A Branch and Bound Approach to Permutation Codes

János Barta Roberto Montemanni
Dalle Molle Institute for Artificial Intelligence
University of Applied Sciences of Southern Switzerland
Galleria 2, 6928 Manno, Switzerland
{*janos.barta, roberto.montemanni*}*@supsi.ch*

Derek H. Smith
Division of Mathematics and Statistics
University of South Wales
Pontypridd, CF37 1DL, Wales, UK
derek.smith@southwales.ac.uk

***Abstract*—The Maximum Permutation Code Problem (MPCP) is a well-known combinatorial optimization problem in coding theory. The aim is to generate the largest possible permutation codes, having a given length n and a minimum Hamming distance d between the codewords. In this paper we present a new branch and bound algorithm, which combines combinatorial techniques with an approach based on group orbits. Computational experiments lead to interesting considerations about the use of group orbits for code generation.**

***Index Terms*—Coding theory; permutation codes; combinatorial optimization.**

I. Introduction

Permutation codes have received considerable attention in recent years, both for their intrinsic interest and because of potential applications to power line communications [1], [2], [3], [4], [5], [6], where permutations are used to ensure that power output remains as constant as possible. As well as white Gaussian noise the codes must combat permanent narrow band noise from electrical equipment or magnetic fields, and impulse noise. Most of the approaches presented in the literature are based on linear programming [7], [8] or on group theory ideas [7], [9], [10], [11], which have more recently been amalgamated with optimization, mainly based on search techniques [12], [13], [14].

Starting from a classical branch and bound approach, in this paper we develop a new exact algorithm for solving the MPCP problem. In addition, we present some techniques for computing upper bounds, which exploit the strong symmetries of permutation groups.

II. The maximum permutation code problem

We call any permutation of the n-tuple $x_0 = [0, 1, ..., n-1] \in \mathbf{N}^n$ a *codeword* of length n and we denote the set of all codewords of length n by Ω_n. From an algebraic point of view the set Ω_n is the orbit of the symmetric group of permutations S_n, i.e.

$$\Omega_n = \{x \in \mathbf{N}^n | x = gx_0,\ g \in S_n\} \tag{1}$$

We call any subset Γ of Ω_n a *permutation code*. The ability of a permutation code to correct errors depends on the minimum Hamming distance between the elements of the code. The Hamming distance $d(x, y)$ between codewords x and y is the number of components that differ in the two codewords. For any code $\Gamma \subseteq \Omega_n$, containing at least two distinct codewords, we define the *code distance* as

$$\delta(\Gamma) = \min_{x,y \in \Gamma, x \neq y} d(x, y). \tag{2}$$

After these preliminary definitions we are ready to formulate the MPCP problem.

Definition 1. Given a codeword length n and a Hamming distance d, the maximum permutation code problem MPCP consists in determining a largest code $\Gamma \subseteq \Omega_n$ with $\delta(\Gamma) \geq d$.

For simplicity in the remainder of the paper we refer to a specific MPCP problem as an (n, d)-problem and we denote the corresponding maximum number of codewords by $M(n, d)$.

As an example, in the $(6, 5)$-problem we are looking for a maximum code Γ of length $n = 6$ with a code distance $\delta(\Gamma) \geq 5$. As reported in [12] the optimal solution of this problem is $M(6, 5) = 18$ and one of the many possible optimal codes is

$$\begin{aligned}\Gamma = \{&[012345], [021453], [034512], [045231], [102534],\\ &[130425], [153240], [205143], [243015], [251304],\\ &[310254], [324105], [341520], [425310], [432051],\\ &[450132], [504321], [513402]\}.\end{aligned}$$

III. Code generation with permutation groups

A. Code generator subgroups

As explained in [3] and [12] an MPCP problem can be transformed into an equivalent maximum clique problem. In fact, with any (n, d)-problem we can associate a graph $G(n, d)$, where the set of nodes is Ω_n and codewords x and y are connected by an arc, iff $d(x, y) \geq d$. It is easy to see that any clique of the graph $G(n, d)$ provides a feasible code and maximum cliques correspond to maximum permutation codes. However, state of the art maximum clique algorithms (see for instance [15] and [16]) are typically able to handle problems of this type with up to 1000 nodes in a reasonable computational time. These problem correspond to (n, d)-problems with $n \leq 6$.

