Upper Bounds for Covering Designs by Simulated Annealing

Kari J. Nurmela and Patric R. J. Östergård
Digital Systems Laboratory
Helsinki University of Technology
SF–02150 ESPOO, FINLAND

Abstract

A $t-(v,m,k,\lambda)$ *covering design* is a pair $(X,\mathcal{A})$, where X is a set of v elements (called *points*) and $\mathcal{A}$ is a multiset of k-subsets of X (called *blocks*), such that every m-subset of X intersects at least λ members of $\mathcal{A}$ in at least t points. It is required that $v \geq k \geq t$ and $v \geq m \geq t$. Such a covering design gives an upper bound on $C_\lambda(v,m,k,t)$, the number of blocks in a minimal $t-(v,m,k,\lambda)$ covering design. In this paper it is shown how simulated annealing, a probabilistic method for solving combinatorial optimization problems, can be used to construct covering designs. Implementation of this method is discussed. The method is used to calculate new upper bounds on $C_1(v,t,k,t)$ for $k \leq 9$, $v \leq 13$. A new exact value, $C_1(11,4,6,4) = 32$, is obtained.

1 Introduction

Let $v \geq k \geq t$ and $v \geq m \geq t$. A $t-(v,m,k,\lambda)$ *covering design* is a pair $(X,\mathcal{A})$, where X is a set of v elements (called *points*) and $\mathcal{A}$ is a multiset of k-subsets of X (called *blocks*), such that every m-subset of X intersects at least λ members of $\mathcal{A}$ in at least t points. We define

$$C_\lambda(v,m,k,t) = \min\{|\mathcal{A}| \mid \text{there is a } t-(v,m,k,\lambda) \text{ covering design } (X,\mathcal{A})\}.$$

The index λ can be omitted if it is 1. Covering designs are here defined in a very general sense. Most of the published results on covering designs have concerned the case $m = t$ (see [20] for an excellent survey). Some results on the case $m > t$ exist [3, 4, 27]; in those papers $\lambda = 1$, and the notation $B(t,k,m,v) = C(v,m,k,t)$ is used.

A $t-(v,m,k,\lambda)$ *packing design* can be analogously defined; it is a pair $(X,\mathcal{A})$, where X is a set of v elements and $\mathcal{A}$ is a multiset of k-subsets of X, such that every m-subset of X intersects *at most* λ members of $\mathcal{A}$ in at least t points. Let

If a design simultaneously is both a packing and a covering design, it is called a *t-design*. We use t, v, k, λ with the same meaning as above ($m = t$). If a t-design with these parameters exists, it is called a $t-(v,k,\lambda)$ design (sometimes $S_\lambda(t,k,v)$). 2-designs are generally referred to as *balanced incomplete block designs* (BIBDs). In this paper packing designs and t-designs are discussed very briefly.

The problem of finding the exact value of $C_\lambda(v,m,k,t)$ is very difficult and has been solved only for small values of the parameters and in some special cases. Such values have been proved by combinatorial arguments and in some cases by computer search [27]. Due to combinatorial explosion even computers are not of much help in the general case, so for most parameters only upper and lower bounds on $C_\lambda(v,m,k,t)$ are known. Upper bounds are obtained by explicitly constructing covering designs.

In this paper we show how a probabilistic method can be used in the search for covering designs. Such an approach is interesting for many reasons. Assume that we for some parameters λ, v, m, k, and t know that $C_\lambda(v,m,k,t) \leq b$. We then try to find a covering design with $b-1$ blocks. If no covering design is found, we can conjecture that no such covering design exists and concentrate on improving the lower bound. If we find a covering design we can try to further improve on the upper bound. When no improvements can be made we can study the record-breaking covering design. Many construction methods have been found by this method, although the role of the computer is seldom mentioned in the published results. Also, probabilistic methods can be used to find nonisomorphic designs with given parameters [15]. Two designs are isomorphic if one can be transformed into the other by a permutation of the blocks and the points.

In Section 2 we give some basic theoretical results about covering designs. In Section 3 we discuss simulated annealing, the probabilistic method that has been used in the search for covering designs. In Section 4 the computer implementation and problems and options related to that are treated. In Section 5 it is shown how the program can be used to find other designs than pure covering designs. In Section 6 we give upper bounds on $C(v,t,k,t)$ for $k \leq 9$, $v \leq 13$; covering designs corresponding to the new upper bounds are listed in the Appendix. The paper is concluded in Section 7.

2 Covering Designs

The definition of covering designs was given in the Introduction. In the subsequent discussion of such designs we adopt the following conventions. The elements of X (the points) are denoted by the numbers $0, 1, 2, \ldots, v-1$. The letter b is used to denote the number of blocks in a $t-(v,m,k,\lambda)$ covering design. A k-set is said to *cover* an m-set (in t points), if their intersection contains at least t points.

Example 1 The family $\{0123, 3456, 5678\}$ is a $2-(9,4,4,1)$ covering design. However, this family of 4-sets is not a minimal design, because $\{0123, 4567\}$ is also such a design. A single 4-set will not suffice, so $C(9,4,4,2) = 2$.

A well-known result for covering designs with $m = t$ is the *Schönheim lower bound* [24]:

$$C_\lambda(v,t,k,t) \geq \left\lceil \frac{v}{k} \left\lceil \frac{v-1}{k-1} \cdots \left\lceil \frac{v-t+1}{k-t+1}\lambda \right\rceil \cdots \right\rceil \right\rceil . \tag{1}$$

A generalization of (1) to the case $m \geq t$ does not seem possible, since the proof of (1) is based on a counting argument that does not work for $m > t$. The following lower bound holds in the general case.

Theorem 1 $C_\lambda(v,m,k,t) \geq \left\lceil \dfrac{\lambda \binom{v}{m}}{\sum_{i=t}^{\min(k,m)} \binom{k}{i}\binom{v-k}{m-i}} \right\rceil .$

Proof: The total number of m-sets is $\binom{v}{m}$. Each block in a $t-(v,m,k,\lambda)$ covering design covers $\sum_{i=t}^{\min(k,m)} \binom{k}{i}\binom{v-k}{m-i}$ m-sets; a simple division now gives a lower bound for the number of blocks in the case $\lambda = 1$. The theorem is obtained by multiplying the result by λ. □

Upper bounds for the number of blocks in covering designs can be obtained by explicitly constructing such designs. The method described in the Introduction can then be used. If we are able to construct a covering design with b blocks we try to construct a covering design with $b-1$ blocks and repeat this procedure until we fail at constructing a design.

