Towards Deciding the Existence of 2–(22,8,4) Designs

Brendan D. McKay
Department of Computer Science
Australian National University
Canberra, ACT 2601, Australia
bdm@cs.anu.edu.au

Stanisław P. Radziszowski*
Department of Computer Science
Rochester Institute of Technology
Rochester, NY 14623, USA
spr@cs.rit.edu

Abstract.

We report on progress towards deciding the existence of $2-(22,8,4)$ designs without assuming any automorphisms. Using computer algorithms we have shown that in any such design every two blocks have nonempty intersection, every quadruple of points can occur in at most two blocks, and no three blocks can pairwise intersect in one point.

1. Overview.

A *design* is a pair $(X, \mathcal{D})$ where X is a v-element set of *points* and $\mathcal{D}$ is a multiset of subsets of X, called *blocks*. A design is called *simple* if it has no repeated blocks. A $t-(v,k,\lambda)$ *design* is a design $(X, \mathcal{D})$ with $|X| = v$, such that the blocks have size k and $|\{K \in \mathcal{D} : T \subseteq K\}| = \lambda$ for all $T \subseteq X$ with $|T| = t$.

The case $t = 2$ defines an important category called *balanced incomplete block designs*, BIBD's. Among BIBD's, $2-(22,8,4)$ is the smallest design whose existence is unsettled [CCK] despite much work done by many authors.

Let $\mathcal{D}$ denote any such design and let N be its 22 by 33 (0,1)-incidence matrix for the remainder of this paper. The point set will be also fixed throughout this paper to $X = \{1, 2, ..., 22\}$. The design $\mathcal{D}$ has $b = 33$ blocks, and each point belongs to $r = 12$ blocks. Hamada and Kobayashi [HK], and later Hall, Roth, van Rees and Vanstone [HRRV] discovered much of the structure of $\mathcal{D}$. In particular, in [HK] it is shown that $\mathcal{D}$ may have only four types of blocks with respect to their intersection pattern with other blocks, as shown in Table 1. For each type of block the table gives the number b_i of blocks intersecting it in i points, for all possible i.

Table 1 clearly indicates that any two blocks intersect in at most 4 points, in particular that $\mathcal{D}$ is necessarily simple. We show that any two blocks must have nonempty intersection, thus eliminating type 4. In [HRRV] it is proved that if B is a block of type 3, and C, D are the two blocks intersecting B in a single point, then $1 \leq |C \cap D| \leq 2$. We show that no three blocks can pairwise intersect in one point, hence here necessarily $|C \cap D| = 2$. This

* supported in part by a grant from the NSA number MDA904-94-H-2009.

Type	b_0	b_1	b_2	b_3	b_4
1	0	0	12	16	4
2	0	1	9	19	3
3	0	2	6	22	2
4	1	0	6	24	1

Table 1. The types of blocks in $\mathcal{D}$.

eliminates one (the easier) of the two subcases of blocks of type 3. Finally, on another path, we prove that every quadruple of points can occur in at most two blocks. All our results were obtained with the extensive use of computer algorithms, described in Section 3.

Each of the computations was done at least twice, using a different program written by each of the two authors. Furthermore, the two rounds agreed exactly on a very large number of intermediate partially constructed designs, as described in Section 4. During these computations we constructed thousands of "near" $2-(22,8,4)$ designs, which are collections of 33 blocks of size 8, hitting each of the 22 points 12 times, each pair of blocks intersecting in 1, 2, 3 or 4 points, and such that at least 227 (out of 231) pairs of points are covered exactly 4 times. The details concerning near designs are presented in Section 2.

Among the various software tools used in this work, the novel ones included an efficient method for eliminating most isomorphs during point by point extensions of partial designs, and a method partitioning the set of points into cells and analysing the possible intersection patterns between the blocks and the cells (Section 3).

Throughout our work we did not assume any automorphism acting on $\mathcal{D}$, but consideration of group actions on partially constructed designs formed a very important factor in obtaining efficient algorithms. We note that several papers have appeared that study possible automorphism groups of $\mathcal{D}$ [Hall, Kap, Lan, LT, Šift]. Currently it is known that any nontrivial automorphism group of $\mathcal{D}$ cannot have elements of odd order, a cycle of length 8, or an element of order 2 fixing 3 blocks.

In Section 5 we report on some other approaches, with which we were so far not able to produce any substantial progress. The main line followed the study of block intersection matrices $N^T N$ for $\mathcal{D}$ developed in [HRRV], and then strengthened by Greig [Gre].

