A Multilevel Cooperative Tabu Search Algorithm for the Covering Design Problem

Chaoying Dai
(Ben) Pak Ching Li
Michel Toulouse
Department of Computer Science,
University of Manitoba
Email: {chaoying, lipakc, toulouse}@cs.umanitoba.ca

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 instance for each level of the multilevel search. A new tabu search algorithm is introduced to optimize the problem instance 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 problem instances. Improved upper bounds have been found for 29 problem instances, 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.

Keywords: Multilevel algorithms, Covering design problem, Tabu search heuristic.

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. A $t-(v,k,\lambda)$ covering design is *optimal* if it has $C_\lambda(v,k,t)$ blocks [21]. The *covering design problem* is the problem of determining the value of $C_\lambda(v,k,t)$. The covering design problem has applications in lottery design, data compression and error-trapping decoding [7].

The value $C_\lambda(v,k,t)$ can be determined using exact search algorithms. In the worst case, an exact algorithm will have to test $O(\binom{v}{k}^b)$ combinations of b blocks, which is prohibitive for all but a few set of parameters. For most parameters, only upper bounds for $C_\lambda(v,k,t)$ are known. Research on improving upper bounds for $C_\lambda(v,k,t)$ relies on different approaches [18], including search heuristics [23, 24].

In this paper, the problem of finding new upper bounds for $C_\lambda(v,k,t)$ is modeled as a global optimization problem. Assume b is an integer strictly smaller than the best known upper bound for $C_\lambda(v,k,t)$. We seek a $t-(v,k,\lambda)$ covering design with $|S|=b$ blocks by minimizing a penalty function defined in terms of the number of t-subsets not covered at least λ times in a given set of b blocks [24]. A set of b blocks is an optimal solution if all t-subsets are covered at least λ times.

We introduce a multilevel cooperative search heuristic for computing optimal solutions for the above penalty function. Multilevel cooperative search [29] borrows from cooperative search and multilevel search. Cooperative search [6] is a set of independent search procedures exploring concurrently the solution space of a problem instance. Cooperation takes place when a search procedure gives away information about its best solutions and another procedure adapts its search trajectory based on this information.

Multilevel search is an adaptation of multilevel methods for numerical approximation [3, 4] to global optimization [30]. The multilevel search framework can be described as followed: During the *coarsening phase*, a original problem instance P_0 with solution space $\mathcal{S}_0$ is projected into smaller prob-

lem instances $P_1, P_2, \ldots, P_l$ by reducing, recursively, the number of decision variables with respect to P_0. The projection is such that the search spaces $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_l$ induced respectively from problem instances $P_1, P_2, \ldots, P_l$ satisfy the relation $\mathcal{S}_l \subset \mathcal{S}_{l-1} \subset \cdots \subset \mathcal{S}_1 \subset \mathcal{S}_0$. During an *initial search phase*, a search heuristic optimizes problem instance P_l and the best solution S_l is remembered. During the *refinement phase*, solution S_i is computed for problem instance P_i by *interpolating* the values of its decision variables from a best solution S_{i+1} of problem instance P_{i+1}. The interpolated solution S_i serves as initial solution for a search procedure optimizing problem instance P_i. The refinement phase interpolates and refines feasible solutions until the values for the decision variables of a solution S_0 can be interpolated from a best solution for P_1. This last interpolation provides an initial solution to a search heuristic optimizing problem instance P_0.

In multilevel cooperative search, the concurrent exploration of solution space $\mathcal{S}_0$ by independent search procedures is restricted to search spaces $S_1, S_2, \ldots S_l$. Our multilevel cooperative search heuristic for covering designs is as followed: Let $\binom{X}{k}$ denote the set of all k-subsets in X. Assume $A_1, A_2, \ldots, A_l$ are subsets of $\binom{X}{k}$ such that $A_l \subset A_{l-1} \subset \cdots \subset A_1 \subset A_0 = \binom{X}{k}$. Let $\mathcal{S}_0 = \{S \subset \binom{X}{k} || S| = b\}$ be the solution space when searching for a $t-(v,k,\lambda)$ covering design with b blocks. Assume $\mathcal{S}_i = \{S \subset A_i || S| = b\}$ is the search space induced by set A_i, $i > 0$. Solution space $\mathcal{S}_0$ and search spaces $S_1, S_2, \ldots S_l$ are optimized concurrently and independently by tabu search procedures. Two categories of operators support cooperation among the search procedures. Interpolation operators regularly re-initialize tabu search optimizing $\mathcal{S}_i$ with a best solution $E_j \in \mathcal{S}_j$, $j > i$. Re-coarsening operators copy the blocks belonging to a best solution $E_i \in \mathcal{S}_i$ into sets $A_{i+1}, A_{i+2}, \ldots, A_l$. Each re-coarsening operation based on a solution E_i can potentially re-define search spaces $\mathcal{S}_{i+1}, \mathcal{S}_{i+2}, \ldots, \mathcal{S}_l$.

