A Multilevel Cooperative Tabu Search Algorithm for the Covering Design Problem

Chaoying Dai, (Ben) Pak Ching Li
Department of Computer Science, University of Manitoba
Winnipeg, Manitoba R3T 2N2, Canada
Michel Toulouse
CIRRELT, Université de Montréal
Montréal, Québec H3C 3J7, Canada

Abstract

We propose a multilevel cooperative search algorithm to compute upper bounds for $C_\lambda(v,k,t)$, the minimum number of blocks in a $t-(v,k,\lambda)$ covering design. Multilevel cooperative search is a search heuristic combining cooperative search and multilevel search. We first introduce a coarsening strategy for the covering design problem which defines reduced forms of an original $t-(v,k,\lambda)$ problem for each level of the multilevel search. A new tabu search algorithm is introduced to optimize the problem at each level. Cooperation operators between tabu search procedures at different levels include new re-coarsening and interpolation operators. We report the results of tests that have been conducted on 158 covering design problems. Improved upper bounds have been found for 34 problems, many of which exhibit a tight gap. The proposed heuristic appears to be a very promising approach to tackle other similar optimization problems in the field of combinatorial design.

1 Introduction

A $t-(v,k,\lambda)$ *covering design* is a pair (X,S), where X is a set of size v, called *points*, and S is a set of k-subsets of X, called *blocks*, such that every t-subset of X is contained in at least λ blocks of S. Let $C_\lambda(v,k,t)$ denote the minimum number of blocks in any $t-(v,k,\lambda)$ covering design.

JCMCC **68** (2009), pp. 33-65

A $t-(v,k,\lambda)$ covering design is *optimal* if it has $C_\lambda(v,k,t)$ blocks [18]. The *covering design problem* is the problem of determining the value for $C_\lambda(v,k,t)$.

The covering design problem has a long history. Many theoretical papers have been written on this problem. This is also a problem that has several important applications in cryptography [22], data compression [9], and lottery design [6]. However, the covering design problem is extremely difficult to solve, solutions exist only for a small set of parameters. For most parameters, only upper bounds for $C_\lambda(v,k,t)$ are known. One research direction to improve upper bounds for $C_\lambda(v,k,t)$ investigates methods that construct explicitly covering designs. Constructive methods include deterministic constructive methods such as dynamic programming techniques [14], implicit enumeration methods such as Branch-&-Cut [16] and heuristic search methods [19, 20]. The object of the present research is the introduction of a new heuristic method for computing upper bounds for the covering design problem.

Our heuristic is a multilevel cooperative search [21, 26], a multi-search technique that can cope efficiently with non-trivial characteristics in some optimization problems. Multilevel cooperative search combines cooperative search [5, 15] and multilevel search [1, 27]. In a cooperative search, several search procedures are run concurrently and independently while optimizing the same cost function. Cooperation is based on operators which allow a particular search procedure to use attributes of solutions found by other search procedures to guide its own search steps. Multilevel search applies multigrid/multilevel methods for numerical approximation [3] to discrete optimization problems. In the initial phase of a multilevel search, the source problem is coarsened into a smaller problem, for example, by reducing the number of decision variables. A best solution (elite solution) is computed for the coarsened problem. Next, a search heuristic for the source problem is initialized with a solution interpolated from the elite solution. This scheme can be applied recursively, coarsening the domain of a coarsened problem P_i in order to find a good solution to initialize search at level i, thus the name multilevel. The recursively coarsened problems are interpreted as a hierarchy of approximations of the source problem.

Multilevel cooperative search algorithms make use of the hierarchy of coarsened problems approximating the source problem to partition the search space for the multi-search. Together, the coarsened problems and the source problem are solved concurrently, for example, as in our case, using one independent sequential search procedure per level. In the present multilevel cooperative search algorithm, a new tabu search heuristic computes, at each level, upper bounds for $C_\lambda(v,k,t)$. This tabu search heuristic

optimizes a simple integer cost function, which models the problem of finding covering designs as a combinatorial optimization problem. This cost function arises directly from the definition of the covering design problem [19]. It is characterized by large plateaus in which solutions "close" to each other in the neighborhood structure of search methods have similar cost. Plateaus are problematic to local search heuristics because of the lack of effective gradient in the neighborhood.

Three cooperation operators complement the neighborhood structure and short-term memories of tabu search to provide search directions in the search space of this cost function. The "interpolation" operator restarts the tabu search procedure at level i whenever it seems to be wandering in a plateau of the cost function. This re-initialization is based on an elite solution computed at the adjacent level $i+1$. We also apply cooperation to identify regions of the search space where overall cost-improving solutions can be made, particularly at the level of the source problem. For this purpose, we introduce two new cooperation operators for multilevel cooperative search: a multilevel re-coarsening operator and a direct interpolation between the source problem and the top most problem of the hierarchy (level l).

Our re-coarsening strategy exploits a characteristic of the covering design problem seen as a subset problem: b, the number of blocks in subsets representing feasible solutions, is very small compared to the whole set of $\binom{v}{k}$ blocks. Therefore, as attributes of the solutions that can be applied to re-coarsen problems, we use blocks of elite solutions from several levels. More precisely, let $E_0, E_1, \ldots, E_{i-1}$ be respectively the elite solution of levels $0, 1, \ldots, i-1$. The "multilevel re-coarsening" operator copies blocks from $EB_i = E_0 \cup E_1 \cup \cdots \cup E_{i-1}$ in a set $A_i \subset \binom{v}{k}$ that defines the coarsened problem at level i. In particular, set A_l is kept very small such that coarsened problem P_l is almost exclusively defined in terms of elite solutions from the other levels. The "direct interpolation" operator restarts tabu search for the source problem with solutions computed by tabu search at level l. Direct interpolation is a very successful strategy to initialize search such that improving solutions are found by tabu search. We credit this operator for the most significant overall cost improvements of this multilevel cooperative search algorithm.

The main contributions of this research are the following: A new heuristic method, a multilevel cooperative search, is introduced for computing upper bounds for $C_\lambda(v,k,t)$. The search method for this heuristic is a new tabu search heuristic, which in our tests, outperformed the optimized simulated annealing code [20]. Contributions are made to the design of multilevel methods for re-optimization, re-coarsening and interpolation. The

direct interpolation operator is quite significant in terms of the originality of the approach, its impacts on the performance of the present multilevel cooperative search algorithm and the possibilities for generalization to other multilevel methods and other problems. Our multilevel cooperative search algorithm is a useful development among constructive methods. Numerical results indicate that this is by far the best search heuristic for computing upper bounds on $C_\lambda(v,k,t)$.

The subsequent sections of this paper are organized as follows. Section 2 provides some background on the covering design problem. Section 3 introduces a mathematical programming model for computing covering designs and a coarsening algorithm to reduce the size of problem instances in this model. Section 4 describes our tabu search algorithm for covering designs. Section 5 describes the multilevel cooperative search algorithm. Section 6 reports experimental results. Finally, we conclude in Section 7.

2 Background on covering design

The study of covering designs began around the end of the 1930's. Turán (see [6]) was one of the first researchers to study covering designs. Since then, many researchers have studied covering designs from various directions. One such direction is the determination of $C_\lambda(v,k,t)$ by means of computer programs. Because the exact value of $C_\lambda(v,k,t)$ has been computed only for small set of values for v, k, t and λ, most research on covering designs has focused on determining the upper and lower bounds for $C_\lambda(v,k,t)$. In this section, we briefly describe some important results about the lower bounds and upper bounds for $C_\lambda(v,k,t)$.

The Schönheim lower bound ($L_\lambda(v,k,t)$) [24] provides a lower bound for $C_\lambda(v,k,t)$ given by:

$$L_\lambda(v,k,t) := \left\lceil \frac{v}{k} \left\lceil \frac{v-1}{k-1} \cdots \left\lceil \frac{v-t+1}{k-t+1}\lambda \right\rceil \cdots \right\rceil \right\rceil \leq C_\lambda(v,k,t).$$

This bound is a very good general lower bound for $C_\lambda(v,k,t)$. For many values of v, k, and t where $C_\lambda(v,k,t)$ is known, $L_\lambda(v,k,t)$ attains the value $C_\lambda(v,k,t)$ [17].

In 1963, Erdős and Hanani [8] conjectured that for fixed values of t and k, where $t < k$.

$$\lim_{n\to\infty} \frac{C_1(v,k,t)\binom{k}{t}}{\binom{v}{t}} = 1.$$

This result was shown to be true in 1985 by Rödl [23], using probabilistic methods. This result implies that $C_1(v,k,t) = (1+o(1))\frac{\binom{v}{t}}{\binom{k}{t}}$.

Various techniques have been used to construct covering designs [14]. One of the earliest constructions involved using finite geometries to construct covering designs. For example, it has been found that the hyperplanes of the affine geometry $AG(t,q)$ form an optimal (q^t, q^{t-1}, t) covering design with $\frac{q^{t+1}-q}{q-1}$ blocks. Another common approach is to use recursive techniques for constructing covering designs from smaller covering designs [19]. For example, if S_1 is a $t-(v-1,k,\lambda)$ covering design and S_2 is a $(t-1)-(v-1,k-1,\lambda)$ covering design, then a $t-(v,k,\lambda)$ covering design can be constructed by taking all blocks from S_2, adding a new point v to each of these blocks, and including all blocks from S_1.