2. Near Designs.

Definition 1. The design $(X, \mathcal{ND})$ will be called a *near design* if it satisfies the following properties:

(a) $|X| = 22$, $|\mathcal{ND}| = 33$,
(b) each block has 8 points,

(c) each point occurs in 12 blocks,
(d) each pair of blocks has nonempty intersection,
(e) each pair of blocks intersects in at most 4 points,
(f) at least 227 pairs of points occur in exactly 4 blocks.

The design $\mathcal{D}$ we search for is a near design in the above sense. Note that by the definition, in any near design there are at most 4 pairs of points that are not covered exactly four times. Using elementary combinatorics one can prove that there cannot be precisely 1, 2 or 3 such pairs of points. The algorithms described in the next section so far produced more than 11000 nonisomorphic near designs, failing however to produce $\mathcal{D}$. This number is steadily growing with almost every new experiment performed. The existence of a large number of near designs is perhaps an explanation of why the search for $\mathcal{D}$ is such a difficult computational problem; this is because it is very hard to devise necessary conditions for intermediate configurations which are not also satisfied if the configuration can complete to a near design. The large number of constructed near designs does not mean that we have found them easily, but rather reflects the fact that we have at our disposition a remarkable cpu power (see Section 4), and our algorithms (see Section 3) are specially devised to produce them.

An incidence matrix of a near design chosen from the output of our programs is presented in Figure 1. Its first three blocks intersect pairwise in one point.

```
    3332212112121212121111212      <- block type

 1  0001111111110000000000000000111000
 2  1101110000001111100000000000100100
 3  1010001100001100011100000000011100
 4  0111001011001010010011000000000010
 5  1001000100100100001011110001000010
 6  1000000010010010000110001101101l0
 7  1000100001100011010100010010010l0
 8  1000101001010100110010100l0000001
 9  1001001010000001101001001110010l00
10  1000010101001010000001101010100l1
11  0100110100000000010011011100011l00
12  0100001100101001000110000111000l1
13  0101100000011000001100101000010l1
14  0100011000010100100101010010100l0
15  0100000110110010111000100010001l0
16  0100000011000101011000001100110l01
17  0010110010100110001101000100000l01
18  0011000001111100000000011110001l0
19  0010100110001001100000110100l0010
20  0010001000010011000001110001011l1
21  0011010000100001110110101000l0000
22  0010010001000000101010000011011l1
```

Figure 1. Example of a near design.

Here, there are two pairs covered 3 times, (20,21) and (19,22), and two pairs covered 5 times, (19,21) and (20,22). All other pairs of points are covered exactly 4 times. This design has a trivial full automorphism group. In addition to conditions (a) through (f), it has a property that only the last 6 blocks are not of the types listed in Table 1. Among the near designs constructed so far, only 4 others have 6 such bad blocks; all the rest have 8–12.

3. Algorithms and Results.

Our general technique for advancing knowledge about $\mathcal{D}$ is composed of three stages:

(A) Choice of a small configuration of blocks to be studied,
(B) Construction of a set of starters, refinements of (A),
(C) Design and implementation of an extender, completing (B).

The choices of (A) are those listed in the abstract. The work on (A) is done without help from the computer. For each choice of (A) we construct by computer a few thousand "starters" (see below and Section 4 for more details), which are much more specific configurations than (A). The starters are filtered according to known necessary conditions, and all isomorphs are removed. Stage (B) is human-intensive, and despite needing a large number of distinct programs, the cpu time needed to complete this stage is rather small (in the order of a few cpu hours). Finally, to finish stage (C), a program called an extender is run on each starter. The extender refines a single starter from (B), and finds a family of near designs that are derivable from this starter, that family being designed to include $\mathcal{D}$ if it is there. An extender is a larger piece of software, and is both human-work and cpu-time intensive.

Our starters, and the intermediate configurations constructed by extenders, are in the form of pd-systems defined below.

Definition 2. A quintuple $S = \langle X, [X_1|X_2|...|X_c], k, b, \{s_{ij} : 1 \le i \le b, 1 \le j \le c\}\rangle$ is called a *pd−system* if the s_{ij} are integers such that S satisfies the following properties:

(1) $[X_1|X_2|...|X_c]$ is a partition of X,

(2) $\sum_{j=1}^{c} s_{ij} = k$, for each $1 \le i \le b$,

(3) $0 \le s_{ij} \le |X_j|$, for each $1 \le i \le b, 1 \le j \le c$.