The penalty function we apply to find covering designs arises directly from the definition of the covering design problem. At least two aspects of this cost function are challenging to neighborhood search heuristics like tabu search. One is that solutions "closed" to each other in any reasonable neighborhood structure often have similar cost, forming large plateaus which offer poor guidance to explore the solution space of problem instances. Second, given a solution S of b blocks, up to $v!$ isomorphic solutions with same cost can be obtained by permuting the points in the b blocks of S. When a solution S is a local optimum, the other solutions from the same permuta-

tion group can constitute a spectacular barrier to local search heuristics on their way toward unexplored and cost improving solutions. In our multilevel cooperative search heuristic, we conjugate tabu search, interpolation and re-coarsening operations to provide favorable conditions to the optimization of this penalty function by local search. Re-coarsening operations fill each set A_i with blocks from best solutions of all search spaces. Next, search focalizes on finding combinations of b blocks among the blocks from best solutions. This locality of the search centered around a set of best solutions provides in turn, through interpolation operations, guidance to explore solution space $\mathcal{S}_0$ as well as search spaces $\mathcal{S}_1$ to $\mathcal{S}_{l-1}$. Focusing search on a small set of blocks make the exploration less prone to visit regions in a same permutation group. Finally, interpolation operations provide an escape way to search when trapped in plateaus of the cost function.

The contributions of this research are the following: A new heuristic method is introduced to compute covering designs. Numerical results are very encouraging and significant considering 1- the number of problem instances for which new upper bounds have been found; 2- the small gaps between lower and upper bounds of many of the problem instances. We describe a mathematical programming formulation for computing covering designs and introduce a coarsening strategy for this combinatorial optimization problem. We experiment with a completely new re-coarsening operator which intensifies search of each problem instance in regions spawn by a few best solutions from different search spaces. A variation of the interpolation operator addresses more directly plateaus in the cost function. Finally, we propose a new tabu search heuristic for covering designs. Tabu search [14, 15] is a well known search heuristic technique which has been applied to a broad range of combinatorial optimization problems. A successful multilevel cooperative search algorithm needs a good search heuristic. Experimental results show the tabu search procedure is competitive with a simulated annealing procedure that has been designed for the covering design problem.

The subsequent sections of this paper are organized as follow. Section 2 provides background information on the covering design problem and the combinatorial optimization model for our search algorithm. In section 3, we describe our coarsening strategy for this optimization problem. Section 4 details our cooperative multilevel tabu search algorithm and Section 5 reports experimental results. Finally, we conclude in Section 6.

2 Background

In this section, we first provide a short background on covering designs. Next, we introduce an integer programming formulation for covering designs and a neighborhood structure for the tabu search heuristic. Finally, we model the typical behavior of search heuristics for covering designs given the cost function and neighborhood structure proposed in this paper.

2.1 Covering designs

The study of covering designs began around the end of the 1930's. Turán (see [7]) 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 ($L_\lambda(v,k,t)$) [27] 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 $C_\lambda(v,k,t)$ [22].

In 1963, Erdős and Hanani [12] 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 [26], 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 [16]. 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. That is, using smaller covering designs to construct larger covering designs [23]. 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 with adding a new point v to all 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 [20] 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.

