

Small Covering Designs by Branch-and-Cut

François Margot
University of Kentucky
Lexington, KY 40506-0027
fmargot@ms.uky.edu

June, 2000
(revised July, 2001)

Abstract

A Branch-and-Cut algorithm for finding covering designs is presented. Its originality resides in the use of isomorphism pruning of the enumeration tree. A proof that no $4 - (10, 5, 1)$ -covering design with less than 51 sets exists is obtained together with all non isomorphic $4 - (10, 5, 1)$ -covering designs with 51 sets.

1 Introduction

Let V be a set of elements of cardinality v and let k and t be integers such that $v \geq k \geq t \geq 0$. Let \mathcal{K} be the set of all k -subsets of V and \mathcal{T} be the set of all t -subsets of V . For $\lambda \geq 1$, a $t - (v, k, \lambda)$ -covering design is a collection \mathcal{C} of sets in \mathcal{K} such that each set in \mathcal{T} is contained in at least λ sets of \mathcal{C} . A $t - (v, k, \lambda)$ -covering design \mathcal{C} is *minimum* if the cardinality of \mathcal{C} is as small as possible. This cardinality, the *covering number*, is denoted by $C_\lambda(v, k, t)$.

Covering designs have a long history and have applications in statistics, coding theory and combinatorics, among others. Numerous theorems give the value of a minimal covering design under certain assumptions on the parameters (see the surveys [13, 21]). For small values of v , complete or implicit enumeration algorithms were used to generate solutions [20]. Heuristic construction algorithms have also been developed to deal with larger values of the parameters (see [14, 15, 16] for example).

Yet, for particular values of the parameters, only lower and upper bounds are available. According to the tables in [13, 14, 21], the following covering numbers for $v \leq 11$ are open: $4 - (10, 5, 1)$, $4 - (11, 6, 1)$, $5 - (11, 6, 1)$, $5 - (11, 7, 1)$. A case point is the $4 - (10, 5, 1)$ -covering design, for which a lower bound of 50 and an upper bound of 51 are known [1, 13, 14, 21].

In this paper, we present a $(0, 1)$ integer linear programming Branch-and-Cut (B&C) approach to the covering design problem for the case $\lambda = 1$. We assume familiarity with the

B&C procedure, as an excellent introduction can be found in [17, 22, 23]. We were able to show that the minimum $4 - (10, 5, 1)$ -covering design has cardinality 51 with a surprisingly small enumeration tree (about 350 nodes) and computing time (about 85 seconds cpu time on a workstation). All covering designs problems with $v \leq 10$ are efficiently solved by this approach ($4 - (10, 5, 1)$ being the most difficult by far), but the open problems with $v = 11$ mentioned above remain unsolved.

The original feature of this B&C approach is the use of isomorphism rejection for pruning the enumeration tree and generating cuts. While isomorphism rejection in backtracking searches has been used in many applications [2, 5, 9, 10, 18, 19], it is not commonly used in a Branch-and-Cut context.

In Section 2, we give a classical formulation of the covering design problem in integer linear programming form. In Section 3, we describe the isomorphism rejection procedure. It is based on a simple lexicographic observation similar to the property of canonical factorizations studied by Seah and Stinson [19] and by Ivanov [5] (see [11, 18] for generalizations). In Section 5, we present the Schönheim bound and related linear inequalities as well as a class of isomorphism inequalities and a separation algorithm. Other details of the algorithm such as branching and exploration strategies are dealt with in Sections 4 and 6. In Section 7, the complete list of non isomorphic minimum $4 - (10, 5, 1)$ -covering designs is given.

To illustrate the general description of the algorithm, we give, at several points in the paper, numbers related to the $4 - (10, 5, 1)$ -covering design problem. If not specified otherwise, these numbers correspond to a run of the B&C using all tools (cuts, pools, isomorphism rejection) and using an upper bound of 51. Since the objective value is an integer, this means that all nodes with a bound strictly larger than 50 are pruned.

2 Integer linear programming formulation

The covering design problem has a natural formulation as an integer linear program (ILP). Let A be a $|\mathcal{T}| \times |\mathcal{K}|$ matrix defined by

$$A_{i,j} = \begin{cases} 1 & \text{if } t\text{-set } i \text{ is contained in } k\text{-set } j, \\ 0 & \text{otherwise.} \end{cases}$$

The problem may thus be formulated as

$$\begin{aligned} \min \quad & \mathbf{1}^T \cdot x & (2.1) \\ \text{s.t.} \quad & Ax \geq \lambda \mathbf{1}, \\ & x \in \{0, \dots, \lambda\}^{|\mathcal{K}|}, \end{aligned}$$

where $\mathbf{1}$ is a vector of all 1's with appropriate dimension. In the remainder of the paper, we will only consider the case $\lambda = 1$, although slight modifications in the presentation would cover the general case. The ILP (2.1) is then a classical $(0, 1)$ covering problem. Standard

ILP solvers will be able to solve this covering design problem only for small values of the parameter v . The difficulty comes from the fact that the LP bound obtained by relaxing the integrality constraints on x is weak and, more importantly, the fact that the matrix A has a large automorphism group. This implies that the underlying polyhedron has a large symmetry group, making cutting and branching procedures inefficient.

To simplify the presentation in the remainder of the paper, it is assumed that the k -sets in \mathcal{K} are lexicographically ordered, starting with the largest such set. For example, if we consider the $2 - (5, 3, 1)$ -covering design problem, the ten 3-sets are ordered as follows:

$$\begin{array}{ll}
 1: & (1 \ 1 \ 1 \ 0 \ 0) \\
 2: & (1 \ 1 \ 0 \ 1 \ 0) \\
 3: & (1 \ 1 \ 0 \ 0 \ 1) \\
 4: & (1 \ 0 \ 1 \ 1 \ 0) \\
 5: & (1 \ 0 \ 1 \ 0 \ 1) \\
 6: & (1 \ 0 \ 0 \ 1 \ 1) \\
 7: & (0 \ 1 \ 1 \ 1 \ 0) \\
 8: & (0 \ 1 \ 1 \ 0 \ 1) \\
 9: & (0 \ 1 \ 0 \ 1 \ 1) \\
 10: & (0 \ 0 \ 1 \ 1 \ 1)
 \end{array}$$

By abuse of notation, k -sets are identified by their index s in this ordering or by the corresponding variable x_s in the ILP.