Given a design (X, Q) consisting of b blocks with k points each, and a partition of the domain X into $[X_1|X_2|...|X_c]$, a pd-system S of Q can be obtained by identifying all points in each cell X_j, and defining s_{ij} as the number of points in the intersection of the $i-$th

block of Q and the cell X_j. Let us consider the sequence $(s_{ij} : 1 \leq j \leq c)$ to be the $i-$th block of S.

We will be interested only in pd-systems leading potentially to $\mathcal{D}$, i.e. those which have 22 points and 33 blocks of size 8 ($v = 22, b = 33, k = 8$). If, in addition, a pd-system S satisfies the conditions

(4) $\sum_{i=1}^{b} s_{ij} = 12|X_j|$, for each $1 \leq j \leq c$,

(5) $\sum_{i=1}^{b} s_{ij}s_{im} = 4|X_j||X_m|$, for each $1 \leq j < m \leq c$,

(6) $\sum_{i=1}^{b} \binom{s_{ij}}{2} = 4\binom{|X_j|}{2}$, for each $1 \leq j \leq c$,

then we will say that S is *a pd-system of* $\mathcal{D}$. Condition (4) enforces that points within cells occur the proper number of times, condition (5) that all the pairs of cells are covered the number of times needed to hit all pairs of points of the form (X_j, X_m) four times, and (6) similarly for pairs within a cell.

The concept of a pd-system generalizes that of a design. In particular, for a partition of X into singletons, any pd-system S of $\mathcal{D}$ is clearly equivalent to $\mathcal{D}$ itself. In the sequel, to denote partitions of X with cells possibly of the same size, we specify the sizes of cells with multiplicities. For example, $[\,5{*}1 \,|\, 5 \,|\, 2{*}6\,]$ denotes a partition where the first five points of X form singletons, followed by a 5-cell and two 6-cells, and similarly $[22{*}1]$ can be considered to be the set X itself.

We say that a block B_i is *full* in a pd-system S iff every cell X_j of S is contained in B_i ($s_{ij} = |X_j|$) or disjoint from B_i ($s_{ij} = 0$). Note that for any full block B, the pd-system defines the value of $|B \cap C|$, for all blocks C. If n is the number of distinct blocks in S, and b is the number of all blocks, then we define the *repetitiveness* of S as

$$rep(S) = b - n.$$

Large $rep(S)$ means many repeated blocks in S, which provides an equivalence between refinements that we can easily exploit to gain efficiency.

Let us illustrate these concepts with the pd-systems used to eliminate one class of type 3 blocks. We have found that there are exactly 51 nonisomorphic $[\,4{*}1 \,|\, 3{*}6\,]$ pd-systems of $\mathcal{D}$, in which we have three full blocks pairwise intersecting in one point. One such pd-system is shown in Figure 2. The columns represent blocks, as before, of which the first three are the full blocks, and the rows represent the cells. An 'r' marks a block which is identical to the next block; the significance of this will be described shortly.

Tasks (B) and (C) are mathematically the same; construction of the starters consists of fitting configuration (A) into the trivial partition [22] and then refining it to the partition of the starter pd-systems, while the role of the extender is to refine in all feasible ways a starter pd-system to pd-systems on the final partition $[22{*}1]$. Hence, in what follows

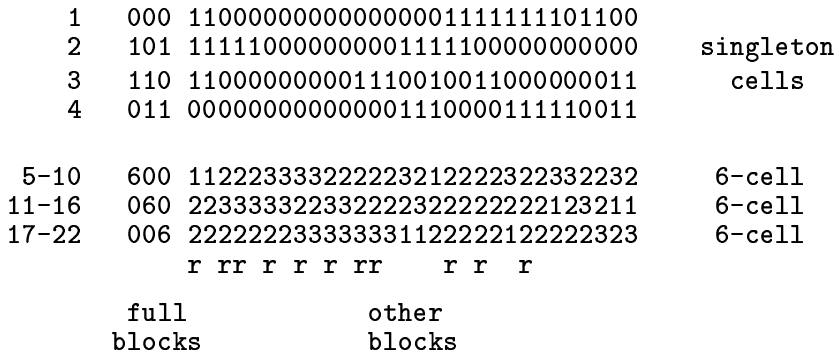

Figure 2. Example of a [4∗1 | 3∗6] pd-system of $\mathcal{D}$

we will focus only on the more difficult task of designing extenders (C), even though, as mentioned at the beginning of this section, the types of software used for (B) and (C) were quite different.