Exact search methods have also been used to construct covering designs. Bate [2] developed a backtracking algorithm to exhaustively search for generalized covering designs to determine $C_\lambda(v,k,t)$. In 2003, Margot [16] used integer programming techniques, branch-and-cut and isomorphism rejection to design an algorithm for computing $C_\lambda(v,k,t)$. However, such algorithms are effective for only a few set of parameters.

Recently, neighborhood based search heuristics (local search, tabu search, simulated annealing) have been developed to compute upper bounds on $C_\lambda(v,k,t)$ [19, 20]. An elaborated version of the simulated annealing algorithm in [20] has been coded and is publicly available. This program is extensively used for seeking improved upper bounds to covering design problems.

3 Problem formulation & coarsening

The problem of finding covering designs is formulated as an integer programming problem. Next, a coarsening strategy is described for this formulation.

3.1 Problem formulation

Let $\binom{X}{k}$ denote the set of all k-subsets in X. Assume b is an integer strictly smaller than the best known upper bound for $C_\lambda(v,k,t)$. The search for a covering design with b blocks is guided by a cost function in $\binom{v}{k}$ integer decision variables. A decision variable d_s models a block $s \in \binom{X}{k}$. The

domain of each decision variable is the set $\{0, 1, \ldots, b\}$ of integers. The value of decision variable d_s represents the number of times block s is included in a solution S.

Let $\binom{X}{t}$ denote the set of t-subsets and assume $t < k$. Then, there are $\binom{v}{t}$ members in $\binom{X}{t}$. The cost $c(S)$ of a set S of b blocks is defined in terms of the difference between λ and the number of times a t-subset is covered by blocks in S. This difference is summed over all the t-subsets:

$$c(S) = \sum_{y \in \binom{X}{t}} \max\{0, \lambda - \sum_{s \in \binom{X}{k}} d_s(y \subset s)\}. \quad (1)$$

When $c(S) = 0$, all t-subsets are covered at least λ times, indicating a $t - (v, k, \lambda)$ covering design with b blocks has been discovered. The integer programming formulation is:

$$\begin{aligned} &\min \textstyle\sum_{y \in \binom{X}{t}} \max\{0, \lambda - \sum_{s \in \binom{X}{k}} d_s(y \subset s)\} \\ &\text{subject to:} \\ &\qquad d_s \in \{0, 1, \ldots, b\} \\ &\qquad \textstyle\sum_{i=1}^{\binom{v}{k}} d_i = b \end{aligned} \quad (2)$$

The cost function in (2) is the reformulation of a discrete optimization model for covering designs introduced in [19]. In this paper, we will refer equivalently to a set S of b blocks as a feasible solution to the above mathematical program. Similarly, the *search space* $\mathcal{S}$ of integer vectors, as defined in (2), is expressed in terms of sets of blocks in $\binom{X}{k}$ as follows:

$$\mathcal{S} = \{S \subset \binom{X}{k} || S | = b\}. \quad (3)$$

3.2 Coarsening strategy

We describe a coarsening strategy for problem instances of the above model. Our coarsening strategy generates a hierarchy of increasingly smaller problems by reducing the size of the initial set of blocks $\binom{X}{k}$ into a nested sequence of increasingly smaller sets $A_1, A_2, \ldots, A_l$ such that $A_l \subset A_{l-1} \subset \cdots \subset A_1 \subset A_0 = \binom{X}{k}$. Each set A_i defines a search space $\mathcal{S}_i = \{S \subset A_i || S| = b\}$ and a problem P_i which is to minimize $c(S)$, $S \in \mathcal{S}_i$. Since

$A_{i+1} \subset A_i$, $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_l$ is a hierarchy of increasingly smaller search spaces where

$$\mathcal{S}_l \subset \mathcal{S}_{l-1} \subset \cdots \subset \mathcal{S}_1 \subset \mathcal{S}_0 = \mathcal{S}. \tag{4}$$

Each solution $S \in \mathcal{S}_i$ is also a solution of $\mathcal{S}_{i-1}, \mathcal{S}_{i-2}, \ldots, \mathcal{S}_0$.

A single data structure, an integer array M of size $\binom{v}{k}$, stores the sets A_0 to A_l. All entries of M are initialized to "0". Then, $|A_1|$ entries in M are selected randomly and are assigned the value "1". Next, $|A_2|$ entries among those with value "1" are selected randomly and assigned value "2". This same action is applied to set A_i in order to compute set A_{i+1} until $|A_l|$ entries of M are assigned value l.

The number of blocks in set A_i is based on the *coarsening factor* cf. This coarsening factor is function of the total number of blocks $\binom{v}{k}$ in the source problem, the number of levels $(l+1)$ and the cardinality of the smallest set A_l. The coarsening factor cf is computed as follows:

$$cf = \frac{\binom{v}{k} - (|A_l| \times (l+1))}{(l+1) \times \frac{l}{2}}. \tag{5}$$

A block s *belongs* to set A_i if $M[s] \geq i$. The number of blocks that belong to A_i is

$$|A_i| = \sum_{j=1}^{\binom{v}{k}} (M[j] \geq i) = (l-i+1) \times |A_l| + \frac{(l-i)(l-i+1)}{2} \times cf. \tag{6}$$

We also define an exclusive partitioning of set $\binom{X}{k}$. A block s *belongs strictly* to A_i if $M[s] = i$. The number of blocks that belong strictly to A_i is $\sum_{j=1}^{\binom{v}{k}} (M[j] = i) = |A_l| + (l-i) \times cf$.

4 Tabu search for covering design

Tabu search [10, 11] is a well known search heuristic technique which has been applied to a broad range of combinatorial optimization problems [12]. Our tabu search heuristic extends the tabu heuristic in [19] with two new tabu lists and a diversification phase.

4.1 Neighborhood structure

A solution $S' \in \mathcal{S}$ is a neighbor of S (denoted as $S' \in \mathcal{N}(S)$) if S' has $b-1$ identical blocks with S and one block $s' \in S'$ differs from a block $s \in S$ by exactly one point. For each block $s \in S$, there are $v-k$ points not in S. Therefore, the size of the neighborhood $\mathcal{N}(S)$ is relatively small:

$$|\mathcal{N}(S)| = b \times k \times (v-k). \tag{7}$$

Since $\mathcal{N}(S)$ is small, tabu search evaluates neighborhoods exactly. Given a current solution S, the best neighbor is selected by applying the following transition rule:

$$S = \min\{c(S') \mid S' \in \mathcal{N}_i(S)\}. \tag{8}$$

To speed-up computation, the neighborhood of each solution $S \in \mathcal{S}$ and the cost of each neighbor is stored in tables [20]. Computing the cost of a neighbor is reduced to reading a few entries in these tables. The mapping function in [25] is used to rank the $\binom{v}{k}$ blocks and to index the tables.

Note, each tabu search procedure at each level i has a different neighborhood structure $\mathcal{N}_i$. The neighborhood of solution $S \in \mathcal{S}_i$ is defined as:

$$\mathcal{N}_i(S) \quad = \quad \begin{cases} \emptyset, & S \notin \mathcal{S}_i \\ S', & S' \in \mathcal{S}_i \text{ and } S' \in \mathcal{N}_0(S). \end{cases} \tag{9}$$

It follows from relations 4 and 9 that if $S \in \mathcal{S}_i$, then the neighborhood of S in $\mathcal{S}_{i-1}, \mathcal{S}_{i-2}, \ldots, \mathcal{S}_0$ is such that

$$\mathcal{N}_i(S) \subseteq \mathcal{N}_{i-1}(S) \subseteq \cdots \subseteq \mathcal{N}_0(S). \tag{10}$$

4.2 Tabu lists

The application of transition rule (8) is a transformation $(s' \rightarrow s)$ of the current solution S that brings a block s' in S and removes from S a block s. Two tabu lists prevent short term cycling. A first tabu list prohibits the reversal of recent transformations. It prohibits the opposite transformation $(s \rightarrow s')$ by storing transformation $(s \rightarrow s')$ in the tabu list. In order to control situations where $(s' \rightarrow s)(s \rightarrow t) \cdots (t \rightarrow s')(s' \rightarrow s)$, the transformation $(s' \rightarrow s)$ is also stored in this first tabu list.

In plateaus of the cost function, some same blocks have tendency to swap in and out from the current solution. A second tabu list disallows

block s' from leaving S for a predefined number of iterations after entering S. This second tabu list greatly improves the performance of our tabu search heuristic compared to the tabu search heuristic in [19]. However, because this tabu list imposes strong limitations on the exploration of the search space, it is kept very short. Furthermore, the second tabu list supersedes the first one. That is, if s' is prohibited from leaving S following a transformation $s' \rightarrow s$, then the opposite transformation $s \rightarrow s'$ cannot occur (See [4] for an application of tabu search with multiple tabu lists).