4-(10, 5, 1)-covering design: A is a 210×252 matrix. The LP bound obtained by relaxing the integrality constraint on x in (2.1) is 42. The smallest known $4 - (10, 5, 1)$ -covering design has 51 sets [1].

3 Isomorphism test

The proposed B&C will branch by fixing the value of one variable x_j to 0 or 1. Since A has a large automorphism group, it is very likely that several nodes in the enumeration tree will correspond to isomorphic problems. Obviously, solving one of these isomorphic problems and pruning the others would result in huge savings. One important goal is to do so without having to keep in memory the list of all non isomorphic subproblems encountered since the start of the algorithm. One way to achieve this is to define, for each isomorphism class of subproblems, one particular subproblem (called *representative* of the class) that will be solved. Given a subproblem, we then just need to be able to decide if it is a representative or not and, in the latter case, we can prune the corresponding node of the B&C. Some care must be taken to ensure that the representative subproblems form a subtree of the B&C tree including the root. For example, Kreher and Stinson give a representative definition (based on a certificate, in their terminology; Section 7.3.2 in [9]) unsuitable for our application. The general approach of isomorphism free generation of combinatorial structures based on representatives was studied by Read [18]. A general theory for isomorphism free generation, developed by McKay, can be found in [11].

Let M be an $m \times n$ $(0, 1)$ -matrix and \mathcal{M} be the set of all matrices isomorphic to M , i.e. all matrices that can be obtained from M by a permutation of its rows followed by a permutation of its columns. The binary number obtained by concatenation of the rows M_1, \dots, M_m of M

is the *score* of the matrix M . Accordingly, the score of a row M_i of M is the binary number associated with M_i . Matrix M is a *representative* of \mathcal{M} if M has the maximal score among all matrices in \mathcal{M} . For $i = 1, \dots, m$, we denote by $M[i]$ the row submatrix of M containing rows $1, \dots, i$ of M .

Note that two matrices in \mathcal{M} have the same score if and only if they are identical. The correctness of the algorithm given below relies on the following lemma (a proof can be found in [10], if necessary):

Lemma 3.1 *Let M be an $(m \times n)$ $(0, 1)$ -matrix. Then*

- (i) *if M is a representative then $score(M_i) \geq score(M_{i+1})$ for $i = 1, \dots, m - 1$;*
- (ii) *M is a representative if and only if $M[i]$ is a representative for $i = 1, \dots, m$.*

Observe that, if the order of the rows of M is fixed, then the maximum score achievable by a permutation of its columns is obtained by ordering them in decreasing order, reading the columns as binary numbers of length m from the top down. (The matrix obtained by this ordering of the columns is denoted by \bar{M} .) It follows that, to compute the representative of \mathcal{M} , one only needs to permute the rows of M in all possible orders, sort the columns in decreasing order and memorize the matrix thus obtained with largest score.

In the B&C application, we do not really need to compute the representative of \mathcal{M} , we just need to decide if M is a representative or not. This can be done by a simple backtracking on the ordering of the rows of M , as illustrated in the following algorithm (see Figure 3). The meaning of the variables is: k is the current number of selected rows of the given matrix M , I is the set of indices of rows of M currently not selected (with an additional dummy index, $m + 1$, to simplify the presentation), $\pi[i]$ is the index of the i th selected row and $last$ is the index of the last row removed during a backtracking step. We extend the notation $M[i]$ to cover the case $i = 0$, i.e. $M[0]$ is an empty matrix.

The validity of this algorithm relies on Lemma 3.1(ii): When the condition $score((\bar{B})_k) > score(M_k)$ is satisfied, then $M[k]$ is not a representative since $(\bar{B})[k-1] = M[k-1]$ (otherwise the algorithm would have stopped or used a backtracking step during the previous iteration), implying $score(\bar{B}) > score(M[k])$. It follows that M is not a representative either. When the condition $score((\bar{B})_k) < score(M_k)$ is satisfied, then $score(\bar{B}) < score((\bar{M})[k])$ and B is not a representative. By Lemma 3.1(ii), it is pointless to try to add more rows to B since all larger matrices will not be a representative either.

Note also that the sorting of the columns of B to compute \bar{B} can get a hot start by storing a partition P_1, \dots, P_s of the columns of $M[k-1]$ into classes, two columns being in the same class if they are identical and a column in P_i being lexicographically larger than a column in P_j if $i < j$. Observe that an ordering of the columns is optimal for $M[k-1]$ if and only if it is obtained from a permutation of the columns in P_1 followed by a permutation of the columns in P_2, \dots , followed by a permutation of the columns in P_s . Then, as $(\bar{B})[k-1] = M[k-1]$, the partition of the columns of B is obtained by a refinement of the partition for $M[k-1]$:

```

Input: An  $m \times n$   $(0, 1)$ -matrix  $M$ .
 $I = \{1, 2, \dots, m, m + 1\}$ ;  $k = 0$ ;  $last = 0$ ;
Repeat
     $j = \min \{i \in I \mid i > last\}$ ;

    If  $(j < m + 1)$  then /* forward step */
         $\pi[k + 1] = j$ ;  $I = I - j$ ;  $last = 0$ ;  $k = k + 1$ ;
    else /* backward step */
        If  $k \geq 1$  then  $I = I \cup \pi[k]$ ;  $last = \pi[k]$ ;  $k = k - 1$ ;
        else Output ‘‘ $M$  is a representative’’; stop.

    Let  $B$  be the matrix with rows  $M_{\pi[1]}, \dots, M_{\pi[k]}$ .

    If  $score((\bar{B})_k) > score(M_k)$  then
        Output ‘‘ $M$  is not a representative’’; stop.

    If  $score((\bar{B})_k) < score(M_k)$  then /* backward step */
         $I = I \cup \pi[k]$ ;  $last = \pi[k]$ ;  $k = k - 1$ ;

```

Figure 1: Algorithm to decide if M is a representative or not.