Let S be a starter pd-system, and let X_j be a cell of $p > 1$ points. We want to split X_j into singletons. Using a solver of systems of 0-1 linear equations, we can find a set Y_j of all binary vectors $y = (y_1, ..., y_{33})$ with 12 ones, such that properties (5) and (6) hold when X_j is split into a singleton, hitting block i iff $y_i = 1$, and a $(p-1)$-cell. It is easy to see that we only need to test (5) when either j or m is the index of the new singleton cell, and that even one of those equations is redundant.

In order to avoid generation of isomorphic configurations due to presence of repeated blocks in the current pd-system R, we found the following very simple and efficient method. Singleton cells of R or its refinements are represented as bit vectors, with (currently) equal blocks being in adjacent positions. Let $t = rep(R)$, and consider a bit vector *eqbit* with t ones indicating the positions of the blocks equal to the next block (as indicated by the 'r's in Figure 2). It is clear that among a set of candidates y, equivalent under permutation of equal blocks, there is exactly one which hits the rightmost possible repeated blocks. Using *eqbit* it takes only four machine instructions each to recognise such rightmost y's and to update *eqbit* when one of them is used to split a cell. We leave the details as an interesting puzzle for the reader.

Now, applying backtracking on candidates from the set Y_j, the entire cell X_j can be split into singletons. We only need to backtrack $p - 1$ levels, as the final split makes two singletons. After each new singleton cell is created, to enforce the conditions (5) and (6) for later partitions, we need only filter out those candidates which don't intersect the current y in 4 blocks. We proceed similarly with other cells, unless we want to deliberately relax some conditions in order to obtain nonempty output at level 22 (see Section 4).

The bubble in backtracking typically occurs when the partition has about 12 singletons and 1, 2 or 3 larger cells. In order to moderate the difficulty of this barrier, we used various modifications of the above skeleton. These included:

- Lazy filtering. One can define the set of candidates Y_j for singletons filling cell X_j, for each cell at every level of backtracking. These sets can be computed lazily, only if required, thus avoiding filtering candidates if a branch of the search dies before reaching cell X_j.
- Monitoring sizes of blocks. Pd-systems define the exact number of points of each block in any cell, namely s_{ij}. Tracing the current sizes of blocks during backtracking gives a powerful look-ahead which can often allow us to prune hopeless branches of the search quite early.
- Block intersection criteria. The properties that no two blocks can intersect in more than 4 points, and the triangle inequality which must hold for the cardinalities of the pairwise intersections of any three blocks [Gre], lead to further pruning. Pd-systems often permit us to compute or estimate the size of intersection of two blocks at an early stage of backtracking.
- Countless structural changes and tune-ups suggested by experiment.

Using the algorithms outlined above, we have obtained the results summarized in the following theorem.

Definition 3. Consider blocks of 8 points in a set of 22 points. Define DB to be a pair of disjoint blocks, T31 to be three blocks which pairwise intersect in one point, distinct for each pair, and TQ to be three blocks whose pairwise intersection is a common set of four points.

Theorem. *In the design $\mathcal{D}$, every two blocks have nonempty intersection, every quadruple of points occurs in at most two blocks, and no three blocks pairwise intersect in one point.*

Proof. With the cases DB, TQ and T31 as choices for (A), we completed steps (B) and (C). In each case the families of constructed near designs did not include $\mathcal{D}$, which shows that $\mathcal{D}$ cannot include any of these configurations. Since no two blocks can intersect in more than 4 points, TQ covers all possible quadruples covered three times. In [HRRV] it is shown that if three blocks share one point then some pair intersects in more than one point, and thus the case T31 covers all triples of blocks which pairwise intersect in one point. For comments on how we have verified the correctness of our implementations see the next section. ∎

4. Computations.

The three cases eliminated by this paper, disjoint blocks (DB), one class of type 3 blocks (T31), and thrice-covered quadruples (TQ), were completed several times with gradually improving algorithms. For each case, at least one independent implementation was developed and run by each author. The data presented below refers to the fastest run for each case.

