Heuristic Construction of Constant Weight Binary Codes

Roberto Montemanni   Derek H. Smith

Technical Report No. IDSIA-12-07

IDSIA / USI-SUPSI
Istituto Dalle Molle di studi sull'intelligenza artificiale
Galleria 2, 6928 Manno, Switzerland

Heuristic Construction of Constant Weight Binary Codes

Roberto Montemanni[*] Derek H. Smith[†]

Abstract

Constant weight binary codes are used in a number of applications. The most fully developed treatment of these codes in the literature is restricted to codes of length at most 28. Only recently have comprehensive results been presented for longer codes. Constructions based on mathematical structure are known for many of the shorter codes. However, such constructions are rarer for lengths greater than 28, and algorithmic constructions are often useful. When a good mathematical structure is known, algorithmic methods will only rarely obtain as many codewords. However, in an application requiring a variety of different code parameters, it may be more convenient to find a moderately good code using a single algorithm than to attempt to elucidate a suitable structure in each case.

This paper considers the problem of finding constant weight codes with the maximum number of codewords from an algorithmic perspective. A set of heuristic and metaheuristic methods is developed which targets the production of codes with lengths between 29 and 63.

After a detailed description of the proposed algorithms, and a discussion of the choices made in their development, some computational experiments are presented. These aim to achieve an understanding of the potential of the novel methods. As a result of these experiments, many new codes are obtained with significantly increased numbers of codewords in comparison with existing constructions. For 10 of these new codes the number of codewords meets a known upper bound, and so these 10 codes are optimal.
Keyword: Constant weight binary codes, lower bounds, heuristic algorithms.

1 Introduction

A *constant weight binary code* is a set of binary vectors of length n, weight w and minimum Hamming distance d. The weight of a binary vector (or *codeword*) is the number of 1's in the vector. The Hamming distance $d(\mathbf{x}, \mathbf{y})$ between two vectors $\mathbf{x}$ and $\mathbf{y}$ is the number of positions in which they differ. The minimum distance of a code is the minimum Hamming distance between any pair of codewords. The maximum possible number of vectors in a constant weight code is usually referred to as $A(n, d, w)$.

[*]R. Montemanni is with the Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA), Galleria 2, CH-6928 Manno, Switzerland. *Email: roberto@idsia.ch*

[†]D.H. Smith is with the Division of Mathematics and Statistics, University of Glamorgan, Pontypridd, Mid-Glamorgan, CF37 1DL, Wales, UK. *Email: dhsmith@glam.ac.uk*

Apart from their important role in the theory of error-correcting codes [11], constant weight codes have also found application in fields as diverse as the design of demultiplexers for nano-scale memories, the construction of frequency hopping lists for use in GSM networks [12] and the design of DNA codes for use in DNA barcoding and DNA computing [10].

Accounts of the theory of constant weight codes can be found in [11, 3]. A detailed account of upper bounds for $A(n, d, w)$ can be found in [1]. Lower bounds for $A(n, d, w)$ are usually obtained constructively. Code constructions for $n \leq 28$ can be found in [3]. In [14] a comprehensive set of constructions was described for the parameter sets appropriate to the frequency hopping application. These parameter sets were $29 \leq n \leq 63$ and $5 \leq w \leq 8$ with $d = 2w - 2$, $d = 2w - 4$ and $d = 2w - 6$. A small number of improvements to the values in [14] can be found in [8] and [7]. In [3] the construction of a code without identifying its mathematical structure is regarded as a "failure". However, algorithmic constructions are useful in applications and for longer codes algorithms often give the best known code. Concerning metaheuristic algorithms, both simulated annealing [6] and tabu search [2] have been used to construct constant weight codes, although the number of results presented in [6, 2] is small. The methods presented here significantly improve these results.

It can be noted from the results in [14] that when no good mathematical construction is available and methods from [3] using permutation groups cease to be feasible in a reasonable run time, then lexicographic search becomes the default method. It usually performs reasonably well, and is certainly much more effective than random search [14]. There are a number of variations of lexicographic search, including the use of reverse lexicographic search and the use of of seed vectors. Thus lexicographic search forms the basis of the first method presented here. In small cases clique search can also be used to find good constant weight codes. In the second method presented here clique search is used in a way which restricts the search to feasible partial problems. The two methods can then be combined in a variable neighbourhood search framework [9].

Section 2 describes the development of the local search algorithms; seed building in 2.1 and clique search in 2.2. In 2.3 these local search algorithms are evaluated. In Section 3 the local search algorithms are developed into a variable neighbourhood search metaheuristic, and the results from the new algorithm are compared with the best of the local search algorithms. In Section 4 the numbers of codewords of new best codes are tabulated and compared with known upper bounds. Ten of the new codes are optimal. Finally, Section 5 presents conclusions.