Take the columns in class order P_1, \dots, P_s , and order columns in P_t by taking first columns with a 1 on row k of \bar{B} , then the ones with a 0 on $(\bar{B})_k$, for $t = 1, \dots, s$. (This is akin to a step of a radix sort algorithm [8]).

The complexity of the above algorithm is, of course, exponential in m . Nevertheless, when the input matrix is relatively small, it is fast enough. (The Repeat loop can be implemented to run in $O(n)$.) The existence of a polynomial time algorithm for testing if a given matrix M is a representative is an open question and it is not even clear if this problem belongs to NP. Note that a polynomial time algorithm returning the representative of the isomorphism class of a given matrix could be used to decide in polynomial time if two given bipartite graphs are isomorphic, a problem whose complexity status is a long lasting open question [3, 6].

Remark 3.2 In our application, any permutation of the entries of the characteristic vector of a set in \mathcal{K} yields a set in \mathcal{K} and we may use the indices of the sets to identify them. Let M be a matrix whose rows are characteristic vectors of sets in \mathcal{K} . Then, maximizing the score of a matrix isomorphic to M amounts to finding the matrix M' isomorphic to M such that the ordered set of indices of its rows is lexicographically minimum, since $score(M_i) < score(M'_i)$ if and only if the index of the set corresponding to M_i is strictly larger than the one corresponding to M'_i .

4-(10, 5, 1)-covering design: The time spent in the representative test algorithm during the B&C amounts to less than 0.5% of the total cpu time.

4 Branching and isomorphism pruning

The proposed Branch-and-Cut works with the $(0, 1)$ ILP (2.1) as starting formulation and will branch by fixing variables to 0 or 1.

Let a be a node of the B&C enumeration tree. Let $F_1^a = \{i_1, \dots, i_p\}$ (resp. $F_0^a = \{j_1, \dots, j_q\}$) be the set of indices of variables fixed by branching decisions to 1 (resp. to 0) at a . Let M_1^a and M_0^a be the $(0, 1)$ -matrices whose columns correspond to the elements of V and whose rows are the characteristic vectors of the sets indexed in F_1^a (resp. F_0^a), with the indices corresponding to the rows in increasing order. Let b be another node and let M_1^b and M_0^b be the corresponding matrices associated with b . The subproblems associated with nodes a and b of the B&C are isomorphic if and only if there exist a permutation π of the elements of V , a permutation σ_1 of the rows of M_1^b and a permutation σ_0 of the rows of M_0^b such that permuting the rows and columns of M_i^b according to σ_i and π yields the matrix M_i^a for $i = 0, 1$.

Unfortunately, using this isomorphism test to identify subproblems that can be pruned during the B&C would require the storage of a maximal set of non isomorphic subproblems generated so far in the enumeration. Moreover, the computation needed to find if π , σ_1 and σ_0 exist is not trivial and would be required for many pairs of subproblems. Using the definition of a representative matrix given in the previous section, we can use a slight relaxation of the isomorphism test that turns out to be practical. The price to pay for the simplification is that we will no longer be free to branch on any variable of the ILP: At depth d of the B&C enumeration tree (the root being at depth 1), the branching variable will have to be x_d (even if the value of x_d in the current solution of the LP relaxation is 0 or 1). The enumeration tree generated by a Branch-and-Bound using this branching strategy and the LP relaxation of (2.1) as bound is called the *full enumeration tree*.

Consider the following pruning test: If M_1^a is not a representative, then prune node a . Assume that the branching variable at depth d is always x_d . If M_1^a is a representative then none of the nodes b between the root node and a are pruned by this test: By the choice of branching strategy, $M_1^b = M_1^a[r]$ for some r and thus, by Lemma 3.1 (ii), M_1^b is a representative. It follows that the nodes left after applying the test form a subtree (called the *restricted enumeration tree*) of the full enumeration tree. Moreover, let a be a node of the full enumeration tree for which F_1^a is an optimal solution to ILP (2.1). Then the representative of M_1^a is a matrix M^* , and, by Lemma 3.1 (i), the indices of the rows of M^* are increasing. By Lemma 3.1 (ii), $M^*[r]$ is a representative for all $r = 1, \dots, |F_1^a|$ and thus there is a node b in the restricted enumeration tree with $M_1^b = M^*$. Hence, the optimal value returned by the B&C on the restricted enumeration tree will be the same as the optimal value of ILP (2.1).

4-(10, 5, 1)-covering design: This isomorphism test prunes 485 nodes while the B&C explores the equivalent of 830 additional nodes of the restricted enumeration tree. (Note: these numbers differ from the numbers mentioned in the introduction and in Section 7, as they refer to nodes of the restricted enumeration tree. As explained in Section 6, the enumeration tree of the final B&C is smaller.)