The length of the two tabu lists is set randomly inside a certain range. This range is determined by an input parameter t. For example, the length of a tabu list can be selected randomly to be one of the values in the range $t-2$ to $t+2$. The first tabu list is less constraining on the search, we usually set t_1, which determines the range of this tabu list, as twice the size of t_2, which determines the range of the second tabu list.

4.3 Termination criterion

The tabu search procedure terminates after completing a predefined number of iterations without improving the best solution.

4.4 Diversification phase

For levels 0 to $l-1$, once the termination criterion is partly satisfied (i.e., once the number of tabu search iterations without improving the best solution reaches half the value of the stopping criterion, line 7 in Figure 1), the cost of the current best solution E is compared with the cost of the initial solution (line 8). If $c(E) < c(sol\_init)$, an improving solution has been found, search continues until the termination criterion is fully satisfied. If $c(E) \not< c(sol\_init)$, it is assumed an improving solution is not likely to be found. Then, tabu search initiates its diversification phase (lines 10 and 11).

During the diversification phase, tabu search executes a predefined number of iterations rs with the following transition rule:

$$S = \min\{c(S') | S' \in \mathcal{N}_i(S) \text{ and } M[s'] = i\}. \tag{11}$$

The condition $M[s'] = i$ ensures that for each transformation $(s' \rightarrow s)$, the block s' entering in the current solution S belongs strictly to A_i. During the diversification phase, blocks entering S cannot be removed from S. This limits the number of iterations during this phase to less than $b+$

1. Once the diversification phase is completed, tabu search applies the same transition rule as before the diversification phase (line 2) until the termination criterion is fully satisfied (line 1).

If the cost of the best solution E found by tabu search after the diversification phase is greater than the cost of the initial solution (line 12), a decision must be made about which solution between E and the initial solution is returned by tabu search. The *explore* parameter guides this decision. The elite solution E is returned if $c(E) \leq c(sol\_init) + explore$ (line 12), otherwise, the initial solution is returned (line 13).

Note, the input parameter $diversify$ in Figure 1 is a Boolean controlling the activation of the diversification phase. When this parameter is turned-off, the heuristic in Figure 1 is an independent tabu search heuristic for computing covering designs. We call this heuristic "one level tabu search".

tabu_search(sol_init,$stop\_tabu$,$diversify$,rs, M, i, $explore$)
0. $E = S = sol\_init$; $stop = 0$;
1. **while** (tabu search stopping criterion not satisfied) **do**
2. $S = \min\{c(S') \mid S' \in \mathcal{N}_i(S)\}$;
3. update tabu lists;
4. **if** $(c(S) \leq c(E))$ **then**
5. $E = S$; $stop = 0$;
6. **else** $stop = stop + 1$;
7. **if** $(stop = \frac{stop\_tabu}{2} \& diversify)$ **then**
8. **if** $(c(E) \not< c(sol\_init))$ **then**
9. $c(E) = \infty$;
10. **for** $(q = 0; q < rs; q++)$ **do**
11. $S = \min\{c(S') | S' \in \mathcal{N}_i(S)$ and $M[s'] = i\}$;
12. **if** $(diversify \& (c(E) > (c(sol\_init) + explore)))$ **then**
13. $E = sol\_init$;
14. **return** E;

Figure 1: Tabu search heuristic for computing covering designs

4.5 Search behavior of the one level tabu search

This is a brief analysis of the search behavior for our one level tabu search. For illustration, assume $\lambda = 1$ and let $u_S(s)$ be a function returning the number of t-subsets covered exclusively by a block $s \in S$ (a t-subset is covered exclusively by a block $s \in S$ if it does not appear in any other block of S). Let s, s' be a pair of blocks such that $s \in S$, $s' \notin S$ and blocks

s, s' differ by exactly one point. We can express the cost of each neighbor $S' \in \mathcal{N}(S)$ in terms of $c(S)$ and the difference between the number of t-subsets covered exclusively by $s \in S$ and the number of t-subsets covered exclusively by $s' \in S \setminus s$:

$$c(S') = c(S) + u_S(s) - u_S(s'). \tag{12}$$

By selecting the best neighbor, transition rule (8) exchanges in S the pair of blocks s, s' minimizing the term $(u_S(s) - u_S(s'))$. Usually, blocks that cover exclusively few t-subsets are quickly removed from S. On the other hand, blocks that cover exclusively many t-subsets tend to form a stable subcollection of blocks that cannot be removed from S. Once blocks with low $u_S(s)$ have been removed from S, the search becomes very much driven by the blocks in $\overline{S}$, the set of blocks s' which differ by exactly one point with a block $s \in S$. For example, to minimize the cost differential $u_S(s) - u_S(s')$, the block s in the transformation $s \rightarrow s'$ selected by the transition rule must cover exclusively a number of t-subsets similar to the number of t-subsets covered exclusively by the block s'. Tabu search iterations execute transformations involving pairs of blocks s, s' where the cost differential $u_S(s) - u_S(s')$ is either zero, a small positive or a small negative value. This yields the characteristic plateaus in the exploration of the cost function.

To scape from such plateaus, the current solution must be subjected to stronger perturbations than simply accepting small cost increasing transitions as in transition rule (8). For example, we can re-initialize the search randomly or re-initialize the search in a new region of the search space by selectively perturbing the decision variables. In this study, cooperation operators perturb the set $\overline{S}$, i.e., the set neighbors of the current solutions. For example, re-coarsening replaces blocks in A_i, which impacts the neighborhood of solutions at level i. Interpolations bring elite solutions to lower levels where there are more blocks, adding blocks to $\overline{S}$. Cooperation operators create conditions for the search to move away from plateaus, this without destroying the information contained in current solutions.

5 Multilevel cooperative search algorithm

This section details our multilevel cooperative search strategy for the covering design problem. Figure 2 gives an overview of the algorithm. Lines 1 to 4 is the initialization phase. Line 1 is the coarsening of the source problem instance. In line 3, a first initial solution S_i is computed for each level i by selecting randomly b blocks among the blocks in set A_i. In line 4, a first elite solution E_i is computed by the tabu search procedure at each

corresponding level.

The **while** loop of line 6 is the main loop of the algorithm, it controls the multilevel search. Variable j (lines 5 and 13) is the loop index. In each iteration, tabu search procedures are run concurrently at all levels. Each tabu search procedure is initialized with a new solution at each loop iteration. An iteration of the main loop starts or ends with the execution of a cooperation operator. Iterations of the main loop are synchronized, iteration $j+1$ starts only once operations at all levels are completed for iteration j. Three consecutive loop iterations involving only re-coarsening operations are performed (lines 7 and 8) for each iteration where interpolations are executed (lines 9 and 10). We name *re-optimization phase* the iterations of the main loop where only re-coarsening operations are executed. Figure 3 gives the pseudo-code of the re-coarsening operations and synchronization of the tabu search procedures during the re-coarsening phase (sub-routine **search_re-coarsening**). We name *interpolation phase* the iterations of the main loop were interpolation operations are executed. In this phase, tabu search at level i is initialized with an elite solution from level $i+1$. It is also during this phase that direct interpolation operations are executed. Computation involved during interpolation phases is described in Figure 6 (sub-routine **interpolation_search**). After every 4 iterations of the main loop, the search at level l is restarted (line 12).

Multilevel cooperative search algorithm( )

1. coarsening phase;
2. **for** $(i = l; i \geq 0; i--)$
3. generate a solution S_i by selecting randomly b blocks in A_i;
4. $E_i =$ **tabu_search**$(S_i, stop\_tabu, false, 0, M, i, explore)$;
5. $j = 1$;
6. **while** (none of multilevel termination criteria is satisfied)
7. **if** $(j \bmod 4 \neq 0)$
8. **search_re-coarsening**$(j, E_R, stop\_tabu, rs, M, explore)$;
9. **if** $(j \bmod 4 = 0)$
10. **interpolation_search**$(j, E_R, stop\_tabu, rs, M, explore)$;
11. **if** $((j \bmod 4 = 1) \& (j \neq 1))$
12. $E_R =$ **restart_search**(j);
13. $j++$;

Figure 2: Main procedure of the multilevel cooperative search algorithm

5.1 Re-optimization phase

The purpose of the re-optimization phase is, for tabu search procedures at levels 1 to l, to re-optimize coarsened problems P_1 to P_l after they have been re-defined by re-coarsening operations. During the re-optimization phase, for each iteration j of the main loop, tabu search at level i is initialized with E_i^{j-1}, the elite solution at level i of iteration $j-1$ in the main loop (lines 3 and 4 in Figure 3). E_i^{j-1} is a solution that has been visited at least once in previous iterations. For this reason, the search strategy during re-optimization phases fit the description of an intensification in promising regions of the search space.

search_re-coarsening$(j, E_R, stop\_tabu, rs, M, explore)$
1. **if** $(j \bmod 4 = 1)$ **then** $E_l^{j-1} = E_R$
2. **for** $(i = 0; i \leq l-1; i++)$ **do**
3. **fork**$(E_i =$ **tabu_search**$(E_i^{j-1}, stop\_tabu, true, rs, M, i, explore))$;
4. $E_l =$ **tabu_search**$(E_l^{j-1}, stop\_tabu, false, 0, M, l, explore)$;
5. **joint**();
6. **for** $(i = 0; i \leq l-1; i++)$ **do**
7. **re-coarsening**(E_i);

Figure 3: Search_re-coarsening procedure

During our investigation, we observed that re-optimization based on repeated initializations of tabu search with E_i^{j-1} works better than a re-optimization strategy based on a single call to the search procedure. This could be explained in part by the flatness of the cost function. Solutions often have more than one best neighbor (at least for the lowest levels). Our tabu search heuristic breaks tights randomly. This, together with the length of tabu lists set randomly and independently at each level and at each iteration of the main loop, ensures the same search does not repeat systematically even when tabu search is re-initialized with the same solution. There is a second and more general rational justifying this re-optimization strategy. Covering designs have a large symmetry group. Consequently, if a problem has a solution with cost 0, the number of covering designs is potentially large. When solutions are plentiful but difficult to find, it is often better to intensify search in a small region of the search space (where there might be a covering design) than to apply a same number of search steps to explore the whole search space.