Therefore, we have to face the issue of generating solutions of larger (n, d)-problems. A possible approach is to reduce the problem size by combining adequately defined subsets of

978-1-4799-3580-2/14/$31.00 ©2014 IEEE

Ω_n, instead of single codewords. In this manner the number of elements to be combined can be reduced, as well as the size of the corresponding clique problem. On the other hand, since not all possible combinations of codewords are considered, in general optimality can not be guaranteed. Suitable subsets aggregating single codewords can be defined as follows.

Definition 2. Given an (n, d)-problem P, we call a set $T \subseteq \Omega_n$ with $|T| = k$ a $k-$code of problem P, if there is no pair of distinct elements $x, y \in T$ with $d(x, y) < d$.

In other words a k-code is a feasible, but not necessarily optimal code of an (n, d)-problem, including also the trivial case of $1-$codes, composed of single codewords. In order to combine k-codes to form larger codes the following compatibility constraint must hold.

Definition 3. A k_1-code T_1 and a k_2-code T_2 of an (n, d)-problem are *compatible*, if

$$d(T_1, T_2) := \min_{x_1 \in T_1, x_2 \in T_2} d(x_1, x_2) \geq d. \tag{3}$$

It is easy to see that the union $(T_1 \cup T_2)$ of a k_1-code T_1 and a compatible k_2-code T_2 provides a $(k_1 + k_2)$-code.

An intensively investigated approach for generating collections of k-codes for (n, d)-problems is the computation of suitable group orbits (see for instance [3] and [12]). For this purpose it is necessary to identify the subgroups of S_n whose orbits are k-codes of the (n, d)-problem P.

Definition 4. Given an (n, d)-problem P, we refer to a subgroup H of S_n as a *code generator group* for problem P, if the orbits of H are k-codes of problem P.

Basically, the use of groups for generating k-codes is suggested by several well-known properties of group orbits. In particular, the orbits of any permutation group H have cardinality $|H|$, are disjoint and form a partition of Ω_n. Furthermore, the orbits of a given permutation group are always *isomorphic*, in the sense that they have the same Hamming distance matrix, if the right ordering of codewords in the orbits is chosen. Due to these properties, the sets of k-codes obtained via permutation groups have remarkable symmetries and, as shown in [3] and [12], in many cases the combination of compatible orbits of a fixed code generator group leads to best or even optimal solutions.

However, it is interesting to remark that in [12] only the 2-3 largest available code generator subgroups are taken into account and solutions are always obtained by combining compatible orbits of one single subgroup. In the next sections we will investigate alternative strategies, such as combining shorter group orbits or even single codewords.

B. The case of MPCP problems (5,5) and (6,5)

In order to obtain a complete list of all code generators of (n, d)-problems, we implemented a procedure in the GAP[1]-language, that identifies all subgroups whose orbits are k-codes.

[1]http://www.gap-system.org

Table I shows a list of (n, d)-problems with the corresponding total number of subgroups of S_n (column 2), the number of subgroup conjugacy classes (column 3), the total number of code generator subgroups (column 4) and the number of code generator conjugacy classes (column 5).

Essentially, Table I suggests that although the total number of subgroups of S_n increases rapidly, subgroups can be classified in a relatively small number of conjugacy classes. The fact that conjugate groups are isomorphic has the important consequence that the sets of orbits of groups belonging to the same conjugacy class are isomorphic too. In other terms, conjugate groups preserve not only the distance structure of single orbits, but also the distance structure between orbits. This fact relies on the fundamental property of the Hamming distance $d(gx, gy) = d(x, y),\ \forall g \in S_n, x, y \in \Omega_n$. If the groups H and L are conjugate, it holds $L = gHg^{-1}$ for some $g \in S_n$. Furthermore, it can be proved that if O_x^H is the H-orbit of $x \in \Omega_n$, its g-permutation corresponds to the L-orbit of gx, that is $g(O_x^H) = O_{gx}^L$. In other terms we obtain the L-orbits simply by applying the permutation g on the H-orbits.

Therefore, if an exact algorithm computes a largest code by combining the orbits of a code generator subgroup H_1, it will necessarily obtain a largest code of the same length by combining the orbits of a conjugate subgroup H_2. In other words, if we look for best codes by combining orbits of single code generator subgroups, it suffices to check one representative for each conjugacy class.

