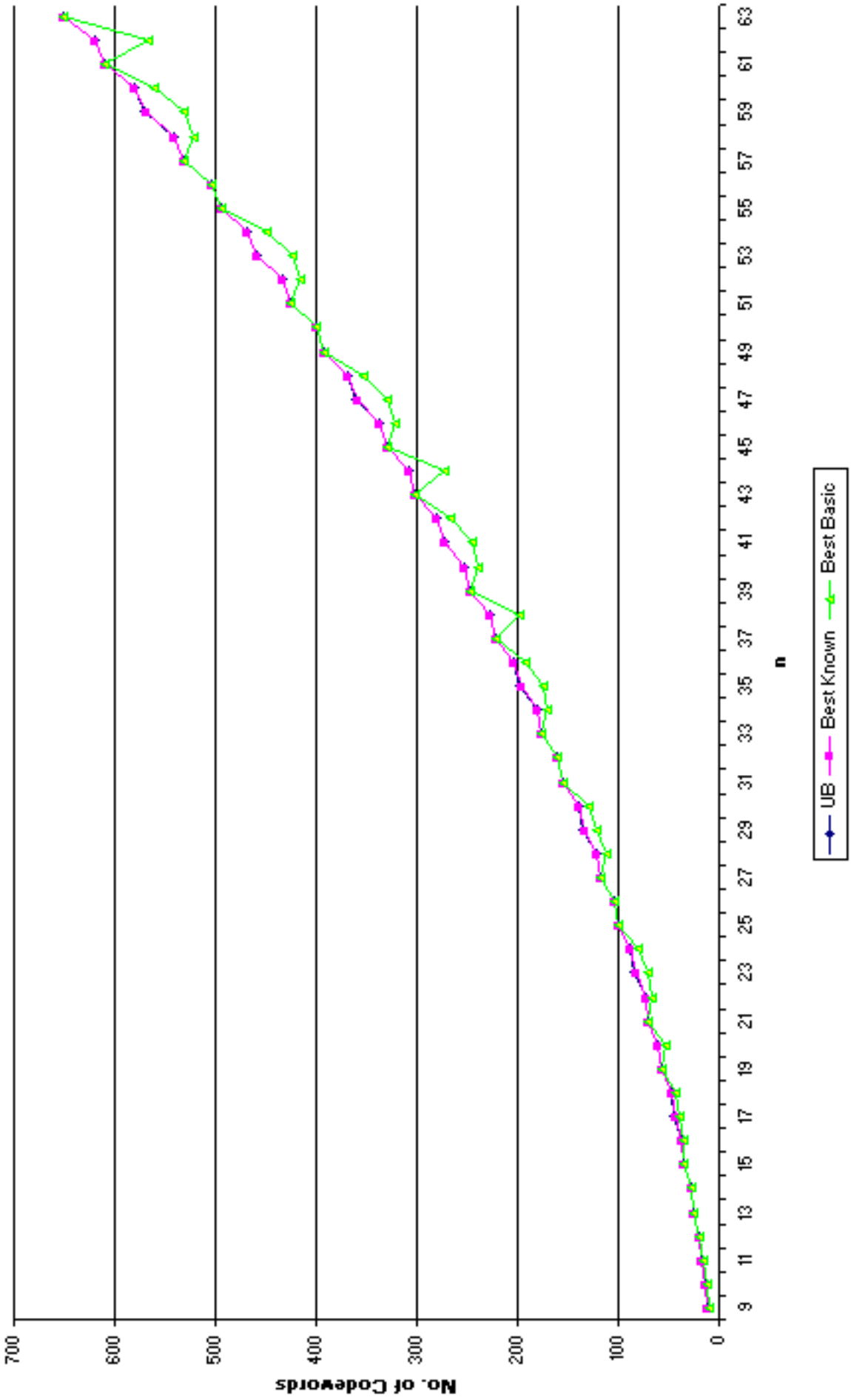

Figure 17: Comparison of the Upper Bound, Best Basic and the Best Known $d = 4, w = 3$

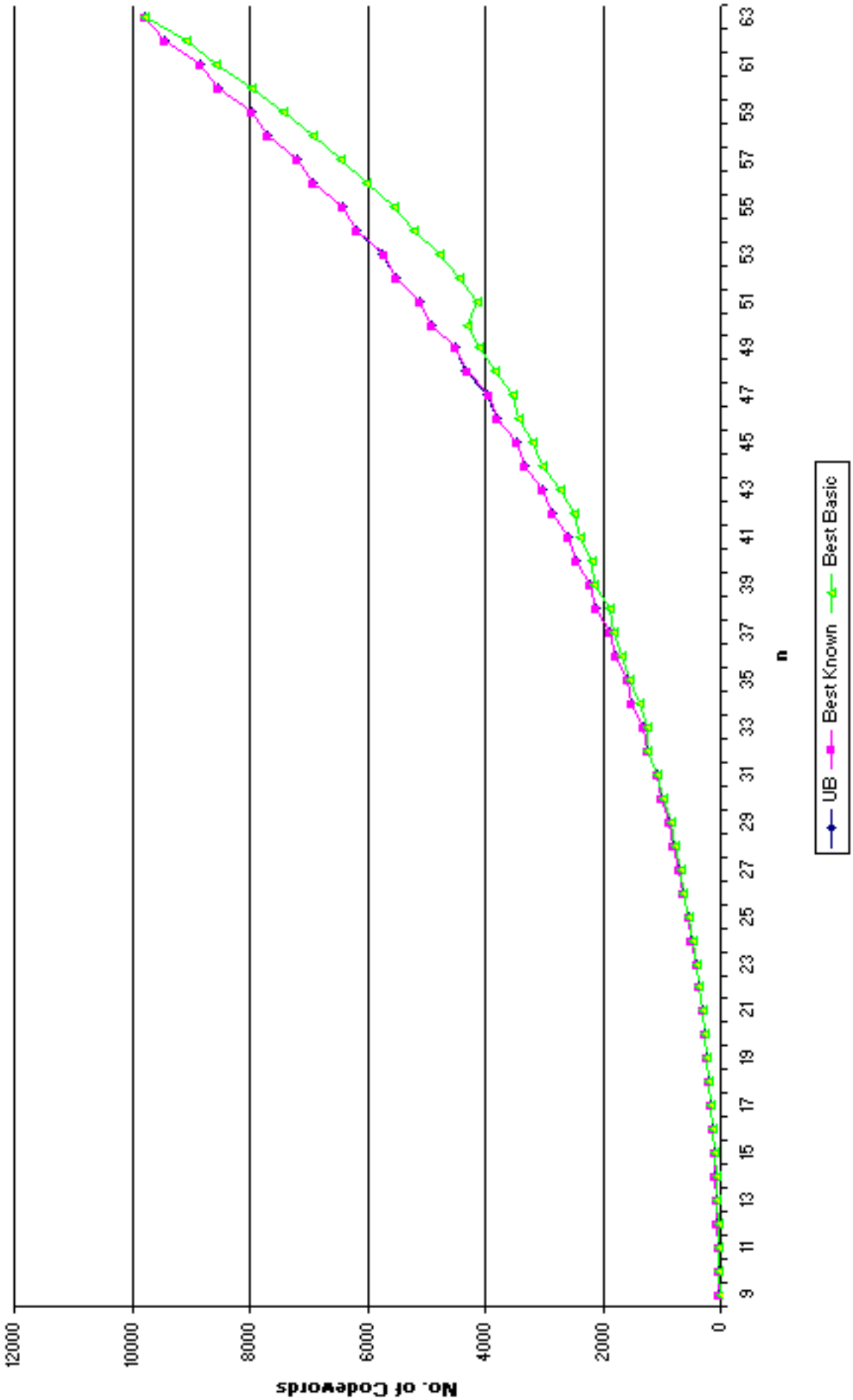

Figure 18: Comparison of the Upper Bound, Best Basic and the Best Known $d = 4, w = 4$

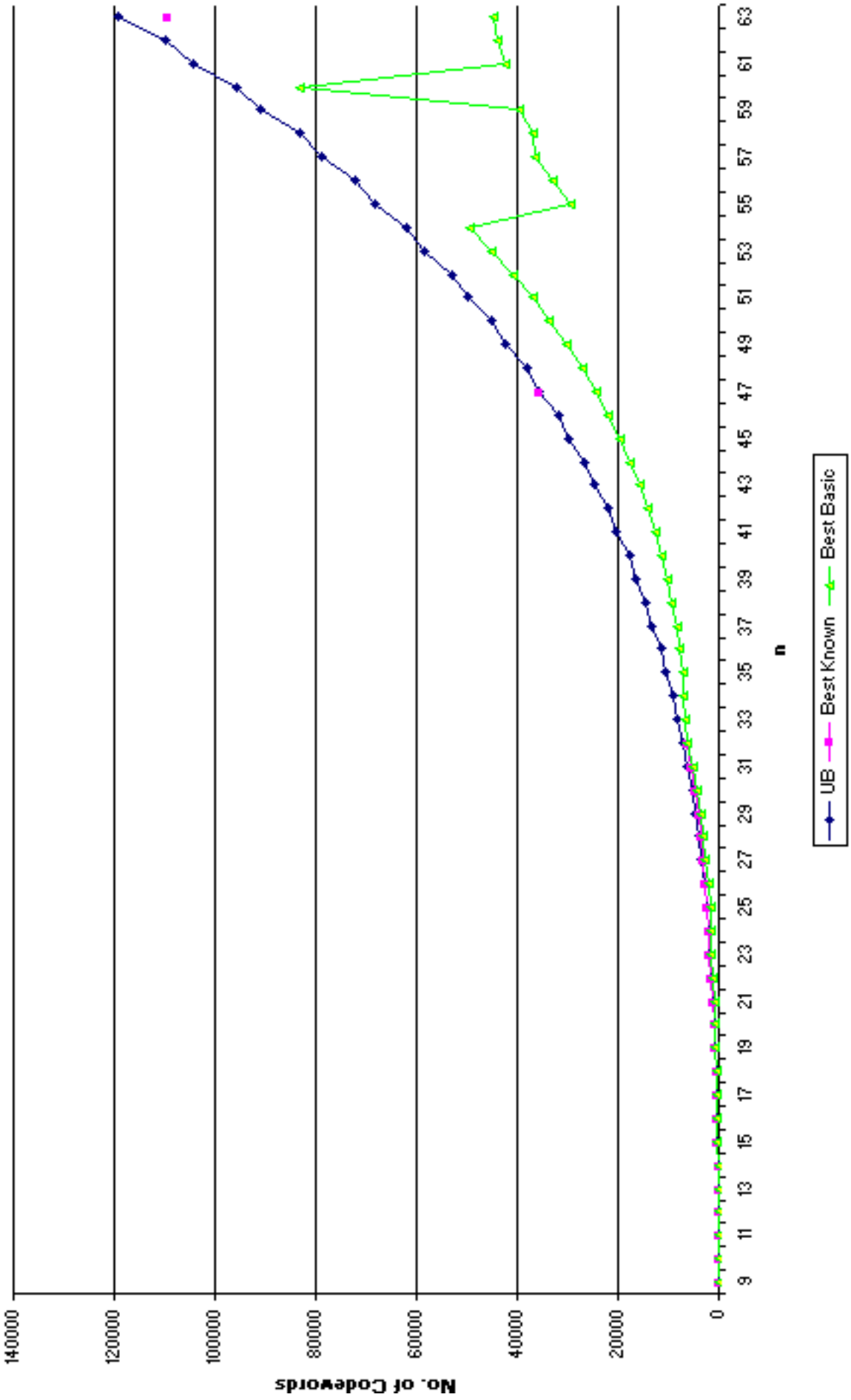

Figure 19: Comparison of the Upper Bound, Best Basic and the Best Known $d = 4, w = 5$

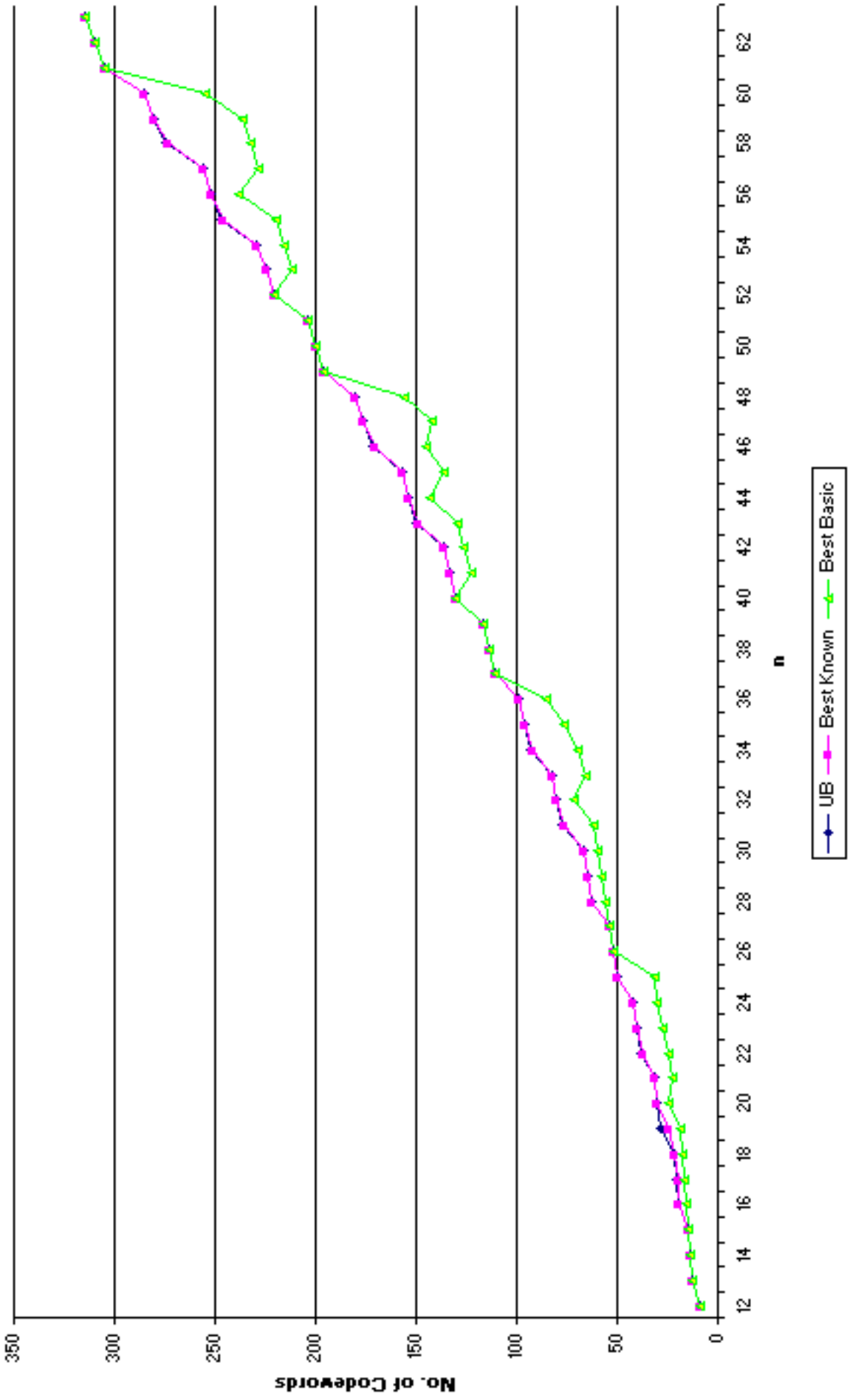

Figure 20: Comparison of the Upper Bound, Best Basic and the Best Known $d = 6, w = 4$

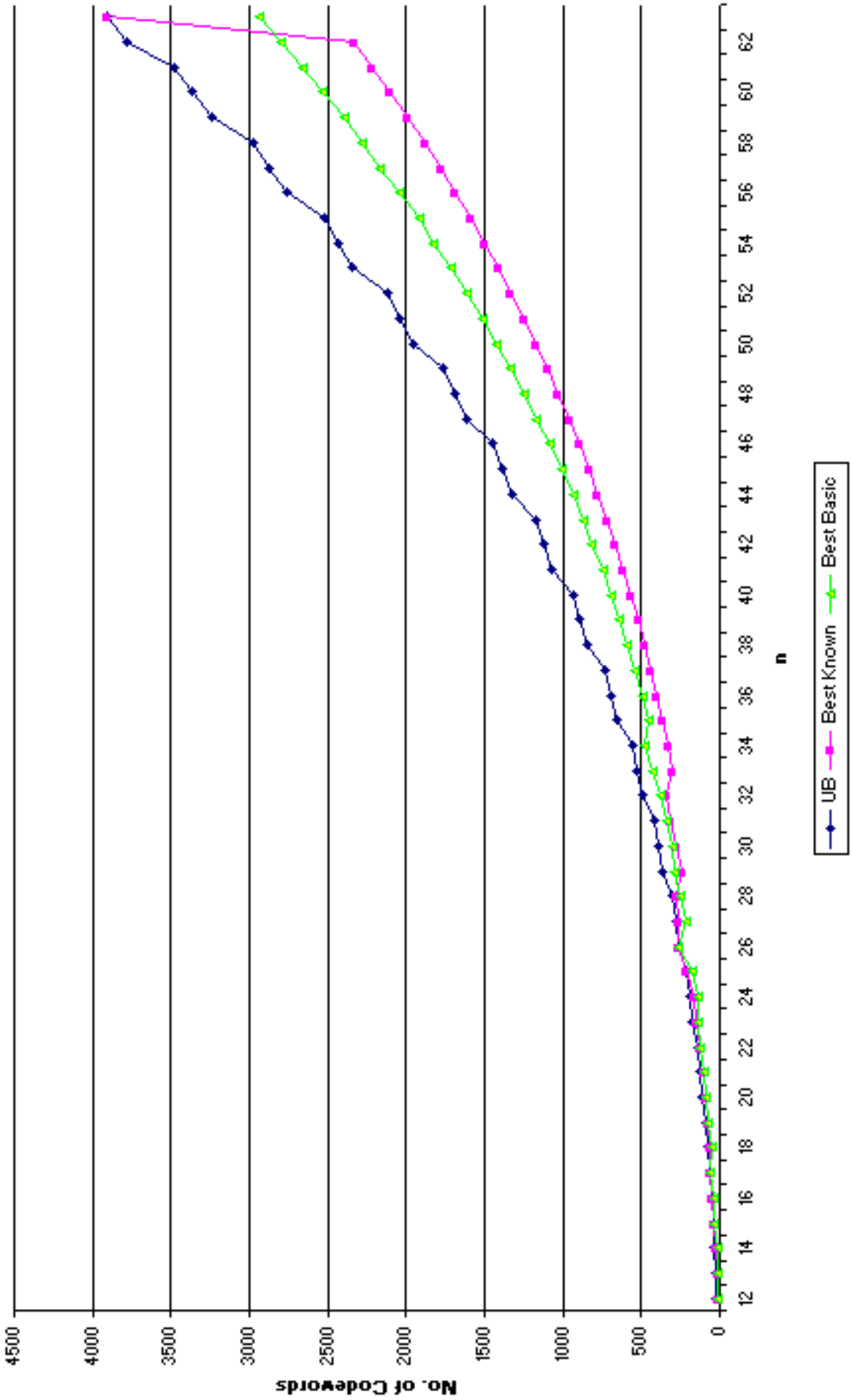

Figure 21: Comparison of the Upper Bound, Best Basic and the Best Known $d = 6, w = 5$

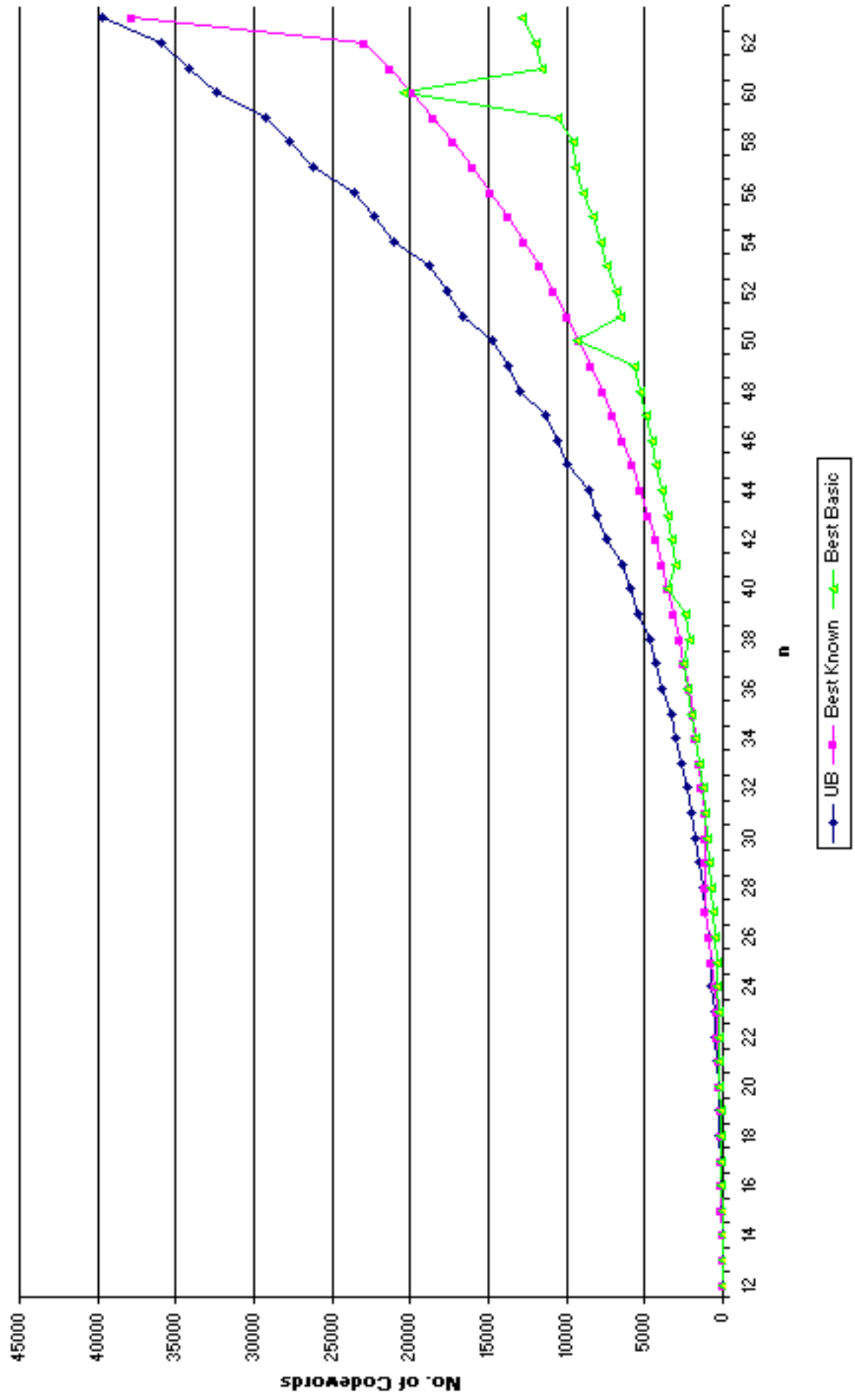

Figure 22: Comparison of the Upper Bound, Best Basic and the Best Known $d = 6, w = 6$