2 Local search algorithms

In this section two families of local search algorithms for maximizing the number of codewords in a constant weight code are developed. Some computational results aiming at understanding the potential of the methods are presented and discussed.

2.1 Seed building

Forward lexicographic order of binary vectors is a dictionary order starting at $00 \cdots 011 \cdots 1$. Vector $\mathbf{x} = x_1 \ldots x_n$ is listed before $\mathbf{y} = y_1 \ldots y_n$ if $x_i < y_i$, where i is the first position in which the two vectors differ. Thus for $w = 2$ and $n = 4$ the order is 0011,0101,0110,1001,1010,1100. Reverse lexicographic order is the reverse of this order.

As observed in the introduction, algorithms that examine all possible codewords in lexicographic order or reverse lexicographic order and incrementally accept codewords that are feasible with respect to already accepted ones, can often produce fairly good codes [14, 8]. Sometimes they can produce very good codes [5]. For this reason the first method presented here is built on these orderings, combined with the concept of *seed codewords* [3]. These seed codewords are an initial set of codewords of weight w to which codewords are added in the given ordering if they satisfy the minimum distance criterion. Given a set of seed codewords, the set of codewords incrementally added by a given ordered search may change radically.

Seeds can be selected at random, but computational results (see Section 2.3) clearly show that there is a better way to proceed. In the seed building algorithm a set of seeds is initially empty, and one random seed is added at a time. If this seed leads to good results it is kept, and a new random seed is designated for testing, increasing the size of the seed set. The same rationale is used to decide whether to keep subsequent seeds or not. In the same way, if after a given number of iterations the quality of the solutions provided by a set of seeds is judged to be not good enough, the most recent seed is eliminated from the set, which results therefore in a reduction in size of the seed set. In this way the set of seeds is expanded or contracted depending on the quality of the solutions provided by the set itself. What happens in practise is that the size of the seed set oscillates through a range of small values.

Pseudo-code for the seed building method (SB) is provided in Figure 1. BestSol and WorkSol represent respectively the best solution retrieved so far, and the working solution. SelOrd is a parameter describing the sequence in which binary vectors are examined (i.e. forward lexicographic, reverse lexicographic or random), SeedSet is the set of seed vectors, while ItrCnt is an iteration counter and ItrSeed is a parameter indicating for how long (in terms of number of iterations) each new seed is tested. CWord is a random codeword used to extend the partial code contained in SeedSet. The algorithms works in an iterative fashion on an adaptive set of seed codewords contained in the set SeedSet, which is initially empty. At each iteration, a new codeword CWord, compatible with those in SeedSet, is generated and the partial solution WorkSol, initialized with the elements of SeedSet and CWord, is expanded by adding feasible binary vectors, examined according to the order defined by parameter SelOrd. If a new best solution is found, the set SeedSet is immediately expanded. Every ItrSeed iterations, the average size of the codes generated with the set SeedSet is checked to determine whether SeedSet is a promising set (in which case it should be augmented) or not (in which case it is reduced by deleting the most recently added seed). The procedure stops after a fixed computation time $Time_{SB}$ has been reached.

2.2 Clique search

The idea at the basis of this local search method is that it is possible to complete, in the best possible way, a partial code by solving a maximum clique problem [13]. More precisely, given a code, a random subset of the codewords of the code is removed, leaving a partial code. It is now possible to identify all the codewords of weight w compatible with those already in the code, and build a graph from these codewords, where codewords are represented by vertices.

```
SeedBuilding(Time$_{SB}$, BestSol, SelOrd, ItrSeed)
    SeedSet := $\emptyset$;
    ItrCnt := 0;
    While(computation time $<$ Time$_{SB}$)
        ItrCnt := ItrCnt + 1;
        CWord := random codeword compatible with SeedSet;
        WorkSol := SeedSet $\cup$ $\{$CWord$\}$;
        Complete WorkSol by adding feasible binary vectors examined
            according to the criterion SelOrd;
        If(|WorkSol| $>$ |BestSol|)
            BestSol := WorkSol;
            SeedSet := SeedSet $\cup$ $\{$CWord$\}$;
            ItrCnt := 0;
        EndIf;
        If(ItrCnt = ItrSeed)
            AclSeed := Average code size in the last ItrCnt iteration;
            AclAll := Average code size from the beginning;
            If(AclSeed > AclAll)
                BVect := random codeword compatible with SeedSet;
                SeedSet := SeedSet $\cup$ $\{$BVect$\}$;  /*added in last position*/
            Else
                If(|SeedSet| > 0)
                BVect := Codeword in last position of SeedSet;
                SeedSet := SeedSet $\setminus$ BVect;
                EndIf
            EndIf
            ItrCnt := 0;
        EndIf;
    EndWhile;
```

Figure 1: The *Seed Building* algorithm.