The second line of Table I shows that S_5 has 156 subgroups, divided in 19 conjugacy classes. However, problem $(5, 5)$ has only 7 code generator subgroups, belonging to 2 conjugacy classes. A more detailed analysis of the 7 code generators, reported in Table II, reveals that 6 of them are isomorphic to the cyclic group C_5 and belong to the same conjugacy class. Each of them generates 24 orbits of length 5 in Ω_5. Since the optimum of problem $(5, 5)$ is $M(5, 5) = 5$, the 6 C_5-subgroups provide a set of 144 maximum permutation codes for problem $(5, 5)$. It is worth remarking that in the list of code generators we always include the trivial case of the 1-element identity group I, whose orbits are single codewords. At this point the question arises, whether the 144 orbits obtained via C_5-subgroups are all maximum codes of problem $(5, 5)$ or not. In Section V we will show that there are many other 5-codes for problem $(5, 5)$, which can only be obtained as a combination of the 1-element orbits of the identity group I.

Another extremely interesting case is problem $(6, 5)$. As reported in Table I, among the 56 subgroup conjugacy classes there are only 6 classes of code generators for problem $(6, 5)$. Table II shows for each class of code generators the corresponding orbit length (column 3), the number of orbits per group (column 4), the number of groups in the class (column 5) and the group structure (column 6). Since the maximum code length for problem $(6, 5)$ is $M(6, 5) = 18$ (see for instance [12]), looking at Table II we can draw the conclusion that the maximum codes of problem $(6, 5)$ can not be single group orbits. In fact, the largest code generator subgroups belong to class 1 and 2 with an orbit length 6.

TABLE I
NUMBER OF CODE GENERATOR SUBGROUPS OF (n, d)-PROBLEMS

n	d	Subgroups	Conj. classes	Code gen.	Code gen. classes
5	4	156	19	54	7
5	5	156	19	7	2
6	4	1455	56	701	23
6	5	1455	56	152	6
7	4	11300	96	5075	32
7	5	11300	96	1538	11
8	4	151221	296	78809	128
8	5	151221	296	29350	37

TABLE II
(n, d)-PROBLEMS AND THEIR CODE GENERATOR CLASSES

n	d	Class	Orb.length	# Orbits	# Groups	Structure
5	5	1	5	24	6	C_5
		2	1	120	1	I
6	5	1	6	120	60	C_6
		2	6	120	20	S_3
		3	5	144	36	C_5
		4	3	240	20	C_3
		5	2	360	15	C_2
		6	1	720	1	I

However, it is interesting that no maximum code can be constructed by unifying 3 orbits of any of these 6-element subgroups. Obviously class 3 subgroups are not suitable for generating maximum codes, because the orbits have length 5 and 5 does not divide $M(6, 5)$. This example suggests that in general large subgroups are not the best choice for code generation. The main advantage of large subgroups is the small number of orbits, which reduces the size of the combinatorial problem. On the other side, short orbits offer more chances of suitable combinations.

IV. A BRANCH AND BOUND ALGORITHM FOR PERMUTATION CODES

A. Detailed description

In this section we present a new branch and bound algorithm (BBcodes) for computing a best code of an (n, d)-problem by combining compatible elements of a given pool $\mathcal{C}$ of k-codes, where k has a fixed value. The algorithm is conceived in such a way that it can handle any pool of k-codes, as for instance collections of group orbits, of single codewords or arbitrary lists of k-codes. Algorithm BBcodes builds and visits a binary search tree, and each search tree node q can be identified by the following elements:

- $In(q)$: set of compatible k-codes of the pool $\mathcal{C}$, that any solution associated with the search subtree rooted at q must contain. We denote by $\Gamma(q)$ the code made up of the k-codes in set $In(q)$ and by $|In(q)|$ the number of k-codes in set $In(q)$.
- $Rem(q)$: remainder set of k-codes, which are compatible with $\Gamma(q)$, but not yet processed. These elements can be chosen for the next branching step.

The algorithm starts by generating the root r of the search tree, with set $Rem(r)$ containing all k-codes of the pool $\mathcal{C}$ and $In(r) = \emptyset$. We denote by Q the set of the open search tree nodes. The algorithm terminates as soon as $Q = \emptyset$.