Very few results on the construction of covering designs by computer have been published. In [20, p. 394] some approaches are mentioned, but there is still a lot to do in this area. We hope that this paper at least partly fills this gap.

2.1 Applications of Covering Designs

In discussions of applications of covering designs, two fields are generally mentioned. They can be used as lottery schemes (see, e.g., [21, 23]) and for error-trapping decoding (see [8]).

Covering designs can be used, for example, in Finnish lotto. In Finnish lotto (as today) 7 numbers out of 37 are chosen randomly, and if you have guessed at least 4 numbers correctly, you win. Finnish lotto supports the use of lottery schemes. There are six different schemes available (called 'rake systems'), which are $5-(v,5,7,1)$ covering designs, v ranging from 12 to 18 and b (the number of blocks in the designs or, equivalently, the number of rows in the schemes) ranging from 60 to 600. Therefore, it is guaranteed that if you guess 5 numbers correctly, there is at least one row in

In the case of error-trapping decoding, covering designs with $m = t$, $\lambda = 1$ can be used. The number of blocks in the covering design used gives the complexity of the decoding procedure; this justifies the importance of finding minimal covering designs. We do not further discuss this application here but refer the reader to the aforementioned reference.

3 The Simulated Annealing Algorithm

Simulated annealing (SA) is a combinatorial optimization method. The aim in combinatorial optimization is to find the best solution among a finite number of possible solutions. Simulated annealing is good at finding near-optimal solutions with a reasonable amount of computing. Note that we do not know for sure, if the best solution found is the global optimum. This restricts the use of the algorithm to cases where a good local optimum is also acceptable.

3.1 Local Optimization

The basic concepts of simulated annealing come from local optimization, also called iterative improvement. A combinatorial optimization problem can be characterized by a set of *solutions* and a *cost function*, which attaches a cost to every solution. Our goal is to find a solution with minimum cost. (A maximization problem can be turned into a minimization problem easily by reversing the sign of the cost function.)

In local optimization one must first define a procedure to perturb the solutions in order to get new ones. Perturbation is accomplished by small changes to the current solution. A solution S' is a *neighbor* of another solution S, if S' can be obtained from S by one such change. The *neighborhood* of a given solution S is the set of all neighbors of S.

The local optimization algorithm takes an initial solution, which can be random or based on some heuristic procedure, and then starts to investigate its neighborhood. If it finds a solution in the neighborhood whose cost is lower than the present cost, it moves to that solution. This procedure is iterated until no further improvements can be made.

The solution this algorithm finds is necessarily a *local* optimum, but it is usually not the global optimum. We can of course apply the method many times with different initial solutions and hope that the best solution found is the global optimum.

3.2 Simulated Annealing

Simulated annealing can be viewed as an extension to local optimization. If the proposed new solution is equal to or better than the current solution, it is accepted.

1. Get an initial solution S and an initial temperature T.
2. While *stop criterion* not satisfied do the following:
   (a) While *inner loop criterion* not satisfied do the following:
      i. Select a neigbor S' of S.
      ii. Let $\Delta = cost(S') - cost(S)$.
      iii. If $\Delta \leq 0$ (downhill move), set $S = S'$.
      iv. If $\Delta > 0$ (uphill move), set $S = S'$ with probability $e^{-\Delta/T}$.
   (b) Reduce temperature T.
3. Return S.

Figure 1: Simulated annealing.

If the proposed new solution is worse than the current solution, it is even then accepted with probability

$$P_{\text{accept}} = \exp\left(\frac{-\Delta}{T}\right) = \exp\left(-\frac{cost(S') - cost(S)}{T}\right), \tag{2}$$

where $cost(S)$ denotes the cost function assigned to the current solution, $cost(S')$ denotes the cost function assigned to the proposed next solution, Δ is the cost change, and the temperature T is simply a control parameter in the same units as the cost function. Ideally, when local optimization is trapped in a poor local optimum, simulated annealing can 'climb' out of the poor local optimum. In the beginning the value of T is relatively large so that many cost-increasing moves are accepted in addition to cost-decreasing moves. During the optimization process the temperature is decreased gradually so that fewer and fewer cost-increasing moves are accepted.

We get the simulated annealing algorithm in Figure 1. For a comprehensive treatment of the simulated annealing algorithm and its analogical physical counterpart, see [1, 7, 11, 13, 16].

The criterion for stopping the iteration (detecting when the process is 'frozen') in step 2 in Figure 1 varies in the literature. One possible approach is to stop when the percentage of accepted moves is below a certain level for a given number of successive executions of steps 2(a) and 2(b), and no improvement of the best solution found so far is recorded.

In step 2(a)i, S' is usually selected randomly among all the neighbors of the current solution, with the same probability for all neighbors. However, with a complicated neighborhood structure, a nonuniformly random selection might be appropriate.

A common inner loop criterion 2(a) is to perform the loop a constant (L) number of times. A rule of thumb is to take L equal to or proportional to the size of the

The new temperature in step 2(b) is usually calculated according to

$$T' = rT, \tag{3}$$

where T' is the new temperature and r is typically in the range $0.95 \ldots 0.99$. Some other *cooling schedules* have also been studied (see, for example, [10]), but this *geometric cooling* seems to be not only one of the most efficient, but also very robust.

It can be shown that simulated annealing converges asymptotically to the set of globally optimal solutions if certain conditions are fulfilled [1, Ch. 2–3]. Unfortunately, pushing the probability of finding a global optimum provably to a given level requires very slow cooling schedules and long iterations at each temperature. Hence it can take more time than what it would take to accomplish an exhaustive search in the solution space. Further, we still do not know when we actually have reached one of the global optima. Nevertheless, in practise, SA often finds very good local optima with a reasonable amount of computing.

The main problems while implementing the simulated annealing algorithm are:

Selection of the cost function. Cost functions usually employ some heuristic related to the problem at hand. If there are difficulties while applying the algorithm, it is difficult to say whether the problem is too hard or whether the cost function is poor.

Efficiency. Perturbations and the corresponding cost change calculations should be simple enough to perform, so that the algorithm can perform iterations fast.

4 Constructing Covering Designs

In this section we show how we have chosen problem-specific parameters, like the cost function, and parameters of the simulated annealing algorithm, like the initial temperature. Data structures used are also discussed briefly.