Two vertices are connected if and only if the Hamming distance between the codewords is at least d. It is then possible to run a maximum clique algorithm on the graph in order to complete the partial code in the best possible way. Heuristic or exact methods can be used to solve the maximum clique problem. Here the exact algorithm presented by Carraghan and Pardalos [4] is used. It is important that the search remains feasible, so if the computation time is greater than the parameter $Time_{MC}$, the execution is truncated and the largest clique retrieved so far is used.

Pseudo-code for the Clique Search (CS) method is provided in Figure 2. BestSol and WorkSol represent respectively the best solution retrieved so far, and the working solution. Parameter CSRem represents the target percentage of codewords that have to be deleted from the solution WorkSol before it is reconstructed by the maximum clique method described above. In practice the codewords deleted are selected randomly and CSRem is the mean percentage of codewords deleted. The Clique Search algorithm is based on an itera-

```
CliqueSearch(Time$_{CS}$, BestSol, CSRem, Time$_{MC}$)
    While(computation time $<$ Time$_{CS}$)
        WorkSol := BestSol;
        Delete $\lfloor$|WorkSol|$\cdot$ CSRem/100$\rfloor$ random codewords from WorkSol;
        Build the graph $G = \{V, E\}$ where:
            $V$ =$\{$all codewords of weight $w$ compatible with
                the (partial) code WorkSol$\}$;
            $E = \{\{i, j\}|$dist$(i, j) \geq d\}$;
        Let Clique be a solution of the maximum clique problem
            on $G$ corresponding to a set of codewords. The maximum
            computation time is Time$_{MC}$ seconds;
        WorkSol := WorkSol $\cup$ Clique;
        If(|WorkSol| $>$ |Bestsol|)
            BestSol := WorkSol;
        EndIf;
    EndWhile;
```

Figure 2: The *Clique Search* algorithm.

tive mechanism and runs until a given computation time, regulated by parameter Time$_{CS}$, is reached.

2.3 Preliminary computational experiments

In this section a first set of computational experiments is described. These experiments aim to determine the potential of the local search procedures described in Sections 2.1 and 2.2. The algorithms are run on a small, but representative, subset of problems.

All of the algorithms discussed in the present paper have been implemented in ANSI C. The experiments discussed in this section have been run on an Intel Core Duo 2.0 GHz / 1 GB RAM machine.

Table 1 is organized by blocks of rows. The first column is used to identify the problem and the computation time (in seconds) allowed for each algorithm. The computation times have been selected to allow the methods to reach a relatively steady state, where further improvements are unlikely to be found easily. Notice that this time will be assigned to the input parameters Time$_{SB}$ and Time$_{CS}$ of the Seed Building and Clique Search procedures, respectively. The second column describes the method and columns three to five give the results.

For each problem there are five rows, one for each of the following methods:

- RND: Reverse lexicographic search is repeatedly run with a set of between one and eight random seed codewords. This algorithm is run for comparison purposes;
- SB_{REV}: Seed Building with reverse lexicographic order;
- SB_{FWD}: Seed Building with forward lexicographic order;

- SB_{RND}: Seed Building with random order (i.e. a random order for the codewords is fixed and is followed during each search for feasible codewords);
- CS: Clique Search.

Notice that algorithms SB_{REV}, SB_{FWD} and SB_{RND} are variants of the Seed Building procedure obtained with different settings of parameter SelOrd. For each row (algorithm), the average, the best and the worst results for the number of codewords (over ten runs) are reported.

Other parameters are set as follows for the experiments reported in this section: ItrSeed = 20 (Seed Building methods) CSRem = 20 and Time_{MC} = 30 (Clique Search algorithm) for all the problems considered. These values were suggested by preliminary tuning tests.

An examination of Table 1 suggests, first of all, that the Seed Building methodology can help in retrieving better results than algorithm RND. This can be inferred by comparing rows RND and SB_{REV} (which are basically the same method, and differ from each other in the seed selection: random or guided). Further examination suggests that algorithm RND provides in some cases more robust results, i.e. method SB_{REV} has a worse worst case performance. On the other hand, SB_{REV} almost always provides the best maximum result when compared with RND. In terms of average performances, SB_{REV} is almost never worse than RND. This indicates that it might be preferable to have relatively short runs of the Seed Building method instead of a single longer one; diversification should lead to better best results, and therefore to the complete dominance of SB_{REV} over RND. This will be taken into consideration in the design of the algorithm described in Section 3.1.