Essentially, algorithm BBcodes processes an open search tree node $q \in Q$ by computing a lower bound $LB(q)$ and an upper bound $UB(q)$ of the maximum code length on the subtree rooted at q. The next search tree node can be selected by means of different strategies. We implemented both, a breadth-first and a depth-first strategy.

The bounds are calculated as follows. If node q is a leaf of the search tree, that is $Rem(q) = \emptyset$, the lower and the upper bound coincide with $|In(q)|$. Otherwise, we determine a lower bound for node q by running a maximum clique algorithm (MC) on set $Rem(q)$. In other terms, algorithm MC unifies as many compatible k-codes of $Rem(q)$ as possible in a given maximum CPU time $Tmax$. For this purpose any maximum clique algorithm can be used. In our implementation we apply the algorithm proposed in [17]. Let M be the maximum number of k-codes provided by MC. We obtain a lower bound $LB(q) := |In(q)| + M$, because we can unify the k-codes belonging to $In(q)$ with the best solution found by MC in set $Rem(q)$. In the case that MC terminates within the maximum time $Tmax$, we get a tight upper bound $UB(q) = LB(q)$. Otherwise we set as a default upper bound $UB(q) := |In(q)| + |Rem(q)|$.

At this point algorithm BBcodes checks, whether the lower bound $LB(q)$ improves the current best value $LBbest$ and in this case updates it. Furthermore, if node q is dominated, that is $UB(q) \leq LBbest$, it is pruned from set Q.

If node q is not dominated, a branching step is carried out. Basically, an element $T \in Rem(q)$ is chosen (in our implementation in a lexicographic order) and two new search tree nodes q' and q'' are generated: the first one includes T in the solution, the second one excludes it.

It is clear that if an element T is included in $In(q')$, all k-codes $T' \in Rem(q)$ with $d(T, T') < d$ can be eliminated, because they are not compatible with T. Therefore it is useful to compute the neighbourhood $N(T) = \{T' \in \mathcal{C} | d(T, T') < d\}$ of a k-code T. Formally, the new search tree nodes are characterized by $In(q') := In(q) \cup \{T\}$, $Rem(q') := Rem(q) \setminus N(T)$, $In(q'') := In(q)$ and $Rem(q'') := Rem(q) \setminus \{T\}$. Finally, the new nodes q' and q'' are added to Q and the processed node q is deleted.

B. Pseudocode of algorithm BBcodes

The algorithm BBcodes can be formalized in the following pseudocode:

Step 0. *Root generation*
Generate the root r of the search tree and insert it in the set of search tree nodes to be visited. Set $Q := \{r\}$, $In(r) := \emptyset$, $Rem(r) := \mathcal{C}$, $LBbest := 0$. Go to Step 1.

Step 1. *Bounding*
If $Q = \emptyset$, the search is terminated. The optimal code length is $LBbest$. Otherwise choose the next search tree node $q \in Q$ to be processed (according to Section IV-A).
If $Rem(q) = \emptyset$, a leaf has been reached. Set $LB(q) := UB(q) := |In(q)|$ and go to Step 2.

Otherwise run a maxclique algorithm (MC) on set $Rem(q)$ for a fixed maximum time $Tmax$ (see Section IV-A). Let M be the maximum obtained via algorithm MC.
Lower bound: $LB(q) := |In(q)| + M$.
Upper bound: if MC terminates within $Tmax$: $UB(q) := LB(q)$. Otherwise $UB(q) := |In(q)| + |Rem(q)|$. Go to Step 2.

Step 2. *Update and pruning*
Update: if $LB(q) > LBbest$, update $LBbest := LB(q)$.
Pruning: if $UB(q) \leq LBbest$, prune dominated node q, set $Q := Q \setminus \{q\}$ and go to Step 1. Otherwise go to Step 3.

Step 3. *Branching*
According to Section IV-A select a k-code $T \in Rem(q)$ for the branching step and compute its neighbourhood $N(T)$. Generate two new search tree nodes q' and q'' with $In(q') := In(q) \cup \{T\}$, $Rem(q') := Rem(q) \setminus N(T)$, whereas $In(q'') := In(q)$ and $Rem(q'') := Rem(q) \setminus \{T\}$. Update search tree nodes set $Q := Q \cup \{q', q''\} \setminus \{q\}$. Go to Step 1.