5 Schönheim's bound and isomorphism inequalities

Let $v_0 \in V$ and let \mathcal{S} be an optimal $t - (v, k, 1)$ -covering design. Let $M(\mathcal{S})$ be the matrix whose columns are elements of V and whose rows are the characteristic vectors of the sets in \mathcal{S} . If $t \geq 1$, observe that the sets $S - v_0$, for all $S \in \mathcal{S}$ with $v_0 \in S$, form a $(t - 1) - (v - 1, k - 1, 1)$ -covering design of the elements in $V - v_0$. Hence, the number of sets in \mathcal{S} containing v_0 is at least $C_1(v - 1, k - 1, t - 1)$, the minimum cardinality of a $(t - 1) - (v - 1, k - 1, 1)$ -covering design. It follows that the number of 1's in each column of $M(\mathcal{S})$ is at least $C_1(v - 1, k - 1, t - 1)$. Since $M(\mathcal{S})$ has $C_1(v, k, t)$ rows and each of these rows contains exactly k 1's, we get

$$v C_1(v - 1, k - 1, t - 1) \leq k C_1(v, k, t) \quad (5.2)$$

and thus

$$\left\lceil \frac{v}{k} \right\rceil C_1(v - 1, k - 1, t - 1) \leq C_1(v, k, t). \quad (5.3)$$

Iterating this relation and observing that $C_1(v, k, 0) = 1$, we get

$$L_1(v, k, t) = \left\lceil \frac{v}{k} \left\lceil \frac{v - 1}{k - 1} \cdots \left\lceil \frac{v - t + 1}{k - t + 1} \right\rceil \cdots \right\rceil \leq C_1(v, k, t). \quad (5.4)$$

The number $L_1(v, k, t)$ is called the *Schönheim's lower bound* [13, 21]. Although this bound does not translate into a very useful linear inequality for the B&C, the observations made to establish the relation (5.2) translate into interesting inequalities, provided that the values of $C_1(v - p, k - p, t - p)$ are known for some $1 \leq p \leq t - 1$. Indeed, if, for $U \subseteq V$, we denote by $\mathcal{K}(U)$ the set of all k -sets containing U then the inequality

$$\sum_{i \in \mathcal{K}(v_0)} x_i \geq C_1(v - 1, k - 1, t - 1) \quad (5.5)$$

is valid. More generally, if U is a subset of u elements in V with $1 \leq u \leq t - 1$, then the following inequality, called a *Schönheim's inequality*,

$$\sum_{i \in \mathcal{K}(U)} x_i \geq C_1(v - u, k - u, t - u) \quad (5.6)$$

is valid. The number of all these inequalities is

$$\binom{v}{1} + \binom{v}{2} + \cdots + \binom{v}{t - 1},$$

exponential in t , but small enough to be manageable in our application. We store these inequalities in a pool and use a brute force separation algorithm.

4-(10, 5, 1)-covering design: As the following values are known [13, 21], $C_1(9, 4, 3) = 25$ [12], $C_1(8, 3, 2) = 11$, $C_1(7, 2, 1) = 4$, we use inequalities (5.6) for $u = 1, 2, 3$. The number of inequalities is 10 for $u = 1$, 45 for $u = 2$ and 120 for $u = 3$. Adding these 175 inequalities to the linear relaxation of (2.1) yields an LP with an optimal (non integer) solution of 50, a big improvement over the optimal solution of 42 of the original relaxation. The B&C adds 682 of these cuts during the run (since inequalities are sometimes deleted from the current LP, as explained in Section 6, adding several times the same inequality is possible). The time spent for their separation accounts for less than 1.2% of the total cpu time.

Let us now turn to a second type of inequalities, called *isomorphism inequalities*. Let a be a node of the enumeration tree, with F_1^a as set of indices of variables currently fixed to 1 by branching decisions and M_1^a be the associated $(0, 1)$ -matrix, using the same notation as in Section 3. Let G^a be the set of variables that are not fixed to 0 at node a . Suppose that there exist $J = \{j_1, \dots, j_p\} \subseteq G^a$ and an associated matrix $M(J)$ such that the representative $M^*(J)$ of the equivalence class of $M(J)$ has a strictly larger score than $M_1^a[p]$. By Remark 3.2, this means that the ordered set of the indices of the rows of $M^*(J)$ is lexicographically smaller than the p smallest indices in F_1^a . Let b be the node in the full enumeration tree with $F_1^b = J \cup F_1^a$ and $F_0^b = F_0^a$. Then b (or one of its ancestors) is pruned by the isomorphism test of Section 3, and thus no leaf c of the descendants of a in the restricted enumeration tree has $J \subseteq F_1^c$. It follows that the inequality

$$\sum_{j \in J} x_j \leq |J| - 1 \tag{5.7}$$

is valid in this subtree. Moreover, if the whole restricted enumeration tree is explored by a depth-first search, always selecting first the son d where the branching variable is set to 1, then the sets F_1^d are enumerated in lexicographic order, starting with the smallest one. It follows that if an inequality (5.7) is generated at a , it is valid for the rest of the enumeration, i.e. it can be considered global.

Let \bar{x} be the current optimal solution of the LP associated with a and let $G \subseteq \{i \in \{1, \dots, |\mathcal{K}|\} \mid \bar{x}_i > 0\}$ (the motivation for using G instead of G^a will become clear later; at this point, simply note that $G \subseteq G^a$). The separation algorithm for the isomorphism inequalities is similar to the backtracking used to decide if a matrix is a representative or not (see Figure 5). Here, we try to find a subset J of q indices in G , with $q \leq |F_1^a|$, $\bar{x}(J) > |J| - 1$ and such that the associated matrix $M(J)$ has a representative with a better score than $M_1^a[q]$. We use the same notation as in the previous algorithm, with additional variables $sum_x[k]$ which give the sum of the entries $\bar{x}_{\pi[i]}$ for $i = 1, \dots, k$. The indices in G are ordered in increasing order and $G[j]$ returns the j th index in G .

It is now time to discuss the way to pick the set $G \subseteq G^a$. Any choice is valid, but G should be large enough so that inequalities are generated and small enough so that the backtracking remains reasonably fast. Note that including in G an index i with $\bar{x}_i = 0$ is pointless and that finding a subset J with $\bar{x}(J) > |J| - 1$ is more probable when J is a subset of the indices i with \bar{x}_i relatively large. Thus, a sensible choice is to set G to all indices i such that $\bar{x}_i > \delta$ for some $\delta > 0$.