A second clear indication of Table 1 is that it is impossible to identify any dominance among the Seed Building and Clique Search methods considered (although algorithm SB_{FWD} seems to perform somewhat worse than the other Seed Building methods). This might be seen as disappointing since one cannot concentrate on any one method. On the other hand, this lack of dominance can be useful. From the perspective of a Variable Neighbourhood Search (VNS) method (see, for example, [9]), the presence of different local search methods, equally powerful and with different neighbourhoods can be very useful. Following this logic, in Section 3.1 a way that local searches can be combined together in a VNS framework will be described.

3 A metaheuristic approach

In this section a metaheuristic algorithms is presented that combines the local search procedures described in Section 2 in a Variable Neighbourhood Search framework. Computational experiments comparing the results of the new metaheuristic algorithm with those of the individual local search methods discussed in Section 2, are presented in Section 3.2.

3.1 Variable neighbourhood search

Variable neighbourhood search (VNS) methods have been demonstrated to perform well and are robust (see [9]). Such algorithms work by applying different local search algorithms one

Table 1: Results for local search algorithms.

Problem	Method	Average	Best	Worst
A(29,4,5) time = 1600	RND	3782.2	3796	3774
	SB_{REV}	3782.2	3800	3773
	SB_{FWD}	3737.5	3738	3737
	SB_{RND}	3804.7	3816	3792
	CS	3757.8	3785	3745
A(29,6,5) time = 90	RND	249.4	251	248
	SB_{REV}	251.8	254	251
	SB_{FWD}	247.0	248	246
	SB_{RND}	250.2	253	248
	CS	247.9	252	245
A(29,6,6) time = 1180	RND	866.7	870	864
	SB_{REV}	868.4	872	866
	SB_{FWD}	846.0	847	845
	SB_{RND}	845.3	848	842
	CS	861.8	868	856
A(29,8,5) time = 15	RND	30.0	30	30
	SB_{REV}	30.2	31	30
	SB_{FWD}	30.0	30	30
	SB_{RND}	31.5	32	31
	CS	30.8	32	30
A(29,10,7) time = 180	RND	36.2	37	36
	SB_{REV}	35.5	37	35
	SB_{FWD}	34.0	35	33
	SB_{RND}	35.9	37	35
	CS	36.4	37	35
A(45,4,5) time = 3260	RND	22958.4	23698	21153
	SB_{REV}	23021.2	23678	21591
	SB_{FWD}	19640.1	19666	19627
	SB_{RND}	24803.1	24975	24589
	CS	19917.3	19975	19854
A(45,6,5) time = 190	RND	1018.4	1023	1015
	SB_{REV}	1019.2	1026	1014
	SB_{FWD}	1004.4	1010	1003
	SB_{RND}	1025.5	1037	1014
	CS	1009.8	1012	1008
A(45,6,6) time = 2400	RND	5933.3	5936	5933
	SB_{REV}	5933.3	5936	5933
	SB_{FWD}	5818.3	5824	5817
	SB_{RND}	5751.7	5830	5681
	CS	5935.4	5937	5934
A(45,8,5) time = 40	RND	74.2	75	73
	SB_{REV}	74.5	75	74
	SB_{FWD}	67.1	69	65
	SB_{RND}	77.3	79	76
	CS	52.9	57	48
A(45,10,7) time = 400	RND	169.1	170	169
	SB_{REV}	169.1	170	169
	SB_{FWD}	163.8	167	163
	SB_{RND}	159.4	162	156
	CS	169.1	170	169

```
VNS(Time$_{VNS}$, Time$_{SB}$, Time$_{CS}$, Time$_{MC}$, ItrSeed, CSRem,
        $P_{REV}$, $P_{FWD}$, $P_{RND}$)
    BestSol := $\emptyset$;
    While(computation time $<$ Time$_{VNS}$)
        SelOrd := order for the examination of codewords selected at
            random with probabilities given by $P_{REV}$, $P_{FWD}$ and $P_{RND}$;
        SeedBuilding(Time$_{SB}$, BestSol, SelOrd, ItrSeed);
        CliqueSearch(Time$_{CS}$, BestSol, CSRem, Time$_{MC}$);
    EndWhile;
```

Figure 3: The *VNS* algorithm.

after the other, aiming at differentiating the characteristics of the search-spaces visited (i.e. changing the neighbourhood).