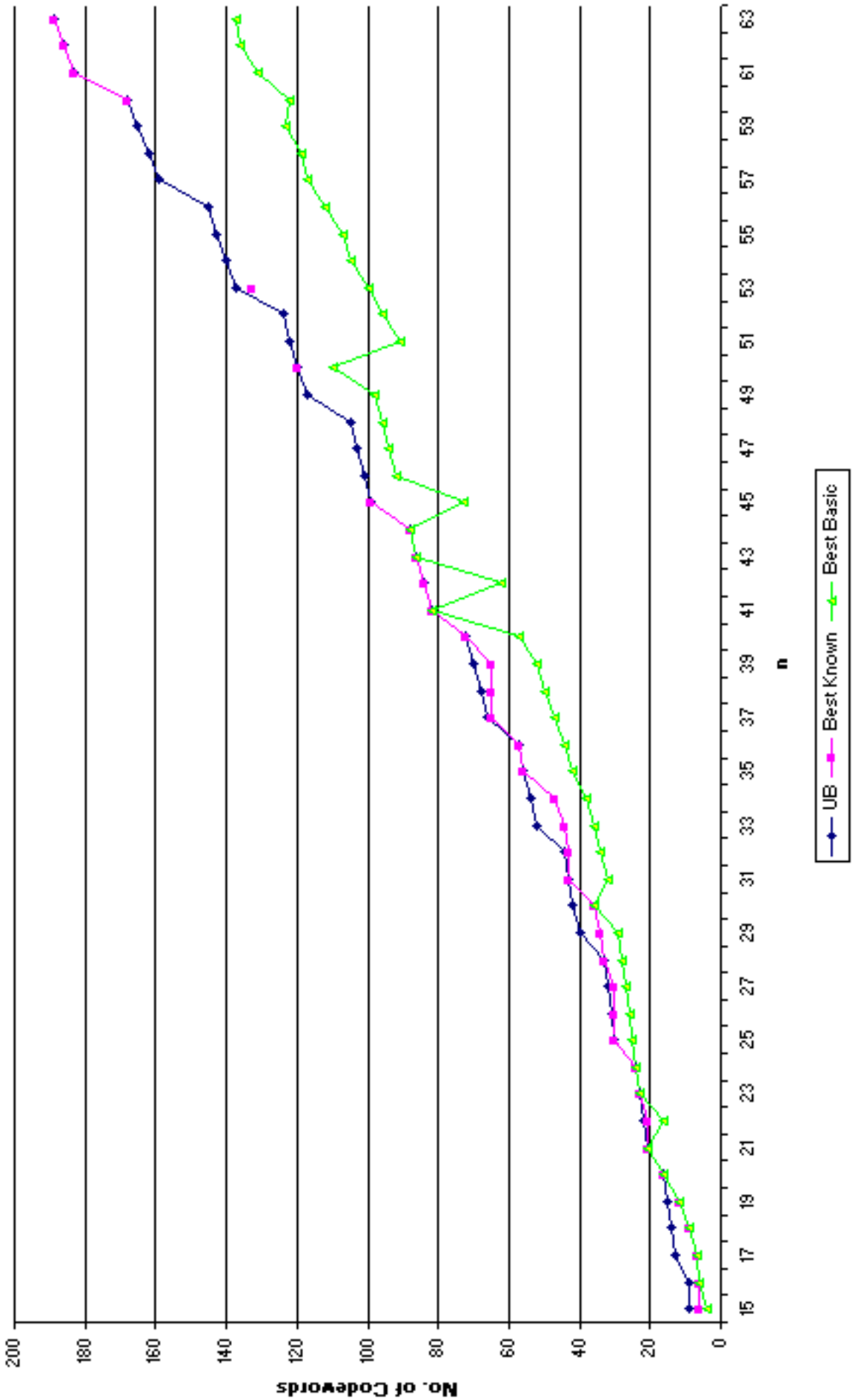

Figure 23: Comparison of the Upper Bound, Best Basic and the Best Known $d = 8, w = 5$

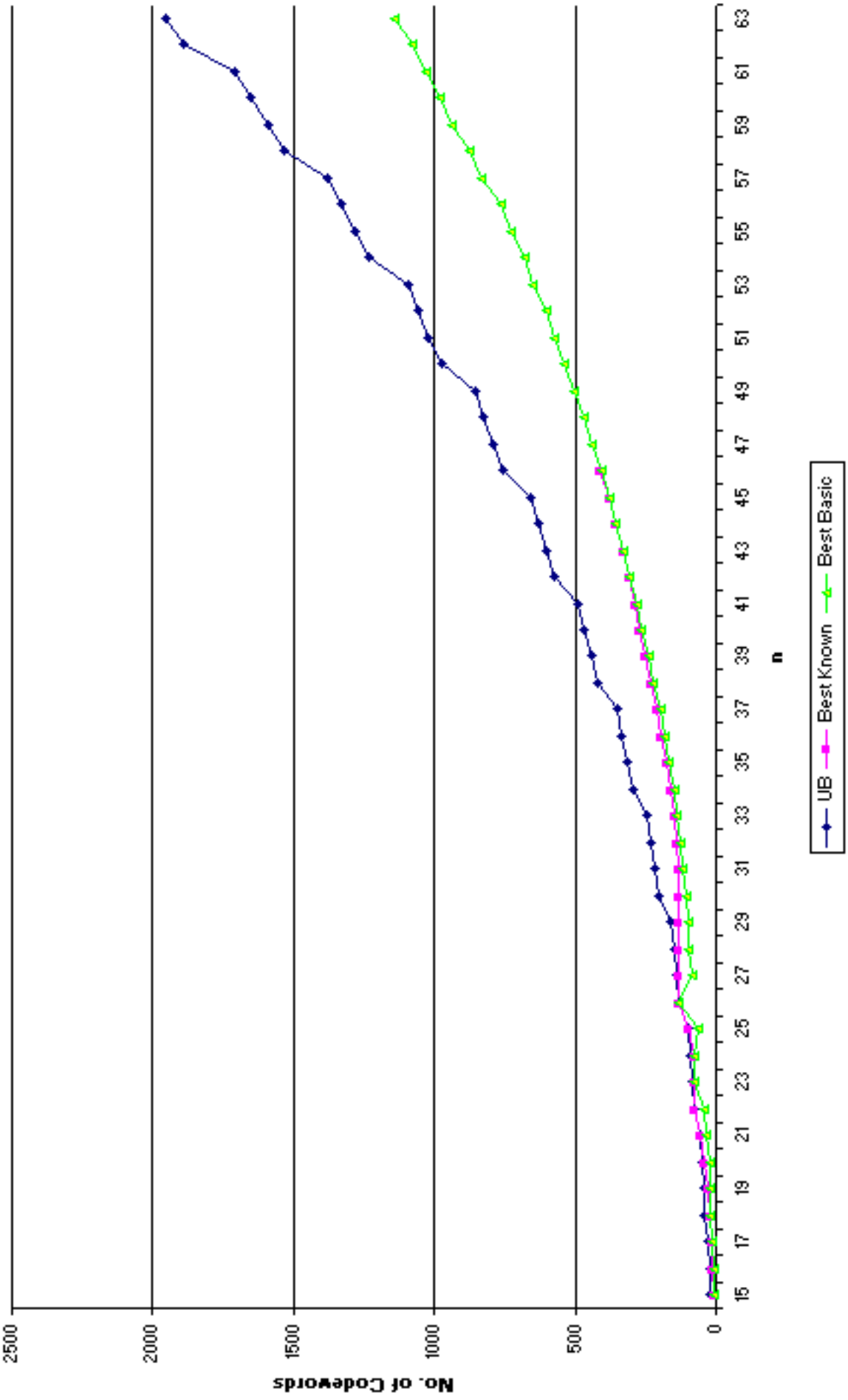

Figure 24: Comparison of the Upper Bound, Best Basic and the Best Known $d = 8, w = 6$

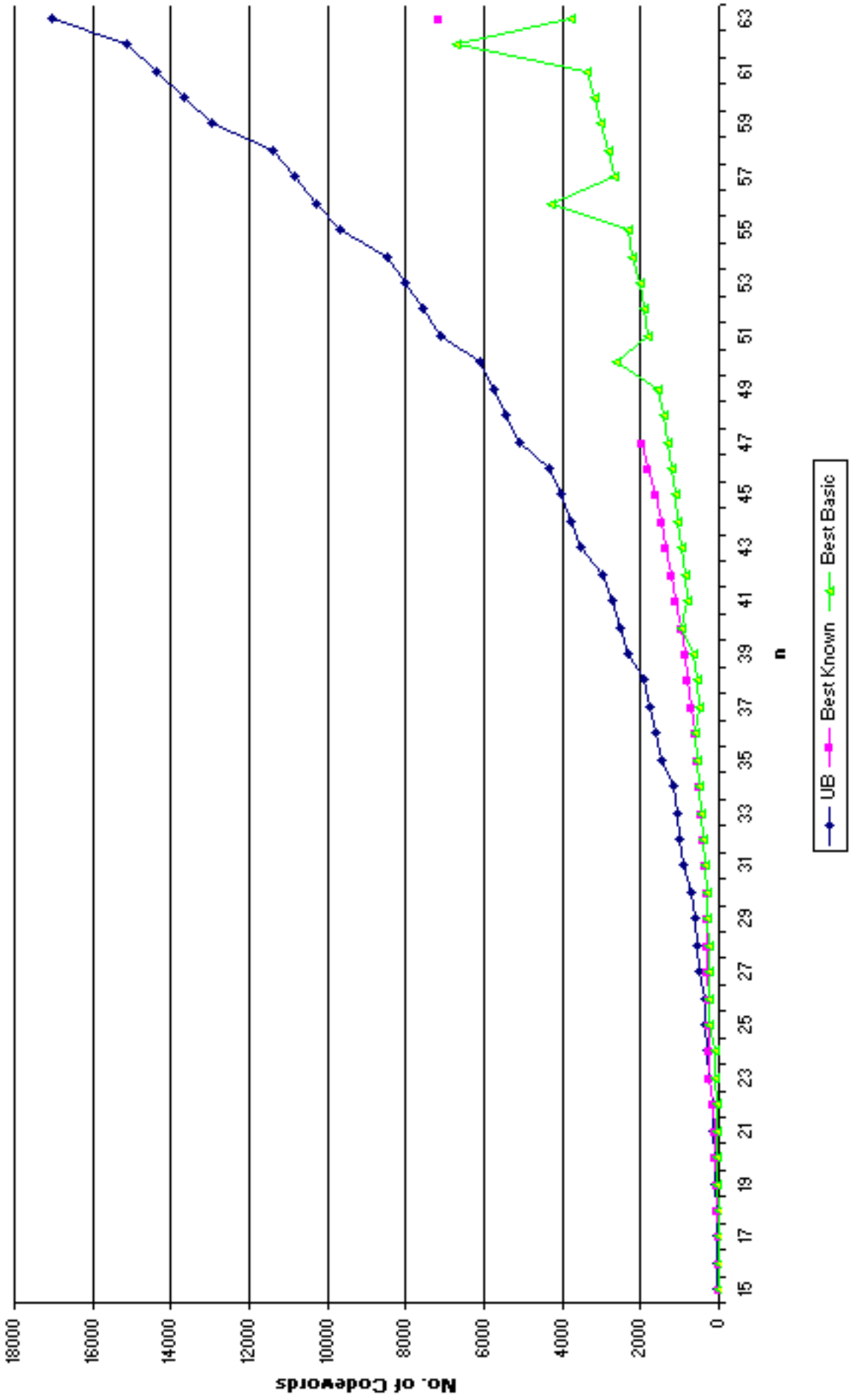

Figure 25: Comparison of the Upper Bound, Best Basic and the Best Known $d = 8, w = 7$

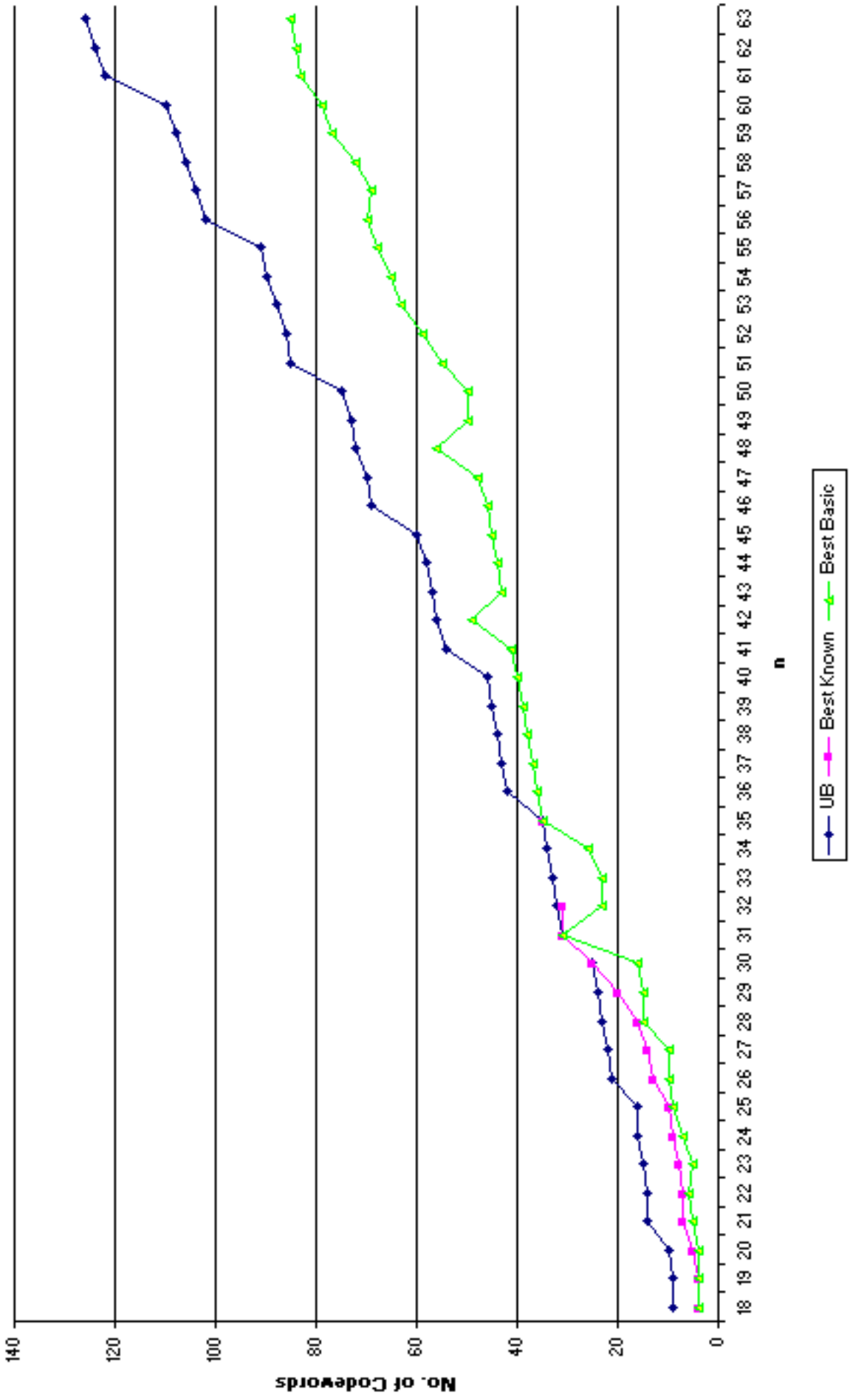

Figure 26: Comparison of the Upper Bound, Best Basic and the Best Known $d = 10, w = 6$

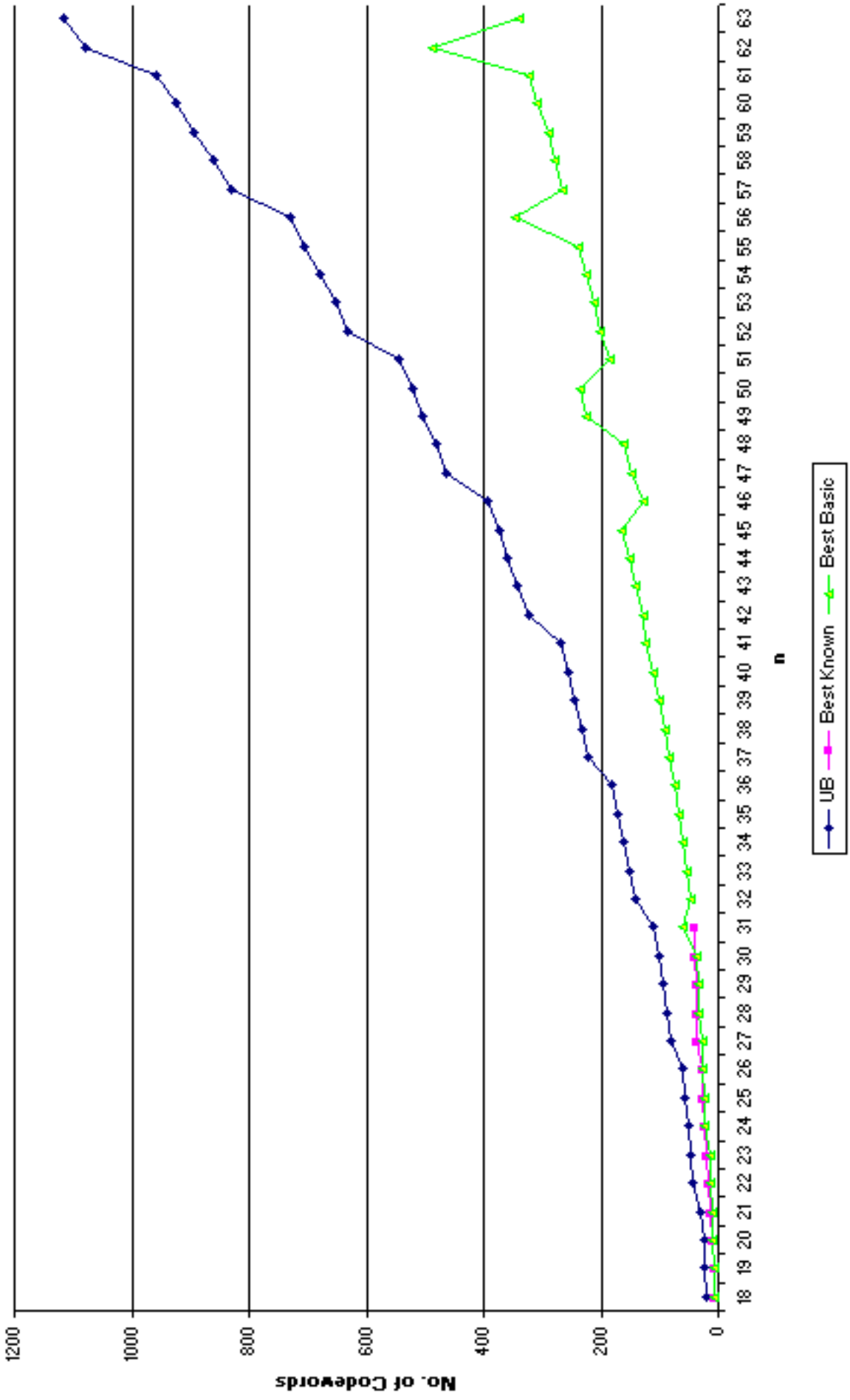

Figure 27: Comparison of the Upper Bound, Best Basic and the Best Known $d = 10, w = 7$

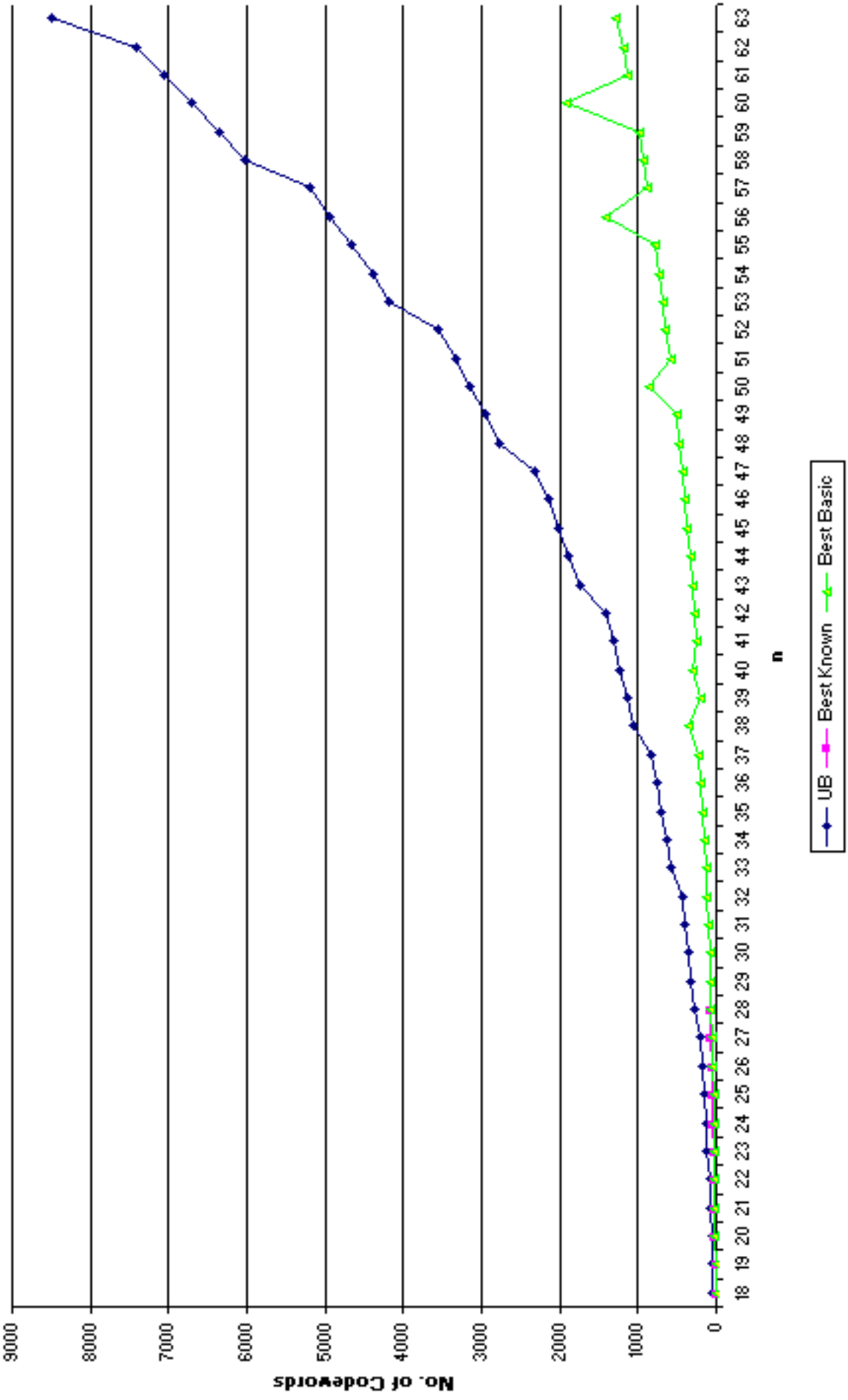

Figure 28: Comparison of the Upper Bound, Best Basic and the Best Known $d = 10, w = 8$