In our implementation of SA the general guidelines given in the previous section were followed. Various different approaches were also considered, however, no significant improvement could be recorded.

4.1 Cost Function

In the search for covering designs we let a solution S be a multiset of k-subsets of X; the cardinality of the multiset is b. Covering designs cover all m-subsets of v points λ times. One possible cost function for a solution S is the number of m-sets that are covered less than λ times:

where X_m denotes the family of all m-subsets of the set X of v points. However, it seems logical to give a higher cost for an m-set that is covered $\lambda - 2$ times than for an m-set that is covered $\lambda - 1$ times. This inspires the following cost function:

$$cost(S) = \sum_{x \in X_m} \max\Big(0,\ \ \lambda - |\{y \in S : |x \cap y| \geq t\}|\Big). \tag{5}$$

For $\lambda = 1$, (4) and (5) coincide. The intuitive argument in favor of (5) has turned out to hold, so this cost function has been used. We then try to minimize this cost function by using simulated annealing. If the cost ever reaches 0, a covering design has been found and the iteration can be ended.

4.2 Neighborhoods

The main difficulty in implementing the cost function (5) and the neighborhood structure to be described here is efficiency. We rely on an extensive tabulation of neighborhood and covering data.

To be able to efficiently store neighbors, etc., we rank individual k-sets and m-sets lexicographically, so that different subsets have different ranks and the ranks vary from 0 to $\binom{v}{k} - 1$ and $\binom{v}{m} - 1$, respectively. Algorithms for this approach has been described, for example, in [25, pp. 162–163]. The algorithms use binomial coefficients, which have been computed in an array.

The most natural neighborhood structure is as follows. Suppose we have a solution S. Then we can make another solution by first selecting one of the k-sets belonging to S, and by replacing a point in this k-set by a point not belonging to this k-set. We define *the neighborhood of* S as the set of all solutions obtained from S by one such change. The *neighborhood size* is the cardinality of this set, which is $bk(v-k)$, where b is the number of k-sets in a solution and v and k are as before. It is also possible to consider one k-set at a time and try to make a change to it as described before. This reduces the neighborhood size to $k(v-k)$.

4.3 Complexity Issues

The calculation of the cost of a given solution is the most critical part in implementing the simulated annealing algorithm. If it cannot be done efficiently, the performance of the algorithm will be poor. Usually, the most convenient (and efficient) way of computing the cost is to compute only the *change* of cost, when the solution is altered.

In order to compute the change of cost efficiently, we tabulate the data needed to answer to the following questions:

- How many k-sets in the current solution cover a given m-set?
- Which k-sets can be obtained from a given k-set by one element change?

With this information, we can calculate the cost change by traversing through the m-sets which are covered by the changed k-set. There are

$$\sum_{i=t}^{\min(k,m)} \binom{k}{i} \binom{v-k}{m-i}$$

such sets, so the time required to calculate the cost change varies with the parameters of the design and is exponential with respect to v in the worst case.

The space complexity of our approach is also exponential with respect to v in the worst case. The space demands usually set the limits for the parameters of the design. However, with suitable parameter selection (especially when b is large and m equals t) the time demand can take over.

See [22] for a more detailed discussion of the data structures and the implementation of the cost change calculation.

4.4 Initial Temperature

The selection of initial temperature is very important. If we select too high an initial temperature, then the program will spend a lot of time in the beginning of the process without progress of any kind. If we select too low an initial temperature, then the quality of the final solutions will deteriorate.

It is difficult to give the initial temperature directly, because the value of a suitable temperature will depend on the problem instance, the neighborhood structure, the initial solution and the cost function used.

It has been proposed that the algorithm should be given the probability of acceptance of *all moves* (see, e.g., [1]), but in this case it leads to difficulties. If the desired probability is smaller than the relative number of moves that do not increase the cost, then the temperature converges below zero and the iteration formula [1, p. 60] does not work. In this problem there are then sometimes a narrow range of valid initial probabilities.

That is why we here present a more robust and intuitively more appealing initial temperature approximation method. We give the wanted initial probability of acceptance of *cost increasing moves*. Then we approximate the initial temperature according to the formulas below.

If there are n cost increasing moves around the initial solution, then we can approximate the number of moves that would be accepted n_{acc} by

$$n_{\text{acc}} = n \exp\left(\frac{-\overline{\Delta\, cost_+}}{T}\right) \tag{6}$$

where $\overline{\Delta\, cost_+}$ denotes the average change of cost in cost increasing moves around the current solution. Then the acceptance ratio χ is

$$\chi = \frac{n_{\mathrm{acc}}}{n} \approx \exp\left(\frac{-\overline{\Delta\, cost_+}}{T}\right),$$

from which we obtain

$$T \approx \frac{-\overline{\Delta\, cost_+}}{\ln \chi}. \tag{7}$$

Now we set χ to the desired initial probability and calculate an estimate for $\overline{\Delta\, cost_+}$ by taking randomly a given number of neighbors (a suitable value is for example 100) of the initial solution and calculating the average change of cost of those moves that would increase the cost. An approximation of the initial temperature T is then given by (7). Of course, (6) is only a rough estimate of the average number of accepted changes, so the initial temperature given by (7) is only an approximation of the right initial temperature. The average change of cost in cost increasing moves also varies among distinct solutions. In this problem the variation is not a serious problem, but in some other problems one may have to take into account many solutions, not just the initial one, to get good initial temperatures.

This approach is intuitively very appealing, because we are mainly interested in how many moves are accepted despite of that they will increase the cost. We are not interested in how many moves would increase cost and how many would not. If we want that 'practically all moves should be accepted in the beginning', we can set the probability of acceptance for cost increasing moves to, say, 0.8. We do not want to think: 'In this problem instance there are about 60% of moves around the initial solution that will be accepted in any case, so we can set the initial probability between about 60% and 100%, let's say 90%.'

5 Constructing Packing Designs and t-Designs

Assume that $\lambda = 1$. Then a design is a $t-(v, m, k, 1)$ packing design exactly when for any two blocks there are at least $d = 2k - 4t + 2m + 2$ points that belong to exactly one of the blocks (called the distance between the blocks). The blocks can thus be considered as codewords of a *constant weight code* with minimum distance d. Probabilistic methods such as simulated annealing have successfully been used to construct such codes (designs) [9, 12].