In the current context, there are two families of local searches, with a total of four possible methods. Three of them are Seed Building procedures (as discussed in Section 2), where different orders for the examination of codewords are considered. The fourth one is the Clique Search algorithm. The Seed Building type of algorithm is alternated with the Clique Search algorithm, each time starting from the best solution retrieved since the beginning. The ordering used for Seed Building is selected randomly, according to chosen probability parameters for the orders. The aim is to obtain better solutions than those retrieved by running the single local search procedures alone. The complete metaheuristic algorithm is summarized by the pseudo-code presented in Figure 3. VNS runs for Time$_{VNS}$ seconds.

3.2 Computational experiments comparing VNS with local search

In this section a set of computational experiments is described which aims to achieve an understanding of whether the metaheuristic approach described in this section is capable of improving the results of the local search methods described in Section 2. The benchmarks considered are those already used for the experiments discussed in Section 2.3. The average, the best and the worst results obtained over ten runs are again reported. The experiments discussed in this section have been carried out on an Intel Core Duo 2.0 GHz / 1 GB RAM machine.

Table 2 is organized, in a similar way to Table 1, by blocks of rows. The first column is used to identify the problem, and there are two rows for every problem, one for each of the following methods:

- *Best$_{LS}$*: The best results obtained by all the local search methods discussed in Section 2. Notice that not all the results reported on a single line have necessarily been obtained by the same algorithm. In each case it is the best result obtained. For example, given a problem, the result reported in the column "Best" might have been obtained by a Seed Building method, while that reported in the column "Average" might come from the Clique Search algorithm.

Table 2: Results for metaheuristic algorithms.

Problem	Method	Average	Best	Worst
A(29,4,5)	$Best_{LS}$	3804.7	3816	3792
time = 1600	VNS	3806.6	3832	3787
A(29,6,5)	$Best_{LS}$	251.8	254	251
time = 90	VNS	253.0	254	250
A(29,6,6)	$Best_{LS}$	868.4	872	866
time = 1180	VNS	873.5	881	862
A(29,8,5)	$Best_{LS}$	31.5	32	31
time = 15	VNS	32.3	33	31
A(29,10,7)	$Best_{LS}$	36.4	37	36
time = 180	VNS	36.8	38	36
A(45,4,5)	$Best_{LS}$	24803.1	24975	24589
time = 3260	VNS	24914.9	24975	24587
A(45,6,5)	$Best_{LS}$	1025.5	1037	1015
time = 190	VNS	1035.0	1039	1031
A(45,6,6)	$Best_{LS}$	5935.4	5937	5934
time = 2400	VNS	5956.7	5937	5935
A(45,8,5)	$Best_{LS}$	77.3	79	76
time = 40	VNS	78.7	80	78
A(45,10,7)	$Best_{LS}$	169.1	170	169
time = 400	VNS	169.9	170	169

- VNS: The Variable Neighbourhood Search method presented in Section 3.1.

For each row (algorithm), the average, the best and the worst results over ten runs are reported together with the computation time Time$_{VNS}$ allowed. This time is the same as that already quoted in Table 1. After some tuning experiments, the values of the remaining parameters were selected as ItrSeed = 20 (Seed Building methods) CSRem = 20, Time$_{MC}$ = 30 (Clique Search algorithm), $P_{REV} = 0.4$, $P_{FWD} = 0.3$, and $P_{RND} = 0.3$. For A(29,4,5), A(29,6,6), A(45,4,5) and A(45,6,6) Time$_{SB}$ = Time$_{CS}$ = 150. For A(29,6,5), A(29,10,7), A(45,6,5) and A(45,10,7) Time$_{SB}$ = Time$_{CS}$ = 15. For A(29,8,5) and A(45,8,5) Time$_{SB}$ = Time$_{CS}$ = 3. The analysis of Table 2 shows that in all but worst case results the VNS method outperforms the local search methods, even on the assumption that the best local search method is used.

3.3 Comparison with simulated annealing and tabu search

A comparison can be made of the results of runs of the VNS algorithm (similar to those in Section 3.2) with the simulated annealing and tabu search algorithms presented in [6] and [2] respectively. The results are presented in Table 3. The first column describes the problem and the remaining columns give the number of codewords obtained by the variable neighbourhood search (VNS), simulated annealing (SA) and tabu search (TS) algorithms respectively. The improved performance of the VNS algorithm over simulated annealing and tabu search is apparent.

Table 3: Comparison with simulated annealing and tabu search.