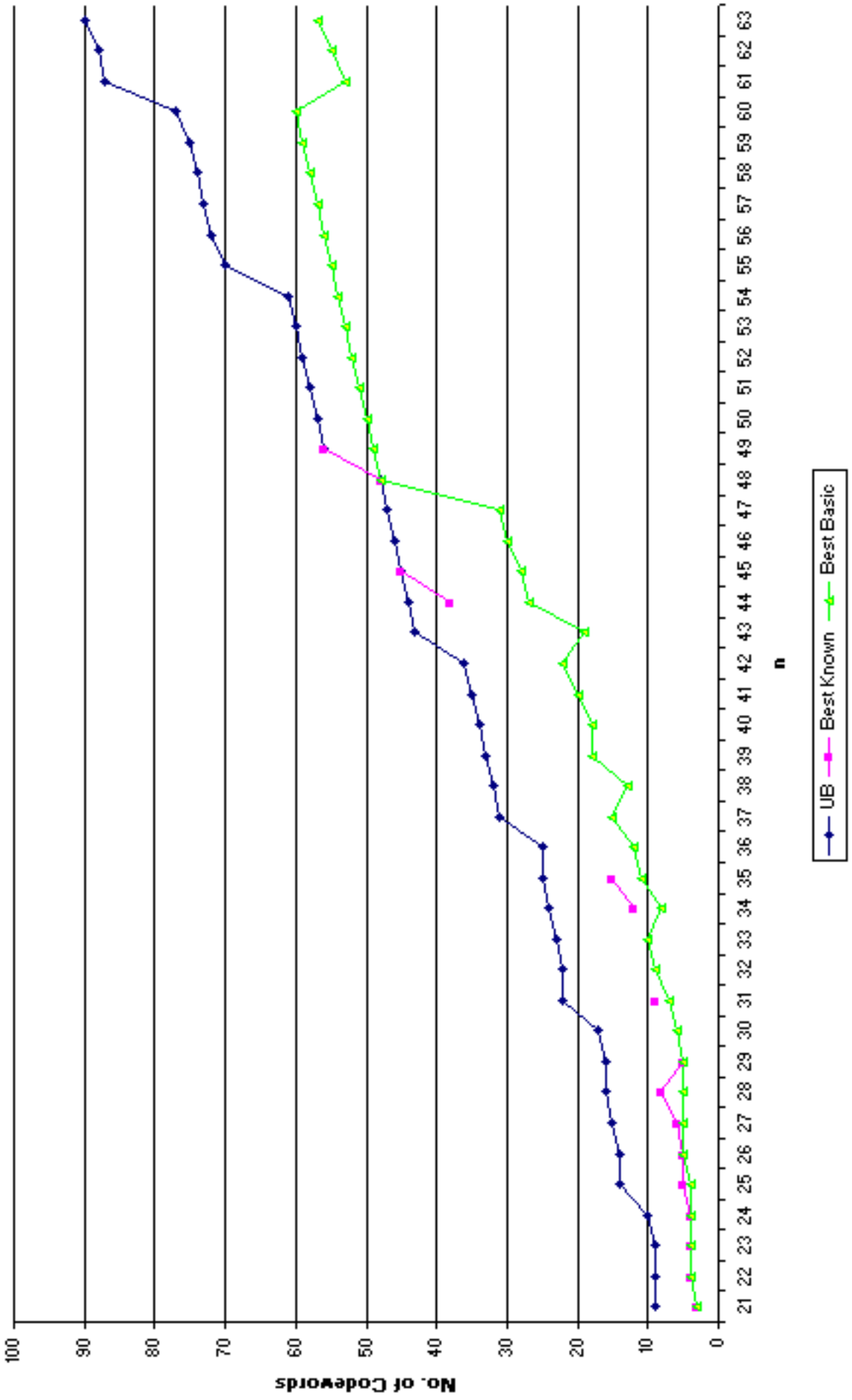

Figure 29: Comparison of the Upper Bound, Best Basic and the Best Known $d = 12, w = 7$

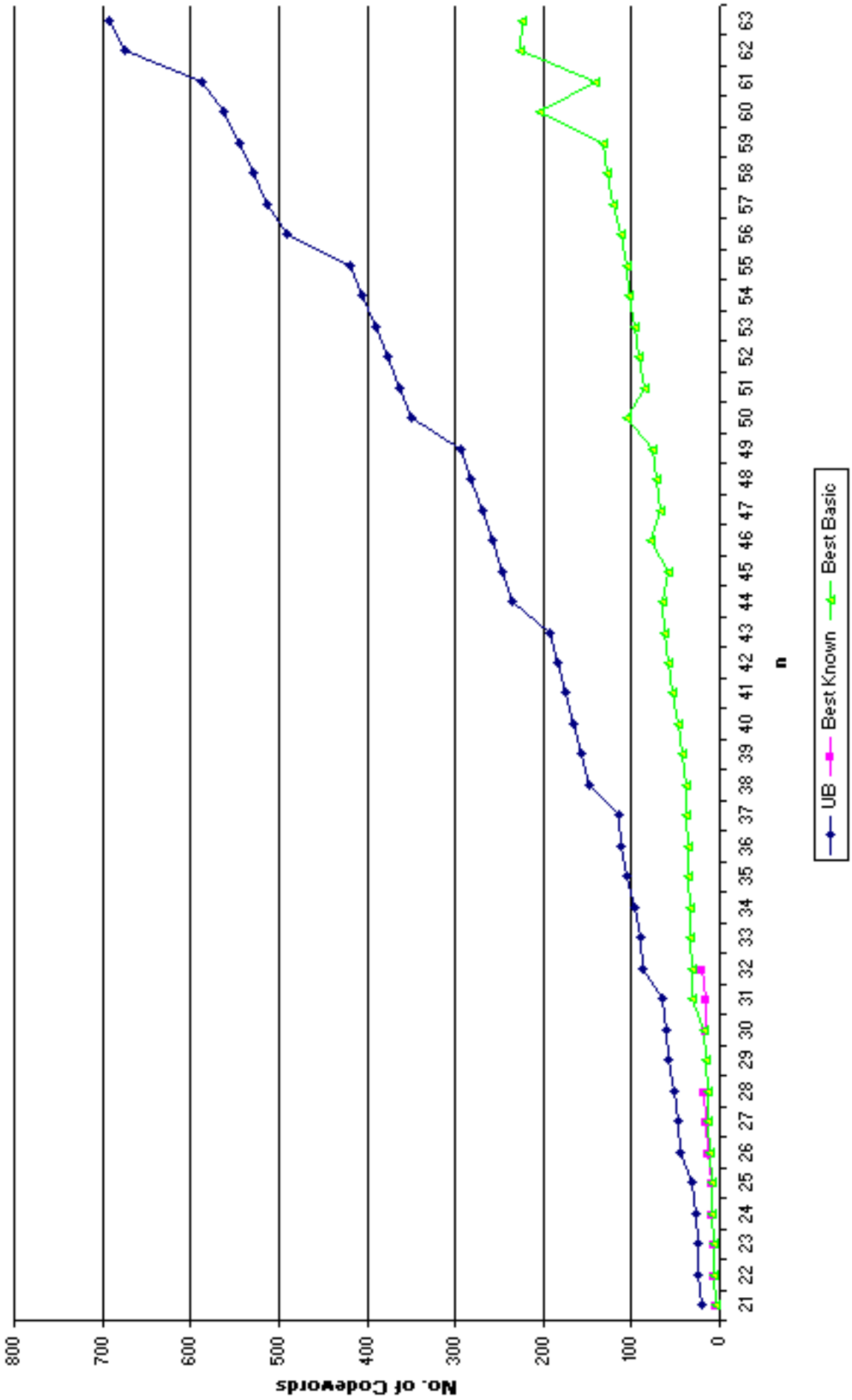

Figure 30: Comparison of the Upper Bound, Best Basic and the Best Known $d = 12, w = 8$

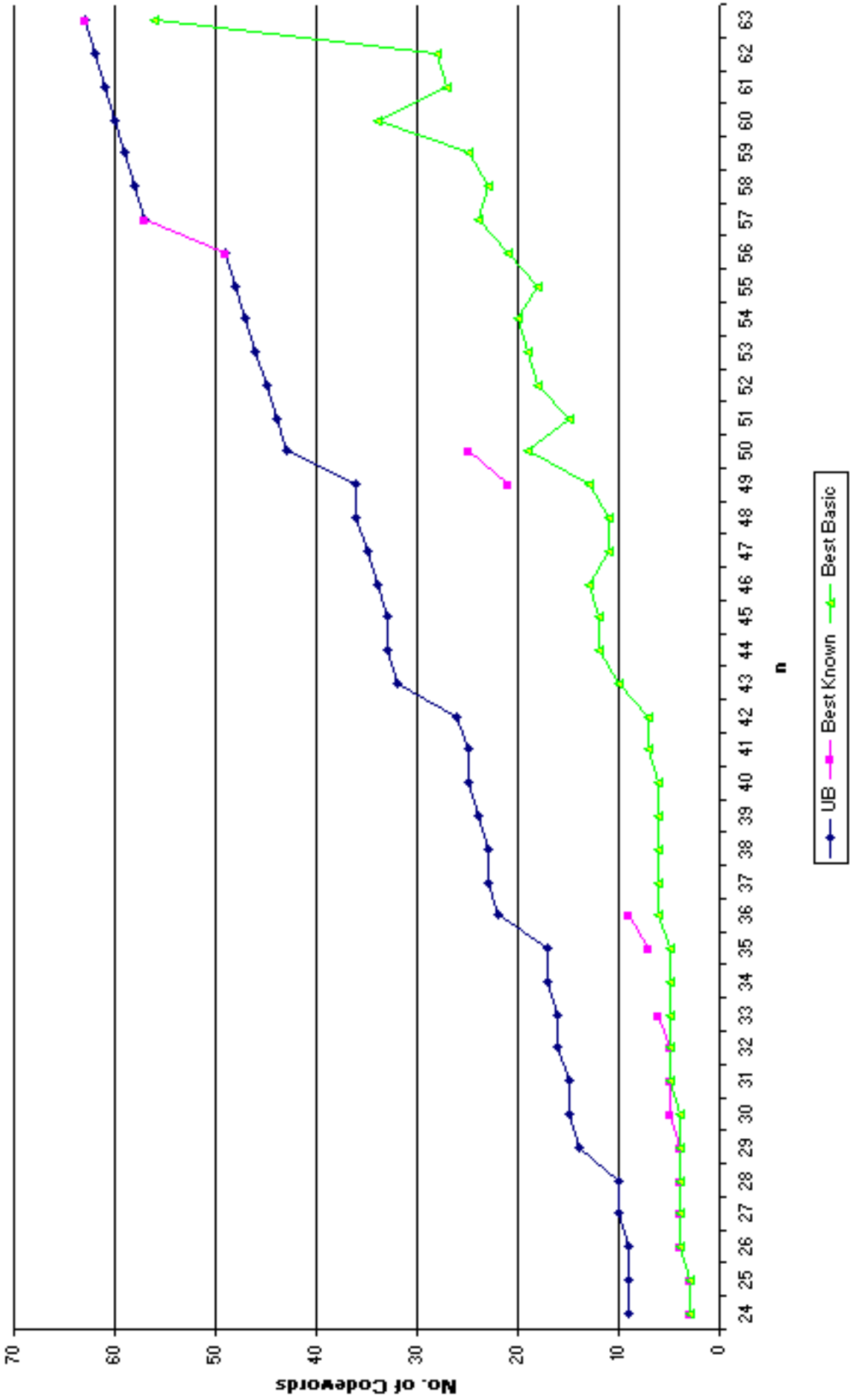

Figure 31: Comparison of the Upper Bound, Best Basic and the Best Known $d = 14, w = 8$

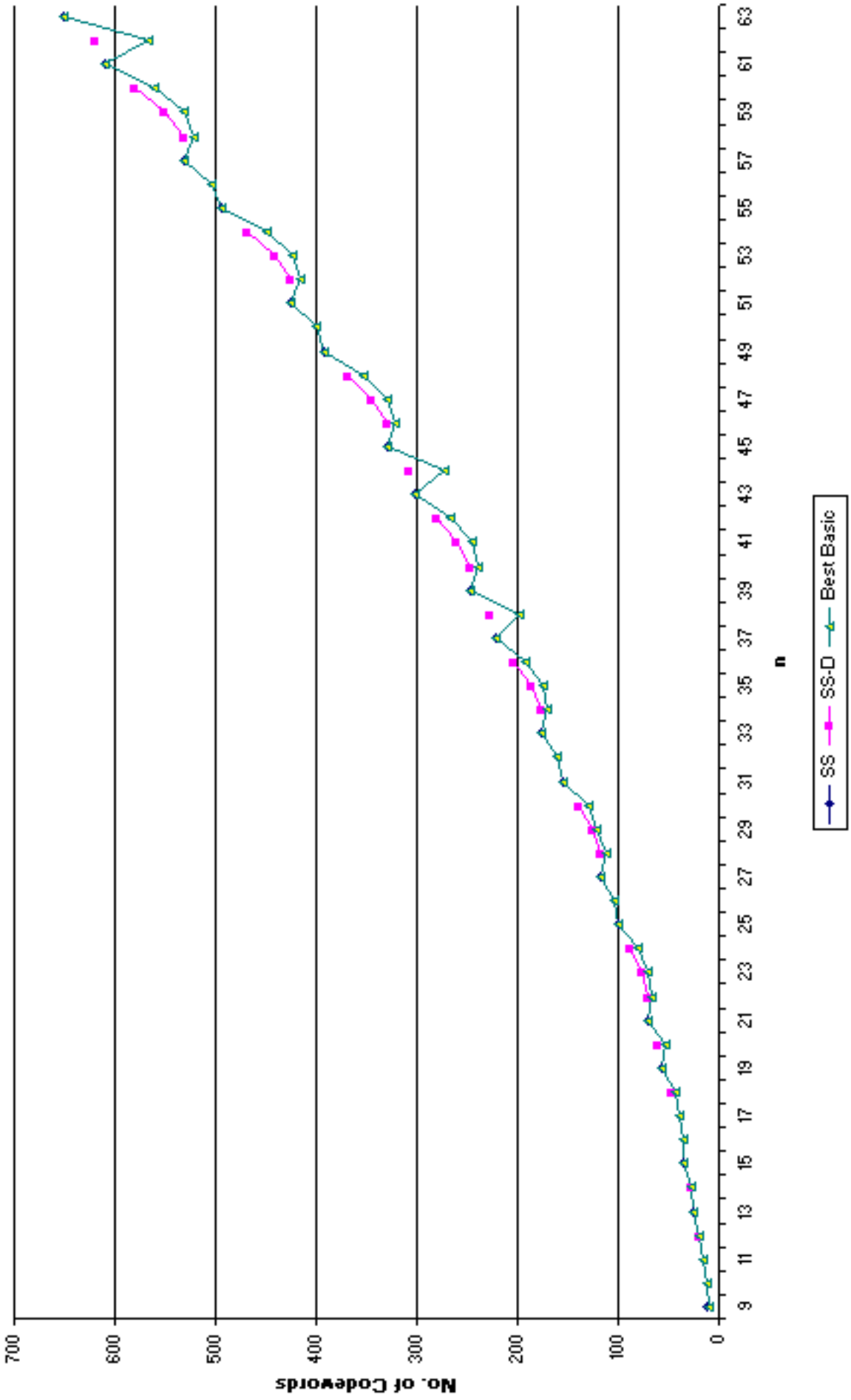

Figure 32: Comparison of Steiner System codes, Steiner System Derived and the Best Basic $d = 4, w = 3$

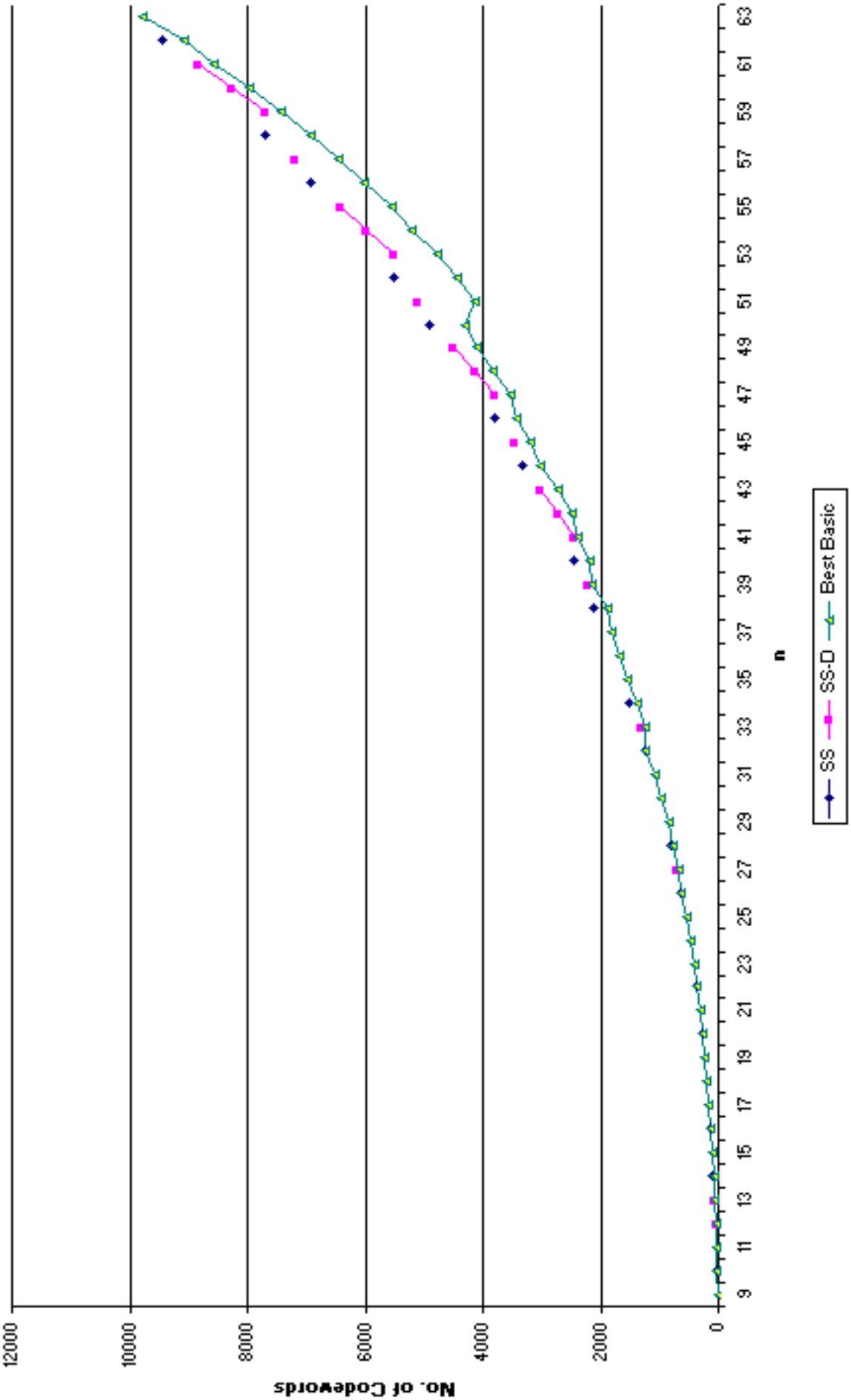

Figure 33: Comparison of Steiner System, Steiner System Derived and the Best Basic $d = 4, w = 4$