In Figure 3, the Boolean input parameter $diversify$ is set to *true* for levels 0 to $l-1$ (line 3) (it is set to false for level l, line 4). The diversification phase in tabu search also helps to prevent that the search repeats systemat-

ically. Diversification provides the re-coarsening operator with new blocks that update the sets A_1 to A_l, changing the content of $\overline{E_i^{j-1}}$, the set of blocks which differ by exactly one point with a block in solution E_i^{j-1}.

In Figure 3, line 3, **fork** is a parallel programming primitive instructing the operating system to create an independent computing thread. There are $l+1$ independent threads including the thread for the master program, each thread runs one tabu search procedure. The **joint** operation of line 5 is a synchronization primitive forcing the master program to wait on line 5 until all tabu search procedures have completed before the execution of the loop controlling re-coarsening operations (line 7).

5.1.1 Re-coarsening operator

Assume elite solution E_i is the current best known solution of problem P_i. Re-coarsening proceeds in two steps (Figure 4). In step 1, each block $s \in E_i$ is copied in A_l by setting the value of $M[s]$ to l (line 5). In step 2, blocks are removed from set A_l (line 3). This proceeds as follows. The value of $M[s]$ for $s \in E_i$ ranges from i to l. Given $s \in E_i$ such that $M[s] = g$, a block $s' \in A_l$ is selected randomly and the value of entry $M[s']$ is changed to g (line 4). The purpose of this second step is to keep constant the cardinality of sets A_i to A_l.

re-coarsening(E_i)
  $E_i' = E_i$;
  while ($E_i' \neq \emptyset$)
1. randomly choose a block $s \in E_i'$;
2. $E_i' = E_i' \setminus s$;
   if ($M[s] \neq l$) **then**
3. randomly choose $s' \in A_l$ such that $s' \notin E_l \cup E_{l-1} \cup \cdots \cup E_i$;
4. $M[s'] = M[s]$;
5. $M[s] = l$;

Figure 4: Re-coarsening operator

Figure 5 illustrates a re-coarsening operation. In this figure, elite solution E_0 at level 0 re-coarsen problems P_1 and P_2 at level 1 and level 2 respectively. In Figure 5.a, elite solution E_0 contains, among others, blocks 8, 10, 11 and 32, all belonging to set A_0. Blocks 8 and 11 belong strictly to A_0 and consequently have value 0 in M. Block 32 belongs strictly to A_1 and has value 1 while block 10 belongs strictly to A_2 and has value 2.

After copying E_0 in set A_2 (Figure 5.b), all the blocks of E_0 belong strictly to A_2 (Figure 5.c). To keep constant the cardinality of sets A_0 to A_2, the value of block 18 at level 2 is changed to 1 and the value of blocks 6 and 48 is changed to 0 (Figure 5.c). Figure 4 provides a pseudo-code illustrating the implementation of this re-coarsening operator.

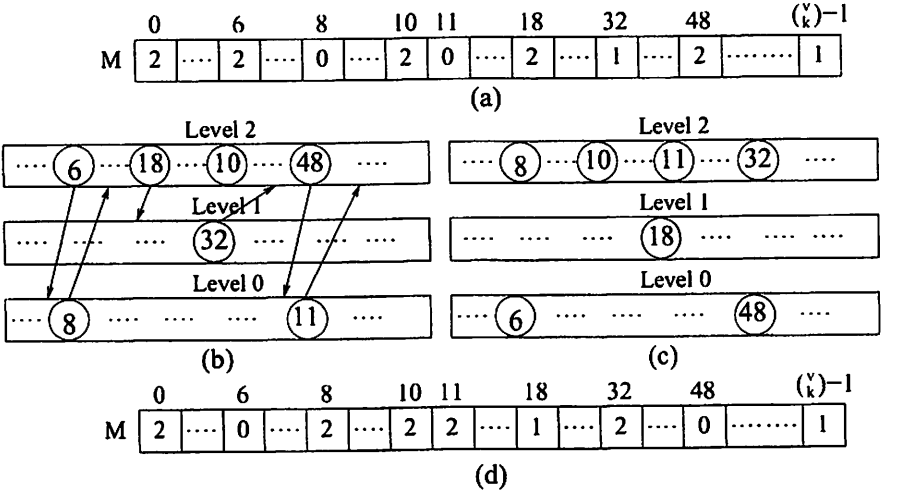

Figure 5: Example of a re-coarsening operation

Each re-coarsening operation copies an elite solution E_{i-1} into sets A_i to A_l, redefining coarsened problems P_i to P_l. The re-coarsening operations (line 7 of Figure 3) copy to each set A_i blocks from $EB_i = E_0 \cup E_1 \cup \cdots \cup E_{i-1}$, which could bring up to $b \times (i-1)$ new blocks in A_i. In practice, this number is often significantly smaller because many blocks in E_0 to E_{i-1} already belong to set A_i. Nonetheless, re-coarsening operations change significantly the problems at the top most levels, particularly at level l.

5.2 Interpolation phase

Interpolation is a class of cooperation operators that provide initial solutions to search procedures for levels 0 to $l-1$. Interpolation operations initialize the search in promising regions of the search space as well as diverting the search away from entrapments in local optima or plateaus of the cost function, or entrapments in other forms of confined regions of the

interpolation_search(j)
1. **for** ($i = 0; i \leq l-1; i++$) **do**
2. **fork**(E_i = **tabu_search**(E_{i+1}^{j-1},*stop_tabu*, $false$, 0, M, i, *explore*));
3 **joint**();
4. **for** ($i = 0; i \leq l-1; i++$) **do**
5. **re-coarsening**(E_i);
6. E_l = **tabu_search**(E_l^{j-1},*stop_tabu*, $false$, 0, M, l, *explore*);
7. E_0 = **tabu_search**(E_l,*stop_tabu*, $false$, 0, M, 0, *explore*);
8. **re-coarsening**(E_0);

Figure 6: Interpolation phase

search space.

In each interpolation phase, two different interpolation operators are executed: *interpolation* between adjacent levels i and $i+1$ and *direct interpolation* between level 0 and level l. Interpolation phases start by the concurrent execution of interpolation operations between adjacent levels (line 2) followed by the concurrent execution of tabu search procedures for levels 0 to $l-1$. Once search based on interpolated solutions is completed (line 3), re-coarsening operations bring the best solutions to level l. A search is performed at level l (line 6). Interpolation phases conclude with a direct interpolation operation between level 0 and level l. The search at level 0 is initialized with E_l. Note, diversification is turn off in tabu search procedures during an interpolation phase.

5.2.1 Interpolations between adjacent levels

This is a familiar operator to multilevel methods. Coarsened problem P_{i+1} is a closed approximation of P_i. It follows that elite solution E_{i+1}^{j-1} identifies a region of $\mathcal{S}_i$ worthy of further exploration because E_{i+1}^{j-1} is also a good solution for problem P_i. Then, the search at level i improves the cost of E_{i+1}^{j-1} by performing a small number of cost-improving transformations $s \rightarrow s'$ of E_{i+1}^{j-1} in the region of the search space $\mathcal{S}_i$ identified by E_{i+1}^{j-1}. We describe one case where tabu search transitions lead to cost improvements of elite solution E_{i+1}^{j-1}.

Let $S \in \mathcal{S}_{i+1}$ be a current solution of tabu search at level i. We have $c(E_i) < c(E_{i+1}^{j-1})$ (line 2) if the following condition is satisfied: For at least one transformation $(s \rightarrow s')$, there is a block $s' \in \overline{S}$ such that $s' \notin A_{i+1}$ and the cost differential $u_S(s) - u_S(s')$ is smaller than any other pair of

blocks s, s' where $s' \in A_{i+1}$. When the coarsening factor cf is sufficiently large, there are several solutions $S \in \mathcal{S}_{i+1}$ that satisfy the above condition, consequently, as observed empirically, such block s' is often found.