Problem	VNS	SA	TS
A(22,10,9)	32	23	23
A(23,10,7)	19	18	15
A(23,10,8)	29	28	21
A(23,10,9)	41	24	28
A(23,10,11)	56	39	37
A(24,10,8)	35	33	25
A(24,10,9)	52	24	36

4 New tables of constant weight binary codes

The metaheuristic algorithm presented in Section 3 gives substantial improvements to the lower bounds for many cases within the parameter ranges studied in [14] (and also matches or improves some lower bounds which recently appeared in [7, 8]). In the tables which follow the best lower bounds obtained during the development of the local search and VNS algorithms are reported. Typical settings for the VNS algorithm were ItrSeed = 20 (Seed Building) CSRem = 20, Time$_{MC}$ = 30 (Clique Search). Other parameters were typically $P_{REV} = 0.55$, $P_{FWD} = 0.35$, and $P_{RND} = 0.1$ except for the case $w = 8$, $d = 14$ where $P_{REV} = 1.0$ was used. Many of the new best results were obtained by a programme of long runs of the VNS algorithm. For these runs the time values (in seconds) depended on n: Time$_{SB}$ $= 505 \times (n/29)^7$ Time$_{CS}$ $= 505 \times (n/29)^7$ Time$_{VNS}$ $= 16600 \times (n/29)^5$ were used. These very long run times were barely adequate to allow seed building to work effectively for $n \geq 60$ but were probably excessive for many smaller values of n, where the best result was found early in the run. The results have been obtained on a selection of processors with similar characteristics: Dual AMD Opteron 250 2.4GHz with 4GB RAM, Dual Intel Xeon 2.66 GHz with 4GB RAM and Intel Pentium 4 2.5GHz with 1GB RAM.

These results are summarized in Tables 4-14. Problems are identified in the first column of these tables, while the previous lower bounds (Old LB), the new lower bounds (New LB) and the best upper bounds available (UB) are reported in the remaining columns. Bold entries in the (NewLB) column denote new optimal codes. The upper bounds contain some improvements to the upper bounds stated in [14], which simply quoted the first Johnson bound. The value quoted in the current paper is the smallest of (i) the first Johnson bound [3], (ii) the second Johnson bound [3], (iii) equation (18) of [3], (iv) equations (25)-(27) of [1] and (v) Theorems 12 and 13 of [1]. In fact for the parameter sets studied here and in [14], the second Johnson bound is always less than or equal to the other upper bounds except in the 5 cases (n,d,w)=(53,6,5), (47,8,7), (56,10,6), (62,10,7) and (38,10,8). In these cases Theorems 12 and 13 of [1] together with non-existence of certain Steiner systems [3, 1] allow the second Johnson bound to be reduced by 1. The codes corresponding to the new improved lower bounds are available at http://data.research.glam.ac.uk/projects/ [1].

Tables 12 and 14 contain optimal codes. A(30,12,7)=9 was recently noted in [7].

[1]Also available at http://www.idsia.ch/~roberto/BCWC07.zip.

Table 4: Improved constant weight binary codes, A(n,6,5).

Problem	Old LB	New LB	UB
A(41, 6, 5)	755	779	1066
A(42, 6, 5)	817	841	1117
A(43, 6, 5)	874	910	1169
A(44, 6, 5)	941	975	1320
A(45, 6, 5)	1009	1030	1386
A(46, 6, 5)	1097	1114	1444
A(47, 6, 5)	1172	1181	1616
A(48, 6, 5)	1254	1269	1689
A(49, 6, 5)	1343	1347	1764
A(50, 6, 5)	1429	1459	1960
A(51, 6, 5)	1517	1543	2040
A(52, 6, 5)	1617	1654	2121
A(53, 6, 5)	1719	1758	2341
A(54, 6, 5)	1822	1840	2430
A(55, 6, 5)	1936	1948	2519

A(39,14,8)=10, A(41,14,8)=11 and A(42,14,8)=12 were recently noted in [8]. The other ten optimal codes are new. Note that it was erroneously stated in [8] that A(45,14,8)=14 is optimal. In fact the second Johnson bound is 15 and the code found here has 15 codewords.

5 Conclusions

The generation of constant weight binary codes has been considered from an algorithmic perspective. Local search and metaheuristic methods have been presented and discussed. A computational study has shown the potential of the methods.

The novel methodologies produced codes which improved the best known lower bounds for many cases with $29 \leq n \leq 63$. In particular, 10 new optimal codes are obtained which were previously unknown.

References

[1] E. Agrell, A. Vardy, and K. Zeger. Upper bounds for constant-weight codes. *IEEE Transactions on Information Theory*, 46(7):2373–2395, 2000.

[2] J.A. Bland and D.J. Bayliss. Modelling constant weight codes using tabu search. *Appl. Math. Modelling*, 33(1):667–672, 1997.