As can be seen in the formula for d, a $t-(v, m, k, 1)$ packing design is also a $(t + t') - (v, m + 2t', k, 1)$ packing design ($t' \in \mathcal{Z}$, such that $t + t' \leq m + 2t'$). We can thus restrict our discussion to the case $t = m$. Now each k-set covers $\binom{k}{t}$ t-sets. Then, if the number of uncovered t-sets is $\binom{v}{t} - b\binom{k}{t}$, no t-set can be covered more than once (still, $\lambda = 1$), so we have a packing. We can try to find such a packing by searching

number of the uncovered sets reaches $\binom{v}{t} - b\binom{k}{t}$. If $\lambda > 1$ there is no straightforward way to find packings by considering distances between blocks, however, we can use a set covering approach that is a generalization of the approach in the case $\lambda = 1$.

As t-designs are simultaneously both packing and *covering* designs, our approach can be used directly in searches for such designs. However, we can do somewhat better by making use of the fact that each point in a t-design occurs in r blocks, where r can be calculated from the parameters of the design. See [14] for a discussion of how to make use of this fact in searches for t-designs by simulated annealing.

6 Tables of Upper Bounds

In Tables 1 through 8, upper bounds and exact values for $C(v, t, k, t)$, $k \leq 9$, $v \leq 13$ are given. Upper bounds are preceded by $\leq$. The new upper bounds were determined by explicitly constructing corresponding designs, which are listed in the Appendix. Previously known upper bounds, $C(11, 6, 7, 6) \leq 84$ and $C(12, 7, 8, 7) \leq 126$ [5, 6], are preceded by $\leq$ and typeset in *italic* (we reached these upper bounds in this study, too, but we could not improve them). Some of the upper bounds for $k = 6$, $k = 7$, and $k = 8$ have also been proved by different constructors of lottery systems. We do not know whether the designs found by us are isomorphic to these systems.

The exact values have been gathered from [20], which summarizes some known exact values for $C(v, t, k, t)$. Some of the exact values listed here are outside the boundaries of the parameters in [20]. These values have been obtained as follows.

The trivial result $C(v, t, v, t) = 1$ gives the upmost diagonal in every table. The result $C(v, t, v - 1, t) = t + 1$ is easy to prove and that gives us the second diagonals. In 1941 Turán [31] solved a graph theoretic problem which gives us the proof that $C(v, t, v - 2, t)$ meets the Schönheim lower bound. That result gives us the third diagonal in our tables. It is trivial that $C(v, 1, k, 1) = \lceil \frac{v}{k} \rceil$, so we readily have the values for the first columns in our tables. The exact values in the second and third columns follow from the results in [17, 18, 19, 26, 29]; see also [20, Tables 4.1 and 4.2]. The entry $C(11, 4, 6, 4) = 32$ is now known to be exact, because we found a corresponding design and it was shown in [2] that $32 \leq C(11, 4, 6, 4) \leq 33$. Other exact values are $C(12, 6, 9, 6) = 22$ and $C(13, 3, 7, 3) = 13$ obtained in [21] and [30], respectively.

The designs that give the new upper bounds in these tables are listed in the Appendix in a compressed notion. The designs can be retrieved in the following way: first form all possible k-subsets out of v points and sort them lexicographically in a list. Now let the compressed notion of the design be $s_1, s_2, \ldots, s_b$, the design containing b blocks. The first block of the design is the $(s_1 + 1)$th entry in the list. The ith block can be found by taking the $(i - 1)$th block in the list and moving forward in the list $s_i + 1$ positions. In this compressed notation, the sequence $a, a, \ldots, a$ (j times) is abbreviated as a^j.

	m,k,t							
v	$1,2,1$	$2,3,2$	$3,4,3$	$4,5,4$	$5,6,5$	$6,7,6$	$7,8,7$	$8,9,8$
2	1							
3	2	1						
4	2	3	1					
5	3	4	4	1				
6	3	6	6	5	1			
7	4	7	12	9	6	1		
8	4	11	14	20	12	7	1	
9	5	12	25	30	30	16	8	1
10	5	17	30	$\leq$51	50	45	20	9
11	6	19	47	66	$\leq$100	$\leq$ *84*	63	25
12	6	24	57	113	132	$\leq$177	$\leq$ *126*	84
13	7	26	78	$\leq$157	245	$\leq$264	$\leq$297	$\leq$185

Table 1: Values of $C(v,k-1,k,k-1)$, $k \leq 9$, $v \leq 13$.

	m,k,t						
v	$1,3,1$	$2,4,2$	$3,5,3$	$4,6,4$	$5,7,5$	$6,8,6$	$7,9,7$
3	1						
4	2	1					
5	2	3	1				
6	2	3	4	1			
7	3	5	5	5	1		
8	3	6	8	7	6	1	
9	3	8	12	12	9	7	1
10	4	9	17	20	20	12	8
11	4	11	$\leq$20	32	$\leq$34	$\leq$29	15
12	4	12	$\leq$29	$\leq$41	$\leq$59	$\leq$51	$\leq$40
13	5	13	$\leq$34	$\leq$66	$\leq$88	$\leq$104	$\leq$79

Table 2: Values of $C(v,k-2,k,k-2)$, $k \leq 9$, $v \leq 13$.

	m,k,t					
v	$1,4,1$	$2,5,2$	$3,6,3$	$4,7,4$	$5,8,5$	$6,9,6$
4	1					
5	2	1				
6	2	3	1			
7	2	3	4	1		
8	2	4	4	5	1	
9	3	5	7	6	6	1
10	3	6	10	10	8	7
11	3	7	11	$\leq$17	16	10
12	3	9	$\leq$15	$\leq$24	$\leq$27	22
13	4	10	$\leq$21	$\leq$30	$\leq$43	$\leq$41

Table 3: Values of $C(v, k-3, k, k-3)$, $k \leq 9$, $v \leq 13$.

	m,k,t				
v	$1,5,1$	$2,6,2$	$3,7,3$	$4,8,4$	$5,9,5$
5	1				
6	2	1			
7	2	3	1		
8	2	3	4	1	
9	2	3	4	5	1
10	2	4	6	5	6
11	3	6	8	9	7
12	3	6	$\leq$11	12	12
13	3	7	13	$\leq$19	$\leq$21

Table 4: Values of $C(v, k-4, k, k-4)$, $k \leq 9$, $v \leq 13$.