5.2.2 Direct interpolations between levels 0 and l

Direct interpolation is a new interpolation operator for multilevel cooperative algorithm and multilevel methods. The reasons this operator provides good initial solutions differ from adjacent interpolations. Coarsened problem P_l is a very distant (coarse) approximation of the source problem P_0, therefore E_l is not very specific about a region of the source problem worthy of further exploration. Furthermore, the cost of the elite solution E_l computed in line 6 is usually high, higher than the cost of any elite solution copied in A_l. In terms of cost, elite solution E_l is not a good solution at level 0, not even a very good solution for level l. Nonetheless, elite solution E_l is a good initial solution for the following two reasons: Solution E_l contains sub-collections of blocks from different elite solutions copied in A_l. These sub-collections approximate promising regions of the search space $\mathcal{S}_0$. Let $\overline{u_{E_l}} = \frac{\sum_{n=1}^{b} u_{E_l}(s)}{b}$ be the average (mean) number of t-subsets covered exclusively by blocks in E_l. The standard deviation σ_{E_l} of $u_{E_l}(s_1), u_{E_l}(s_2), \ldots, u_{E_l}(s_b)$ is much smaller than the corresponding standard deviation for random initial solutions and smaller than the standard deviation of elite solutions copied in A_l. This property of E_l delays the entrapment of tabu search in a confined region of the search space for the source problem.

The case of level l is particular as the search at this level cannot be re-initialized by interpolation operations. Rather, at the beginning of each re-optimization phase, a new initial solution S_R is computed at level l by selecting randomly b blocks almost entirely among blocks from elite solutions that have been recently in set A_l (see Section 5.3). Elite solution E_l derives from this initial solution. Without lost of generality, assume S_R is composed of a sub-collection E_{p_1} of $\frac{b}{2}$ blocks from E_p and a second sub-collection E_{q_1} of $\frac{b}{2}$ blocks from E_q where E_p and E_q are elite solutions that have been copied in A_l by re-coarsening operations. Since S_R is a random initial solution, we have $c(S_R) > c(E_p)$ and $c(S_R) > c(E_q)$. The cost of S_R is higher because pairs of blocks s_{p_1}, s_{q_1} for $s_{p_1} \in E_{p_1}$ and $s_{q_1} \in E_{q_1}$ cover the same t-subsets each in E_{p_1} and in E_{q_1}. Consequently, since these blocks cover the same t-subsets in S_R, everything else been equal, we have $u_{S_R}(s_{p_1}) < u_{E_p}(s_{p_1})$ and $u_{S_R}(s_{q_1}) < u_{E_q}(s_{q_1})$. Under the assumption that pairs of blocks like s_{p_1}, s_{q_1} occur uniformly among blocks of E_{p_1} and E_{q_1}, there will be blocks $s \in S_R$ for which $u_{S_R}(s) > \overline{u_{S_R}}$ but $u_{S_R}(s) < u_{E_p}(s)$

(respectively $u_{S_R}(s) < u_{E_q}(s)$). In words, the number of t-subsets covered exclusively is smaller in general for blocks in S_R (compared to E_p and E_q) including for the blocks $s \in S_R$ that cover exclusively a large number of t-subsets in E_p and E_q.

Standard deviation is lower for E_l because of the following three factors. First, several blocks that cover exclusively many t-subsets in S_R also belong to E_l. As explained in Section 4.5, these blocks form a stable sub-collection of blocks that cannot be easily removed from the current solution. Second, if there are blocks $s \in S_R$ such that $u_{S_R}(s) = 0$ or $u_{S_R}(s)$ is very small, they will usually be the first one to be replaced by tabu search and will not occur in E_l. This is because blocks $s \in S_R$ that cover exclusively few t-subsets minimize the term $(u_S(s) - u_S(s'))$ in equation (12), even when the value of $u_S(s')$ is not so high. Finally, in the sequence of tabu search transitions between the initial random solution S_R and the elite solution E_l, tabu search cannot introduce blocks that cover exclusively many t-subsets in the current solution S. This is because the small size of A_l limits considerably the number of neighbors each solution has in the search space $\mathcal{S}_l$. In a small set like A_l, the cardinality of $\overline{S}$ is also very small, $\overline{S}$ does not have blocks s' that cover exclusively many t-subsets. This is why in practice, the cost of E_l is usually higher than elite solutions found at lower levels.

Conditions at level 0 are ideal to improve the cost of E_l. The set A_0 is considerably larger than A_l, there are plenty of solutions in $\mathcal{S}_l$ that have cost-improving neighbors at level 0. The transfer of E_l to level 0 brings many new blocks in $\overline{E_l}$. The high cost of E_l creates enormous opportunities at level 0 to find blocks $s' \notin A_l$ for a current solution S at level 0 for which the cost differential $u_S(s) - u_S(s')$ is smaller than any other pair of blocks s, s' where $s' \in A_l$. On the other hand, the low standard deviation for E_l guides tabu search towards good solutions for the source problem. The term $(u_S(s) - u_S(s'))$ in equation (12) is not minimized by small values of $u_S(s)$. Consequently, the minimization of $(u_S(s) - u_S(s'))$ at level 0 is more likely to compete for block $s' \in \overline{S}$ with the highest value $u_S(s')$. There are more blocks available in the current solution S at level 0 to choose from in order to perform cost-improving transitions, it is then much more difficult to trap tabu search in a particular region of the search space. Finally, independently of the standard deviation of E_l, blocks in E_l that may trap tabu search in a region of the search space belong to elite solutions E_p and E_q. This is because, some blocks in E_{p_1} (respectively E_{q_1}) complement each other very well in covering t-subsets and will not be removed from the current solution S.

In summary, an interpolation between adjacent levels i and $i+1$ provides a good initial solution E_{i+1}^{j-1} to search at level i because E_{i+1}^{j-1} is already a

good solution for level i. Solution E_l in direct interpolation is a good initial solution because it leads to a more extensive exploration of the search space. Indeed, the high cost of E_l means that the search at level 0 has the potential to perform several cost-improving transitions and to reach a broad range of regions in the search space. Furthermore, blocks in E_l are such that search is not easily trapped in plateaus of the cost function. In our tests, search following direct interpolations (line 7) makes the largest cost improvements to the overall best solution for the multilevel cooperative search, larger than adjacent interpolations between levels 0 and 1 (line 2).

5.3 Initializing search at level l

Some form of re-initialization at the top most level is needed in a multilevel cooperative search algorithm. The restart operation (line 12 Figure 2) re-initializes the search at level l based on a solution interpolated from a meta-set of blocks called the set R (for "restart" set). The set R is handled as a special level above level l, though the blocks in R do not constitute a subset of A_l. Blocks in R are:

1. the best overall elite solution in each of the main loop most recent iterations;
2. blocks selected randomly among the blocks that belong strictly to A_0;
3. blocks from A_l.

Once the set R is formed (line 1 Figure 7), an initial solution S_R is obtained by selecting randomly b blocks from R (line 2). Then, tabu search explores the search space generated by the set R. Tabu search returns elite solution E_R (line 12 Figure 2), which signals the beginning of a new re-optimization phase. The search at level l is initialized with E_R (line 1 in Figure 3). Since R is not a subset of A_l, to obtain a feasible initial solution at level l, the blocks in E_R which do not belong to A_l are copied in A_l.

Since re-initialization of level l is based on set R rather than A_l, it is less likely that elite solution E_l will be one of the elite solutions copied in A_l by re-coarsening operations. The number of blocks from A_0 copied in R is used as a control parameter of this algorithm. For example, if we want the search at level l to be independent of the search history, this number is set with high values.

restart_search(j);
1. Build set R;
2. Compute an initial solution S_R by selecting randomly b blocks from R;
3. **while** (termination tabu search criterion for restart_search not satisfied)
   E_R = **tabu_search**(S_R,*stop_R*, $false$, 0, M, 0, *explore*);

Figure 7: Restarting search at level l

6 Experimentation

We compare the upper bounds obtained using our algorithm with upper bounds obtained by different constructive techniques for covering design (see [2, 14, 16, 19, 20] for a survey of these techniques). Up to date results for covering design are published in La Jolla Covering Repository Tables [13]. These tables report the best known upper bounds for $C_1(v,k,t)$ (referred as $C(v,k,t)$ from now on) for problems with parameters $v \leq 32$, $k \leq 16$, $t \leq 8$ and $\lambda = 1$. They first appeared in [14]. Since then, they are constantly updated on the web site [13], where they are independently validated before being uploaded. Computation times are not reported in La Jolla Tables. We report our computation times for runs where new upper bounds are found. We have also set the same maximum CPU time for all runs, this is one of the stopping criteria of our algorithm. Computation times and the maximum CPU time provide an appraisal of the computational resources spent for our tests[1].