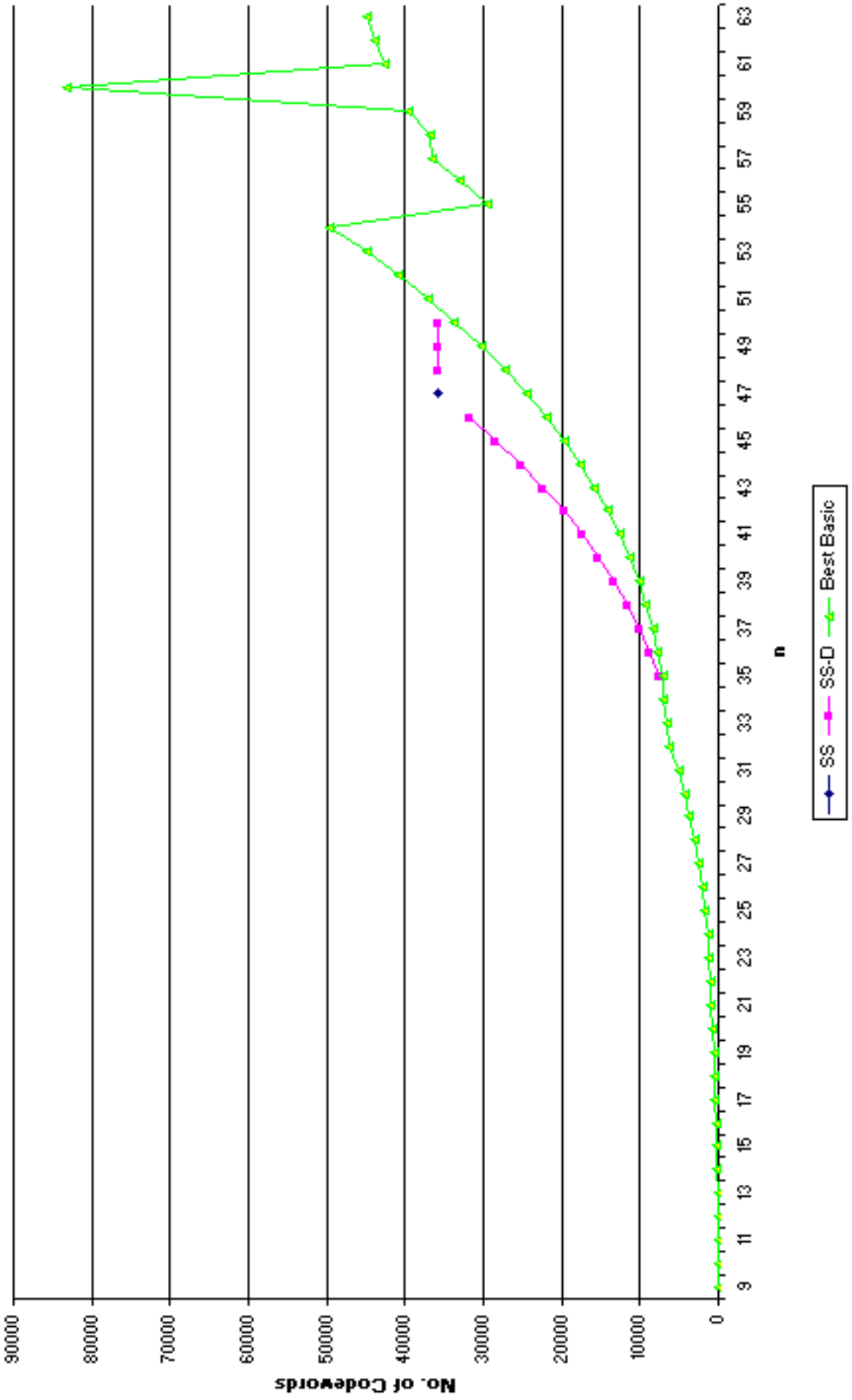

Figure 34: Comparison of Steiner System, Steiner System Derived and the Best Basic $d = 4, w = 5$

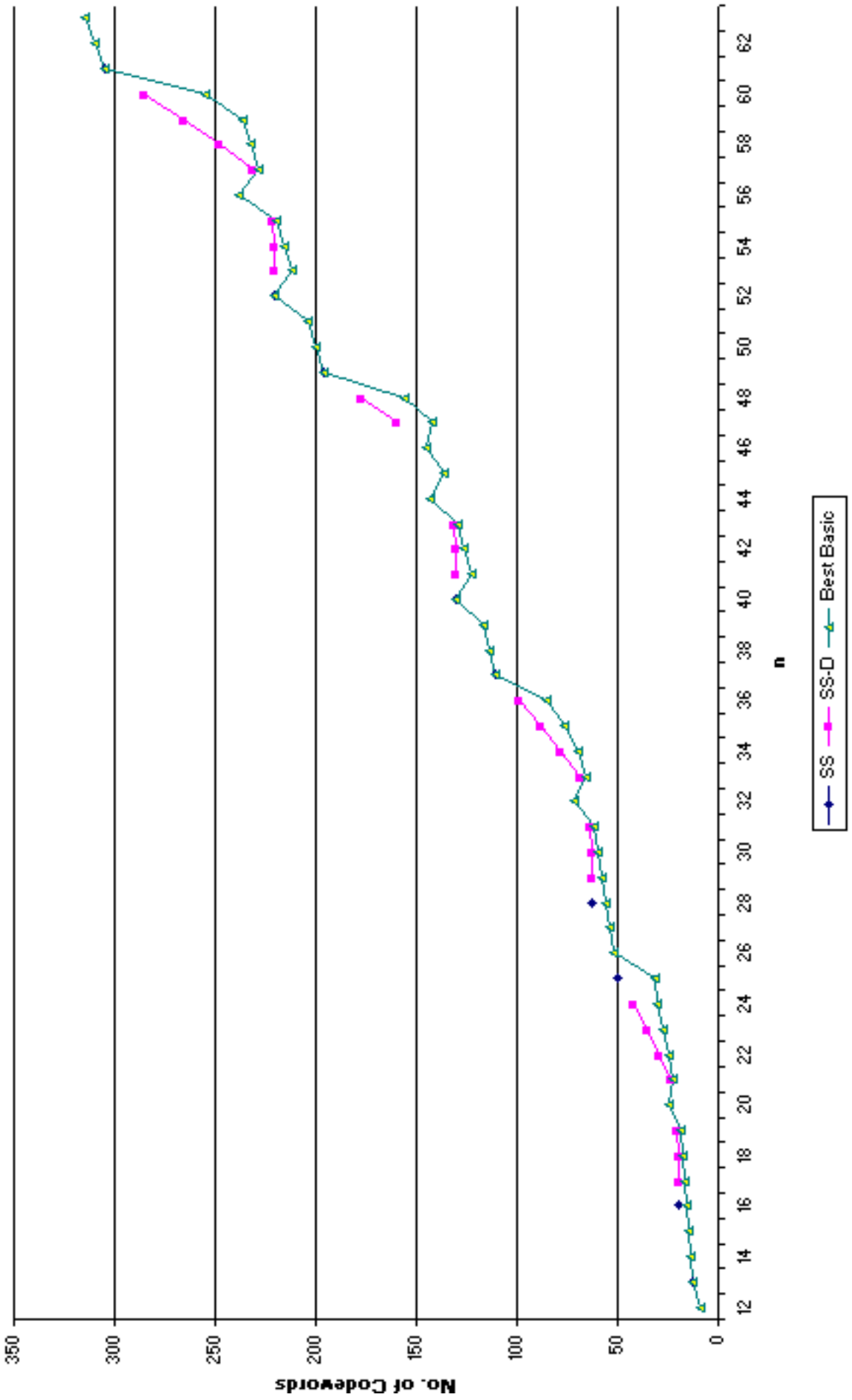

Figure 35: Comparison of Steiner System, Steiner System Derived and the Best Basic $d = 6, w = 4$

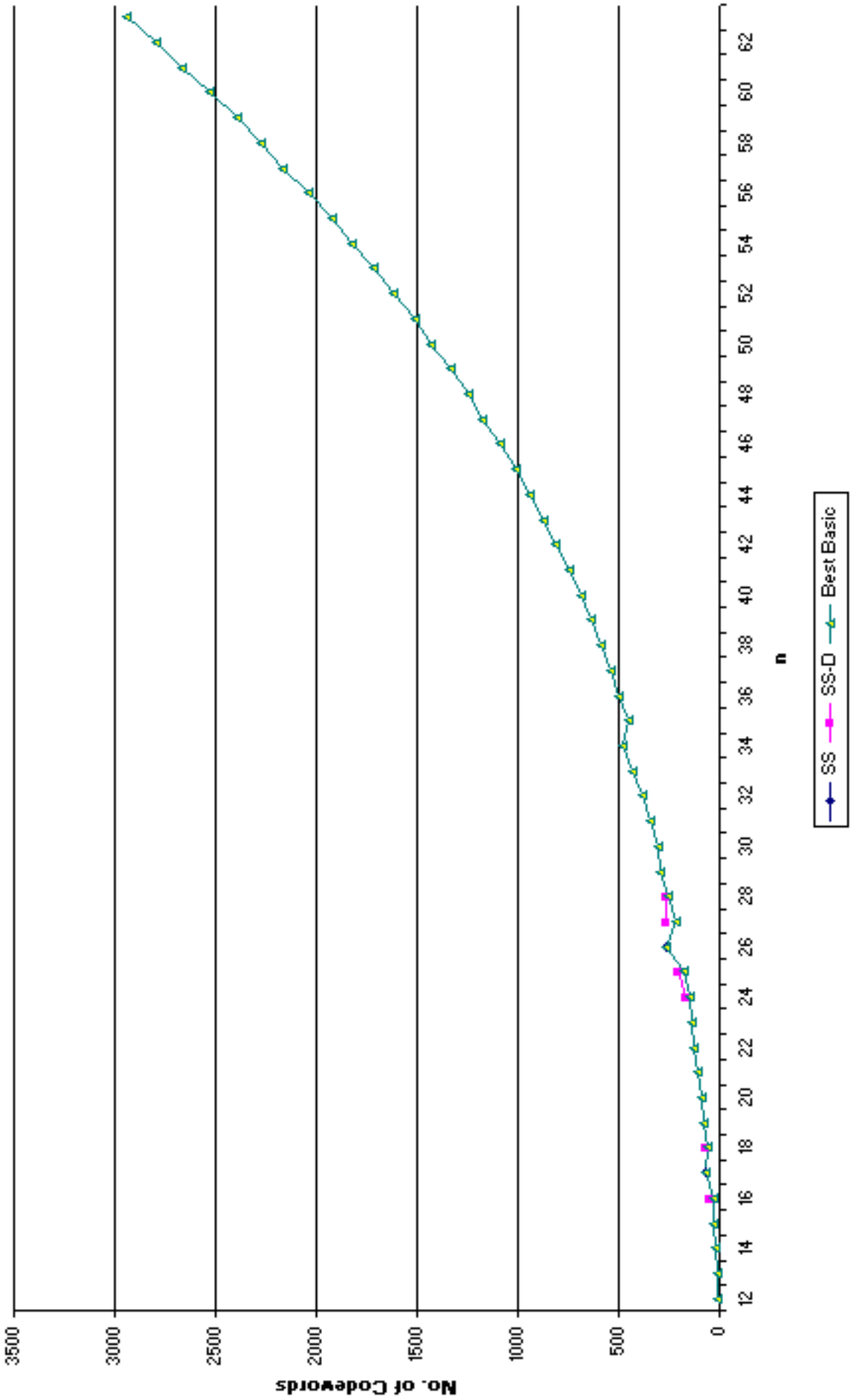

Figure 36: Comparison of Steiner System, Steiner System Derived and the Best Basic $d = 6, w = 5$

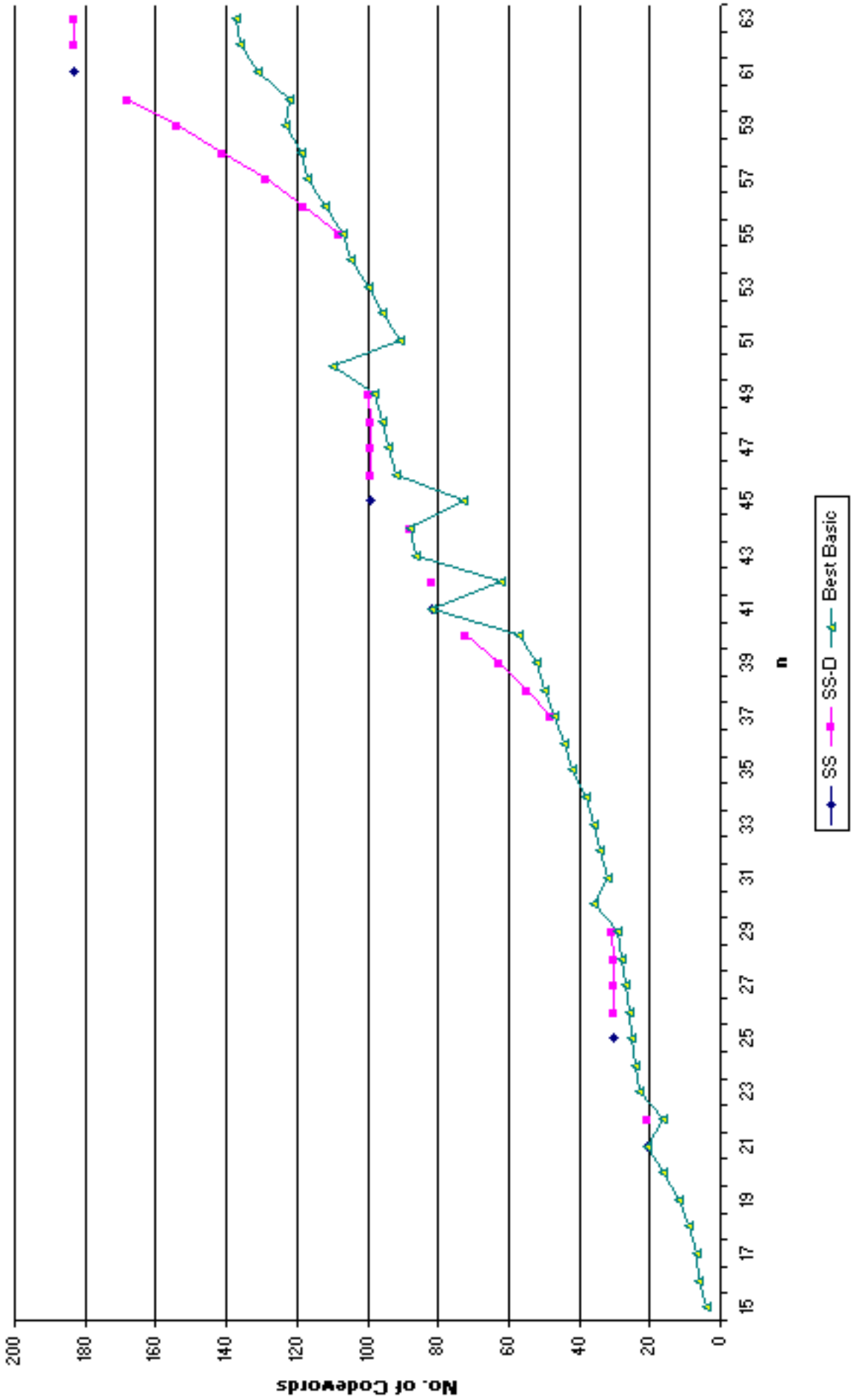

Figure 37: Comparison of Steiner System, Steiner System Derived and the Best Basic $d = 8, w = 5$

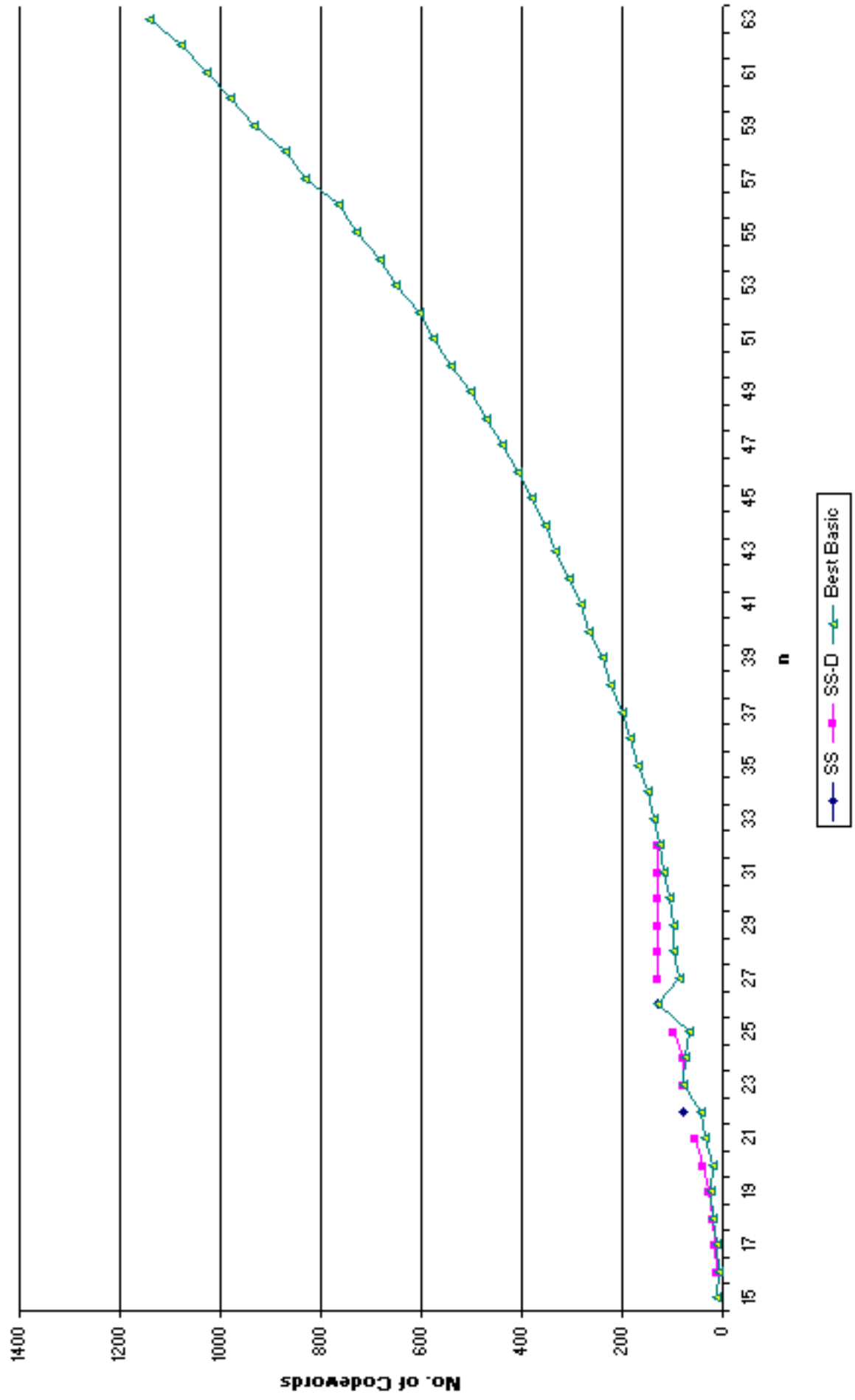

Figure 38: Comparison of Steiner System, Steiner System Derived and the Best Basic $d = 8, w = 6$

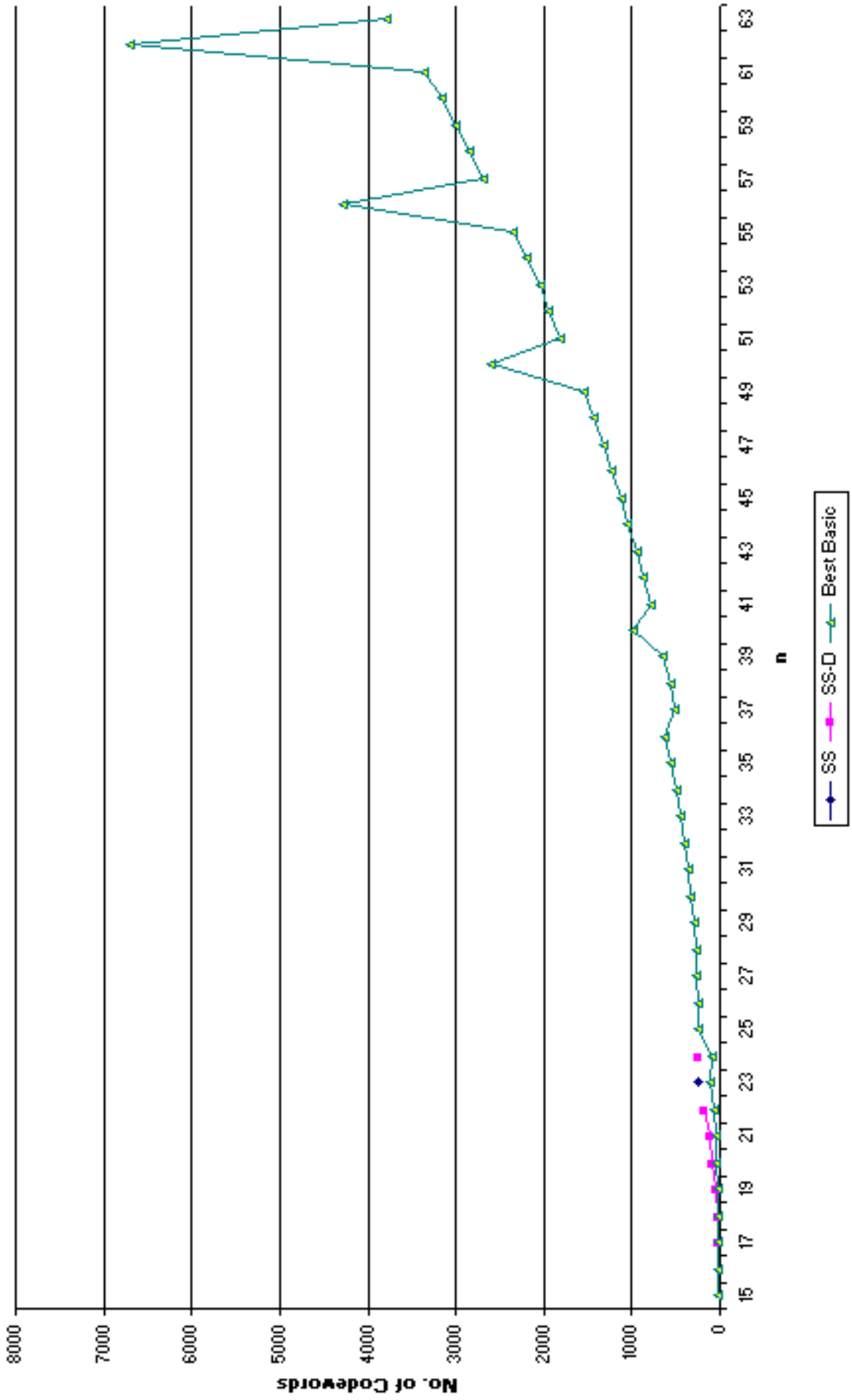

Figure 39: Comparison of Steiner System, Steiner System Derived and the Best Basic $d = 8, w = 7$