Three additional remarks on this algorithm: First, the use of a vector $sum_x[k]$ instead of

```

Input: An  $m \times n$   $(0, 1)$ -matrix  $M_1^a$ , an ordered set of  $p$  indices  $G \subseteq G^a$ , a
 $p \times n$   $(0, 1)$ -matrix  $M$  whose  $i$ th row is the set corresponding to  $G[i]$ , the
current optimal LP solution  $\bar{x}$ .
 $I = \{1, 2, \dots, p, p+1\}$ ;  $sum\_x[0] = 0$ ;  $k = 0$ ;  $last = 0$ ; define  $\bar{x}_{G[p+1]} = 1$ .
Repeat
     $j = \min \{i \in I \mid i > last, sum\_x[k] + \bar{x}_{G[i]} > k\}$ ;
    If  $(j < p+1)$  then /* forward step */
         $\pi[k+1] = j$ ;  $sum\_x[k+1] = sum\_x[k] + \bar{x}_{G[j]}$ ;
         $I = I - j$ ;  $last = 0$ ;  $k = k+1$ ;
    else /* backward step */
        If  $k \geq 1$  then  $I = I \cup \pi[k]$ ;  $last = \pi[k]$ ;  $k = k-1$ ;
        else stop.
    Let  $B$  be the matrix with rows  $M_{\pi[1]}, \dots, M_{\pi[k]}$ .
    If  $score((\bar{B})_k) > score((M_1^a)_k)$  then /* Output and backward step */
        Output  $\pi[1], \dots, \pi[k]$ ;  $I = I \cup \pi[k]$ ;  $last = \pi[k]$ ;  $k = k-1$ ;
    If  $score((\bar{B})_k) < score((M_1^a)_k)$  or  $k = m$  then /* backward step */
         $I = I \cup \pi[k]$ ;  $last = \pi[k]$ ;  $k = k-1$ ;

```

Figure 2: Separation algorithm for isomorphism inequalities cutting \bar{x} .

a single variable, say sum_x , avoids the update of sum_x when performing a backtracking step. This prevents rounding errors propagating during the course of the algorithm, as entries in \bar{x} are rational numbers. Second, it may happen that the same set of indices appears several times in the output of the algorithm, since different ordering of the indices may yield a matrix with score better than the given matrix. A simple example, in the case of $2 - (5, 3, 1)$ -covering designs, would be $F_1^a = \{1, 6\}$, $G = \{1, 6, 7, 8\}$ with $\bar{x}_1 = \bar{x}_6 = 1$, $\bar{x}_7, \bar{x}_8 \geq 0.5$ (see Section 2 for the sets corresponding to these indices). The algorithm will output the ordered sets $(1, 7)$, $(1, 8)$, $(7, 1)$, $(7, 8)$, $(8, 1)$ and $(8, 7)$. Non minimal sets can also be in the output: For a $2 - (6, 3, 1)$ -covering design problem, let $F_1^a = \{1, 8, 15\}$, $G = \{1, 8, 15, 16\}$ and $\bar{x}_1 = \bar{x}_8 = \bar{x}_{15} = 1$ and $\bar{x}_{16} > 0$. The corresponding sets are:

$$\begin{array}{ll}
 1: & (1 \ 1 \ 1 \ 0 \ 0 \ 0) & 15: & (0 \ 1 \ 0 \ 1 \ 0 \ 1) \\
 8: & (1 \ 0 \ 0 \ 1 \ 1 \ 0) & 16: & (0 \ 1 \ 0 \ 0 \ 1 \ 1)
 \end{array}$$

The ordered sets $(8, 15, 16)$ (the matrix \bar{B} has rows $(1, 8, 14)$) and $(15, 16)$ (the matrix \bar{B} has rows $(1, 2)$) will both be in the output. Since the isomorphism cut generated by $\{15, 16\}$ implies the one generated by $\{8, 15, 16\}$, removing duplicates and non minimal sets in the output is advisable. Third, to generate only inequalities “clearly” cutting \bar{x} , we reinforce the

test $(\text{sum}_x[k] + \bar{x}_i > k)$ to $(\text{sum}_x[k] + \bar{x}_i > k + \Delta)$, for some $1 > \Delta > 0$.

4-(10, 5, 1)-covering design: Using the isomorphism inequalities reduces the number of nodes of the B&C by roughly 60%, from 989 to 345. The B&C generates 197 isomorphism inequalities. The time spent for their separation amounts to 8% of the total cpu time. We use $\delta = 0.25$, $\Delta = 0.25$ and stop the algorithm as soon as 30 distinct cuts have been generated.

6 Branch-and-Cut

Our implementation of the B&C procedure consists essentially of four stages: Solving the LP relaxation, separating Schönheim’s and isomorphism inequalities, removing non basic inequalities, and branching. The first three (also called an *iteration*) are performed until the calls to the separation algorithms are unable to find an inequality violated by the current LP solution. We use the software ABACUS (version 2.2) developed by Thienel [7, 22] as generic implementation of all B&C steps (isomorphism rejection excepted).

Inequalities are stored in pools: one pool for the initial formulation (2.1), one for Schönheim’s inequalities and one for the isomorphism inequalities. To speed up the solution of the LP, it is customary to remove non essential inequalities from it. At a node a , an inequality in a pool is either active or not active, depending on its presence in the LP currently solved. In our case, the inequalities in the initial formulation (2.1) are always kept active, but Schönheim’s and isomorphism inequalities may become inactive. When an inequality is inactive, it is still stored in its pool and we have the option, once the solution \bar{x} of the current LP is obtained, to check if the inactive inequalities in a pool would cut \bar{x} . If an inactive inequality cutting \bar{x} is found, it is included in the current LP by setting it active again. This is called a *pool separation algorithm*.