	m,k,t			
v	$1,6,1$	$2,7,2$	$3,8,3$	$4,9,4$
6	1			
7	2	1		
8	2	3	1	
9	2	3	4	1
10	2	3	4	5
11	2	4	5	5
12	2	5	6	$\leq$8
13	3	6	$\leq$10	$\leq$10

	m,k,t		
v	$1,7,1$	$2,8,2$	$3,9,3$
7	1		
8	2	1	
9	2	3	1
10	2	3	4
11	2	3	4
12	2	3	4
13	2	4	6

Table 6: Values of $C(v, k-6, k, k-6)$, $k \leq 9$, $v \leq 13$.

	m,k,t	
v	$1,8,1$	$2,9,2$
8	1	
9	2	1
10	2	3
11	2	3
12	2	3
13	2	3

Table 7: Values of $C(v, k-7, k, k-7)$, $k \leq 9$, $v \leq 13$.

	m,k,t
v	$1,6,1$
9	1
10	2
11	2
12	2
13	2

Table 8: Values of $C(v, k-8, k, k-8)$, $k \leq 9$, $v \leq 13$.

The new designs presented have been found by a computer program developed during the research. The program is written in ANSI C and it is available via anonymous FTP server `saturn.hut.fi` in directory `pub/cover`. The directory contains a file named `README` which includes brief instructions how to compile and use the program (see [22] for more details). The program can be freely used whenever the following stipulations are followed:

- The program may be used for research only. No commercial use is allowed.
- Whenever the program has been used to obtain results that are published, a reference should be made to this work.
- Bugs in the program should be reported to `Kari.Nurmela@hut.fi` or `Patric.Ostergard@hut.fi`. We also appreciate if improvements and new features are sent to the same e-mail addresses.

7 Conclusions

The implementation of the method presented in this paper relies on extensive tabulation of data. The space complexity thus sets limits on the parameter range for the designs, although with suitable parameter values (for example when b is large) the time demand takes over. Thus, some of the designs with many blocks are probably not minimal covering designs.

The possibility of using probabilistic methods to find designs and to develop construction methods out of these was mentioned in the Introduction. We can also go in the other direction and evaluate proposed construction methods. For example, in [8] a method is introduced that is used to show that $C(15,3,6,3) \leq 45$. We were able to prove that $C(15,3,6,3) \leq 31$ by finding a corresponding covering design. The search for this design took only approximately two minutes of CPU time on a SUN SPARCstation 2. This indicates that the method presented in [8] does not work very well, at least not for these parameters.

During the research it was noticed that with very small parameter values local optimization worked more efficiently than SA—this also goes in accordance with the results in [28]. However, when the solution space grows, SA performs better than local optimization.

As indicated earlier, there is a point when the space and/or time requirements make the usage of this approach intractable or even impossible. This limit could be pushed, e.g., by imposing some structure on the design we search for. However, such considerations are not within the scope of this work.

Appendix

$C(10, 4, 5, 4) \leq 51$
3, 1, 3, 1, 7, 4, 1, 7, 2, 4, 8, 0, 3, 9, 2, 4, 0, 8, 4, 5, 4, 2, 0, 7^2, 1, 4, 5^2, 2, 7, 2, 3, 6, 3, 2, 7, 2, 5^2, 7, 0, 2, 4, 5, 2^2, 8, 2, 9, 5

$C(11, 3, 5, 3) \leq 20$
15, 5, 37, 25, 11, 23, 21, 36, 2, 22, 17, 34, 39, 16, 23, 1, 43, 23, 8, 38

$C(11, 4, 6, 4) \leq 32$
9, 7, 27, 0, 20, 6, 20, 26, 1, 14, 2, 15, 30, 3, 17^3, 13, 7, 3, 7, 26, 9, 26, 4, 5, 20, 6, 45, 5^2, 17

$C(11, 5, 6, 5) \leq 100$
5, 4, 2, 0, 3, 5, 3, 4, 7, 0^2, 11, 3, 4, 6, 0^2, 9, 1, 0, 5, 11, 3, 0, 8, 1, 12, 0, 1, 3, 6, 12, 0^2, 2, 6^3, 0, 5, 0, 4, 0, 6, 2, 9, 1, 2, 3, 8, 0, 1, 3, 0, 6, 0, 6, 9, 6, 3^2, 2, 9, 3, 0^2, 1, 8, 1, 7, 4, 0, 14, 0, 4, 0^2, 6, 0, 8, 3, 4^2, 11, 0^2, 4, 0, 7, 0, 4, 1, 5, 10, 3, 0, 5, 2^2, 3

$C(11, 4, 7, 4) \leq 17$
5, 7, 23, 28, 25, 21, 26, 7, 31, 2, 41, 12, 11, 30, 9, 10, 19

$C(11, 5, 7, 5) \leq 34$
8, 20, 0, 8^2, 3, 2, 16, 11, 0, 9, 10, 13, 17, 7^2, 11, 4^2, 12, 3, 20, 2, 8, 10, 20, 7, 4, 7, 6, 11, 15, 3, 7

$C(11, 6, 8, 6) \leq 29$
6, 5, 7, 3^2, 4, 7, 3, 6^2, 0, 1, 4, 5, 13, 3, 4, 7, 5, 4, 11, 0^2, 1^2, 4, 9, 6, 8

$C(12, 3, 5, 3) \leq 29$
9, 17, 12, 41, 86, 31, 40, 6, 69, 5, 15, 7, 8, 29, 12, 19, 51, 17, 40, 6, 41, 21, 6, 11, 15, 49, 20, 2, 77

$C(12, 3, 6, 3) \leq 15$
70, 24, 78, 98, 73, 32, 60, 51, 17, 55, 22, 76, 64, 77, 84

$C(12, 4, 6, 4) \leq 41$
16, 30, 6, 31, 24, 20, 24, 26, 16, 14, 18, 14, 25, 37, 18, 14, 20, 34, 4, 16, 21, 26, 21, 16, 5, 18, 37, 33, 11, 49, 14, 18, 14, 16, 26, 46, 6, 24, 32, 13, 14

$C(12, 3, 7, 3) \leq 11$
95, 61, 30, 32, 44, 96, 44, 50, 113, 127, 64

$C(12, 4, 7, 4) \leq 24$
32, 4, 75, 19, 38, 61, 24, 13, 1, 56, 26, 58, 29, 6^2, 31, 14, 47, 10, 58, 27, 26, 27, 72