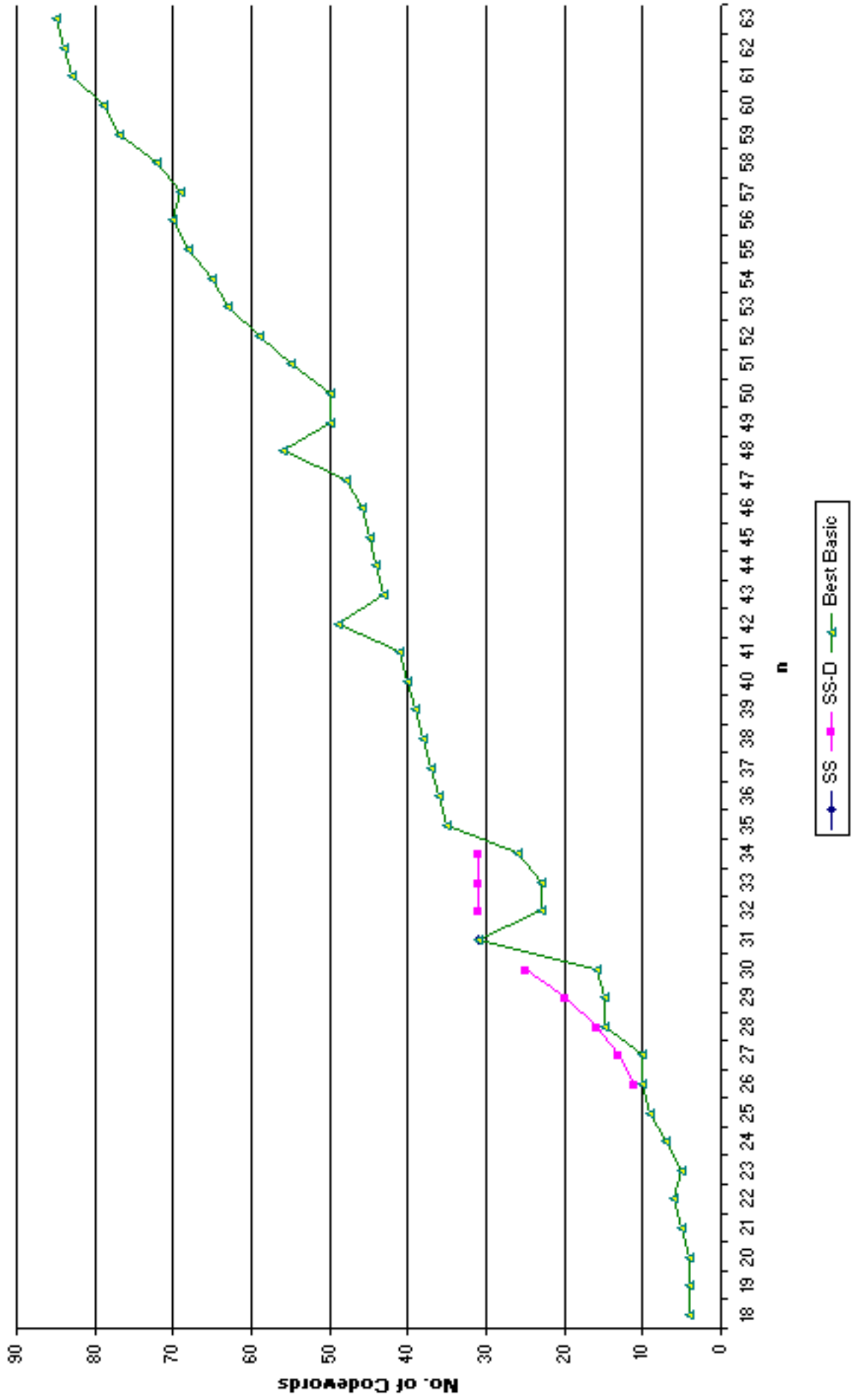

Figure 40: Comparison of Steiner System, Steiner System Derived and the Best Basic $d = 10, w = 6$

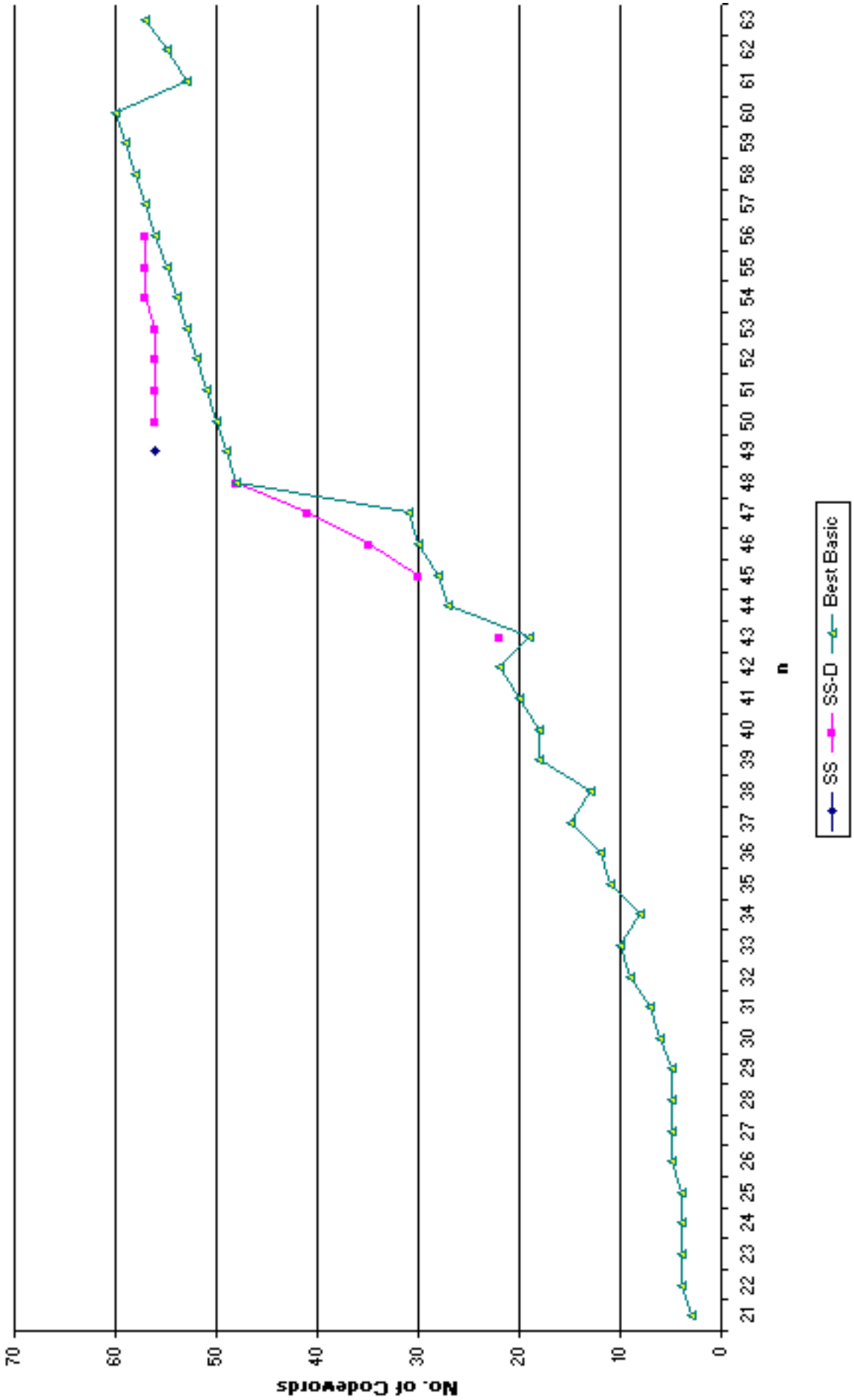

Figure 41: Comparison of Steiner System, Steiner System Derived and the Best Basic $d = 12, w = 7$

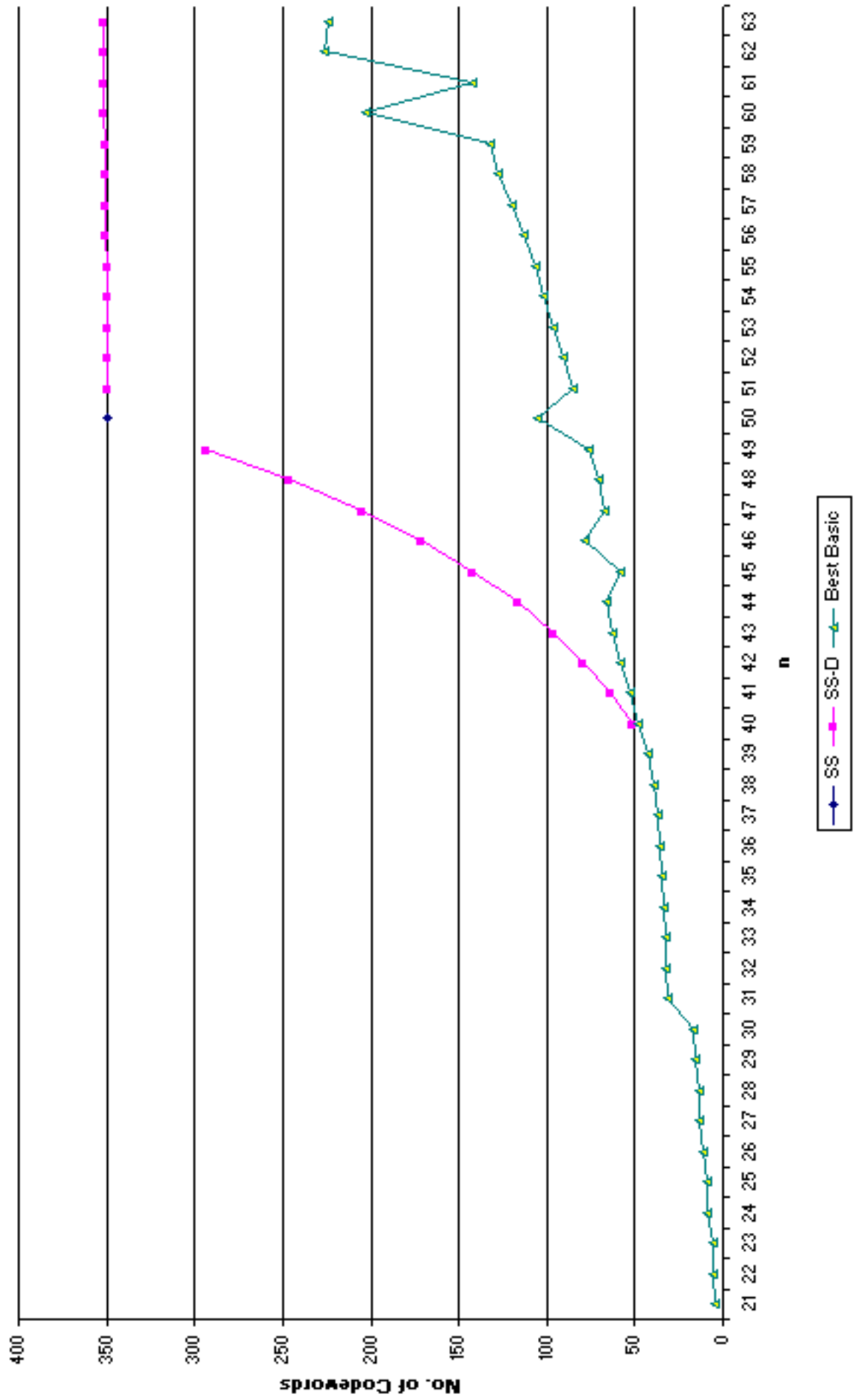

Figure 42: Comparison of Steiner System, Steiner System Derived and the Best Basic $d = 12, w = 8$

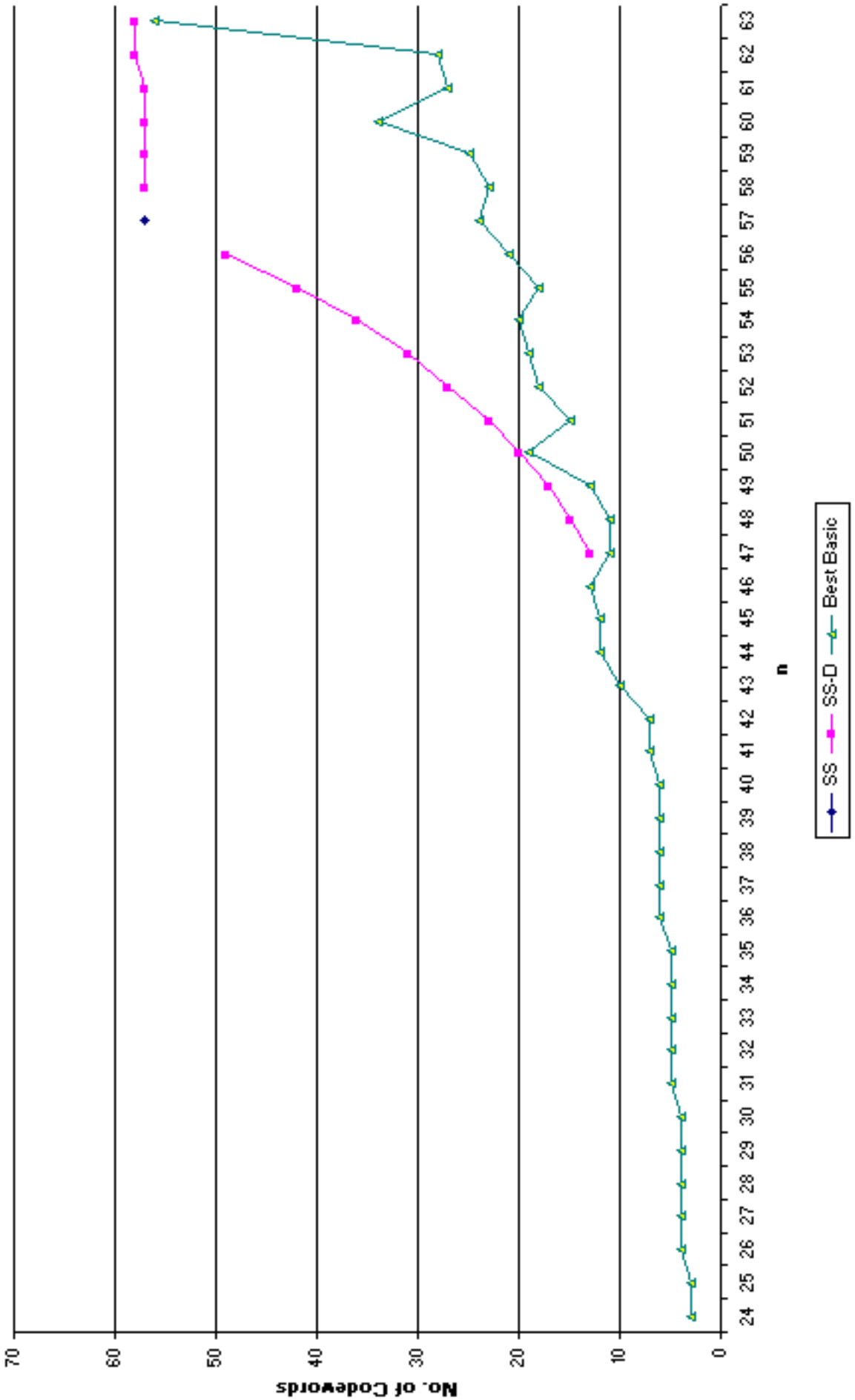

Figure 43: Comparison of Steiner System, Steiner System Derived and the Best Basic $d = 14, w = 8$

10.1 Commentary on results

The cyclic construction is clearly capable of finding the best results of the basic methods in many cases. However, even when the method does complete, the results can sometimes be poor. For example, it produces few codewords for $d = 10, w = 6$ (Table 19) with $n < 31$ and does not start to perform consistently until $n \geq 35$. Similarly when $d = 12, w = 7$ (Table 22) the method is not productive until $n \geq 48$ with the exception of some isolated cases ($n = 21, 28$ and 35). When $d = 14, w = 8$ (Table 24) the method finds codewords for $n = 24, 32$ and 48 but does not start to perform consistently at all within the specified range. Results from the tables show that in the overlap one cases mentioned above, the isolated cases are occurring only where $n \bmod w = 0$, the resulting number of codewords is n/w.

As w gets bigger it becomes more difficult to find a cyclic codeword which satisfies the properties with overlap one. However as n becomes larger the cyclic codewords become easier to find, resulting in more results for larger values of n. This explains the lack of codewords found for small values of n in the overlap one cases. The same is true to a lesser extent for overlaps two and three.

When the cyclic construction does produce codewords the results tend to be very good in comparison with the other basic methods, in some cases they are equal to the best known results and the upper bound. Examination of Table 10 ($d = 4, w = 3$) will show that with the exception of $n = 12$ the cyclic method equals the upper bound at all values of n for which a Steiner system exists. In other cases the method frequently reaches the upper bound and best known results regardless of the existence of Steiner systems. Some of the best known results can be improved upon with the results of the cyclic construction. For example, in Table 14 for $27 < n < 35$ the cyclic construction significantly improves upon the best known results, when $n = 35$ the improvement is an additional 142 codewords.

Binary lexicographic search provides a more complete set of results than the cyclic construction. With the parameters defined here the results of this method are limited only by time constraints. Binary lexicographic search does not produce results as favourable as those produced by the cyclic construction. However in cases where the cyclic construction has failed to produce a result binary lexicographic search has produced good results which in some cases overtake the best known results. For example, examination of Table 14 ($d = 6, w = 5$) will show that for $31 < n < 62$ the binary lexicographic method consistently improves upon the best known results.

As expected the random search results fall a little behind the binary lexicographic

search results. Similarly, the same is true of the non binary/mixed search. However in both cases the run time is favourable, and both methods are able to provide adequate results in cases where the cyclic construction and binary lexicographic search take too long to compute.

Figures 2 to 16 compare the results from the basic methods. These graphs show that in general, where the cyclic construction produces results, the method produces the best results of the basic methods. However the method is also the most erratic. For example, Figure 8 shows that the cyclic method equals the best known results in several places, however there are also many results which fall below the binary lexicographic search. The nature of the cyclic construction can result in significant differences in the number of codewords found in neighbouring values of n. In particular in Figure 11 it can be seen that the number of codewords at $n = 34$ is 0, while the number of codewords at $n = 35$ is 35. Such a difference in neighbouring results can be explained by the effect of finding one new codeword with the cyclic construction. For every additional codeword found with this method, each distinct cycle of the codeword is also added. So in the case mentioned above, when a codeword was found for $n = 35$, this resulted in 35 distinct cycles of the codeword being added.

Figures 17 to 31 show the comparison of the best results from the basic methods, the best known results and the upper bound. For the first four cases, Figures 17 to 20 there is little difference between the upper bound and the best known results. The best basic results meet the upper bound on several occasions and remain reasonably close in the other cases. Examination of Figure 19 will show that the best basic results produce some spikes in the data. This is where the random search has been used instead of the binary lexicographic search, but at $n = 60$ the binary lexicographic search result has been included resulting in a spike. If the binary lexicographic search were used to produce all these results it is expected that a smooth line would join up the points where $n = 54$ and $n = 60$. If this were the case then clearly the best basic results would lie much closer to the upper bound. From Figure 24 ($d = 8, w = 6$) onwards it can be seen that the best basic results are beginning to improve consistently upon the best known results.

Figures 32 to 43 illustrate the results for Steiner system codes, codes derived from Steiner systems and the best basic results. The Steiner system results and other results derived from them are included where they improve upon the best basic results. The figures show that where the results for the best basic method produce a "dip" in the graph the Steiner system results in some cases are able to fill in this gap, which results in a smoother graph.

The choice of which method to use is largely dependent on the run time available.

The cyclic construction is the best method to use for smaller cases, i.e. small values of d, w and n. Binary lexicographic search provides a more complete and consistent set of results. The fastest results, but with only reasonable quality can be obtained by the non binary/mixed search. Random search is also useful when binary lexicographic search is too slow.

11 Conclusion

This report has successfully addressed four aspects of the problem of constructing constant weight codes and applying them to frequency hopping lists in radio networks:

1. Identifying the requirements for frequency hopping lists in radio systems, and particularly in GSM mobile telephone networks. For other networks where search techniques will be impractical, the work of section 6 may ultimately prove particularly useful.

2. Describing a number of basic search techniques which are easily accessible to the Radio Engineer and evaluating them for the parameters relevant to GSM networks. The techniques appear to be very successful for small w, possibly less successful for larger w.

3. Developing theoretical methods of generating constant weight codes from non-binary and mixed codes. In combination with lexicographic search the techniques give very fast (but not especially strong) results for GSM relevant parameters. Certain codes meeting the Griesmer bound have been shown to give constant weight codes which are asymptotically optimal. These codes may prove particularly useful for radio systems with larger numbers of frequencies and larger required hopping list lengths.

4. Identifying cases where Steiner systems lead to particular improvements on the best of the basic methods. Relatively simple constructions are given in most cases. The number of cases where the improvement is very significant is small. Indeed the result $A(50, 12, 8) = 350$ which originally motivated this work has been shown to be the most significant example.

Ada code for some of the search techniques described is included as an appendix to this report.

It can be noted here that the basic cyclic method could easily be complemented by a basic method for *extended cyclic codes* [7], which consist of cycles of length $n-1$ and a single fixed point. Time restrictions prevented this extension from being considered in the project. Other codes obtained from permutation groups are described in [7].

Following the conclusion of this report, good or reasonably good hopping lists can be constructed for all relevant cases. The most direct work to pursue next is the evaluation of hopping list GSM assignments obtained by assigning the lists obtained from constant weight codes. These need to be compared with other techniques, such as that described in [3]. The use of these hopping lists in other radio systems should be compared with random hopping over all frequencies.

For application to GSM networks, one modification of the approach developed here might prove useful. When not hopping the frequencies assigned to two transmitters in the same cell normally require a minimum separation (typically 3 channels). It might be beneficial to consider constant weight codes with a built in "internal separation requirement". Thus codewords could not contain substrings $\ldots 11 \ldots$, or $\ldots 101 \ldots$. Some preliminary experiments have confirmed that simply deleting invalid codewords from the constant weight codes generated here is not a good technique. These experiments have also suggested that the maximum number of codewords attainable divided by $A(n, d, w)$ ranges from 0.75 to 1.0.

References

[1] E. Agrell, A. Vardy and K. Zeger, Upper Bounds for Constant-Weight Codes, IEEE Trans. Inform. Theory, vol. 46, no. 7, Nov. 2000, pp.2373–2395.

[2] S.M. Allen, D.H. Smith and S. Hurley, Lower bounding techniques for frequency assignment, Discrete Math. Vol. 197/198, (1999), pp. 41–52.