2.2 Problem formulation

Let $\binom{v}{k}$ be the number of k-subsets in $\binom{X}{k}$. An integer programming formulation in $\binom{v}{k}$ integer decision variables is used for the problem of finding a covering design with b blocks for $t-(v,k,\lambda)$. Each decision variable d_s in the formulation corresponds to a member $s \in \binom{X}{k}$. The domain of the decision variables is the set $\{0,1,\ldots,b\}$ of integers. The value of decision variable d_s represents the number of times the corresponding block $s \in \binom{X}{k}$ is included in the set S of b blocks.

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 $c(S) = 0$, then all t-subsets are covered at least λ times, indicating we have discovered a $t-(v,k,\lambda)$ covering design with b blocks. The integer

programming formulation is as followed:

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

In this paper, we refer equivalently to a set S of b blocks as a feasible solution to the above mathematical programming formulation. Similarly, the *solution space* $\mathcal{S}$ of integer vectors, as defined in (2), is expressed in terms of sets of blocks in $\binom{X}{k}$ as followed:

$$\mathcal{S} = \{S \subset \binom{X}{k} || S| = b\}. \tag{3}$$

2.3 Neighborhood structure

The set $\mathcal{N}(S)$ such that $\mathcal{N}(S) \subset \mathcal{S}$ represents the neighborhood of solution $S \in \mathcal{S}$. To compute $\mathcal{N}(S)$, a mapping function selects m points among the k points of a block $s \in S$ and replaces these points by m other points from the $v-k$ points not belonging to s. A solution S' is a neighbor of S, denoted $S' \in \mathcal{N}(S)$, if S' has $b-1$ identical blocks with S and one block which differs by exactly m points. The size $|\mathcal{N}(S)|$ of the neighborhood $\mathcal{N}(S)$ is constant for all solutions $S \in \mathcal{S}$ and is given by

$$|\mathcal{N}(S)| = b \times \binom{k}{m} \times \binom{v-k}{m}. \tag{4}$$

The value of m is constant and ranges in $1 \leq m \leq \min(k, v-k)$. Each value of m defines a *neighborhood structure* among the solutions of $\mathcal{S}$. In the present research, the neighborhood structure of solution space $\mathcal{S}_0$ is defined by $m = 1$.

2.4 Neighborhood search heuristics

Neighborhood search is a local search heuristic in which search steps proceed according to a neighborhood structure. In the context of a neighborhood search, $\mathcal{N}(S)$ is a set of *candidate solutions* from which a solution is selected

before initiating the next search step. A *pivoting rule* determines which solution $S' \in \mathcal{N}(S)$ is to become the *current solution* of the next search step. For example, in this paper we use the following pivoting rule:

$$S = \min\{c(S') \mid S' \in \mathcal{N}(S)\}. \tag{5}$$

In this pivoting rule, the selection of the next current solution is strongly biased by the cost function. This is typical of neighborhood search heuristics. When neighborhoods are small, each search step can be computed efficiently. Iterations of the search steps, under the guidance of the cost function, usually drive the exploration towards good sub-optimal solutions in small polynomial time. However, for different reasons, there are cost functions which cannot be optimized efficiently in this local search scheme.

Without lost of generality, 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 of solution 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. According to (2) and the neighborhood structure in Section 2.3, the cost $c(S')$ of each candidate solution $S' \in \mathcal{N}(S)$ is the difference between the number of t-subsets covered exclusively by $s \in S$ and the number of t-subsets covered exclusively by $s' \in S \setminus s$:

$$c(S') = c(S) + u_S(s) - u_S(s'). \tag{6}$$

The execution of the pivoting rule in (5) will exchange in S the pair of blocks s, s' which minimizes the cost differential $u_S(s) - u_S(s')$.

When the current solution S of the initial search step is obtained by selecting randomly b blocks, we observed that for several search steps, the pivoting rule finds pairs of blocks s, s' such that $u_S(s) - u_S(s') < 0$. But sequences of improving search steps are not typically occurring while optimizing the cost function (2). Let $\overline{S}$ be the set of blocks $s' \notin S$ such that $s' \in \overline{S}$ differs by exactly one point with at least one block in S. Each time a cost improving search step occurs, a block $s' \in \overline{S}$ is substituted by a block $s \in S$ which covers exclusively only few t-subsets with respect to the current solution S. Each cost improving search step introduces in $\overline{S}$ a block which covers few blocks exclusively. When all blocks $s' \in \overline{S}$ are such that $u_S(s')$ is small, minimizing the cost differential $u_S(s) - u_S(s')$ forces the pivoting rule to select a block $s \in S$ for which $u_S(s)$ is small. Then, search steps are

executed with pairs of blocks s, s' such that $u_S(s)$ and $u_S(s')$ are small, and 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, where search steps have little consequences in the cost of the next current solution.