Other implementation details are given below:

- a) **Solving the LP relaxation.** We use the commercial package CPLEX (version 6.6) [4].
- b) **Separation.** We generate inequalities by calling three separation algorithms: pool separation for Schönheim’s inequalities, pool separation for isomorphism inequalities, and the separation algorithm described in Section 5. A separation algorithm is stopped as soon as it has generated 30 inequalities. At each node, we call the three separation algorithms during the first iteration, but only the third one for the subsequent iterations.
- c) **Removing inequalities.** Active Schönheim’s and isomorphism inequalities in the current LP relaxation that were not in the optimal basis for 10 consecutive iterations become inactive.
- d) **Branching.** As explained in Section 3, the order of the branching variables is fixed, selecting the variables in increasing order of their index. Hence, if we consider branching

from node a , we have $F_1^a \cup F_0^a = \{1, 2, \dots, i(a)\}$ for some integer $i(a)$. Instead of working with the restricted enumeration tree, we perform the isomorphism rejection test before branching on variable $x_{i(a)+1}$ and creating the two subproblems. This saves the time of generating sons that will be pruned immediately. In other words, we find the index

$$j = \min \{i \in \mathcal{K} - (F_1^a \cup F_0^a) \mid F_1^a \cup i \text{ is a representative}\},$$

using the algorithm of Section 3 for each candidate index i . (If no such index j exists, then we prune a without branching). Let $J = \{i(a) + 1, \dots, j - 1\}$. We create the two subproblems b and c by setting

$$F_1^b = F_1^a \cup j, \quad F_0^b = F_0^a \cup J \quad \text{and} \quad F_1^c = F_1^a, \quad F_0^c = F_0^a \cup J \cup j.$$

As explained in Section 5, isomorphism inequalities may be considered valid for the whole tree from their generation point if the nodes of the restricted enumeration tree are explored in depth first order (DFS), always selecting first the subproblem where the branching variable is set to 1. Hence, we follow this exploration strategy. For many ILP, other strategies (best node first, for example) are far better than DFS. However, the difference between strategies is minimal when the B&C is started with the optimal value of the ILP as cutoff point (i.e. the B&C is used only to prove optimality of a known solution instead of generating it and proving its optimality). Since the upper bounds known for the “small” open covering design problems are believed to be optimal, the constraint on the exploration strategy is likely irrelevant. For the same reason, the chance that a heuristic construction procedure would generate a better solution than the ones already known is remote. Hence we did not consider to implement such a heuristic, although this is usually an important component of an efficient B&C.

7 Results

Running the described B&C algorithm for the $4 - (10, 5, 1)$ -covering design problem, pruning nodes as soon as their associated LP relaxation has value strictly larger than 50, we obtain a proof that no solution better than the best known solution of 51 exists. There are only 345 nodes in the enumeration tree and the cpu time (in seconds) is distributed as follows (the machine used is an HP-J5000 running HP-UX10.20 with two 440MHz PA-8500 RISC CPUs):

Total cpu time:	83.08
LP cpu time:	72.55
Pool separation for Schönheim’s inequalities:	0.93
Pool separation for isomorphism inequalities:	0.67
Separation for isomorphism inequalities:	6.77
Representative test algorithm:	0.32

Since the result is negative, there might be doubts about the accuracy of the pruning due to possible rounding errors. To alleviate these doubts, we also run the B&C, pruning nodes whose LP relaxation has value strictly larger than 51 (instead of 50). Moreover, it branches until all variables that are not fixed have value 0, or until the node is pruned by isomorphism rejection, i.e. the enumeration tree contains all non isomorphic solutions with value 51 (or better; but, of course, no solution with value 50 is found). This run took about 100 hours of cpu time and has an enumeration tree of about 1.2 million nodes. It must be clear that the design of the B&C presented in this paper is not aimed at enumerating all non isomorphic optimal solutions to a covering design problem. It is therefore likely that improvements of these numbers are achievable.

We obtain the following 40 non isomorphic $4 - (10, 5, 1)$ -covering designs with cardinality 51 (we list the indices of the k -sets in the design, assuming a lexicographic ordering of the sets):

1 : { 1, 2, 3, 4, 5, 6, 37, 38, 42, 45, 49, 52, 53, 56, 72, 74, 78, 79, 82, 86, 90, 95, 96, 101, 103, 105, 109, 145, 146, 151, 153, 155, 159, 164, 168, 169, 172, 176, 180, 198, 202, 205, 209, 212, 213, 216, 232, 249, 250, 251, 252 }

2 : { 1, 2, 3, 4, 5, 6, 37, 38, 42, 45, 49, 52, 53, 56, 72, 75, 76, 81, 83, 85, 89, 94, 98, 99, 102, 106, 110, 144, 148, 149, 152, 156, 160, 165, 166, 171, 173, 175, 179, 198, 202, 205, 209, 212, 213, 216, 232, 249, 250, 251, 252 }

3 : { 1, 2, 3, 4, 5, 6, 37, 38, 42, 45, 49, 52, 53, 56, 74, 78, 79, 82, 86, 90, 95, 96, 101, 103, 105, 109, 112, 145, 146, 151, 153, 155, 159, 164, 168, 169, 172, 176, 180, 197, 198, 202, 205, 209, 212, 213, 216, 249, 250, 251, 252 }

4 : { 1, 2, 3, 4, 5, 6, 37, 38, 42, 45, 49, 52, 53, 56, 74, 78, 79, 82, 86, 90, 95, 96, 101, 103, 105, 109, 112, 145, 146, 151, 153, 155, 159, 164, 168, 169, 172, 176, 180, 198, 202, 205, 209, 212, 213, 216, 218, 234, 250, 251, 252 }

5 : { 1, 2, 3, 4, 5, 6, 37, 38, 42, 45, 49, 52, 53, 56, 74, 78, 79, 82, 86, 90, 95, 96, 101, 103, 105, 109, 112, 145, 146, 151, 153, 155, 159, 164, 168, 169, 172, 176, 180, 198, 202, 205, 209, 212, 213, 216, 219, 233, 250, 251, 252 }

6 : { 1, 2, 3, 4, 5, 6, 37, 38, 42, 45, 49, 52, 53, 56, 74, 78, 79, 82, 86, 90, 95, 96, 101, 103, 105, 109, 115, 145, 146, 151, 153, 155, 159, 164, 168, 169, 172, 176, 180, 197, 198, 202, 205, 209, 212, 213, 216, 248, 249, 251, 252 }