When we say that agreement was reached at some level k (number of points), it means the following in reference to two extenders. For each starter, each of the two extenders produced a set of partial designs on k points, with the partial designs satisfying some agreed subset of the known necessary conditions chosen to make the number of partial designs large but manageable. We checked that in each case the sets of partial designs were isomorphic, though they were always distinct and often many isomorphs were produced with distinct multiplicities. Such an agreement for all starters gave high confidence in the correctness of the computations. By the *search space size* we understand the number of partial designs (or pd-systems) constructed during one computation.

DB. We first obtained 40492 starter pd-systems for the partition $[\,8{*}1 \mid 8 \mid 6\,]$, where the two disjoint blocks formed the first 8 singletons and the 8-cell, respectively. The extenders first split the 8-cell into singletons requiring all pairs on 16 points to be covered exactly 4 times, then, with some conditions relaxed, an agreement between the two implementations was obtained at level 20. Obviously, an attempt was always made to proceed to 22 points. The time needed to produce and verify the starters was negligible when compared to about 0.8 cpu years for the fastest DB computation. The search space size was about $4.6 * 10^{11}$.

T31. The 51 $[\,4{*}1 \mid 3{*}6\,]$ pd-systems (one of them used in an example in Section 3) were first refined to 13484 $[\,5{*}1 \mid 5 \mid 6 \mid 6\,]$ starter pd-systems. This was done by splitting one of the three 6-cells into a singleton and a 5-cell. We took great care in choosing this splitting, in order to produce starters S with large $rep(S)$ in the refined pd-system. The extenders filled the first 5-cell, then one of the 6-cells, then with relaxed conditions (weaker than needed for near designs) were run to 22 points. An agreement between the two implementations was obtained at level 22. The fastest run in this case used about 1 cpu year. The search space size was approximately $4.6 * 10^{11}$.

TQ. There were 11814 $[\,10{*}1 \mid 12\,]$ starter pd-systems in this case, where the first 10 points are disjoint from the three blocks covering the same quadruple of points. A few nontrivial properties of the remaining 12 bit vectors of length 33 were derived by hand and hardwired into the extenders. An agreement between the two implementations was reached at levels 19 and 22. The fastest run in this case used 743 cpu hours. The search space size was approximately $2.8 * 10^{10}$.

In all three cases, the value of $rep(S)$ proved to be critical for the efficiency of an extender processing each starter S. Typically, an increment by one of the value of *rep* halved the cpu time.

The similar apparent difficulty of cases DB and T31 shown above should be amended by a comment that for T31 we used substantially better algorithms for manipulation of pd-systems, hence the case T31 is actually more difficult than DB. The case TQ is definitely the easiest one.

As described in [MR], a special version of the graph isomorphism system *nauty* [Mc1] was prepared for testing partial designs for isomorphism. It was easily extended to work on pd-systems and proved very efficient.

The computer times cited above refer to Sun Microsystems Sparcstation One and Two computers. We had at our disposition about 150 computers (including 54 Sparc Two's) dominated by the latter two types. Our computations were completed quickly by running all available machines simultaneously. So far we have used about 15 cpu years for all work related to $\mathcal{D}$. A single elimination of DB, T31 and TQ was obtained in just about 2 cpu years, hence now we could do it in one week of real time.

The management of jobs running on many computers at once was done using the first author's scheduling system *autoson* [Mc2].

5. Other Approaches.

If N is the incidence matrix of $\mathcal{D}$, then the square matrix $S = N^T N$ of order 33, called the block intersection matrix, has entry $S_{ij} = p$ iff blocks i and j intersect in p points. Hall, Roth, van Rees and Vanstone [HRRV] pioneered a very promising approach involving the properties of principal minors of S. In particular, they constructed 13 5-tuples of blocks, called C_5's, in which each point occurs an even number of times, and proved that any $\mathcal{D}$ must contain at least one of them. The C_5's correspond to principal minors of order 5 in S. This approach was generalized and refined by Greig [Gre], and his new technique reduced the number of these unavoidable minors from 13 to 7. Each of his 7 intersection minors has a unique realization as a 5-tuple of blocks, up to isomorphism. We have repeated all the computations from [HRRV] and [Gre], and, despite some technical errors found in [HRRV], obtained the same 7 final configurations.