C. Calculation of upper bounds for the case $k = 1$

Choosing the 1-element identity group I as a code generator subgroup, means that algorithm BBcodes is run with the pool of single codewords $\mathcal{C} = \Omega_n$. This is an important special case, because if the algorithm terminates, it necessarily provides the optimal solution of the (n, d)-problem considered. On the other hand, as it will appear in the next section, due to the size of the pool $\mathcal{C}$ the execution of the algorithm becomes time consuming even for small values of n.

The algorithm BBcodes described in Section IV-A provides only the naive upper bound $UB(q) := |In(q)| + |Rem(q)|$, when the maximum clique procedure MC does not terminate. In the final version of the algorithm we integrated the calculation of a tighter upper bound.

The upper bound that we implemented is based on the idea that Ω_n can be partitioned in n subsets $S_0, ..., S_{n-1}$, in such a way that for a fixed $t \in \{0, ..., n-1\}$ we define $S_i = \{x \in \Omega_n | x(t) = i\}$. In other terms, S_i contains all codewords with the t-th component having the value i. Since the partition is obtained by fixing the value of one component of the codewords, it is clear that the sets S_i are isomorphic to Ω_{n-1}. Furthermore, as the sets S_i form a partition of Ω_n it is well-known that an upper bound of $M(n, d)$ can be obtained by summing up the upper bounds on the subset S_i (see for instance [8]), i.e.

$$M(n, d) \leq n \cdot M(n-1, d). \tag{4}$$

It is interesting to remark that by means of equation (4) the best upper bound of problem $(7, 5)$ ($M(7, 5) \leq 140$, reported in [12]) can be improved to $M(7, 5) \leq 7 \cdot 18 = 126$.

Essentially, we added to the algorithm BBcodes a function that computes an upper bound on the set of codewords $Rem(q)$, whenever the maximum clique algorithm MC does not terminate. In this case a partition $S_0, ..., S_{n-1}$ of the set $Rem(q)$ is generated, such that $S_i = \{x \in Rem(q) | x(t) = i\}$. For each subset S_i an upper bound $UB(S_i)$ is calculated by applying algorithm MC or by means of the known upper bound of $M(n-1, d)$. The upper bound for the search tree node q becomes

$$UB(q) = |In(q)| + \sum_{i=0}^{n-1} UB(S_i). \tag{5}$$

As the index t of the fixed component in the codewords can be varied, there are n different partitions that can be generated. Algorithm BBcodes computes for each partition an upper bound and finally chooses the lowest one.

V. Computational experiments

We present some preliminary results obtained with the branch and bound method BBcodes. The algorithm has been encoded in ANSI C and all the tests reported in this section have been carried out on a computer equipped with an Intel Core *i*5 2.3 GHz processor and 8 GB of memory.

The first issue addressed with algorithm BBcodes is the generation of permutation codes for (n, d)-problems by combining the orbits of a given code generator subgroup. In particular, our aim is to compare the performance of the largest code generator subgroups with smaller subgroups. As a first step we computed in GAP the complete list of the conjugacy classes of code generator subgroups for the (n, d)-problems considered. For each conjugacy class the set of the orbits of a given representative has been generated. Then algorithm BBcodes has been run, choosing different sets of orbits as pools of k-codes. Table III reports the results obtained on (n, d)-problems $(6, 5)$ and $(7, 5)$ using a depth-first version of algorithm BBcodes. Each row of the table corresponds to a run of the algorithm BBcodes on a pool $\mathcal{C}$ of k-codes obtained from a given code generator subgroup of the (n, d)-problem considered.

Column 3 and 4 report the length of the k-codes, that is the orbit length and the number of k-codes in the pool $\mathcal{C}$, i.e. the number of orbits of the code generator. Column 5 shows the length of the best code obtained during the execution of the algorithm. For instance, problem $(6, 5)$ has a lower bound of 6 in the first conjugacy class, which means that the best code contains only one orbit of the pool. Similarly, the lower bound 10 in the third row indicates that the solution is formed by 2 compatible 5-codes. Columns 6-8 are performance indicators of the branch and bound algorithm: the percentage of dominated or solved search tree nodes, the CPU time required to obtain the best lower bound and, finally, the CPU time needed to terminate the branch and bound procedure. If algorithm BBcodes terminates, the best lower bound corresponds to an optimum. Whereas the entry '-' in the last column indicates that the algorithm did not terminate in the maximum allowed computation time of 300 seconds.