$C(12, 5, 7, 5) \leq 59$
16, 18, 0, 15, 29, 4, 8, 14, 8, 13, 28, 7^2, 26, 36, 9, 5, 17, 6, 4, 9, 10, 21, 14, 4, 12, 11, 15, 1, 26, 8, 15, 2, 13, 8, 14, 6, 20, 3, 11, 15, 7^2, 9, 13, 11, 7, 14, 29, 3, 4, 5, 16, 10, 9, 4, 24, 15, 36

$C(12, 6, 7, 6) \leq 177$
4, 2^2, 3, 1, 0, 5, 7, 4, 3, 5, 2^2, 3^2, 6, 3, 6, 3, 5, 0^2, 8, 2, 6, 0, 3, 2, 3, 7, 1, 7, 5, 0, 1, 12, 0^2, 4, 6, 4, 0, 2, 4, 2, 6, 7^2, 4, 0, 2, 0, 3^2, 1, 6, 0^2, 14, 5, 1, 2^3, 4, 0, 6, 8, 0, 4, 3, 16,

0^2, 1, 5, 0^2, 1, 5, 3, 9, 2, 4, 5, 6, 11, 1, 2^2, 5, 0, 5, 3^2, 2, 0, 10, 5, 0, 5, 2, 5, 6, 0^2, 2, 1, 8, 4, 6, 2^2, 3, 2, 7, 0, 4, 0, 5, 3, 9, 3, 0, 8, 7, 9, 7, 0, 2, 1, 0, 1, 3, 5, 4, 10, 2, 1, 4^2, 1, 4, 3, 2, 3^2, 0, 10, 1^2, 2, 3, 11, 0, 3, 0, 5, 3^2, 2, 4, 1, 2, 7^2, 3, 6, 0, 5, 1, 3, 1, 8, 5, 10, 1

$C(12, 5, 8, 5) \leq 27$
16, 9, 6, 32, 13, 3, 29, 13, 7, 31, 4, 21, 10, 30, 39, 18, 29, 9, 18, 19, 25, 10, 2, 40, 8^2, 10

$C(12, 6, 8, 6) \leq 51$
7, 3, 7^2, 13, 4, 12, 8, 2, 29, 1, 12, 10, 2, 9, 13, 4, 9, 7, 2, 13, 11, 2, 17, 3, 7, 24, 1, 3, 30, 2, 14, 9, 11, 1, 11, 12, 5, 2^2, 10^2, 5, 14, 4, 2, 23, 1, 10, 3, 17

$C(12, 4, 9, 4) \leq 8$
13, 17, 19, 28, 2, 38, 33, 26

$C(12, 7, 9, 7) \leq 40$
1, 6, 4^2, 6, 4, 0, 3, 6, 8, 5, 0, 11, 3, 0, 5, 3, 12, 1, 18, 6, 0, 2, 4, 9, 0, 2, 1, 5, 2, 3, 4^2, 2, 4, 6, 7^2, 4^2

$C(13, 3, 5, 3) \leq 34$
76, 38, 12, 64, 21, 0, 67, 4, 47, 29, 19, 25, 76, 12, 40, 41, 1, 43, 42, 37, 76, 11, 38, 86, 18, 46, 2, 29, 24, 12, 78, 26, 9, 101

$C(13, 4, 5, 4) \leq 157$
2, 3, 12, 10, 4, 5, 7, 9, 17, 3, 10^2, 12, 8, 4, 2, 14, 2, 9, 13, 12, 4, 6, 13, 2^2, 19, 0, 11, 0, 23, 0, 5, 4, 16, 8, 18, 0^2, 3, 6^2, 2, 5, 29, 0, 2, 4, 8, 3, 4, 14, 10, 13, 8, 3, 6, 4, 16, 0, 3, 12, 6, 3, 7^2, 6, 8^2, 6, 2, 5, 3, 16, 10, 6, 8, 0, 10, 5, 7, 10, 7, 6^2, 5, 3, 6^2, 3, 11, 7, 20, 2, 9, 5, 8, 4, 23, 0, 5^2, 13, 15, 7, 5, 11, 13, 6, 17, 6, 0, 7, 12, 11, 5, 16, 1, 0, 13^2, 3, 17, 0^2, 1, 5, 0, 12, 3, 6, 1, 2^2, 21, 5, 10^2, 2^2, 6, 3, 12, 5, 7, 9, 6, 9, 1, 2, 24, 2, 3, 24, 1^2, 6

$C(13, 3, 6, 3) \leq 21$
81, 5, 102, 43, 156, 42, 66, 38, 117, 66, 103, 43, 63, 81, 189, 39, 89, 112, 103, 31, 92

$C(13, 4, 6, 4) \leq 66$
23, 5, 17, 57, 22, 31, 25, 2, 19, 5, 6, 55, 14, 81, 4, 12, 0, 76, 87, 0, 12, 4, 20, 0, 12, 4, 115, 25, 2, 19, 113, 2, 14, 6, 8, 20, 1, 17, 6, 140, 8, 13, 3, 102, 20, 13, 3, 6, 19, 1, 15, 49, 3, 15, 65, 3, 11, 61, 7, 6, 32, 1, 24, 1, 52, 34

$C(13, 4, 7, 4) \leq 30$
22^2, 82, 74, 4, 20, 7, 38, 110, 45, 29, 9, 82, 166, 29, 59, 71, 94, 79, 7, 91, 74, 40, 58, 146, 21, 55, 40, 24, 6

$C(13, 5, 7, 5) \leq 88$
4, 13, 39, 14, 9, 5, 2, 49, 11, 13, 39, 11, 0, 4, 3, 9, 17, 11, 17, 13, 6, 36, 4, 13, 44, 4, 35, 17, 26, 25, 22, 38, 4, 31, 9, 17, 18, 7, 49, 31, 3, 30, 23, 43, 6, 20^2, 26, 49, 13, 12, 8, 13, 31, 50, 2, 14, 7, 3, 51, 35, 11, 33^2, 3, 45, 0^2, 3^2, 21, 11, 8, 4, 24, 7, 28, 14, 17^2, 38, 5, 44, 32, 10, 4, 22, 9