The above technique gave apparently strong results for 5-tuples, but seems to be rather weak for larger minors. Among several failed attempts to apply mass computations in this direction was the following. The same argument as in [HRRV] implies that any $\mathcal{D}$ must contain a C_8, i.e. an 8-tuple of blocks hitting each point even number of times. Even using some combinatorial filters in addition to the elimination methods developed in [HRRV] and [Gre], there are about 200000 unavoidable nonisomorphic minors that could correspond to C_8's. Unfortunately, most of these minors correspond to very many C_8's. We have not found any feasible way of completing block extensions of all these 8-tuples. We also tried further extensions of intersection minors to orders 11 and 12, where an additional determinant condition [Con] can be applied, to no avail. Very large numbers of minors of order 12 satisfying all known conditions were produced, and it was still not clear how to efficiently complete the computation for each case. On another path, the computation of all possible intersection minors of order 12 corresponding to 12 blocks hitting a fixed point, and satisfying all necessary conditions known to us, also proved infeasible.

We conclude that this algebraical method will not work by itself, unless coupled with something more. Without having a very clear idea how to do it, we feel there should be an approach which applies the method of unavoidable intersection minors jointly with some kind of generalization of pd-systems.

Bate, Hall and van Rees [BHR], on yet another path towards $\mathcal{D}$, studied the so-called even partial designs, which in our terminology are equivalent to pd-systems with two cells, such that every block intersects both cells in even number of points. The authors of [BHR] first defined a set of even partial design parameters, unavoidable in $\mathcal{D}$, and then they eliminated some special subcases. We think that their approach could be easily modeled with our tools.

The future success of this project is not clear yet. There are a few subcases which are definitely accessible, such as the case where some 3-set is covered four times, but they are not very essential for future progress. Much more important, but much more difficult, is the remaining class of type 3 blocks. At the time of writing, our best programs could finish it in about one century of cpu time, but we hope this can be reduced. Blocks of type 1 and 2 are considerably harder still.

References.

[BHR] J. A. Bate, M. Hall, Jr., G. H. John van Rees, Structures within $(22, 33, 12, 8, 4)$-designs, *J. Combinatorial Mathematics and Combinatorial Computing*, **4** (1988) 115-122.

[CCK] Y. M. Chee, C. J. Colbourn and D. L. Kreher, Simple t-designs with $v \leq 30$, *Ars Combinatoria*, **29** (1990) 193–261.

[Con] W. S. Connor, Jr., On the structure of balanced incomplete block designs, *Ann. Math. Statist.*, **23** (1952) 57–71; corrected *ibid*, **24** (1953) 135.

[Gre] M. Greig, An improvement to Connor's criterion, *preprint.*

[Hall] M. Hall, Jr., Are designs $(22, 33, 12, 8, 4)$ necessarily rigid ? , *Congressus Numerantium*, **69** (1989) 243–244.

[HRRV] M. Hall, Jr., R. Roth, J. van Rees and S. Vanstone, On designs $(22, 33, 12, 8, 4)$, *J. Combinatorial Theory, Ser. A*, **47** (1988) 157–175.

[HK] N. Hamada and Y. Kobayashi, On the block structure of BIB designs with parameters $v = 22$, $b = 33$, $r = 12$, $k = 8$, and $\lambda = 4$. *J. Combinatorial Theory, Ser. A*, **24** (1978) 75–83.

[Kap] S. N. Kapralov, Combinatorial $2-(22, 8, 4)$ designs with automorphisms of order 3 that fix a point, in *Mathematics and Mathematics Education* (in Bulgarian), Slnchev Bryag, 1987, 453–458.

[Lan] I. N. Landgev, Block designs $(22, 8, 4)$ with automorphisms of order 2 and self-orthogonal codes over $GF(3)$, in *Mathematics and Mathematics Education* (in Bulgarian), Slnchev Bryag, 1990, 109–111.

[LT] I. N. Landgev and V. D. Tonchev, Automorphisms of $2-(22, 8, 4)$ designs, *Discrete Math.*, **77** (1989) 177–189.

[Mc1] B. D. McKay, Nauty Users' Guide (Version 1.5), *Technical Report* TR-CS-90-02, Computer Science Department, Australian National University, 1990.

[Mc2] B. D. McKay, Autoson, a private batch system for workstation networks (Version 1.2), *Technical Report* TR-CS-94-10, Computer Science Department, Australian National University, 1994.

[MR] B. D. McKay and S. P. Radziszowski, The nonexistence of $4-(12,6,6)$ designs, in *Computational and Constructive Design Theory*, ed. W. Wallis, Kluwer Publ., *to appear*.

[Šift] J. Šiftar, On 2-groups operating on a $2-(22,8,4)$ design, *Radovi-Matematicki*, **7** no. 2 (1991).