The results on problem $(6, 5)$ show that the best lower bounds are provided by small subgroups and that optimal solution can be obtained only by combining 1-codes, i.e. single codewords. However, it is clear that the execution of the algorithm BBcodes with short orbits is more time consuming, as the size of the pool $\mathcal{C}$ increases. All lower bounds of problem $(6, 5)$ are optimal values, because the

TABLE III
ALGORITHM BBCODES, LOWER BOUNDS USING ORBIT SETS

n	d	Class	k	$\|\mathcal{C}\|$	Best LB	Pruned (%)	Sec (LB)	Sec (End)
6	5	1	6	120	6	100	0.00	0.00
		2	6	120	6	100	0.00	0.00
		3	5	144	10	100	0.01	0.01
		4	3	240	12	100	0.02	0.02
		5	2	360	16	100	1.21	1.21
		6	1	720	18	49	2.92	-
7	5	1	42	120	42	100	0.00	0.00
		2	21	240	42	100	0.00	0.01
		3	14	360	42	100	0.01	0.01
		4	10	504	40	50	0.01	0.89
		5	7	720	77	50	0.22	30.85
		6	6	840	42	49	1.06	-
		7	6	840	42	49	0.85	-
		8	5	1008	65	49	2.34	-
		9	3	1680	66	47	2.32	-
		10	2	2520	64	45	104.36	-
		11	1	5040	59	39	46.25	-

TABLE IV
ALGORITHM BBCODES, CASE $k = 1$

n	d	Gap (%)	Solved (%)	Pruned (%)	Sec (LB)	Sec (End)
5	3	0	4	47	0.02	0.03
5	4	0	28	28	0.01	0.01
6	3	0	0	49	0.39	0.79
6	4	37	0	49	53.50	-
6	5	0	22	27	0.45	-
6	6	0	1	49	0.00	0.22
7	3	0	0	49	2.97	6.00
7	4	22	14	3	95.56	-
7	5	23	10	39	49.45	-
7	6	0	34	15	0.86	0.92
7	7	0	4	45	0.02	-

algorithm terminated for all subgroups except for $k = 1$, but in this last case the optimal value 18 was reached.

Problem $(7, 5)$ is solved for the largest subgroups (conjugacy classes 1-5) and it is interesting to remark that the best lower bound 77 is provided by the pool of 7-codes of class 5. However, it can be expected that the lower bounds of the smallest code generators (classes 6-11) underestimate the optimum, because the algorithm did not terminate in the allowed computation time.

Looking at columns 7 and 8 it is interesting to remark that usually the algorithm finds quickly the best solution, but it requires a high computational effort for terminating the binary search, due to a tailing effect on the branches q'' of the search tree, that is the nodes which exclude given k-codes. The smallest instances, having 100% of pruned nodes, are solved directly by the maximum clique algorithm integrated in BBcodes within the allowed time $Tmax = 2$ seconds.

As already mentioned in Section IV, in the case of 1-codes the execution of the algorithm BBcodes becomes time consuming and often does not terminate. Table IV shows the performance of the depth-first version of algorithm BBcodes on several (n, d)-problems. In this test we set a threshold of 50 codewords for the maximum size of the problems solved by the maximum clique algorithm MC. The table reports the percentage gap between the lower bound of BBcodes and the best known lower bound (column 2), the percentage of the search tree nodes solved by MC (column 3), the percentage of the nodes pruned by means of the upper bound based on partitioning (column 4) and finally the computation times (columns 5 and 6). Again, the entry '-' means that the maximum allowed time of 300 seconds was reached.

The percentage of pruned nodes reported in Table IV shows that the upper bounds provided by the partitioning method are useful for identifying dominated nodes. Moreover, it can be observed that usually a best lower bound with a low gap is obtained in a short time, but there are some cases (like problems $(6, 4)$, $(7, 4)$ and $(7, 5)$), whose lower bounds improve slowly and therefore they require long computation times. An important characteristic of an (n, d)-problem seems to be its total number of maximum codes. There are problems having only few optimal solutions. In these cases the choice of the starting configuration of codewords becomes crucial. In other cases, like for instance problem $(6, 5)$, there are thousands of maximum codes and the algorithm BBcodes has no difficulty to identify one of them.