7 : { 1, 2, 3, 4, 5, 6, 37, 38, 42, 45, 49, 52, 53, 56, 74, 78, 79, 82, 86, 90, 95, 96, 101, 103, 105, 109, 116, 145, 146, 151, 153, 155, 159, 164, 168, 169, 172, 176, 180, 197, 198, 202, 205, 209, 212, 213, 216, 247, 249, 251, 252 }

8 : { 1, 2, 3, 4, 5, 6, 37, 38, 42, 45, 49, 52, 53, 56, 74, 78, 79, 82, 86, 90, 95, 96, 101, 103, 105, 109, 145, 146, 151, 153, 155, 159, 164, 168, 169, 172, 176, 180, 197, 198, 202, 205, 209, 212, 213, 216, 247, 248, 249, 250, 251 }

9 : { 1, 2, 3, 4, 5, 6, 37, 38, 42, 45, 49, 52, 53, 56, 74, 78, 79, 82, 86, 90, 95, 96, 101, 103, 105, 109, 145, 146, 151, 153, 155, 159, 164, 168, 169, 172, 176, 180, 197, 198, 202, 205, 209, 212, 213, 226, 230, 246, 247, 248, 249 }

10 : { 1, 2, 3, 4, 5, 6, 37, 38, 42, 45, 49, 52, 53, 56, 74, 78, 79, 82, 86, 90, 95, 96, 101, 103, 105, 109, 145, 146, 151, 153, 155, 159, 164, 168, 169, 172, 176, 180, 197, 198, 202, 205, 209,

205, 209, 212, 213, 249, 250, 251, 252 }

25 : { 1, 2, 3, 4, 5, 6, 37, 38, 42, 45, 49, 52, 53, 74, 78, 79, 82, 86, 90, 95, 96, 101, 103, 105, 109, 111, 112, 145, 146, 151, 153, 155, 159, 161, 164, 168, 169, 172, 176, 180, 198, 202, 205, 209, 212, 213, 218, 234, 250, 251, 252 }

26 : { 1, 2, 3, 4, 5, 6, 37, 38, 42, 45, 49, 52, 53, 74, 78, 79, 82, 86, 90, 95, 96, 101, 103, 105, 109, 111, 112, 145, 146, 151, 153, 155, 159, 161, 164, 168, 169, 172, 176, 180, 198, 202, 205, 209, 212, 213, 219, 233, 250, 251, 252 }

27 : { 1, 2, 3, 4, 5, 6, 37, 38, 42, 45, 49, 52, 53, 74, 78, 79, 82, 86, 90, 95, 96, 101, 103, 105, 109, 111, 115, 145, 146, 151, 153, 155, 159, 161, 164, 168, 169, 172, 176, 180, 197, 198, 202, 205, 209, 212, 213, 248, 249, 251, 252 }

28 : { 1, 2, 3, 4, 5, 6, 37, 38, 42, 45, 49, 52, 53, 74, 78, 79, 82, 86, 90, 95, 96, 101, 103, 105, 109, 111, 116, 145, 146, 151, 153, 155, 159, 161, 164, 168, 169, 172, 176, 180, 197, 198, 202, 205, 209, 212, 213, 247, 249, 251, 252 }

29 : { 1, 2, 3, 4, 5, 6, 37, 38, 42, 45, 49, 52, 53, 74, 78, 79, 82, 86, 90, 95, 96, 101, 103, 105, 109, 111, 145, 146, 151, 153, 155, 159, 161, 164, 168, 169, 172, 176, 180, 197, 198, 202, 205, 209, 212, 213, 247, 248, 249, 250, 251 }

30 : { 1, 2, 3, 4, 5, 6, 37, 38, 42, 45, 49, 52, 53, 74, 78, 79, 82, 86, 90, 95, 96, 101, 103, 105, 109, 111, 145, 146, 151, 153, 155, 159, 161, 164, 168, 169, 172, 176, 180, 198, 202, 205, 209, 212, 213, 217, 218, 234, 250, 251, 252 }

31 : { 1, 2, 3, 4, 5, 6, 37, 38, 42, 45, 49, 52, 53, 74, 78, 79, 82, 86, 90, 95, 96, 101, 103, 105, 109, 111, 145, 146, 151, 153, 155, 159, 161, 164, 168, 169, 172, 176, 180, 198, 202, 205, 209, 212, 213, 217, 232, 249, 250, 251, 252 }

32 : { 1, 2, 3, 4, 5, 6, 37, 38, 42, 45, 49, 52, 53, 74, 78, 79, 82, 86, 90, 95, 96, 101, 103, 105, 109, 111, 145, 146, 151, 153, 155, 159, 161, 164, 168, 169, 172, 176, 180, 198, 202, 205, 209, 212, 213, 217, 233, 248, 250, 251, 252 }

33 : { 1, 2, 3, 4, 5, 6, 37, 38, 42, 45, 49, 52, 53, 74, 78, 79, 82, 86, 90, 95, 96, 101, 103, 105, 109, 111, 145, 146, 151, 153, 155, 159, 161, 164, 168, 169, 172, 176, 180, 198, 202, 205, 209, 212, 213, 217, 233, 249, 250, 251, 252 }

34 : { 1, 2, 3, 4, 5, 6, 37, 38, 42, 45, 49, 52, 53, 74, 78, 79, 82, 86, 90, 95, 96, 101, 103, 105, 109, 121, 145, 146, 151, 153, 155, 159, 164, 168, 169, 172, 176, 180, 191, 197, 198, 202, 205, 209, 212, 213, 230, 246, 247, 248, 249 }

35 : { 1, 2, 3, 4, 5, 6, 37, 38, 42, 45, 49, 52, 53, 74, 78, 79, 82, 86, 90, 95, 96, 101, 103, 105, 109, 121, 145, 146, 151, 153, 155, 159, 164, 168, 169, 172, 176, 180, 195, 197, 198, 202, 205, 209, 212, 213, 226, 246, 247, 248, 249 }