[3] P. Björklund, P. Värbrand and D. Yuan, Optimal GSM network planning with frequency hopping, Fifth INFORMS Telecommunications Conference, Boca Raton, Florida, U.S.A., March 5–8, 2000

[4] G.T. Bogdanova, A.E. Brouwer, S.N. Kapralov and P.R.J. Östergard , Error-correcting codes over an alphabet of four elements, Des. Codes Cryptogr. Vol. 23 (2001), pp. 333-342.

[5] G.T. Bogdanova and P.R.J. Östergard, Bounds on codes over an alphabet of five elements, Discrete Math. Vol. 240 (2001), pp. 12-19.

[6] A. E. Brouwer, H. O. Hamalainen, P. R. J. Östergard and N. J. A. Sloane, Bounds on Mixed Binary/Ternary Codes, IEEE Trans. Information Theory, Vol. 44 (1998), pp. 140-161.

[7] A.E. Brouwer, J.B. Shearer, N.J.A. Sloane and W.D. Smith, A New Table of Constant Weight Codes IEEE Trans. Inform. Theory, vol. 36, no. 6, pp. 1334–1380, Nov. 1990.

[8] R. Carraghan and P. Pardalos, An exact algorithm for the maximum clique problem, Operations Research Letters, 9(1990) pp. 375–382.

[9] C.J. Colbourn and J.H. Dinitz, The CRC handbook of combinatorial designs, Boca Raton, Florida, CRC Press, 1996.

[10] J.H. Conway and N.J.A. Sloane, Lexicographic Codes; Error-Correcting Codes from Game Theory, IEEE Trans. Inform. Theory, vol. 32, pp.337–348, May 1986.

[11] COST231 Final Report, `http://www.lx.it.pt/cost231/final_report.htm`

[12] P. Dembowski, Finite geometries, Berlin, Springer-Verlag, 1968.

[13] S.M. Dodunekov, Minimum block length of a linear q-ary code with specified dimension and code distance, Problems of Info. Transmission, Vol. 20, No. 4 (1984) pp. 239–249.

[14] N. Q. A. L. Györfi and J.L. Massey, Constructions of binary constant-weight cyclic codes and cyclically permutable codes, IEEE Transactions on Information Theory Vol. 38 (1992), pp. 940-949.

[15] W.K. Hale, New spectrum management tools, *in* Proc. IEEE International Symposium on Electromagnetic Compatibility, IEEE Press, Piscataway, N.J., (1981), pp. 47–53.

[16] H. Hanani, The existence and construction of balanced incomplete block designs, Ann. Math. Statist. 32 (1961), pp. 361–386.

[17] S. Hurley, D.H. Smith and S.U. Thiel, FASoft: A system for discrete channel frequency assignment, Radio Science, Vol. 32, No. 5, (1997), pp. 1921–1939.

[18] T.A. Lanfear, Graph theory and radio frequency assignment, NATO EMC Analysis Project No. 5, NATO Headquarters, 1110 Brussels, 1989.

[19] R. Leese and S. Hurley, Methods and Algorithms for Radio Channel Assignment, Oxford University Press, to appear.

[20] F.J. MacWilliams and N.J.A. Sloane, The theory of error-correcting codes, Amsterdam, Elsevier, 1996, ninth edition.

[21] B.H. Metzger, Spectrum management technique, *presented at* 38th National ORSA meeting, Detroit, MI, Fall 1970.

[22] R. Montemanni, D.H. Smith and S.M. Allen, Lower bounds for fixed spectrum frequency assignment, Annals of Operations Research, to appear.

[23] S. Perkins, A.L. Sakhnovich, and D.H. Smith, Mixed Codes: Bounds, Constructions and Some Applications, submitted for publication.

[24] O. Pretzel, Error Correcting Codes and Finite Fields, Oxford, Clarendon Press, 1992.

[25] P. R. J. Östergard and M. K. Kaikkonen, New single-error-correcting codes, IEEE Trans. Inform. Theory, Vol. 42, (1996), pp. 1261–1262.

[26] F.S. Roberts, T-colorings of graphs: recent results and open problems, Discrete Mathematics, Vol. 93, (1991), 229–245.

[27] J. Samuelsson, Planning frequency hopping networks using next generation automatic frequency planning tools, Powerpoint presentation, Comopt AB, Michael Löfmans Gata 6, SE-254 38 Helsingborg, Sweden, April 1999.

[28] D.H. Smith and S. Hurley, Bounds for the frequency assignment problem, Discrete Math., 167/168, (1997), pp. 571–582.

[29] D.H. Smith, S.M. Allen, S. Hurley, W.J. Watkins, Frequency Assignment: Methods and Algorithms, Proceedings of the NATO RTA SET/ISET Symposium on Frequency Assignment, Sharing and Conservation in Systems (Aerospace), Aalborg, Denmark, October 1998, NATO RTO-MP-13, (1999), pp. K-1 – K-18.

[30] D.H. Smith, S. Hurley and S.M. Allen, A new lower bound for the channel assignment problem, IEEE Trans. Veh. Technol., Vol. VT–49, No. 4, (2000), pp. 1265–1272.

[31] D.H. Smith, S. Hurley and S.U. Thiel, Improving heuristics for the frequency assignment problem, European Journal of Operational Research, Vol. 107, (1998), pp. 76–86.

[32] G. Solomon and J.J. Stiffler, Algebraically punctured cyclic codes, Inform. Contr., Vol. 8 (1965), pp. 170–179.

[33] Table of Constant Weight Binary Codes, http://www.research.att.com/~njas/codes/Andw

[34] Table of Bounds for Constant Weight Binary Codes, http://www.s2.chalmers.se/~agrell/bounds/cw.html

[35] T. Toftegård Nielsen, J. Wigard, P.H. Michaelsen and P. Mogensen, Slow frequency hopping solutions for GSM networks of small bandwidth, 48th IEEE Vehicular Technology Conference, 1998, 1321–1325

[36] G.J.M. van Wee, Bounds on packings and coverings by spheres in q-ary and mixed Hamming spaces, J. Combin. Theory, Ser. A, Vol. 57 (1991), pp. 117–129.

[37] G.J.M. van Wee, On the non-existence of certain perfect mixed codes, Discrete Math. Vol. 87 (1991), pp. 323–326.

[38] S.B. Wicker and V.K. Bhargava, Reed-Solomon codes and their applications, IEEE Press, New York, 1994.

[39] J.A. Zoellner and C.L. Beall, A breakthrough in spectrum conserving frequency assignment technology, IEEE Trans. Electromagnetic Compatibility, EMC-19, 3, (1977), pp. 313–319.

12 Appendix

In this appendix some Ada code for some of the basic methods is presented, largely without description. Ada is selected here for its readability and portability. This software has run successfully using the Gnat Ada95 compiler.

12.1 Code for binary lexicographic search

This is the Ada program *Kxdemo5* that generated the results in Table 1. It is set up for (28,10,7) to give two A-values with a_1 random and going through the vectors in simple lexicographic order.

```
with text_io;                           use text_io;
with ada.long_integer_text_io;          use ada.long_integer_text_io;
with ada.integer_text_io;               use ada.integer_text_io;
with ada.float_text_io;                 use ada.float_text_io;
with ada.short_short_integer_text_io;   use ada.short_short_integer_text_io;
with Ada.Numerics.Discrete_Random;
with Calendar;                          use Calendar;


procedure Kxdemo5 is


                    n :  constant short_short_integer  := 28 ;
                    d :  constant short_short_integer  := 10 ;
                    w :  constant short_short_integer  :=  7 ;

   no_of_random_starts :  constant integer             :=  2 ;

             overlaps :  constant short_short_integer  := w-d/2;

 subtype n_values is short_short_integer range 1 .. short_short_integer(n);
 package Random_n_values is new Ada.Numerics.Discrete_Random(n_values);
 use Random_n_values;
 G : Generator;


 type ONE_D is array(1..w)   of short_short_integer;
 type TWO_D is array(1..4680) of ONE_D;
 B : ONE_D;
 A : TWO_D;


   iii : long_integer;
     i : integer;
  flag : boolean;
 count : short_short_integer;
 start,endtime:float;


 function Possible_word(n,w :in short_short_integer; G:in generator) return ONE_D is
```

```
                A  : ONE_D;
               ok  : boolean;
              i,x : short_short_integer;
begin
  A(1):=random(G);  i:=1;
  loop exit when i=w;
       ok:=false;
       loop exit when ok=true;
             x:=random(G); ok:=true;
             for j in 1..i loop
                   if A(j)=x then ok:=false; exit; end if;
             end loop;
       end loop;
      i:=i+1; A(i):=x;
  end loop;
  return A;
end possible_word;



begin -- main program

start:=float(seconds(clock));
put("n= ");put(n,4);     put("  d= ");put(d,3);     put("   w= ");put(w,3);
new_line(2);put("               overlaps= ");put(overlaps,2);new_line;
put("number of random starts= ");put(no_of_random_starts,3);new_line(2);
put("             list of vectors              ");
put("vector no.  no.of vectors tried  time");new_line;

-- could put, say   count:=random(G); count:=random(G);  ...   to vary the start


for random_start in 1..no_of_random_starts loop
   a(1):=Possible_Word(n,w,G);  i:=1;
   put("first a ## "); for ip in 1..w loop put(a(1)(ip),4); end loop;
   iii:=0;

   for p1 in 1 .. (n-w+1)  loop
    for p2 in p1+1..(n-w+2) loop
     for p3 in p2+1..(n-w+3) loop
      for p4 in p3+1..(n-w+4) loop
       for p5 in p4+1..(n-w+5)  loop
        for p6 in p5+1..(n-w+6)  loop
         for p7 in p6+1..(n-w+7)  loop
           --for p8 in p7+1..(n-w+8)  loop
            --for p9 in p8+1..(n-w+9)  loop
             --for p10 in p9+1..(n-w+10)  loop
              --for p11 in p10+1..(n-w+11)  loop
               --for p12 in p11+1..(n-w+12)  loop

                  iii:=iii+1;
                  b:=(p1,p2,p3,p4,p5,p6,p7);

                  flag:=false;
                  for k in reverse 1 .. i loop
                       count:=0;
                       for L in 1 .. w loop
                          for M in 1 .. w loop
                             if a(k)(L)=b(M) then count:=count+1; exit; end if;
                          end loop; --M
                          if count=overlaps+1 then flag:=true ; exit; end if;
                       end loop; -- L
```

```
                    if flag=true then exit; end if;
                end loop; --k

                if flag=false then
                    i:=i+1; a(i):=b;          new_line; put("b i j ##### ");
                    for ip in 1..w loop put(a(i)(ip),4); end loop;
                    put("     ");put(i,4);put("  ");put(iii,8);put(" ########");
                    put((1.0/60.0)*(float(seconds(clock))-start),6,2,0);
                end if;

              --end loop;--p12
             --end loop;--p11
            --end loop;--p10
           --end loop;--p9
          --end loop;--p8
         end loop;--p7
        end loop;--p6
       end loop;--p5
      end loop;--p4
    end loop;--p3
   end loop;--p2
  end loop;--p1

  new_line(2);put("first a :");for i in 1..w loop put(a(1)(i),4); end loop;
  new_line;put("loop no = ");put(random_start);
  endtime:=float(seconds(clock));new_line;put("time was(in mins):");
  put((1.0/60.0)*(endtime-start),8,2,0);new_line;put("overlaps= ");put(overlaps,2);
  put("                 n= ");put(n,3);put("  d= ");put(d,2);
  put("  w= ");put(w,2); put("   A estimate= ");put(i,3); new_line(4);

end loop; -- randomstart

end kxdemo5;
```

This is variation of the previous program *kxdemo*, set up for (28,6,7) to give two A-values with $a_1, \ldots, a_4$ random and then simple lexicographic order.

```
with text_io;                          use text_io;
with ada.long_integer_text_io;         use ada.long_integer_text_io;
with ada.integer_text_io;              use ada.integer_text_io;
with ada.float_text_io;                use ada.float_text_io;
with ada.short_short_integer_text_io;  use ada.short_short_integer_text_io;
with Ada.Numerics.Discrete_Random;
with Calendar;                         use Calendar;

procedure a67 is
        n :  constant short_short_integer  := 28 ;
        d :  constant short_short_integer  :=  6 ;
        w :  constant short_short_integer  :=  7 ;
      lbA :  constant integer := 10 +       4680 ;  -- an approx. lower bound for A(n,d,w)

 olaps :  constant short_short_integer  := w-d/2;
 subtype nvals is short_short_integer range 1 .. n;
 package Rand_n is new Ada.Numerics.Discrete_Random (nvals); use Rand_n; G : Generator;

 type ONE_D is array(1..w)   of short_short_integer;
 type TWO_D is array(1..lbA) of ONE_D;
 type Aarray is array(1..lbA) of integer;
```

```
 Avals:Aarray;
 b : ONE_D;  -- a possible vector
 a : TWO_D;  -- the vectors with the correct property
 iii : long_integer; -- counts the number of vectors tried
 i,Amax,Amaxloop: integer; --i ends up as an estimate of A(n,d,w),Amax is max of several i-vals
 flag : boolean;
 count : short_short_integer;
 start,endtime:float;
 olaps_plus1:short_short_integer:=olaps+1;

 function Possible_word(n,w:in short_short_integer;G:in generator) return ONE_D is
    A  : ONE_D;  ok  : boolean;  i,x  : short_short_integer;
 begin --Reset (G);????
   A(1):=random(G);  i:=1;
   loop exit when i=w;
     ok:=false;
     loop exit when ok=true;
       x:=random(G); ok:=true;
       for j in 1..i loop
          if A(j)=x then ok:=false; exit; end if;
       end loop;
     end loop;
     i:=i+1; A(i):=x;
   end loop;
   return A;
 end possible_word;

begin
put("  n= ");put(n,4);put(" d= ");put(d,3);put(" w= ");put(w,3);put("  olaps= ");put(olaps,2);
new_line; count:=random(G);count:=random(G);
Amax:=0; start:=float(seconds(clock));
for i in 1..lbA loop Avals(i):=0; end loop;

for ggg in 1..2 loop
  a(1):=Possible_Word(n,w,G); i:=1;
  for j in 1..3 loop
   flag:=true;
   loop exit when flag=false;
     b:=Possible_word(n,w,G);

     for k in reverse 1 .. i loop
          count:=0;
          for L in 1 .. w loop
            for M in 1 .. w loop
              if a(k)(L)=b(M) then count:=count+1; exit; end if;
            end loop;
            if count=olaps_plus1 then  goto here; end if;
          end loop;
     end loop;
     flag:=false; i:=i+1; a(i):=b;
    <<here>> null;

  end loop; -- loop exit when
 end loop; -- j

 iii:=0;flag:=false;
 for p1 in 1 .. (n-w+1)  loop
  for p2 in p1+1..(n-w+2) loop
   for p3 in p2+1..(n-w+3) loop
    for p4 in p3+1..(n-w+4) loop
     for p5 in p4+1..(n-w+5)  loop
      for p6 in p5+1..(n-w+6)  loop
```

```
      for p7 in p6+1..(n-w+7)  loop
       --for p8 in p7+1..(n-w+8)  loop
       -- for p9 in p8+1..(n-w+9)  loop
       --  for p10 in p9+1..(n-w+10)  loop
       --   for p11 in p10+1..(n-w+11)  loop
       --    for p12 in p11+1..(n-w+12)  loop

        iii:=iii+1; b:=(p1,p2,p3,p4,p5,p6,p7);

        for k in reverse 1 .. i loop
          count:=0;
          for L in 1 .. w loop
            for M in 1 .. w loop
              if a(k)(L)=b(M) then count:=count+1; exit; end if;
            end loop;
            if count=olaps_plus1 then  goto there; end if;
          end loop;
        end loop;

       i:=i+1; a(i):=b;
       --new_line;for ip in 1..w loop put(a(i)(ip),4);end loop;put(" ");put(i,4);put(" ");
       -- put(iii,8);put((1.0/60.0)*(float(seconds(clock))-start),8,2,0);

      <<there>> null;

      --      end loop;--p12
      --     end loop;--p11
      --    end loop;--p10
      --   end loop;--p9
      --  end loop;--p8
       end loop;--p7
      end loop;--p6
     end loop;--p5
    end loop;--p4
   end loop;--p3
  end loop;--p2
 end loop;--p1

 --new_line(1);for ik in 1..i loop for ij in 1..w loop put(a(ik)(j),3);end loop;end loop;
 --new_line;--put(" loopno=");put(ggg,3);--endtime:=float(seconds(clock));
 --put(" t(m)");put((1.0/60.0)*(endtime-start),5,2,0);--new_line;
 --put("olaps=");put(olaps,2);put(" n=");put(n,3);put(" d=");put(d,2);
 --put(" w=");put(w,2); --put(" Aest.=");put(i,3);

 if i>Amax then Amax:=i;Amaxloop:=ggg ;end if;
 Avals(i):=Avals(i)+1;

end loop;--ggg

new_line;put("################       Amax=");put(Amax);
put("  in loop=");put(Amaxloop);new_line;
for i in lbA/2..lbA loop put(i,5); put(Avals(i),8); new_line; end loop;
endtime:=float(seconds(clock));
put(" t(m)");put((1.0/60.0)*(endtime-start),5,2,0);--new_line;

end a67;
```

This is variation of *A67*, set up for (14,6,7) to give the best of 1000 A-values, with a_1, a_2 random and then going through in the order starting with a '2' in first position

and ending with '1' in first position.

```
with text_io;                           use text_io;
with ada.long_integer_text_io;          use ada.long_integer_text_io;
with ada.integer_text_io;               use ada.integer_text_io;
with ada.float_text_io;                 use ada.float_text_io;
with ada.short_short_integer_text_io;   use ada.short_short_integer_text_io;
with Ada.Numerics.Discrete_Random;
with Calendar;                          use Calendar;

procedure f_n_6_7tt is
        n :  constant short_short_integer  := 14 ;
        d : constant short_short_integer   := 6 ;
        w :  constant short_short_integer  :=7 ;
       startp1:constant short_short_integer:=2;
 overlaps :  constant short_short_integer  := w-d/2;

      subtype n_values is short_short_integer range 1 .. short_short_integer(n);
      package Random_n_values is new Ada.Numerics.Discrete_Random (n_values);
      use Random_n_values;
      G : Generator;

 type ONE_D is array(1..w)   of short_short_integer;
 type TWO_D is array(1..4680) of ONE_D;
 type Aarray is array(1..4680) of integer;
 Avals:Aarray;
        B : ONE_D;
        A : TWO_D;

 --oldp1,oldp2,oldp3:short_short_integer;
      iii : long_integer;
        i,Amax,Amaxloop : integer;
     flag : boolean;
    count : short_short_integer;
start,endtime:float;
olaps_plus1:short_short_integer:=overlaps+1;

 function Possible_word(n,w:in short_short_integer;G:in generator) return ONE_D is --G : Generator; ??
                  A  : ONE_D;
                 ok  : boolean;interchange_made:boolean:=true;
             --   m, max_no_of_pairs : integer;
        i,x : short_short_integer;  --small,temp : short_short_integer;

 begin --Reset (G);????
    A(1):=random(G);  i:=1;
   loop exit when i=w;
     ok:=false;
     loop exit when ok=true;
       x:=random(G); ok:=true;
       for j in 1..i loop
          if A(j)=x then ok:=false; exit; end if;
       end loop;
     end loop;
     i:=i+1; A(i):=x;
   end loop;
   return A;
 end possible_word;

begin put("n= ");put(n,4);put("  d= ");put(d,3);
put("   w= ");put(w,3); new_line;new_line;
put("               overlaps= ");put(overlaps,2);new_line;
```

```
count:=random(G);count:=random(G);
Amax:=0;
start:=float(seconds(clock));
for i in 1..4680 loop Avals(i):=0; end loop;

 for ggg in 1..1000 loop
   a(1):=Possible_Word(n,w,G);i:=1;
   for j in 1..1 loop
flag:=true;
   loop exit when flag=false;

        b:=Possible_word(n,w,G);
        flag:=false;
      for k in 1 .. i loop
          count:=0;
          for L in 1 .. w loop
            for M in 1 .. w loop
             if a(k)(L)=b(M) then count:=count+1; exit; end if;
            end loop;
            if count=overlaps+1 then flag:=true ; exit; end if;
          end loop;
          if flag=true then exit; end if;
      end loop;
      if flag=false then
              i:=i+1; a(i):=b; --new_line; put("b i j         # ");
              --for ip in 1..w loop put(a(i)(ip),4); end loop;
              --put("     ");put(i,4);put("  ");put(j,8);
              --put((1.0/60.0)*(float(seconds(clock))-start),8,2,0);
      end if; --ii:=j;
  end loop;
  end loop;
--  --put(" time so far (mins) ");put((1.0/60.0)*(float(seconds(clock))-start),8,2,0);

----oldp1:=0; oldp2:=0; oldp3:=0;
iii:=0;flag:=false;
for p1 in startp1 .. (n-w+1)  loop
   --p1:=p1a mod 23 ;-- should be in ..n-w+1
for p2 in p1+1..(n-w+2) loop -- should be in ..n-w+2
for p3 in p2+1..(n-w+3) loop
for p4 in p3+1..(n-w+4) loop
for p5 in p4+1..(n-w+5)  loop
for p6 in p5+1..(n-w+6)  loop
for p7 in p6+1..(n-w+7)  loop
--for p8 in p7+1..(n-w+8)  loop

--for p9 in p8+1..(n-w+9)  loop
--for p10 in p9+1..(n-w+10)  loop
--for p11 in p10+1..(n-w+11)  loop
--for p12 in p11+1..(n-w+12)  loop

iii:=iii+1;
--if oldp1=p1 and oldp2=p2 and oldp3=p3 then iii:=iii+long_integer(50-p8); exit; end if;
b:=(p1,p2,p3,p4,p5,p6,p7);


      for k in reverse 1 .. i loop
          count:=0;
          for L in 1 .. w loop
            for M in 1 .. w loop
          --    if a(k)(L)<b(M) then exit; end if;  -- ##  al check
             if a(k)(L)=b(M) then count:=count+1; exit; end if;
            end loop;
```

```
            if count=olaps_plus1 then  goto there; end if;
          end loop;
          --if flag=true then exit; end if;
          --if flag=true then goto here; end if;

      end loop;
      --if flag=false then
              i:=i+1; a(i):=b; --new_line; put("b i j ######### ");
            --  for ip in 1..w loop put(a(i)(ip),4); end loop;
             --put("     ");put(i,4);put("  ");put(iii,8);put(" ####");
             -- put((1.0/60.0)*(float(seconds(clock))-start),6,2,0);
     --         oldp1:=p1; oldp2:=p2; oldp3:=p3;
     -- end if;
    <<there>> null;
--end loop;--p12
--end loop;--p11
--end loop;--p10
--end loop;--p9
--end loop;--p8
end loop;--p7
end loop;--p6
end loop;--p5
end loop;--p4
end loop;--p3
end loop;--p2
end loop;--p1

--new_line(1);put("a1,2:");for i in 1..w loop put(a(1)(i),3); end loop;
--            put("  ");   for i in 1..w loop put(a(2)(i),3); end loop;

--new_line;
--put(" loopno=");put(ggg,3);
--endtime:=float(seconds(clock));
--put(" t(m)");put((1.0/60.0)*(endtime-start),5,2,0);--new_line;
--put("olaps=");put(overlaps,2);put(" n=");put(n,3);put(" d=");put(d,2);
--put(" w=");put(w,2);
--put(" Aest.=");put(i,3);


iii:=0;flag:=false;
for p1 in 1..startp1-1  loop
   --p1:=p1a mod 23 ;-- should be in ..n-w+1
for p2 in p1+1..(n-w+2) loop -- should be in ..n-w+2
for p3 in p2+1..(n-w+3) loop
for p4 in p3+1..(n-w+4) loop
for p5 in p4+1..(n-w+5)  loop
for p6 in p5+1..(n-w+6)  loop
for p7 in p6+1..(n-w+7)  loop
--for p8 in p7+1..(n-w+8)  loop

--for p9 in p8+1..(n-w+9)  loop
--for p10 in p9+1..(n-w+10)  loop
--for p11 in p10+1..(n-w+11)  loop
--for p12 in p11+1..(n-w+12)  loop

iii:=iii+1;
--if oldp1=p1 and oldp2=p2 and oldp3=p3 then iii:=iii+long_integer(50-p8); exit; end if;
b:=(p1,p2,p3,p4,p5,p6,p7);


      for k in reverse 1 .. i loop
          count:=0;
```

```
          for L in 1 .. w loop
            for M in 1 .. w loop
          --    if a(k)(L)<b(M) then exit; end if;  -- ##  al check
             if a(k)(L)=b(M) then count:=count+1; exit; end if;
            end loop;
            if count=olaps_plus1 then  goto there2; end if;
          end loop;
          --if flag=true then exit; end if;
          --if flag=true then goto here; end if;

      end loop;
      --if flag=false then
              i:=i+1; a(i):=b; --new_line; put("b i j ######### ");
            --  for ip in 1..w loop put(a(i)(ip),4); end loop;
             --put("     ");put(i,4);put("  ");put(iii,8);put(" ####");
             -- put((1.0/60.0)*(float(seconds(clock))-start),6,2,0);
     --         oldp1:=p1; oldp2:=p2; oldp3:=p3;
     -- end if;
    <<there2>> null;
--end loop;--p12
--end loop;--p11
--end loop;--p10
--end loop;--p9
--end loop;--p8
end loop;--p7
end loop;--p6
end loop;--p5
end loop;--p4
end loop;--p3
end loop;--p2
end loop;--p1


if i>Amax then Amax:=i;Amaxloop:=ggg ;end if;
Avals(i):=Avals(i)+1;

--if ggg=30 then
--   new_line;
--   for ik in 1 .. i loop
--     for ij in 1 .. w loop
--        put(a(ik)(ij),5);
--     end loop;
--     new_line;
--   end loop;
--end if;

end loop;--ggg
new_line;put("################       Amax=");put(Amax);
put("  in loop=");put(Amaxloop);new_line;
for i in 20..43 loop put(i,4); end loop; new_line;
for i in 20..43 loop put(Avals(i),4);  end loop;new_line;
endtime:=float(seconds(clock));
put(" t(m)");put((1.0/60.0)*(endtime-start),5,2,0);--new_line;
put(" started p1 loop at ");put(startp1,4);
end f_n_6_7tt;
```