$C(13, 6, 7, 6) \leq 264$
0, 14^2, 4, 11, 3, 2^3, 5, 7, 1, 0, 4, 15, 6^2, 0, 15, 3, 2, 3, 5, 2, 3, 2, 5, 1, 12, 3, 4, 3, 16, 3, 12, 4, 12, 9, 3, 6, 4, 19, 3, 2, 0, 2, 5, 7, 1, 8, 13, 1, 6, 1, 3, 8, 1, 0, 3, 7^2, 4, 3, 6, 17, 3, 5, 6, 4, 20, 6, 2, 12^2, 4, 3, 0, 5^2, 0, 8, 1, 7, 16, 0, 8, 7, 2, 1, 23, 0, 5, 8, 15, 2, 21, 0^4, 5, 1, 2, 5, 1, 4, 2^2, 12, 3, 5, 6, 3, 12, 4, 7, 4, 1, 8, 4, 14, 12, 1, 10, 5^2, 7, 8, 1, 2, 5, 1, 4,

2^2, 5, 0, 4, 9, 2, 1, 8, 14, 2, 7, 13, 4, 8, 12, 0, 3, 12, 9, 4, 5, 0^2, 2, 7, 14, 7^2, 2, 6, 16, 8, 9, 1, 5, 6, 1, 8, 10^2, 2, 14, 7, 0^2, 3, 7, 5, 9, 7, 4, 0, 2^2, 3, 2, 12, 3, 5^2, 4, 0, 5, 3, 2, 23, 1, 6, 10, 14, 0, 7, 6, 7, 4, 3, 6, 11, 8^2, 12, 9, 1, 0, 1, 2, 3, 9, 4, 1^2, 2, 11, 1, 8, 5, 3, 18, 6, 4^3, 2^2, 5, 3, 5, 6^2, 0, 4, 5, 4^2, 3, 9, 0, 16, 4^2, 1, 12, 9, 2, 10, 1, 8, 7, 5, 7

$C(13, 3, 8, 3) \leq 10$
61, 173, 172, 81, 9, 176, 146, 44^2, 286

$C(13, 4, 8, 4) \leq 19$
37, 7, 105, 111, 95, 48, 26, 138, 101, 48, 53, 61, 12, 77, 18, 70, 66, 24, 74

$C(13, 5, 8, 5) \leq 43$
24, 30, 19, 9, 13, 54, 17, 16, 30, 10, 4, 12, 29, 36, 74, 41, 37, 5, 29, 13, 36, 6, 33, 26, 41, 57, 29, 9, 36, 84, 21, 28, 27, 32, 50, 20, 39, 46, 4, 56, 3, 35, 0

$C(13, 6, 8, 6) \leq 104$
12, 14, 12, 14, 11^2, 10, 5, 14, 7, 9, 16, 11, 8, 17, 5, 21, 19, 9, 0, 5, 21, 23, 7, 2, 14, 4, 3, 7, 32, 3, 5, 31, 2, 44, 0, 12, 10, 0, 4, 6, 2, 19, 12, 18, 25, 4, 6, 10, 2, 13, 6, 9, 28, 21, 4^2, 14, 13, 25, 15, 6^2, 8, 14, 8, 10, 23, 8^2, 4^2, 11, 12, 22, 4, 1, 10, 14, 7, 12, 13, 18, 20, 23^2, 3, 4, 8, 0, 7, 2, 10, 15, 37, 14, 5, 8, 7, 24, 8, 23, 1, 3

$C(13, 7, 8, 7) \leq 297$
3, 5, 2, 3, 2, 7, 0, 2, 0, 8, 9, 5, 1^2, 3, 5, 2, 0, 7, 4, 2, 12, 0^2, 4, 3, 2, 3, 1, 3^2, 2^2, 3, 4, 3, 2, 4, 3, 2^3, 0^2, 7, 9, 0, 4, 1, 0, 1, 13, 5, 1, 3, 6, 3, 6, 0^2, 2, 1, 2^2, 4, 0, 1, 15, 0, 4, 3^2, 1^3, 8, 3, 10, 1, 4, 1, 4, 1, 2^2, 3^2, 2, 0, 10, 3, 1, 0, 4, 2, 1, 0, 10, 1, 8, 2, 1^2, 7, 2, 1, 2, 3, 1, 5, 2^2, 4, 10, 6, 8, 1, 2, 8, 1, 3, 1^2, 0, 4, 9, 2, 1^2, 6^2, 7, 1, 6, 2, 1, 5, 4, 3, 7, 5, 6, 0, 1^2, 0, 4, 1, 4, 2^2, 5^2, 4, 3, 4, 3, 4, 6, 0^2, 1, 10, 0^2, 2, 6, 5, 1, 10, 1, 2, 4, 6, 0^2, 1, 0, 5, 4, 7, 0^3, 7, 2, 9, 1, 2, 7, 11, 2, 7, 0^2, 4, 0, 2, 4, 8, 1, 21, 1, 4^2, 6, 1, 2, 1^3, 4, 3, 17, 1, 2, 3, 1, 0, 1, 4^2, 1^2, 6^2, 0, 1, 2, 1, 5, 0, 7, 9, 4, 7, 1, 3, 8, 2, 1, 9, 6, 7, 0, 1, 2, 8, 7, 6, 0^3, 3, 1, 0, 4, 1, 4, 3, 2, 4, 9, 4, 10, 5, 6, 0, 8, 0, 1^2, 2, 5, 2, 1, 12, 5, 1^2, 3, 1, 11, 4, 11, 9, 1, 2, 4, 0^4, 3, 2, 0, 4

$C(13, 4, 9, 4) \leq 10$
106, 14, 15, 39, 85, 97, 117, 27, 34, 84

$C(13, 5, 9, 5) \leq 21$
2, 29, 9, 72, 11, 18, 2, 133, 2, 21, 11, 70, 67, 4, 12, 25, 9, 15, 135, 13, 12

$C(13, 6, 9, 6) \leq 41$
10, 17, 7, 31, 13, 16, 8, 19, 23, 4, 25, 20, 44, 6, 3, 26, 11, 6, 2, 34, 7, 24, 8, 22, 4, 23, 18, 2, 9, 47, 31, 0, 6, 4, 37, 19^2, 4, 5, 46, 5

$C(13, 7, 9, 7) \leq 79$
9, 1, 13, 8, 10, 0, 2, 9, 11, 4, 2, 15, 8, 3, 24, 7, 4, 2, 21, 6, 13, 3, 11, 1, 11, 4, 5^2, 8, 3, 17, 13, 2, 6, 7, 4, 1, 8, 12, 7^2, 15, 3, 7^2, 3, 11, 4, 10, 5, 24, 11, 16, 7, 2, 4, 8^2, 7^2, 13, 9, 5, 4, 17, 2, 17, 0, 12, 15, 9, 3^2, 11, 13, 3, 5, 12, 4