[3] A.E. Brouwer, J.B. Shearer, N.J.A. Sloane, and W.D. Smith. A new table of constant weight codes. *IEEE Transactions on Information Theory*, 36:1334–1380, 1990.

[4] R. Carraghan and P. Pardalos. An exact algorithm for the maximum clique problem. *Operations Research Letters*, 9:375–382, 1990.

Table 5: Improved constant weight binary codes, A(n,6,6).

Problem	Old LB	New LB	UB
A(32, 6, 6)	1331	1369	2213
A(33, 6, 6)	1528	1560	2706
A(34, 6, 6)	1740	1771	2992
A(35, 6, 6)	1973	1998	3249
A(36, 6, 6)	2240	2264	3906
A(37, 6, 6)	2539	2560	4261
A(38, 6, 6)	2836	2860	4636
A(39, 6, 6)	3167	3208	5479
A(40, 6, 6)	3545	3575	5926
A(41, 6, 6)	3964	3983	6396
A(42, 6, 6)	4397	4419	7462
A(43, 6, 6)	4860	4890	8005
A(44, 6, 6)	5378	5414	8572
A(45, 6, 6)	5933	5959	9900
A(46, 6, 6)	6521	6552	10626
A(47, 6, 6)	7160	7194	11311
A(48, 6, 6)	7845	7869	12928
A(49, 6, 6)	8568	8605	13793
A(50, 6, 6)	9348	9380	14700
A(51, 6, 6)	10175	10210	16660

Table 6: Improved constant weight binary codes, A(n,8,5).

Problem	Old LB	New LB	UB
A(31, 8, 5)	36	39	43
A(32, 8, 5)	38	42	44
A(33, 8, 5)	44	45	52
A(52, 8, 5)	110	114	124
A(54, 8, 5)	117	123	140
A(55, 8, 5)	121	128	143

Table 7: Improved constant weight binary codes, A(n,8,6).

Problem	Old LB	New LB	UB
A(37, 8, 6)	199	212	351
A(38, 8, 6)	222	236	418
A(39, 8, 6)	244	254	442
A(40, 8, 6)	275	281	466
A(41, 8, 6)	294	297	492
A(43, 8, 6)	343	347	602
A(44, 8, 6)	355	381	630
A(45, 8, 6)	381	403	660
A(46, 8, 6)	411	432	759
A(47, 8, 6)	440	463	791
A(48, 8, 6)	477	494	824
A(49, 8, 6)	501	527	857
A(50, 8, 6)	542	567	975
A(51, 8, 6)	576	606	1020
A(52, 8, 6)	609	640	1057
A(53, 8, 6)	650	687	1095
A(54, 8, 6)	682	726	1233
A(55, 8, 6)	729	768	1283
A(56, 8, 6)	766	815	1334
A(57, 8, 6)	830	866	1377
A(58, 8, 6)	872	912	1537
A(59, 8, 6)	935	965	1593
A(60, 8, 6)	982	1019	1650
A(61, 8, 6)	1028	1077	1708
A(62, 8, 6)	1079	1130	1891
A(63, 8, 6)	1143	1195	1953

Table 8: Improved constant weight binary codes, A(n,8,7).

Problem	Old LB	New LB	UB
A(30,8,7)	327	340	681
A(31,8,7)	363	375	885
A(32,8,7)	403	418	992
A(33,8,7)	444	466	1079
A(34,8,7)	498	516	1175
A(35,8,7)	553	570	1470
A(36,8,7)	622	637	1620
A(37,8,7)	696	718	1776
A(38,8,7)	785	795	1905
A(39,8,7)	869	893	2328
A(40,8,7)	977	999	2525
A(41,8,7)	1095	1110	2729
A(42,8,7)	1206	1227	2952
A(43,8,7)	1347	1365	3526
A(44,8,7)	1478	1503	3784
A(45,8,7)	1639	1653	4050
A(46,8,7)	1795	1813	4337
A(47,8,7)	1987	2001	5095
A(48,8,7)	2173	2197	5424
A(49,8,7)	2376	2399	5768
A(50,8,7)	2603	2615	6121
A(51,8,7)	2839	2866	7103
A(52,8,7)	3101	3118	7577
A(53,8,7)	3376	3384	8003
A(54,8,7)	3651	3667	8447
A(55,8,7)	3941	3989	9687
A(56,8,7)	4270	4318	10264
A(59,8,7)	5384	5386	12954

Table 9: Improved constant weight binary codes, A(n,10,6).