Tests have been conducted on a subset of 158 covering design problems in La Jolla Tables [13] for which $C(v,k,t)$ was unknown. We have limited our experimentation to this subset of problems primarily for the following two reasons: Computer memory requirements for storing the tables of neighbors (Section 4) exceeded the capacity of several of our computers for $v > 20$. Consequently, we have limited our tests to problems where $v \leq 20$. The most computationally intensive operation of this multilevel cooperative search algorithm is to evaluate neighborhoods: $|\mathcal{N}(S)| = b \times k \times (v-k))$. In this expression, b can vary from 4 to 18497 in problems where $v \leq 20$. Such large variation would have had different impacts on our tests. For example, it would have been difficult to calibrate the control parameters. We have limited our tests mostly to problems where $b < 200$, this represents about 80% of the problems for which $v \leq 20$.

Tests have been conducted on 41 dedicated sequential computers. Con-

[1]Our CPU times cannot be used to compare our runs with each others as tests have been conducted on processors with different clock rates.

current computing threads interleave on the same processor. The 41 computers belong to the following computer architectures: Sun UltraSPARC 10 300-MHz, Sun UltraSPARC 5 400-MHz, Sun UltraSPARC III 600-MHz, Pentium III 866MHz and Pentium IV 2.2 GHz. This experimentation section first describes the control parameters of our multilevel cooperative search algorithm and the calibration procedure. Next, numerical results are reported and analyzed.

6.1 Control parameters

Table 1 lists the control parameters of the multilevel cooperative search algorithm and the range of the values that have been used during the tests. The first row of Table 1 refers to the stopping criteria of the multilevel cooperative search (MCP). There are two stopping criteria:

1. CPU time;
2. total number of iterations in the main loop of the algorithm (line 6 Figure 2).

A third implicit stopping criterion is the discovery of a new upper bound for $C(v, k, t)$. The range in Table 1 refers to the total number of loop iterations. The maximum CPU time was set to 168 hours. The search ends whenever one of these two stopping criteria is satisfied.

Second row is the number of levels. Row 3 is the number of blocks in A_l. This number should be at least $b \times (l + 3)$. We can choose to have $|A_l| > b \times (l + 3)$. Three options are available to fill A_l:

1. copy in A_l randomly selected blocks that belong strictly to A_0;
2. keep the blocks already in A_l;
3. combine the first two options.

Control parameter r_l in row 4 states the number of random blocks that are copied in A_l. When $r_l > 0$, random blocks are copied in A_l at the beginning of the restart_search sub-routine in a similar way as for the re-coarsening operator. In fact, copying random blocks in A_l is a randomized re-coarsening, coarsened problems P_1 to P_l are redefined by the r_l blocks. If $|A_l| > b \times (l + 3) + r_l$, the difference are blocks that have not been kick out of A_l by the different re-coarsening operations.

	Parameters	*Typical range*
1	Stopping criteria for MCP	$600 - 1000$
2	Number of levels (l)	$3 - 6$
3	Size of A_l	$b \times (l+3) - 3000$
4	Number of random blocks in A_l (r_l)	$20 - 400$
5	Stopping criterion for tabu search	$500 - 800$
6	Pivot t_1 of the first tabu list	$10 - 12$
7	Pivot t_2 of the second tabu list	5
8	# iterations during diversification phase (rs)	$40 - 60$
9	Exploration factor (*explore*)	$0 - 20$
10	Size of the restart set R	$120 - 1200$
11	# of random blocks in R (R_r)	$20 - 100$
12	Depth of the tabu search exploration in R	$2 - 60$
13	Phases before unconditional diversification (UD)	$3 - 5$
14	# iterations during UD	$40 - 60$
15	# of blocks fixed after UD	$20 - 50$
16	Exploration factor for UD	$20 - 50$

Table 1: Control parameters and range of their values

Parameters in rows 5 to 9 are control parameters for tabu search during re-optimization and interpolation phases. Parameter 5 specifies the value of the stopping criterion for tabu search (see Section 4). Parameters 6 and 7 specify the value of the pivots around which the length of the tabu lists can be set randomly in a range pre-specified in the code of the algorithm. Parameter 8 is the number of tabu search iterations during the diversification phase of the algorithm. This number is expressed in percentage of b. Parameter 9 specifies the value of the exploration factor.

Parameters in rows 10, 11 and 12 control the restart_search operator. Parameter 10 specifies the size of R. Parameter 11 specifies the number of blocks which are selected randomly among blocks that belong strictly to A_0. Parameter 12 specifies the stopping criterion of the tabu search run in R. The stopping criterion is specified in the same terms as for parameter 5, i.e., the number of tabu iterations performed without improving the current best solution.

In some re-optimization phases, the diversification phase of tabu search is performed unconditionally. Furthermore, a certain number of blocks introduced in the current solution S during the diversification phase are prohibited from leaving S even after the diversification phase is completed. The

execution of a unconditional diversification phase is triggered after a predefined number of iterations of the main loop (Figure 2) have been executed without improving the overall best solution of the multilevel cooperative search. Parameter 13 specifies the number of loop iterations required to trigger a re-optimization phase with unconditional diversification. Parameter 14 has the same interpretation as parameter 8. Parameter 15 identifies the number of blocks which cannot be removed from S after the diversification phase. Parameter 16 has the same interpretation as parameter 9. These implementation details are not shown in the algorithm.

The procedure to calibrate the control parameters is as follows. Initially, tests have been conducted on a reduced set of problems of small size. Runs with the best solutions (some with new upper bounds) were kept separately. A preliminary range of settings was derived from this separated set of runs. For each parameter, the lower bound of the range was set to be the minimum value of this parameter in the separated set of runs. Similarly, the upper bound of the range of each control parameter was set to be the maximum value of this parameter in the separated set of runs. Two other sets of calibration runs have been conducted. One set of runs aimed at adjusting the initial range to problems of larger size (essentially widening the ranges). A last set of runs has been conducted to analyze the response of the multilevel cooperative search algorithm to changes in the values of parameters 4, 8, 9, 11, 13, 14, 15, 16. This last calibration essentially aimed at finding control parameter settings most likely to yield new upper bounds in the time frame allocated to each test run (168 hours). As apparent in our numerical results, these settings may not be suitable to find large improvements in the upper bounds. No attempt has been made to optimize the settings for specific problems.

6.2 Numerical results

Comparisons with the best known upper bounds in La Jolla Tables are reported in Tables 2 to 5. From left to right, the header of each column refers respectively to the problem with parameters (v, k), lower bound (LB), best known upper bound (UB) at the time of our tests, size of the covering design we tested (b), number of t-subsets not covered by our best solution (c). When an entry in the fifth column is 0, it means that all t-subsets have been covered, b is a new upper bound for the problem instance with parameters (v, k, t). Each time a new upper has been found for b, the test was restarted with $b = b - 1$. Tables 2 to 5 report the smallest value of b for which we have found a new upper bound.

(v,k)	LB	UB	b	c	(v,k)	LB	UB	b	c
(12,5)	27	29	28	2	(18,11)	9	10	9	3
(13,5)	32	34	33	4	(19,5)	103	108	107	2
(13,6)	20	21	20	2	(19,6)	57	63	62	0
(14,5)	37	43	42	3	(19,7)	33	35	34	4
(14,6)	24	25	24	1	(19,8)	24	27	26	1
(15,5)	54	56	55	2	(19,9)	15	17	16	8
(15,6)	30	31	30	1	(19,10)	13	14	13	9
(16,5)	61	65	64	3	(19,11)	9	11	10	9
(16,6)	35	38	37	2	(20,5)	124	133	132	14
(16,7)	23	24	23	3	(20,6)	64	72	71	0
(17,6)	43	44	43	1	(20,7)	43	45	44	2
(17,7)	25	27	26	12	(20,9)	20	21	20	8
(17,8)	17	18	17	8	(20,10)	14	15	14	4
(18,7)	31	33	32	2	(20,11)	11	12	11	16
(18,9)	14	16	15	1	(20,12)	9	10	9	10
(18,10)	11	12	11	8					

Table 2: Tests for $t = 3$

In general, we found fewer new upper bounds for small v, k, t, though these problems are smaller and easier to solve. But for many of these problems, the gaps between lower and upper bounds are close, improving the upper bound may amount proving lower bound. This is difficult, since it involves either 1- using an exhaustive search or 2- proving that all possible collections of b blocks does not yield a covering design. When v, k, t are small, it may be possible to prove lower bounds, however, most of these have been done, and the ones left over are hard. Nonetheless, new upper bounds are reported for small v, k, t, some where the gap is very close (see for example entry $(17, 11)$ in Table 3 and entry $(13, 9)$ in Table 5). an indication our method is quite competitive. This becomes more obvious for medium size problems where we improve the upper bound of several problems (Table 4 and problems on the left side of Table 5).

In Tables 2 to 5, we observe that the number of t-subsets not covered generally increase for larger values of v and k. The number of iterations executed by the main loop is a factor. For example, problem instance $(20, 15)$ for $t = 8$ met the CPU time termination criterion after completing less than 200 iterations. A second factor is the smaller effort spent to calibrate for larger problem instances. Finally, our algorithm didn't perform well for some particular problems like for example $(14, 7)$ and $(15, 7)$ in Table 4. This likely also happens for larger problems.