12.2 Code for the cyclic construction

This is the code which implements the cyclic construction. Code for the maximum weighted clique finding software is not given here. This code uses a package `C_Comms` which effects file copying using a call to a C procedure. Other methods of implementing the file copying may be more convenient with other compilers. The procedure *Call_Copy_Files* which is imported from C is not given here.

```
with Text_Io; use Text_Io;
with Interfaces.C;

package C_Comms is

   procedure Call_Copy_Files;

   procedure Call_Clear_Screen;

   pragma Import(Convention => C,
      Entity => Call_Copy_Files,
      External_Name =>"Call_Copy_Files",
      Link_Name => "_Call_Copy_Files");

   pragma Import(Convention => C,
      Entity => Call_Clear_Screen,
      External_Name =>"Call_Cls",
      Link_Name => "_Call_Cls");

   pragma Linker_Options("caller.o");

   procedure Copy_Files(File1,File2:String);

   procedure Clear_Screen;

end C_Comms;

------------------------------------------------------------------------

with Text_Io; use Text_Io;
with Interfaces.C;

package body C_Comms is


   ------------------------------------------------------------------------

   procedure Copy_Files(File1,File2:String) is
      Batfile:Text_Io.File_Type;
   begin
      Create(File=>Batfile,Mode=>Out_File,Name=>"Callcopy.bat");
      put(batfile,"@ echo off");
      new_line(batfile);
      Put(Batfile,"copy ");
      Put(Batfile,File1);
      Put(Batfile," ");
      Put(Batfile,File2);
      Close(Batfile);
      Call_Copy_Files;
   end Copy_Files;
```

```
   ------------------------------------------------------------------------

   procedure Clear_Screen is
   begin
      Call_Clear_Screen;
   end Clear_Screen;

   ------------------------------------------------------------------------

end C_Comms;
 ------------------------------------------------------------------------
with  Text_Io;use Text_Io;
with C_Comms,Ada.Calendar;
use Ada.Calendar;

procedure Code_G_D is

   package Iio is new Integer_Io(Integer);use Iio;
   package Fio is new Float_Io(Float);use Fio;

   subtype Bit is Integer range 0..1;
   N:constant Integer:=61;      -- word length
   W:constant Integer:=5;
   T:constant Integer:=2;
   Number_Of_Twords:Integer:=0;
   Number_Of_Wwords:Integer:=0;
   Number_Of_New_Wwords:Integer:=0;
   type New_Array is array(1..N) of Integer;
   type New_Array7 is array(1..N) of New_Array;
   type New_Array2 is array(Positive range <>) of Integer;
   type Array_W is array(1..W) of Integer;
   type New_Array4 is array(Positive range <>) of New_Array;
   Max:Integer:=100000;    -- highest array size for new_array4;
   Flag:Boolean:=False;

   Txtfile:Text_Io.File_Type;


   Ch:Character;

   Seconds,Seconds1,Seconds2:Duration;
   Time1,Time2:Ada.Calendar.Time;

   ---------------------------------------------------------------------------

   -- determines if two codewords are in lexographic order
   -- returns true if C1 < C2

   function Order(C1,C2:in New_Array;P:in Integer) return Boolean is
   begin
      if C1=C2 then
         return True;              -- true if C1=C2
      elsif C1(P) = C2(P) then
         return Order(C1,C2,P+1);  -- if the first bits are the same then Order(C1(2..n),C2(2..n))
      elsif C1(P) = 1 then
         return True;              -- true if C1=1 and C2/=1
      elsif C2(P) = 1 then
         return False;             -- false if C1/=1 and C2=1
      else
         return True;
      end if;
   end Order;
```

```
-------------------------------------------------------------------------

-- initialise codeword and put into file codeword.dat with 1 in

procedure Initialise is
   Txtfile:Text_Io.File_Type;
begin
   Create(File=>Txtfile,Mode=>Out_File,Name=>"codeword.dat");
   Iio.Put(Txtfile,1,2);   -- create codeword with 1 in
   Close(Txtfile);
end Initialise;

-------------------------------------------------------------------------

-- deletes the files no longer used

procedure Delete_Files is
   Txtfile:Text_Io.File_Type;
begin
   Text_Io.Open(File=>Txtfile,Mode=>In_File,Name=>"codeword.dat");
   Text_Io.Delete(Txtfile);
   Text_Io.Open(File=>Txtfile,Mode=>In_File,Name=>"temp.dat");
   Text_Io.Delete(Txtfile);
end Delete_Files;

-------------------------------------------------------------------------

--returns lexographically minimal shift of a codeword

function Lexorder(C:in New_Array) return New_Array is
   C1:New_Array;
   D:New_Array;                  -- D is the minimal shift
   Temp:Integer;
begin
   C1:=C;
   D:=C;                         -- set D to be the input shift of the codeword
   for I in 1..N-1 loop
      Temp:=C1(N);               -- creates the next shift of the codeword
      C1(2..N):=C1(1..N-1);
      C1(1):=Temp;
      if not Order(D,C1,1) then  -- if C1 < D new minimal shift found set D=C1
         D:=C1;
      end if;
   end loop;
   return D;
end Lexorder;

-------------------------------------------------------------------------

function Number_Of_Ones(C:in New_Array) return Integer is
begin
   for I in 1..N loop
      if C(I)/=1 then
         return I-1;
      end if;
   end loop;
end Number_Of_Ones;


-------------------------------------------------------------------------
```

```
-- generates the t-sets (twords.dat) and the w-sets (codeword.dat)

procedure Generate_Codewords(M:in Integer) is
   subtype New_Array5 is New_Array4(1..Max);
   A:Integer;
   Txtfile1,Txtfile2,Txtfile3:Text_Io.File_Type;
   B,C,C1:New_Array;
   K:Integer:=0;
   Num:Integer:=0;
   Inc:Integer:=0;       -- counts the number of new codewords
   Bool:Boolean:=True;   -- flag set to false if a new codeword equals a previous codeword
   Index:Integer:=0;     -- records the position of the last change in O
begin

   Put_Line("Generating Codewords");

   Open(File=>Txtfile1,Mode=>In_File,Name=>"codeword.dat");
   Create(File=>Txtfile2,Mode=>Out_File,Name=>"temp.dat");

   while not End_Of_File(Txtfile1) loop

      C(1..N):=(others=>0);

      for I in 1..M-1 loop
         Iio.Get(Txtfile1,A); -- reads the previous set of codewords from file
         C(A):=1;
      end loop;

      Index:=0;

      for I in 1..N loop
         C1:=C;
         if C1(I) = 1 then
            Index:=Index+1;
         else
            if Index = M-1 then
               C1(I):=1;
               B:= Lexorder(C1);
               if B=C1 then
                  for J in 1..N loop
                     --Put(Txtfile2,C1(J),2);
                     if C1(J)=1 then
                        Iio.Put(Txtfile2,J,3);
                     end if;
                  end loop;
                  Inc:=Inc+1;

                  ----                     put(inc);
                  New_Line(Txtfile2);
               end if;
            end if;
         end if;
      end loop;
   end loop;
   --put("N");put(num);
   --put("I");put(inc);
   Close(Txtfile1);
   Close(Txtfile2);

   --get(ch);

   C_Comms.Copy_Files("temp.dat","codeword.dat");
```

```
   -- if this is the t'th iteration of the code the Twords are found
   if M=T then
      C_Comms.Copy_Files("codeword.dat","twords.dat");
      Number_Of_Twords:=Inc;
   end if;

   -- if this is the w'th iteration of the code the Wwords are found
   if M=W then
      C_Comms.Copy_Files("codeword.dat","wwords.dat");
      Number_Of_Wwords:=Inc;
   end if;

end Generate_Codewords;

-------------------------------------------------------------------------------

-- counts the number of times a tword (ct) covers a wword(cw)

procedure Cover(Ct,Cw:New_Array;P:out Integer) is
   Bool,Bool2:Boolean:=True;
   Temp:Integer;
   A,B:New_Array;
   Num:Integer:=0;
   Inc:Integer:=0;
begin
   A:=Ct;
   P:=0;

   for J in 1..N loop
      Bool:=True;
      Inc:=0;
      for I in 1..N loop
         if Inc = T then
            exit;
         else
            if A(I)=1 then
               Inc:=Inc+1;
               if Cw(I)=0 then
                  Bool:=False;   -- check if A covers Cw
                  exit;
               end if;
            end if;
         end if;

      end loop;
      if Bool then            -- if A covers Cw
         B:=Cw;
         for I in 1..N-1 loop
            Temp:=B(N);
            B(2..N):=B(1..N-1);  -- create a shift of the codeword
            B(1):=Temp;
            if B=Cw and J>=I then
               Bool2:=False;     -- checks that there isnt a duplicate shift in the cycle
               exit;
            end if;
         end loop;
         if Bool2 then
            P:=P+1;              -- if no duplicate shift already counted then increase by 1
         end if;
      end if;
```

```
      if P>1 then
         exit;
      end if;

      Temp:=A(N);
      A(2..N):=A(1..N-1);        -- create a shift of the codeword
      A(1):=Temp;

   end loop;

end Cover;

-------------------------------------------------------------------------

-- generates a matrix (matrix.dat)
-- the entries show how many times a Tword covers a Wword

procedure Generate_Matrix is
   Txtfile1,Txtfile2,Txtfile3:Text_Io.File_Type;
   Ct,Cw:New_Array;
   Times_Covered:Integer;
   Num:Integer:=0;
   Bool:Boolean:=True;
   B:Integer:=9;
   A:Integer;
   F:Integer;
begin

   New_Line;
   Put_Line("Generating Matrix");

   Open(File=>Txtfile1,Mode=>In_File,Name=>"twords.dat");
   Open(File=>Txtfile2,Mode=>In_File,Name=>"wwords.dat");
   Create(File=>Txtfile3,Mode=>Out_File,Name=>"matrix.dat");

   while not End_Of_File(Txtfile2) loop

      Num:=Num+1;
      Put(Num);

      Cw(1..N):=(others=>0);
      for I in 1..W loop
         Iio.Get(Txtfile2,A); -- reads the previous set of codewords from file
         Cw(A):=1;
      end loop;

      Bool:=True;
      Reset(Txtfile1);

      --while not End_Of_File(Txtfile1) loop
      for J in 1..Number_Of_Twords loop
         --         num:=num+1;
         --         put(num);
         if Bool then
            Ct(1..N):=(others=>0);
            for I in 1..T loop
               Get(Txtfile1,F);       -- read Tword from file
               Ct(F):=1;
            end loop;
            Cover(Ct,Cw,Times_Covered);    -- check how many times a Tword covers a Wword
            if Times_Covered>1 then
               Times_Covered:=B;
```

```
               Bool:=False;
            end if;
            Put(Txtfile3,Times_Covered,2);
         else
            Put(Txtfile3,B,2); -- and writes to file
         end if;
      end loop;
      New_Line(Txtfile3);
   end loop;

   Close(Txtfile1);
   Close(Txtfile2);
   Close(Txtfile3);

end Generate_Matrix;

-------------------------------------------------------------------------------

-- remove lines from the matrix that include an entry greater then 1

procedure Remove_Matrix_Entries_Greaterthan_1 is
   Txtfile1,Txtfile2,Txtfile3,Txtfile4:Text_Io.File_Type;
   subtype New_Array3 is New_Array2(1..Number_Of_Twords);
   B:New_Array3;
   C:Array_W;
   Bool:Boolean:=True;
   Q:Integer:=0;
begin

   New_Line;
   Put_Line("Ammending Matrix");

   Open(File=>Txtfile1,Mode=>In_File,Name=>"wwords.dat");
   Open(File=>Txtfile2,Mode=>In_File,Name=>"matrix.dat");
   Create(File=>Txtfile3,Mode=>Out_File,Name=>"wwords2.dat");
   Create(File=>Txtfile4,Mode=>Out_File,Name=>"matrix2.dat");

   for I in 1..Number_Of_Wwords loop
      Bool:=True;
      for K in 1..W loop
         Get(Txtfile1,C(K));       -- read Wword from file
      end loop;

      for J in 1..Number_Of_Twords loop
         Get(Txtfile2,B(J));       -- read matrix row from file
         if B(J)>1 then
            Bool:=False;           -- checks if matrix row has element > 1
         end if;
      end loop;

      if Bool then                 -- if no element in row > 1
         Q:=Q+1;                   -- counts the number of new Wwords
         for K in 1..W loop
            Put(Txtfile3,C(K),3);  -- write the Wword to file
         end loop;
         New_Line(Txtfile3);
         for J in 1..Number_Of_Twords loop
            Put(Txtfile4,B(J),2);  -- write the matrix row to file
         end loop;
         New_Line(Txtfile4);
      end if;
   end loop;
```

```
   Number_Of_New_Wwords:=Q;

   Close(Txtfile1);
   Close(Txtfile2);
   Close(Txtfile3);
   Close(Txtfile4);

end Remove_Matrix_Entries_Greaterthan_1;

-------------------------------------------------------------------------

-- counts the number of times the position of '1's overlap in two codewords

procedure Number_Of_Times_Overlap(C1,C2:in New_Array2;Q:in out Integer) is
begin
   Q:=0;
   for I in 1..Number_Of_Twords loop
      if C1(I)=1 and C2(I)=1 then
         Q:=1;
         exit;
      end if;
   end loop;
end Number_Of_Times_Overlap;

-------------------------------------------------------------------------

procedure Create_Var_And_Con_Files (Bool:in out Boolean) is
   Txtfile1,Txtfile2,Txtfile3,Txtfile4:Text_Io.File_Type;
   subtype New_Array3 is New_Array2(1..Number_Of_Twords);
   subtype New_Array4 is New_Array2(1..Max);
   B,C:New_Array3;
   Overlap:Integer:=0;
   D:New_Array4;           -- variable array
   --      bool:boolean:=false;
begin

   New_Line;
   Put_Line("Creating Variable and Constraint Files");

   C_Comms.Copy_Files("matrix2.dat","temp.dat");
   Open(File=>Txtfile1,Mode=>In_File,Name=>"matrix2.dat");
   Open(File=>Txtfile2,Mode=>In_File,Name=>"temp.dat");
   Create(File=>Txtfile3,Mode=>Out_File,Name=>"mat.ctr");
   Create(File=>Txtfile4,Mode=>Out_File,Name=>"Mat.Var");

   D(1..Max):=(others => 0);                -- initialise variable array

   for I in 1..Number_Of_New_Wwords loop

      Put(I);

      for J in 1..Number_Of_Twords loop
         Get(Txtfile1,B(J));                -- read in matrix row
      end loop;

      Reset(Txtfile2);

      for K in 1..Number_Of_New_Wwords loop

         for J in 1..Number_Of_Twords loop
            Get(Txtfile2,C(J));             -- read in matrix row
```

```
         end loop;

         if K>I then                          -- compare with matrix rows later than current row
            Number_Of_Times_Overlap(B,C,Overlap);   -- count number of times the two rows overlap
            if Overlap = 0 then               -- can be joined if they dont overlap
               Put(Txtfile3,I);               -- write the data to the constraint file
               Put(Txtfile3,K);
               Put(Txtfile3," >  0 ");
               New_Line(Txtfile3);
               D(I):=9;                       -- i and j are in the variable array
               D(K):=9;
               Bool:=True;
            end if;
         end if;

      end loop;
   end loop;

   for I in 1..Max loop
      if D(I)=9 then
         Put(Txtfile4,I);
         Put(Txtfile4," 0");
         New_Line(Txtfile4);
      end if;
   end loop;


   Close(Txtfile1);
   Close(Txtfile2);
   Close(Txtfile3);
   Close(Txtfile4);

end Create_Var_And_Con_Files;

-------------------------------------------------------------------------------

procedure Find_Number_Of_Codewords is
   Txtfile1,Txtfile2,Txtfile3:Text_Io.File_Type;
   Line1:String(1..50);
   L1:Integer;
   A,P:Integer;
   D:New_Array;
   C:New_Array7;
   Temp:Integer;
   V:Integer:=0;
   Bool:Boolean:=False;
   Num_Of_Codewords:Integer:=0;
begin

   Open(File=>Txtfile1,Mode=>In_File,Name=>"temp.log");
   Open(File=>Txtfile2,Mode=>In_File,Name=>"wwords2.dat");
   Create(File=>Txtfile3,Mode=>Out_File,Name=>"codeword.dat");

   Get_Line(Txtfile1,Line1,L1);

   loop
      begin
         Get(Txtfile1,P);
         Put(P);

      exception
         when Data_Error =>
```

```
            exit;

         when End_Error =>

            exit;
      end;


      Reset(Txtfile2);
      while not End_Of_File(Txtfile2) loop
         V:=V+1;
         D(1..N):=(others=>0);
         for J in 1..W loop
            Get(Txtfile2,A);
            D(A):=1;
         end loop;
         if V=P then
            Put(V);
            New_Line;
            V:=0;
            C(1):=D;

            for I in 1..N-1 loop
               Temp:=C(I)(N);
               C(I+1)(2..N):=C(I)(1..N-1);
               C(I+1)(1):=Temp;
            end loop;
            for I in 1..N loop
               for J in 1..I-1 loop
                  if C(I)=C(J) then
                     Bool:=True;
                  end if;
               end loop;
               if not Bool then
                  for K in 1..N loop
                     Put(Txtfile3,C(I)(K),2);
                  end loop;
                  Num_Of_Codewords:=Num_Of_Codewords+1;
                  New_Line(Txtfile3);
               end if;
               Bool:=False;
            end loop;
            exit;
         end if;
      end loop;

   end loop;
   Close(Txtfile1);
   Close(Txtfile2);
   Close(Txtfile3);

   New_Line;
   Put("Number of codewords: ");
   Put(Num_Of_Codewords);

end Find_Number_Of_Codewords;

-------------------------------------------------------------------------

procedure Number_Of_Orbits is
   Txtfile3,Txtfile1,Txtfile2:Text_Io.File_Type;
   E,E1,D:New_Array;
```

```
   C:New_Array7;
   Bool:Boolean:=False;
   Num_Of_Codewords:Integer:=0;
   P:Integer:=0;
   K:Integer:=0;
   A,Temp:Integer;
begin
   Open(File=>Txtfile1,Mode=>In_File,Name=>"wwords2.dat");
   Open(File=>Txtfile3,Mode=>In_File,Name=>"mat.var");
   Create(File=>Txtfile2,Mode=>Out_File,Name=>"mat.dem");

   Put(Number_Of_New_Wwords);
   while not End_Of_File(Txtfile3) loop

      Get(Txtfile3,P);
      Get(Txtfile3,Temp);

      while not End_Of_File(Txtfile1) loop
         K:=K+1;
         D(1..N):=(others=>0);
         for J in 1..W loop
            Get(Txtfile1,A);
            D(A):=1;
         end loop;
         if K=P then
            C(1):=D;
            for I in 1..N-1 loop
               Temp:=C(I)(N);
               C(I+1)(2..N):=C(I)(1..N-1);
               C(I+1)(1):=Temp;
            end loop;
            for I in 1..N loop
               for J in 1..I-1 loop
                  if C(I)=C(J) then
                     Bool:=True;
                     exit;
                  end if;
               end loop;
               if not Bool then
                  Num_Of_Codewords:=Num_Of_Codewords+1;
                  --              put(num_of_codewords);
               end if;
               Bool:=False;
            end loop;

            Put(Txtfile2,K,4);
            Put(Txtfile2,Num_Of_Codewords,4);
            Num_Of_Codewords:=0;
            New_Line(Txtfile2);
            exit;
         end if;
      end loop;
   end loop;

   Close(Txtfile1);
   Close(Txtfile2);
   Close(Txtfile3);


end Number_Of_Orbits;

-------------------------------------------------------------------------
```

```
procedure Find_Codeword_With_Most_Shifts is
   Txtfile1,Txtfile2:Text_Io.File_Type;
   E,E1,D:New_Array;
   C:New_Array7;
   Bool:Boolean:=False;
   Num_Of_Codewords:Integer:=0;
   Max:Integer:=0;
   A,Temp:Integer;
begin
   Open(File=>Txtfile1,Mode=>In_File,Name=>"wwords2.dat");
   Create(File=>Txtfile2,Mode=>Out_File,Name=>"codeword.dat");


   while not End_Of_File(Txtfile1) loop

      D(1..N):=(others=>0);
      for J in 1..W loop
         Get(Txtfile1,A);
         D(A):=1;
      end loop;
      C(1):=D;
      for I in 1..N-1 loop
         Temp:=C(I)(N);
         C(I+1)(2..N):=C(I)(1..N-1);
         C(I+1)(1):=Temp;
      end loop;
      for I in 1..N loop
         for J in 1..I-1 loop
            if C(I)=C(J) then
               Bool:=True;
               exit;
            end if;
         end loop;
         if not Bool then
            Num_Of_Codewords:=Num_Of_Codewords+1;
            --              put(num_of_codewords);
         end if;
         Bool:=False;
      end loop;
      if Num_Of_Codewords > Max then
         Max:=Num_Of_Codewords;
         E:=D;
      end if;
      Num_Of_Codewords:=0;
      --         put(max);
      --         put(num_of_codewords);
   end loop;

   E1:=E;
   for J in 1..N loop
      Put(Txtfile2,E(J),2);
   end loop;
   New_Line(Txtfile2);
   for I in 1..N-1 loop
      Temp:=E1(N);
      E1(2..N):=E1(1..N-1);
      E1(1):=Temp;
      if E1/=E then
         for J in 1..N loop
            Put(Txtfile2,E1(J),2);
         end loop;
         New_Line(Txtfile2);
```

```
         end if;
      end loop;

      Close(Txtfile1);
      Close(Txtfile2);

      New_Line;
      Put("Number of codewords: ");
      Put(Max);
   end Find_Codeword_With_Most_Shifts;


   ------------------------------------------------------------------------

begin

   Time1:=Ada.Calendar.Clock;

   Initialise;

   for I in 2..W loop
      Put(I);New_Line;
      Generate_Codewords(I);
   end loop;

   Delete_Files;

   Put(Number_Of_Wwords);
   Put(Number_Of_Twords);

   --   Get(Ch);

   Generate_Matrix;

   Remove_Matrix_Entries_Greaterthan_1;

   Text_Io.Open(File=>Txtfile,Mode=>In_File,Name=>"wwords.dat");
   Text_Io.Delete(Txtfile);
   Text_Io.Open(File=>Txtfile,Mode=>In_File,Name=>"twords.dat");
   Text_Io.Delete(Txtfile);
   Text_Io.Open(File=>Txtfile,Mode=>In_File,Name=>"matrix.dat");
   Text_Io.Delete(Txtfile);



   Create_Var_And_Con_Files(Flag);


   New_Line;
   Put("Number of new W-words: ");
   Put(Number_Of_New_Wwords);

   Number_Of_Orbits;

   if Flag then
      New_Line;
      Put_Line("Run clique finding algorithm on Mat.ctr and Mat.var.");
      Get(Ch);
      --      Time1:=Ada.Calendar.Clock;

      Find_Number_Of_Codewords;
   else
```

```
      New_Line;
      --      Time1:=Ada.Calendar.Clock;
      Put_Line("No cliques exist.");
      Find_Codeword_With_Most_Shifts;
   end if;


   Time2:=Ada.Calendar.Clock;
   Seconds:=(Time2-Time1);

   --   Seconds:=Seconds1+Seconds2;

   New_Line;
   Put("Total time taken in seconds: ");
   Fio.Put(Float(Seconds),Aft=>4,Exp=>0);

end Code_G_D;
```