36 : { 1, 2, 3, 4, 5, 6, 37, 38, 42, 45, 49, 52, 53, 74, 78, 79, 82, 86, 90, 95, 96, 101, 103, 105, 109, 121, 145, 146, 151, 153, 155, 159, 164, 168, 169, 172, 176, 180, 195, 197, 198, 202, 205, 209, 212, 213, 230, 246, 247, 248, 249 }

37 : { 1, 2, 3, 4, 5, 6, 37, 38, 42, 45, 52, 53, 54, 74, 76, 81, 84, 85, 86, 90, 95, 98, 99, 102, 103, 111, 112, 145, 148, 149, 152, 153, 161, 164, 166, 171, 174, 175, 176, 180, 197, 198, 202, 205, 212, 213, 214, 249, 250, 251, 252 }

38 : { 1, 2, 3, 4, 5, 6, 37, 38, 42, 45, 52, 53, 54, 74, 76, 81, 84, 85, 86, 90, 95, 98, 99, 102, 103, 111, 113, 145, 148, 149, 152, 153, 161, 164, 166, 171, 174, 175, 176, 180, 197, 198, 202,

205, 212, 213, 214, 248, 250, 251, 252 }

39 : { 1, 2, 3, 4, 5, 6, 37, 38, 42, 45, 52, 53, 54, 74, 76, 81, 84, 85, 86, 90, 95, 98, 99, 102, 103, 111, 114, 145, 148, 149, 152, 153, 161, 164, 166, 171, 174, 175, 176, 180, 197, 198, 202, 205, 212, 213, 214, 247, 250, 251, 252 }

40 : { 1, 2, 3, 4, 5, 6, 37, 38, 42, 45, 52, 53, 54, 74, 76, 81, 84, 85, 86, 90, 95, 98, 99, 102, 103, 111, 145, 148, 149, 152, 153, 161, 164, 166, 171, 174, 175, 176, 180, 197, 198, 202, 205, 212, 213, 214, 247, 248, 249, 250, 251 }

Acknowledgments: Thanks to Don Kreher for introducing me to the covering design problem and for several discussions on isomorphism rejection.

References

- [1] Etzion T., Wei V., Zhang Z., “Bounds on the Sizes of Constant Weight Covering Codes”, *Designs, Codes and Cryptography* 5 (1995), 217–239.
- [2] Gibbons P.B., “Computational Methods in Design Theory”, in: *The CRC Handbook of Combinatorial Designs*, Colbourn C.J., Dinitz J.H., eds., CRC Press (1996), 718–740.
- [3] Garey M.R., Johnson D.S., *Computers and Intractability, a Guide to the Theory of NP-Completeness*, Freeman (1979).
- [4] *ILOG CPLEX 6.5 User’s Manual*, (1999).
- [5] Ivanov A.V., “Constructive Enumeration of Incidence Systems”, *Annals of Discrete Mathematics* 26 (1985), 227–246.
- [6] Johnson D.S., “The NP-completeness Column, an Ongoing Guide”, *Journal of Algorithms* 3 (1982), 288–300.
- [7] Jünger M., Thienel S., “Introduction to ABACUS – A Branch-And-CUt System”, *Operations Research Letters* 22 (1998), 83–95.
- [8] Knuth D.E., *The Art of Computer Programming*, Addison-Wesley (1968).
- [9] Kreher D.L., Stinson D.R., *Combinatorial Algorithms, Generation, Enumeration, and Search*, CRC Press (1999).
- [10] Luetolf C., Margot F., “A Catalog of Minimally Nonideal Matrices”, *Mathematical Methods of Operations Research* 47 (1998), 221–241.
- [11] McKay B.D., “Isomorph-free Exhaustive Generation”, *Journal of Algorithms* 26 (1998), 306–324.
- [12] Mills W.H., “On the Covering of Triples by Quadruples”, *Congressus Numerantium* 10 (1974), 563–581.

- [13] Mills W.H., Mullin R.C., “Coverings and Packings”, in: *Contemporary Design Theory: A collection of Surveys*, Dinitz J.H., Stinson D.R., eds., Wiley (1992), 371–399.
- [14] Nurmela K.J., Östergård P.R.J., “New Coverings of t -sets with $(t+1)$ -sets”, *Journal of Combinatorial Designs* 7 (1999), 217–226.
- [15] Nurmela K.J., Östergård P.R.J., “Covering t -sets with $(t+2)$ -sets”, *Discrete Applied Mathematics* 95 (1999), 425–437.
- [16] Östergård P.R.J., “Constructing Covering Codes by Tabu Search”, *Journal of Combinatorial Designs* 5 (1997), 71–80.
- [17] M.W. Padberg, G. Rinaldi, “A Branch-and-Cut Algorithm for the Resolution of Large Scale Symmetric Travelling Salesman Problems”, *SIAM Review* 33 (1991), 60–100.
- [18] Read R.C., “Every One a Winner or How to Avoid Isomorphism Search When Cataloguing Combinatorial Configurations”, *Annals of Discrete Mathematics* 2 (1978), 107–120.
- [19] Seah E., Stinson D.R., “An Enumeration of Non-isomorphic One-factorizations and Howell Designs for the Graph K_{10} minus a One-factor”, *Ars Combinatorica* 21 (1986), 145–161.
- [20] Stanton R.G., Bates J.A., “A Computer Search for B-coverings”, in: *Combinatorial Mathematics VII, Lecture Notes in Computer Science* 829, Robinson R.W., Southern G.W., Wallis W.D., eds., Springer (1980), 37–50.
- [21] Stinson D.R., “Coverings”, in: *The CRC Handbook of Combinatorial Designs*, Colbourn C.J., Dinitz J.H., eds., CRC Press (1996), 260–265.
- [22] Thienel S., “ABACUS - A Branch-And-CU t System” Ph.D. Thesis, Universität zu Köln (1995).
- [23] L.A. Wolsey, *Integer Programming*, Wiley (1998).