Problem	Old LB	New LB	UB
A(34,10,6)	31	32	34
A(45,10,6)	49	50	60
A(48,10,6)	56	57	72
A(49,10,6)	56	59	73
A(50,10,6)	56	62	75
A(51,10,6)	60	64	85
A(52,10,6)	60	67	86
A(53,10,6)	63	70	88
A(54,10,6)	65	73	90
A(55,10,6)	68	73	91
A(56,10,6)	70	79	101
A(57,10,6)	70	83	104
A(58,10,6)	72	85	106
A(59,10,6)	77	87	108
A(60,10,6)	79	91	110
A(61,10,6)	83	94	122
A(62,10,6)	84	98	124

Table 10: Improved constant weight binary codes, A(n,10,7).

Problem	Old LB	New LB	UB
A(29,10,7)	37	39	95
A(36,10,7)	75	78	180
A(42,10,7)	133	137	324
A(56,10,7)	351	358	728
A(57,10,7)	366	374	830
A(58,10,7)	394	399	861
A(59,10,7)	414	423	893
A(60,10,7)	431	449	925
A(61,10,7)	458	474	958
A(62,10,7)	486	497	1079
A(63,10,7)	514	526	1116

Table 11: Improved constant weight binary codes, A(n,10,8).

Problem	Old LB	New LB	UB
A(30,10,8)	92	93	356
A(33,10,8)	134	140	581
A(34,10,8)	156	162	637
A(35,10,8)	176	182	700
A(36,10,8)	198	205	765
A(37,10,8)	223	230	832
A(38,10,8)	249	259	1053
A(39,10,8)	285	291	1135
A(40,10,8)	318	324	1225
A(41,10,8)	353	362	1317
A(42,10,8)	390	398	1412
A(43,10,8)	432	445	1741
A(44,10,8)	484	487	1892
A(45,10,8)	532	544	2013
A(46,10,8)	590	595	2139
A(47,10,8)	642	656	2314
A(48,10,8)	711	720	2778
A(49,10,8)	776	785	2940
A(50,10,8)	852	858	3150
A(51,10,8)	929	934	3321
A(52,10,8)	1007	1018	3549
A(55,10,8)	1289	1296	4661
A(56,10,8)	1405	1408	4949

Table 12: Improved constant weight binary codes, A(n,12,7).

Problem	Old LB	New LB	UB
A(30,12,7)	8	9	9
A(32,12,7)	9	**10**	10
A(33,12,7)	10	**11**	11
A(36,12,7)	15	**16**	16
A(37,12,7)	16	**17**	17
A(39,12,7)	19	21	22
A(40,12,7)	20	22	25

Table 13: Improved constant weight binary codes, A(n,12,8).

Problem	Old LB	New LB	UB
A(37,12,8)	40	42	115
A(38,12,8)	40	45	147

Table 14: Improved constant weight binary codes, A(n,14,8).

Problem	Old LB	New LB	UB
A(39,14,8)	9	10	10
A(41,14,8)	10	11	11
A(42,14,8)	10	12	12
A(43,14,8)	10	**12**	12
A(44,14,8)	12	**13**	13
A(45,14,8)	12	**15**	15
A(46,14,8)	13	**17**	17
A(47,14,8)	15	**18**	18
A(48,14,8)	18	**19**	19

[5] J.H. Conway and N.J.A. Sloane. Lexicographic codes: error-correcting codes from game theory. *IEEE Transactions on Information Theory*, 32:337–348, 1986.

[6] A.A. El Gamal, L.A. Hemachandra, I. Shperling, and V.K. Wei. Using simulated annealing to design good codes. *IEEE Transactions on Information Theory*, 33(1):116–123, 1987.

[7] I. Gashkov, J.A.O. Ekberg, and D. Taub. A geometric approach to finding new lower bounds of $A(n, d, w)$. *Designs, Codes and Cryptography*, 14, 2007.

[8] I. Gashkov and D. Taub. New optimal constant weight codes. *Electronic Journal of Combinatorics*, 14(1), 2007.

[9] P. Hansen and N. Mladenović. Variable neighbourhood search: principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.

[10] O.D. King. Bounds for DNA codes with constant GC-content. *Electronic Journal of Combinatorics*, 10, 2003.

[11] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes.* North-Holland, Amsterdam - New York - Oxford, 1977.

[12] J.N.J. Moon, L.A. Hughes, and D.H. Smith. Assignment of frequency lists in frequency hopping networks. *IEEE Transactions on Vehicular Technology*, 54(3):1147–1159, 2005.

[13] P. Pardalos and J. Xue. The maximum clique problem. *Journal of Global Optimization*, 4(3), 1994.

[14] D.H. Smith, L.A. Hughes, and S. Perkins. A new table of constant weight binary codes of length greater than 28. *Electronic Journal of Combinatorics*, 13(1), 2006.