$C(13, 8, 9, 8) \leq 185$
2, 1, 9, 0^3, 2, 5, 1, 10, 5, 0^2, 1, 0, 1, 2, 0, 10, 1, 5^2, 0, 4, 0, 4, 2^2, 0, 14, 4^2, 0^3, 1, 0, 1, 7, 4, 0, 2, 0, 5, 7, 0, 1^2, 10, 1^3, 8, 12, 0^2, 1, 2, 4, 0, 5, 4, 1, 4^2, 2, 3, 0, 9, 3, 5^4, 0^2, 1, 0, 1, 4, 0^2, 5, 0, 1, 2, 4, 9, 2, 7, 5, 1^2, 7, 6, 2, 1^2, 0, 1, 2, 6, 1, 5, 4, 2, 1, 0, 11, 1, 0, 4,

0^2, 1, 2, 3, 4^2, 0^2, 2, 9, 0, 6, 4, 1, 12, 0^2, 4, 0, 1, 2, 8, 0, 1, 7, 1, 3^2, 8, 0, 3, 2, 1, 2^2

References

[1] E. Aarts and J. Korst, "Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing," Wiley, Chichester, 1989.

[2] J. A. Bate and G. H. J. can Rees, Some results on $N(4, 6, 10)$ and $N(4, 6, 11)$ and related coverings, *Congr. Numer.* **48** (1985), 25–45.

[3] J. A. Bate and R. G. Stanton, Some algorithmic results on $(2, 3, 3, V)$ designs and $(2, 4, 4, V)$ designs, *Utilitas Math.* **20** (1981), 195–220.

[4] J. A. Bate and R. G. Stanton, A survey of $(2, 2, L, V)$ designs, *Congr. Numer.* **31** (1981), 3–15.

[5] D. de Caen, D. L. Kreher, W. H. Mills, and S. P. Radziszowski, On the covering of t-sets with $(t + 1)$-sets: $C(9, 5, 4)$ and $C(10, 6, 5)$, *Discrete Math.* **92** (1992), 65–77.

[6] D. de Caen, D. L. Kreher and J. Wiseman, On constructive upper bounds for the Turán numbers $T(n, 2r + 1, 2r)$, *Congr. Numer.* **65** (1988), 277–280.

[7] V. Černy, Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm, *J. Opt. Theory Appl.* **45** (1985), 41–51.

[8] A. H. Chan and R. A. Games, (n, k, t)-covering systems and error-trapping decoding, *IEEE Trans. Inform. Theory* **27** (1981), 643–646.

[9] A. A. El Gamal, L. A. Hemachandra, I. Shperling, and V. K. Wei, Using simulated annealing to design good codes, *IEEE Trans. Inform. Theory* **33** (1987), 116–123.

[10] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon, Optimization by simulated annealing: An experimental evaluation; Part I, Graph partitioning, *Oper. Res.* **37** (1989), 865–892.

[11] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, Optimization by simulated annealing, *Science* **220** (1983), 671–680.

[12] K.-U. Koschnick, Some new constant weight codes, *IEEE Trans. Inform. Theory* **37** (1991), 370–371.

[13] P. J. M. van Laarhoven and E. H. L. Aarts, "Simulated Annealing: Theory and Applications," Reidel, Dordrecht, 1987.

[14] R. Mathon, Computational Methods in Design Theory, in "Surveys in Combinatorics," Proc. 13th Br. Comb. Conf., Guildford/UK 1991, LMSLNS 166, London, 1991, pp. 101–117.

[15] R. Mathon, Nonisomorphic designs by simulated annealing, submitted for publication.

[16] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, *J. Chem. Phys.* **21** (1953), 1087–1092.

[17] W. H. Mills, On the covering of pairs by quadruples I, *J. Combin. Theory Ser. A* **13** (1972), 55–78.

[18] W. H. Mills, On the covering of pairs by quadruples II, *J. Combin. Theory Ser. A* **15** (1973), 138–166.

[19] W. H. Mills, Covering designs I: Coverings by a small number of subsets, *Ars Combin.* **8** (1979), 199–315.

[20] W. H. Mills and R. C. Mullin, Coverings and packings, in J. H. Dinitz and D. R. Stinson (Eds.), "Contemporary Design Theory: A Collection of Surveys," Wiley, New York, 1992, pp. 371–399.

[21] M. Morley and G. H. J. van Rees, Lottery schemes and covers, *Utilitas Math.* **37** (1990), 159–165.

[22] K. J. Nurmela and P. R. J. Östergård, Constructing covering designs by simulated annealing, Report B10, Digital Systems Laboratory, Helsinki University of Technology, 1993.

[23] W. Oberschelp, Lotto-Garantiesysteme und Blockpläne, *Math.-Phys. Semesterber.* **19** (1972), 55–67.

[24] J. Schönheim, On coverings, *Pacific J. Math.* **14** (1964), 1405–1411.

[25] D. W. Stanton and D. E. White, "Constructive combinatorics," Springer-Verlag, New York, 1986.

[26] R. G. Stanton and J. G. Kalbfleisch, Coverings of pairs by k-sets, *Internat. Conf. on Combin. Math. Ann. NY Acad. Sci.* **175**, Art. 1 (1970), 366–369.

[27] R. G. Stanton and J. A. Bate, A computer search for B-coverings, R. W. Robinson, G. W. Southern, and W. D. Wallis, (Eds.), "Combinatorial Mathematics VII," LNCS 829, Springer-Verlag, New York, 1980, pp. 37–50.

[28] D. R. Stinson, Hill-climbing algorithms for the construction of combinatorial designs, *Ann. Discrete Math.* **26** (1985), 321–334.

[29] D. T. Todorov, On the covering of pairs by 13 blocks, *C. R. Acad. Bulgare Sci.* **38** (1985), 691–694.

[30] D. T. Todorov and V. D. Tonchev, On some coverings of triples, *C. R. Acad. Bulgare Sci.* **35** (1982), 1209–1211.

[31] P. Turán, Eine Extremalaufgabe aus der Graphentheorie, *Mat. Fiz. Lapok* **48** (1941), 436–452.