12.3 Code for non binary/mixed lexicographic search

```
with Text_Io;
use Text_Io;
procedure Cdwrdsm1 is
   package Integer_Inout is new Integer_Io(Integer);
   use Integer_Inout;
   Outfile: File_Type;
   Cq_1 : constant Integer := 2;
   Cq_2 : constant Integer := 3;
   Cn_1 : constant Integer := 8;
   Cn_2 : constant Integer := 4;
   Ln : constant Integer := Cn_1+Cn_2;
   Lim : constant Integer := 150000;
   Nw : Integer := 0;
   Ix3 : Integer;
   Ix6 : Integer :=1;
   Ix9 : Integer :=1;
   Op : Integer;
   Olp : Integer :=8;
   type Trow is array (1..Ln) of Integer;
   type Tmatrix is array (1..Lim) of Trow;
   Word : Trow;
   Matrix : Tmatrix;
   function Comp (Wy1, Wy2 : Trow) return Integer is
   begin
      Op := 0;
      for I in 1..Ln loop
         if Wy1(I)=Wy2(I) then
            Op := Op+1;
         end if;
         exit when Op=Olp;
      end loop;
      return Op;
   end Comp;
begin
   Create(Outfile,Name=>"c:\results\dict5.txt");
   --matrix(1)(1) :=3;
```

```
   --matrix(1)(2) :=3;
   --matrix(1)(3) :=1;
   --matrix(1)(4) :=1;
   Word(1) := -1;
   for Ix1 in 2..Ln loop
      Word(Ix1) := 0;
   end loop;
   while Nw<Lim loop
      Ix3 := 1;
      while Ix3<Cn_1+1 and Word(Ix3)=Cq_1-1 loop
         Ix3 := Ix3+1;
      end loop;
      if Ix3<Cn_1+1 then
         for Ix8 in 1..Ix3-1 loop
            Word(Ix8) := 0;
         end loop;
         Word(Ix3):= Word(Ix3)+1;
      else
         while Ix3<Ln and Word(Ix3)=Cq_2-1 loop
            Ix3 := Ix3+1;
         end loop;
         if  Word(Ix3)<Cq_2-1 then
            for Ix8 in 1..Ix3-1 loop
               Word(Ix8) := 0;
            end loop;
            Word(Ix3):= Word(Ix3)+1;
         else
            exit;
         end if;
      end if;
      Ix6 := 0;
      for Ix5 in 1..Nw loop
         if Comp(Word, Matrix(Ix5))=Olp then
            Ix6 :=1;
            exit;
         end if;
      end loop;
      if Ix6=0 then
         Nw := Nw+1;
         Matrix(Nw) := Word;
         Put ("Number is now:");
         Put (Nw);
         New_Line;
         Put ("next word is:");
         New_Line;
         Put(Word(1));
         Put(Outfile,Nw,Width=>1);
         Put(Outfile,Word(1),Width=>5);
         for Ix7 in 2..Ln loop
            Put (Word(Ix7));
            Put(Outfile,Word(Ix7),Width=>2);
         end loop;
         New_Line;
         New_Line(Outfile);
      end if;
   end loop;
   Close(Outfile);
end Cdwrdsm1;
```

12.4 Code for constructing codes using theorem 6.15.

```
with Text_Io;
use Text_Io;
procedure Phisrch is
   package Integer_Inout is new Integer_Io(Integer);
   use Integer_Inout;
   Outfile: File_Type;
   Cq_1 : constant Integer := 2;
   Cq_2 : constant Integer := 3;
   Cn_1 : constant Integer := 2;
   Cn_2 : constant Integer := 11;
   Cya : constant Integer := 1;
   Cyb : constant Integer := 7;
   Ds : Integer :=7;
   Par11 : constant Integer := 0;
   Par12 : constant Integer :=1;
   Par21 : constant Integer := 1;
   Par22 : constant Integer := 1;


   Bck_1, Ck_1 : Integer;
   Bck_2, Ck_2 : Integer;

   Ln : constant Integer := Cn_1+Cn_2;
   Lim : constant Integer := 1000;
   Plim : constant Integer := 3000;
   Bpnw : Integer := 0;
   Bnw, Bnw1, Nw,Nw1, Pnw : Integer;
   Ix3, Tx1, Tx2 : Integer;
   Ix6 : Integer;
   Ix9 : Integer;
   Op : Integer;
   Olp : Integer :=Cn_1+Cn_2-Ds+1;
   Cna, Cnb, Na, Nb : Integer;

   type Trow is array (1..Ln) of Integer;
   Word : Trow;
   type Tmatrix is array (1..Lim) of Trow;
   Bmatrix, Matrix : Tmatrix;
   type Taword is array (1..Cn_1*Cn_1+2) of Integer;
   Aword, Baword : Taword;
   type Tbword is array (1..Cn_2*Cn_2+2) of Integer;
   Bword, Bbword : Tbword;
   type Trow1 is array (1..Cn_1+2) of Integer;
   type Tmatrix1 is array (1..Cn_1+2) of Trow1;
   type Trow2 is array (1..Cn_2+2) of Integer;
   type Tmatrix2 is array (1..Cn_2+2) of Trow2;
   Matrixa : Tmatrix1;
   Matrixb : Tmatrix2;
   Kword : Trow;
   type Tmatrixg1 is array (1..Cn_1+2) of Trow1;
   type Tmatrixg2 is array (1..Cn_2+2) of Trow2;
   Matrixg1 : Tmatrixg1;
   Matrixg2 : Tmatrixg2;
   Sln : Integer;
   Zen1, Zen2, Sop : Integer;
   type Trows is array (1..Cn_1+Cn_2) of Integer;
   Sword, Cword : Trows;
   type Tmatrixs is array (1..Lim) of Trows;
```

```
   Smatrix, Bsmatrix : Tmatrixs;

   function Comp (Wy1, Wy2 : Trow) return Integer is
   begin
      Op := 0;
      for I in 1..Ln loop
         if Wy1(I)=Wy2(I) then
            Op := Op+1;
         end if;
         exit when Op=Olp;
      end loop;
      return Op;
   end Comp;
begin
   Create(Outfile,Name=>"c:\results\th310dop.txt");
   for Ck_1 in Par11..Par12 loop
      for Ck_2 in Par21..Par22 loop
         Cna := Ck_1*(Cn_1-Ck_1);
         Cnb := Ck_2*(Cn_2-Ck_2);

         for I in 1..Cn_1-Ck_1 loop
            for R in 1..Cn_1-Ck_1 loop
               if I=R then
                  Matrixa(I)(R) := 1;
               else
                  Matrixa(I)(R) := 0;
               end if;
            end loop;
         end loop;

         for I in 1..Cn_2-Ck_2 loop
            for R in 1..Cn_2-Ck_2 loop
               if I=R then
                  Matrixb(I)(R) := 1;
               else
                  Matrixb(I)(R) := 0;
               end if;
            end loop;
         end loop;
         Aword(1) := -1;
         for I in 2..Cna loop
            Aword(I):=0;
         end loop;
         Na :=0;
         while Bpnw<Plim loop
            for P in 1..Cya loop
               Tx1 :=1;
               while Tx1<Cna and Aword(Tx1)=Cq_1-1 loop
                  Tx1 := Tx1+1;
               end loop;
               if  Aword(Tx1)<Cq_1-1 then
                  for Ix8 in 1..Tx1-1 loop
                     Aword(Ix8) := 0;
                  end loop;
                  Aword(Tx1):= Aword(Tx1)+1;
               else
                  Na :=1;
                  exit;
               end if;
            end loop;
            if Na=1 then
               exit;
```

```
end if;
Bword(1) := -1;

for I in 2..Cnb loop
   Bword(I):=0;
end loop;
Nb :=0;
while Bpnw<Plim loop
   for P in 1..Cyb loop
      Tx1 :=1;
      while Tx1<Cnb and Bword(Tx1)=Cq_2-1 loop
         Tx1 := Tx1+1;
      end loop;
      if  Bword(Tx1)<Cq_2-1 then
         for Ix8 in 1..Tx1-1 loop
            Bword(Ix8) := 0;
         end loop;
         Bword(Tx1):= Bword(Tx1)+1;
      else
         Nb :=1;
         exit;
      end if;
   end loop;
   if Nb=1 then
      exit;
   end if;

   for I in 1..Ck_1 loop
      for R in 1..Cn_1-Ck_1 loop
         Matrixa(R)(Cn_1-Ck_1+I) := Aword((R-1)*Ck_1+I);
      end loop;
   end loop;
   for I in 1..Ck_2 loop
      for R in 1..Cn_2-Ck_2 loop
         Matrixb(R)(Cn_2-Ck_2+I) := Bword((R-1)*Ck_2+I);
      end loop;
   end loop;
   Nw :=0; Ix6 := 1; Ix9 := 1;
   for I in 1..Ck_1 loop
      for R in 1..Cn_1-Ck_1 loop
         Matrixg1(I)(R) := -Matrixa(R)(Cn_1-Ck_1+I);
      end loop;
      for R in Cn_1-Ck_1+1..Cn_1 loop
         if I=R-Cn_1+Ck_1 then
            Matrixg1(I)(R) := 1;
         else
            Matrixg1(I)(R) := 0;
         end if;
      end loop;
   end loop;
   for I in 1..Ck_2 loop
      for R in 1..Cn_2-Ck_2 loop
         Matrixg2(I)(R) := -Matrixb(R)(Cn_2-Ck_2+I);
      end loop;
      for R in Cn_2-Ck_2+1..Cn_2 loop
         if I=R-Cn_2+Ck_2 then
            Matrixg2(I)(R) := 1;
         else
            Matrixg2(I)(R) := 0;
         end if;
      end loop;
   end loop;
```

```
Kword(1) := -1;
for Ix1 in 2..Ck_1+Ck_2 loop
   Kword(Ix1) := 0;
end loop;
while Nw<Lim loop
   Ix3 := 1;
   while Ix3<Ck_1+1 and Kword(Ix3)=Cq_1-1 loop
      Ix3 := Ix3+1;
   end loop;
   if Ix3<Ck_1+1 then
      for Ix8 in 1..Ix3-1 loop
         Kword(Ix8) := 0;
      end loop;
      Kword(Ix3):= Kword(Ix3)+1;
   else
      while Ix3< Ck_1+Ck_2 and Kword(Ix3)=Cq_2-1 loop
         Ix3 := Ix3+1;
      end loop;
      if  Kword(Ix3)<Cq_2-1 then
         for Ix8 in 1..Ix3-1 loop
            Kword(Ix8) := 0;
         end loop;
         Kword(Ix3):= Kword(Ix3)+1;
      else
         exit;
      end if;
   end if;
   ---------------------------
   --------------------------------
   for I in 1..Cn_1 loop
      Word(I):=0;
      for R in 1..Ck_1 loop
         Word(I):= Kword(R)*Matrixg1(R)(I)+ Word(I);
      end loop;
      Word(I) := Word(I) mod Cq_1;
   end loop;

   for I in Cn_1+1..Ln loop
      Word(I):=0;
      for R in 1..Ck_2 loop
         Word(I):= Kword(R+Ck_1)  * Matrixg2(R)(I-Cn_1)+ Word(I);
      end loop;
      Word(I) := Word(I) mod Cq_2;
   end loop;

   Ix6 := 0;
   for Ix5 in 1..Nw loop
      if Comp(Word, Matrix(Ix5))=Olp then
         Ix6 :=1;
         exit;
      end if;
   end loop;
   if Ix6=0 then
      Nw := Nw+1;
      Matrix(Nw) := Word;
   end if;
end loop;
Nw1 := Nw;
Nw := 0;
Sln := Cn_1+Cn_2-Ck_1-Ck_2;
Ix9 := 1;
```

```
Sword(1) := -1;
for Ix1 in 2..Sln loop
   Sword(Ix1) := 0;
end loop;
while Nw<Lim loop
   Ix3 := 1;
   while Ix3<Cn_1-Ck_1+1 and Sword(Ix3)=Cq_1-1 loop
      Ix3 := Ix3+1;
   end loop;
   if Ix3<Cn_1-Ck_1+1 then
      for Ix8 in 1..Ix3-1 loop
         Sword(Ix8) := 0;
      end loop;
      Sword(Ix3):= Sword(Ix3)+1;
   else
      while Ix3<Sln and Sword(Ix3)=Cq_2-1 loop
         Ix3 := Ix3+1;
      end loop;
      if  Sword(Ix3)<Cq_2-1 then
         for Ix8 in 1..Ix3-1 loop
            Sword(Ix8) := 0;
         end loop;
         Sword(Ix3):= Sword(Ix3)+1;
      else
         exit;
      end if;
   end if;

   Sop := Sln;
   Ix6 := 0;
   for I in 1..Nw loop
      Kword(1) := -1;
      for Ix1 in 2..Ck_1+Ck_2 loop
         Kword(Ix1) := 0;
      end loop;
      while Nw<Lim loop
         Ix3 := 1;
         while Ix3<Ck_1+1 and Kword(Ix3)=Cq_1-1 loop
            Ix3 := Ix3+1;
         end loop;
         if Ix3<Ck_1+1 then
            for Ix8 in 1..Ix3-1 loop
               Kword(Ix8) := 0;
            end loop;
            Kword(Ix3):= Kword(Ix3)+1;
         else
            while Ix3< Ck_1+Ck_2 and Kword(Ix3)=Cq_2-1 loop
               Ix3 := Ix3+1;
            end loop;
            if  Kword(Ix3)<Cq_2-1 then
               for Ix8 in 1..Ix3-1 loop
                  Kword(Ix8) := 0;
               end loop;
               Kword(Ix3):= Kword(Ix3)+1;
            else
               exit;
            end if;
         end if;
         Zen1 := 0;
         Zen2 := 0;
         for H in 1..Ck_1+Ck_2 loop
            if Kword(H)=0 then
```

```
               Zen1 := Zen1+1;
            end if;
         end loop;
         for R in 1..Cn_1-Ck_1 loop
            Cword(R) :=Sword(R)-Smatrix(I)(R);
            for F in 1..Ck_1 loop
               Cword(R) :=Cword(R)-Matrixa(R)(Cn_1-Ck_1+F)*Kword(F);
            end loop;
            Cword(R) := Cword(R) mod Cq_1;
         end loop;

         for R in Cn_1-Ck_1+1..Sln loop
            Cword(R) :=Sword(R)-Smatrix(I)(R);
            for F in 1..Ck_2 loop
               Cword(R) :=Cword(R)-Matrixb(R-Cn_1+Ck_1)(Cn_2-Ck_2+F)*Kword(Ck_1+F);
            end loop;
            Cword(R) := Cword(R) mod Cq_2;
         end loop;

         for H in 1..Sln loop
            if Cword(H)=0 then
               Zen2 := Zen2+1;
            end if;
         end loop;
         if Cn_1+Cn_2-Zen1-Zen2 < Sop then
            Sop := Cn_1+Cn_2-Zen1-Zen2;
         end if;
      end loop;
      if Sop < Ds then
         Ix6 := 1;
         exit;
      end if;
   end loop;

   --------------------------------

   if Ix6=0 then
      Nw := Nw+1;
      Smatrix(Nw) := Sword;
   end if;
end loop;
Pnw := Nw1*Nw;
if Pnw>Bpnw then
   Bnw1 := Nw1; Bnw := Nw;
   Bpnw := Pnw;
   Bmatrix := Matrix;
   Bsmatrix := Smatrix;
   Baword := Aword;
   Bbword := Bword;
   Bck_1 := Ck_1; Bck_2 := Ck_2;
end if;
Put("next cycle parameters:");
New_Line;
Put(Ck_1);
Put(Ck_2);
New_Line;
Put(Nw1);
Put(Nw);
New_Line;

Put(Bck_1);
New_Line;
```

```
               Put(Bck_2);
               New_Line;
               Put(Bpnw);
               New_Line;
            end loop;
         end loop;
      end loop;
   end loop;
   Put(Outfile,Bpnw,Width=>1);
   New_Line(Outfile);
   Put(Outfile,Bnw1,Width=>1);
   New_Line(Outfile);
   Put(Outfile,Bnw,Width=>1);
   New_Line(Outfile);
   Put(Outfile,Bck_1,Width=>1);
   New_Line(Outfile);
   Put(Outfile,Bck_2,Width=>1);
   New_Line(Outfile);
   for Ix7 in 1..Cn_1-Bck_1 loop
      for Ix8 in 1..Bck_1 loop
         Put(Outfile,Baword((Ix7-1)*Bck_1+Ix8),Width=>2);
      end loop;
      New_Line(Outfile);
   end loop;
   New_Line(Outfile);
   for Ix7 in 1..Cn_2-Bck_2 loop
      for Ix8 in 1..Bck_2 loop
         Put(Outfile,Bbword((Ix7-1)*Bck_2+Ix8),Width=>2);
      end loop;
      New_Line(Outfile);
   end loop;
   New_Line(Outfile);
   New_Line(Outfile);
   ------------------------------------------------------------
   for Ix7 in 1..Bnw1 loop
      for Ix8 in 1..Ln loop
         Put(Outfile,Bmatrix(Ix7)(Ix8),Width=>2);
      end loop;
      New_Line(Outfile);
   end loop;
   New_Line(Outfile);
   New_Line(Outfile);
   ------------------------------------------------------------------------
   for Ix7 in 1..Bnw loop
      for Ix8 in 1..Ln-Bck_1-Bck_2 loop
         Put(Outfile,Bsmatrix(Ix7)(Ix8),Width=>2);
      end loop;
      New_Line(Outfile);
   end loop;
   New_Line(Outfile);



   Close(Outfile);
end Phisrch;
```