Since the branch and bound algorithm BBcodes carries out an exhaustive search, it can be easily modified in order to collect all optimal solutions of a given (n, d)-problem, whose optimal value $M(n, d)$ is known. An interesting experiment on problem $(5, 5)$ revealed that it has 1344 different maximum codes of length 5. As already mentioned in Section III-B only 144 of them are C_5-orbits. This result suggests that in some cases there may exist optimal solutions that can not be obtained as a combination of large subgroup orbits.

VI. CONCLUSION

An exact algorithm for the solution of the maximum permutation code problem has been presented in this paper. The algorithm proposed is a branch and bound method applied on suitable sets of group orbits. Computational experiments show that the orbits of large subgroups of S_n are not always the best choice for generating maximum permutation codes. Furthermore, for the special case of single codewords ($k = 1$) a useful technique for computing upper bounds has been presented.

REFERENCES

[1] R.F. Bailey. "Error-correcting codes from permutation groups," Discrete Mathematics, vol. 309, pp. 4253–4265, 2009.
[2] I.F. Blake. "Permutation codes for discrete channels," IEEE Transactions on Information Theory, vol. 20, no. 1, pp. 138–140, 1974.
[3] W. Chu, C.J. Colbourn and P. Dukes. "Constructions for permutation codes in powerline communications," Designs, Codes and Cryptography, vol. 32, pp. 51–64, 2004.
[4] C.J. Colbourn, T. Kløve and A.C.H. Ling. "Permutation arrays for powerline communication and mutually orthogonal latin squares," IEEE Transactions on Information Theory, vol. 50, pp. 1289–1291, 2004.
[5] N. Pavlidou, A.J. Han Vinck, J. Yazdani and B. Honary. "Power line communications: state of the art and future trends," IEEE Communications Magazine, vol. 41, no. 4, pp. 34–40, 2003.
[6] J. Quistorff. "A survey on packing and covering problems in the Hamming permutation space," Electronic Journal of Combinatorics, vol. 13, #A1, 2006.

[7] Bogaerts M.: New upper bounds for the size of permutation codes via linear programming. The Electronic Journal of Combinatorics **17**(#R135) (2010).
[8] H. Tarnanen. "Upper bounds on permutation codes via linear programming," European Journal of Combinatorics, vol. 20, pp. 101–114, 1999.
[9] M. Deza and S.A. Vanstone. "Bounds for permutation arrays," Journal of Statistical Planning and Inference, vol. 2, pp. 197–209, 1978.
[10] P. Dukes and N. Sawchuck. "Bounds on permutation codes of distance four," Jourbal of Algebriac Combinatorics, vol. 31, pp. 143–158, 2010.
[11] P. Frankl and M. Deza. "On maximal numbers of permutations with given maximal or minimal distance," Journal of Combinatorial Theory Series A, vol. 22, pp. 352–260, 1977.
[12] D.H. Smith and R. Montemanni. "A new table of permutation codes," Designs, Codes and Cryptography, vol. 63, no. 2, pp. 241–253, 2011.
[13] D.H. Smith and R. Montemanni. "Permutation codes with specified packing radius," Designs, Codes and Cryptography, vol. 69, no. 1, pp. 95–106, 2013.
[14] I. Janiszczak, W. Lempken, P.R.J. Östergård and R. Staszewski. "Permutation codes invariant under isometries," Designs, Codes and Cryptography, to appear.
[15] P.R.J. Östergård. "A new algorithm for the maximum-weight clique problem," Nordic Journal of Computing, vol. 8, no. 4, pp. 424–436, 2001.
[16] P.R.J. Östergård. "A fast algorithm for the maximum clique problem," Discrete Applied Math, vol. 120, pp. 197–207, 2002.
[17] R. Carraghan and P.M. Pardalos. "An exact algorithm for the maximum clique problem," Operations Research Letters, vol. 9, pp. 375–382, 1990.

978-1-4799-3580-2/14/$31.00 ©2014 IEEE