Table 6 gives the CPU time taken by the tests for which improved upper bounds are reported in Tables 2 to 5. From left to right, column headers

(v,k)	LB	UB	b	c	(v,k)	LB	UB	b	c
(12,6)	40	41	40	6	(17,10)	19	23	22	4
(12,7)	20	24	23	1	(17,11)	13	16	15	0
(13,6)	59	66	65	2	(18,7)	111	130	126	0
(13,7)	28	30	29	3	(18,8)	57	66	65	2
(14,6)	75	80	79	8	(18,9)	34	38	37	6
(14,7)	40	44	43	2	(18,10)	24	26	25	8
(14,8)	23	24	23	9	(18,11)	15	19	18	18
(15,6)	93	117	116	1	(19,7)	131	153	152	0
(15,7)	52	57	56	1	(19,8)	74	84	83	3
(15,8)	29	30	29	11	(19,9)	45	48	47	3
(15,9)	19	20	19	9	(19,10)	27	32	31	13
(15,10)	12	14	13	6	(19,11)	19	23	22	10
(16,6)	144	152	151	6	(19,12)	15	17	16	22
(16,7)	69	76	75	1	(20,8)	83	93	92	2
(16,9)	24	26	25	6	(20,9)	54	64	63	0
(16,10)	16	18	17	3	(20,10)	30	36	35	18
(17,7)	85	99	98	6	(20,11)	24	28	27	14
(17,8)	49	54	53	0	(20,12)	15	20	19	36
(17,9)	27	28	27	9	(20,13)	14	16	15	9

Table 3: Tests for $t = 4$

display the problem parameters, the best known upper bound previously to our improvements, the value of the best new upper bound we have found, the number of tabu search iterations executed before finding the new upper bound and finally, the elapsed CPU time in seconds to find the new upper bound. We observe in Table 6 that the new upper bounds are found quite rapidly (problem $(17, 8, 4)$ is the longest, around 147 CPU hours). This is related to the setting of the control parameters. Diversification actions were activated soon after the search failed to find cost-improving solutions in a particular region of the search space. These diversification actions involve calls to randomization and the use of re-coarsening operations with poor quality solutions. On the long term, these actions dilute the global guidance of the search by the cost function and multilevel cooperation. If new upper bounds are to be found, then it will usually be soon in the computation. In fact, the time where new upper bounds are reported to have been found in Table 6 parallel very much the discovery of the last overall best solutions in many of the tests reported in Tables 2 to 5. To make use of more CPU time to compute better upper bounds, one has to reduce the weight that parameters like 4, 9, 11 and 13 have on the search behavior of the multilevel cooperative search algorithm. It has been shown [7] that, under the control of a multilevel cooperative search, heuristics like tabu search are better optimizer once given more CPU time.

(v,k)	LB	UB	b	c	(v,k)	LB	UB	b	c
(11,6)	96	100	99	1	(17,9)	58	80	79	0
(11,7)	33	34	33	3	(17,10)	41	49	48	0
(12,7)	55	59	58	2	(17,11)	25	32	30	0
(13,7)	75	78	77	34	(18,9)	98	113	112	2
(13,8)	33	43	42	0	(18,10)	49	54	53	12
(14,7)	118	138	137	1	(18,11)	32	42	41	6
(14,8)	49	55	54	5	(18,12)	20	24	23	123
(14,9)	28	32	31	4	(19,10)	65	86	83	0
(15,7)	161	189	188	1	(19,11)	42	50	49	0
(15,8)	75	89	88	2	(19,12)	29	38	37	0
(15,9)	39	42	41	15	(19,13)	18	21	20	139
(15,10)	24	27	26	1	(20,10)	90	106	99	0
(16,8)	104	117	116	12	(20,11)	50	65	64	4
(16,9)	52	61	60	0	(20,12)	32	42	41	16
(16,10)	31	37	34	0	(20,13)	24	33	32	4
(16,11)	18	22	21	22	(20,14)	16	18	17	248
(17,8)	147	188	178	0					

Table 4: Tests for $t = 5$

Table 7 reports the outcomes of tests that assess the impacts the randomized steps have on the performance of our algorithm. Problems listed in Table 7 are problems for which a new upper bound have been found by our algorithm. Each row of this table reports the number of tabu search iterations and the CPU time taken by three independent runs of our algorithm using a same setting of control parameters for each run (settings may differ between problems in different rows). There are important variations in the amount of CPU time to find a new upper bound between runs on a same problem. However, a new covering design is always found.

Table 8 compares simulated annealing, our one level tabu search and multilevel cooperative search. Tests have been conducted on problems for which multilevel cooperative search found new upper bounds. Tabu search and simulated annealing procedures are given the same amount of CPU time (in seconds in Table 8) as multilevel cooperative search using computers with same clock rates. We report the number of iterations[2] performed by each procedure as well as the number of t-subsets that have not been covered. Results in Table 8 show that the tabu search procedure is very competitive. These results also confirm that multilevel cooperation substantially improves the performance of tabu search. More extensive tests have been conducted with the one level tabu search where it found independently 5 of the new upper bounds identified by the multilevel cooperative

[2]One iteration of simulated annealing is like one neighbor evaluation for tabu search. Simulated annealing evaluates only one or very few neighbors at each iteration.

$t = 6$

(v,k)	LB	UB	b	c	(v,k)	LB	UB	b	c
(12,7)	165	176	175	2	(17,10)	89	119	118	1
(12,8)	50	51	50	6	(17,11)	48	61	60	7
(13,8)	90	100	99	0	(17,12)	26	36	35	31
(13,9)	38	40	39	0	(18,11)	68	89	88	1
(14,8)	132	151	150	2	(18,12)	38	48	47	55
(14,9)	52	74	70	0	(18,13)	24	28	27	134
(14,10)	27	29	28	32	(19,11)	85	102	101	12
(15,9)	82	100	99	0	(19,12)	51	76	75	3
(15,10)	42	54	52	0	(19,13)	30	42	41	103
(16,9)	134	172	170	0	(19,14)	21	22	21	362
(16,10)	63	77	76	0	(20,12)	70	93	92	0
(16,11)	35	44	43	2	(20,13)	45	66	65	90
(17,9)	197	268	245	0	(20,14)	26	32	31	458

$t = 7$

(v,k)	LB	UB	b	c	(v,k)	LB	UB	b	c
(13,8)	269	297	295	0	(17,12)	50	64	63	4
(13,9)	73	79	78	6	(17,13)	25	26	25	296
(14,9)	140	166	163	0	(18,12)	72	101	100	15
(14,10)	54	57	56	0	(18,13)	36	50	49	168
(15,10)	78	118	115	0	(19,13)	56	84	83	70
(15,11)	37	42	41	49	(19,14)	33	42	41	229
(16,10)	132	167	165	0	(20,13)	79	136	135	210
(16,11)	62	88	85	0	(20,14)	43	60	59	595
(17,11)	98	127	126	12	(20,15)	28	34	33	531

$t = 8$

(v,k)	LB	UB	b	c	(v,k)	LB	UB	b	c
(14,10)	103	119	117	0	(17,13)	37	42	41	396
(15,11)	74	80	79	5	(18,13)	70	103	102	48
(16,11)	114	191	190	5	(18,14)	33	34	33	839
(16,12)	50	59	58	39	(19,14)	49	75	74	246
(17,12)	88	138	137	18	(20,15)	44	57	56	746

Table 5: Tests for $t = 6, 7$ and 8

(v,k,t)	*Previous UB*	*New UB*	*Tabu Iterations*	*CPU Time*
(13,8,5)	43	42	1006263	16780
(13,8,6)	100	99	18833	1639
(13,8,7)	297	295	675823	48331
(13,9,6)	40	39	3962341	199081
(14,9,6)	74	70	920688	20621
(14,9,7)	166	163	170608	10448
(14,10,7)	57	56	12150	891
(14,10,8)	119	117	1977944	144094
(15,9,6)	100	99	260927	70032
(15,10,6)	54	52	266206	74306
(15,10,7)	118	115	292485	42315
(16,9,5)	61	60	148508	20486
(16,9,6)	172	170	362941	177913
(16,10,5)	37	34	2516428	398476
(16,10,6)	77	76	1467652	184592
(16,10,7)	167	165	303568	36961
(16,11,7)	88	85	185564	24160
(17,8,4)	54	53	41396983	529789
(17,8,5)	188	178	354106	91311
(17,9,5)	80	79	2680690	138870
(17,9,6)	268	245	412836	192413
(17,10,5)	49	48	899131	278101
(17,11,4)	16	15	712030	7617
(17,11,5)	32	30	415428	56489
(18,7,4)	130	126	4572362	366055
(19,6,3)	63	62	301834	8853
(19,7,4)	153	152	11560	1138
(19,10,5)	86	83	714778	84262
(19,11,5)	50	49	1813365	191949
(19,12,5)	38	37	4164742	490706
(20,6,3)	72	71	15469025	351503
(20,9,4)	64	63	6815489	445288
(20,10,5)	106	99	2379806	353505
(20,12,6)	93	92	250548	330744

Table 6: Computational times on successful problems

(v,k,t)	b	*First run*		*Second run*		*Third run*	
		Tabu Iterations	*CPU Time*	*Tabu Iterations*	*CPU Time*	*Tabu Iterations*	*CPU Time*
(12,5,3)	29	19226	13	9271	6	7903	5
(13,6,4)	66	58003	212	33264	121	13725	45
(17,11,4)	16	171268	2024	386865	4431	15050	209
(13,8,5)	43	402150	3004	957527	7084	1251202	9185
(19,11,5)	50	175252	18732	89050	10144	18187	2469
(19,12,5)	37	4164742	490706	1377491	154184	1385207	166421
(13,9,6)	40	592921	6106	157792	1651	1803051	18480
(14,10,7)	55	493573	12661	906425	23060	597453	15408

Table 7: Comparison of different runs with same parameter setting

search algorithm.

	Multilevel Cooperative Search		*Simulated Annealing*		*Tabu Search*	
(v,k,t)	*CPU Time*	*Tabu Iterations*	*Iterations*	c	*Iterations*	c
(13,8,5)	839	50313	330061196	4	39824	5
(16,9,5)	20486	148508	4820470893	2	185693	7
(16,10,5)	48867	366183	5727645000	11	315167	4
(17,11,5)	56489	415428	3178988604	27	225053	24
(13,8,6)	1639	18833	1032767980	14	53867	11
(14,9,6)	20621	920688	6007147920	12	215842	8
(16,9,6)	177913	362941	46526949504	185	726651	11
(14,10,7)	891	12150	205696966	10	13098	9
(16,11,7)	24160	185564	2184703573	32	39775	0

Table 8: Comparison between different search heuristics

Research is very active regarding the application of constructive methods to improve upper bounds. New upper bounds are reported regularly on the web site [13]. The bulk of these new results are for large problems, particularly $v > 20$, an uncharted territory for constructive methods until recently. Upper bounds are found because more powerful computers are used and because the gaps between bounds for large values of v is often substantial. Based on results on the right side of Table 5, the current implementation of our algorithm cannot tackle efficiently these larger problems. Executing the current algorithm on a parallel computer, parallelization of the neighborhood evaluation, re-calibration of the control parameters and other simple optimizations will address partially this problem. For the largest problems, the exact evaluation of the neighborhoods will have to be replaced by a selective evaluation of a subset of neighbors. For large values of b, the cost of neighborhood evaluations can be reduced by coarsening the

vector solution S, i.e., by fixing to "1" some of the decision variables.

7 Conclusion

In this paper, we have introduced a multilevel cooperative search heuristic for computing upper bounds on $C_\lambda(v,k,t)$, the minimum number of blocks in any $t-(v,k,\lambda)$ covering design. A new tabu search heuristic for the covering design problem has been used as the underlying search method of this multilevel cooperative search. In our tests, this new tabu search heuristic outperformed simulated annealing, a widely used search heuristic for covering designs. We report numerical results for 158 problems of small and medium size where this new tabu search heuristic is embedded in our multilevel cooperative search heuristic. The algorithm was given a maximum of one week of CPU time on standard desktop computers to find new upper bounds for each tested problem. Multilevel search and cooperation drastically improve the performance of tabu search. New upper bounds have been found for 34 problems. It is at this point the best search heuristic for covering designs.

The multilevel cooperative search algorithm is based on two new cooperation operators: a multilevel re-coarsening operator and a direct interpolation operator. In particular, the approach followed by direct interpolation to compute initial solutions has never been used before by multilevel methods. Empirically, this approach has worked remarkably well for the covering design problem. It will be interesting to confirm weather the initial solutions computed for the direct interpolation operations depend specifically on the present re-coarsening strategy for subset problems. We have reason to believe it will not be the case and that most re-coarsening strategies can be adapted to direct interpolation. We think it will be possible to generalize the direct interpolation operation to other multilevel methods and possibly to other search heuristic methods.

The current multilevel cooperative search strategy can be applied to other combinatorial design problems such as packing design and t design. Similar cost function, neighborhood structure and the same three cooperation operators can be used as well for these problems. There are other subset problems similar to covering design such as feature selection in bioinformatics and data mining. At least, the main framework of the re-coarsening and direct interpolation operations can be applied in a multilevel cooperative search algorithm for such problems. Finally, the potential for the present algorithm to find new upper bounds to covering design problems is far from been exhausted. By setting control parameters differently or specifically,

the present algorithm can be used as useful constructive method to reduce gaps between bounds of covering design problems.

Acknowledgments

Funding for this project has been provided by the Natural Sciences and Engineering Council of Canada through its Research Grant program. We also acknowledge the Center for Research on Transportation of the Université de Montréal, Canada Foundation for Innovation through the Heterogeneous Distributed Computing laboratory at the University of Manitoba and the Department of Computer Science of the University of Manitoba for providing the computing resources for this project.

References

[1] S.T. Barnard and H.D. Simon. A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems. *Concurrency: Pratice & Experience*, 6(2):111–117, 1994.

[2] J.A. Bate. *A Generalized Covering Problem.* PhD thesis, University of Manitoba, 1978.

[3] A. Brandt. Multi-level adaptive solutions to boundary value problems. *Mathematics of Computation*, 31:333–390, 1977.

[4] E.K. Burke, J.D. Landa Silva, and E. Soubeiga. Multi-objective Hyper-heuristic Approaches for Space Allocation and Timetabling. In T. Ibaraki, K. Nonobe and M. Yagiura, editor, *Meta-heuristics: Progress as Real Problem Solvers*, pages 129–158. Springer, 2005.

[5] S.H. Clearwater, T. Hogg, and B.A. Huberman. Cooperative Problem Solving. In B.A. Huberman, editor, *Computation: The Micro and the Macro View*, pages 33–70. World Scientific, 1992.

[6] C. J. Colbourn and J. H. Dinitz, editors. *The CRC Handbook of Combinatorial Designs.* CRC Press, 1996.

[7] T.G. Crainic, Y. Li, and M. Toulouse. A Simple Cooperative Multilevel Algorithm for the Capacitated Multicommodity Network Design. *Computer & Operations Research*, 33(9):2602–2622, 2006.

[8] P. Erdős and H. Hanani. On a limit theorem in combinatorial analysis. *Publicationes Mathematicae Debrecen*, 10:10–13, 1963.

[9] T. Etzion, V. Wei, and Z. Zhang. Bounds on the sizes of constant weight covering codes. *Designs, Codes and Cryptography*, 5:217–239, 1995.

[10] F. Glover. Tabu Search – Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[11] F. Glover. Tabu Search – Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[12] F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publishers, 1997.

[13] D.M. Gordon, G. Kuperberg, and O. Patashnik. La Jolla covering repository tables. http://www.ccrwest.org/cover.html.

[14] D.M. Gordon, G. Kuperberg, and O. Patashnik. New constructions for covering designs. *Journal of Combinatorial Designs*, 3(4):269–284, 1995.

[15] T. Hogg and C. Williams. Solving the Really Hard Problems with Cooperative Search. In *Proceedings of the 11th National Conference on Artificial intelligence (AAAI93)*, pages 231–236. AAAI Press, August 1993.

[16] F. Margot. Small covering designs by branch-and-cut. *Mathematical Programming*, 94:207–220, 2003.

[17] W.H. Mills. Covering designs I: coverings by a small number of subsets. *Ars Combinatoria*, 8:199–315, August 1979.

[18] W.H. Mills and R.C. Mullin. Coverings and packings. In *Contemporary Design Theory: A Collection of Surveys*, pages 371–399. Wiley-Interscience Series in Discrete Mathematics and Optimization, 1992.

[19] K.J. Nurmela. Constructing combinatorial designs by local search. Technical report, Helsinki University of Technology, November 1993.

[20] K.J. Nurmela and P.R.J. Östergård. Constructing covering designs by simulated annealing. Technical report, Helsinki University of Technology, January 1993.

[21] M. Ouyang, M. Toulouse, K. Thulasiraman, F. Glover, and J.S. Deogun. Multilevel Cooperative Search for the Circuit/Hypergraph Partitioning Problem. *IEEE Transactions on Computer-Aided Design*, 21(6):685–693, 2002.

[22] R. Rees, D.R. Stinson, R. Wei, and G.H.J. van Rees. Applications of covering designs: Determining the maximum consistent set of shares in a threshold scheme. *Ars Combinatoria*, 53:225–237, 1999.

[23] V. Rödl. On a packing and covering problem. *European Journal of Combinatorics*, 5:69–78, 1985.

[24] J. Schönheim. On coverings. *Pacific Journal of Mathematics*, 14:1405–1411, 1964.

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

[26] M. Toulouse, K. Thulasiram, and F. Glover. Multi-Level Cooperative Search: A New Paradigm for Combinatorial Optimization and an Application to Graph Partitioning. In *5th International Euro-Par Parallel Processing Conference, volume 1685 of Lecture notes in Computer Science*, pages 533–542. Springer-Verlag, August 1999.

[27] C. Walshaw. Multilevel Refinement for Combinatorial Optimisation Problems. *Annals Oper. Res.*, 131:325–372, 2004.