Search is kept in plateaus by *dominant blocks* $s \in S$ which have the largest $u_S(s)$ values and, consequently, cover exclusively many t-subsets. Dominant blocks are unlikely candidates to leave the current solution. However, some must be removed from the current solution in order to extract the search from a plateau of neighbor solutions having similar cost. A common approach to address this problem is to perturb the current solution beyond the perturbation created by the pivoting rule. For example, one can replace randomly a subset of blocks in the hope that some dominant blocks will be replaced by blocks that do not cover exclusively many t-subsets. This strategy effectively extracts search from the current plateau where it is trapped. Unfortunately, it also destroys information about the search history and does not provided any guidance about re-initializing search in a region of the solution space where cost improvements can be made.

Our penalty function for covering designs is difficult to optimize by local search. Given a neighborhood structure where the size of $\mathcal{N}(S)$ is kept reasonably small, the gradient of the cost function is either null or change sign rapidly. Guidance from the cost function is ineffective. In this paper, we use interpolation operations to assist the cost function by punctually re-initializing the search.

3 Coarsening procedure

In the first phase of any multilevel search algorithm, the original problem instance P_0 is projected into smaller problem instances which approximate P_0. There are at least two general strategies to realize this projection or coarsening of the original problem. The *clustering* approach consists of mapping decision variables of P_0 into clusters. Decision variables that are mapped to the same cluster have the same value. Clustering strategies have first been proposed for graph partitioning problems [1, 5, 19]. Figure 1 illustrates this coarsening strategy. For P_0, nodes represent the original set of decision variables. For P_1 and P_2, nodes are clusters of decision variables from the next lowest level.

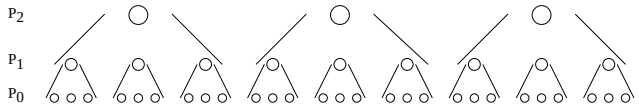

Figure 1: Coarsening based on clustering

For several combinatorial optimization problems, coarsening by clustering decision variables is hardly applicable. In [8], a coarsening strategy is proposed in which projection operators fix the state of some decision variables in the coarsened forms of the original problem. A decision variable is fixed if its value cannot be changed during the subsequent search phases. In Figure 2 below, nodes of P_0 represent the decision variables of a original problem instance. All decision variables are free. In P_1 and P_2, empty nodes and a black nodes represent decision variables which are fixed respectively in state 0 and state 1. Problem instances P_1 and P_2 are coarsened versions of P_0 because they have fewer free decision variables.

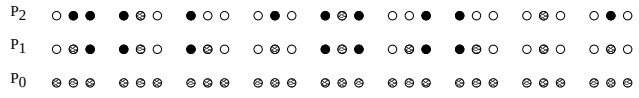

Figure 2: Coarsening based on fixing the state of decision variables

Our coarsening strategy for the problem formulation in (2) fixes to 0 the state of some decision variables. When the state of a decision variable d_s is fixed to 0, the corresponding block $s \in \binom{X}{k}$ cannot be included in solution S. Fixing the state of decision variables to 0 is equivalent to reduce the set $\binom{X}{k}$ to a set $A_1 \subset \binom{X}{k}$ from which blocks can be included in S. The recursive application of this coarsening procedure yields sets of blocks $A_1, A_2, \ldots, A_l$ such that:

$$A_l \subset A_{l-1} \subset \cdots \subset A_0 = \binom{X}{k}. \tag{7}$$

A set of block A_i defines an instantiation P_i of the problem formulation in (2) consisting to find a set of b blocks belonging to A_i such that the number of t-subsets not covered at least λ times is minimized.

The algorithm implementing this coarsening strategy represents sets A_0 to A_l using a single data structure, an integer array M of size $\binom{v}{k}$. Initially, all entries of M are set 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 procedure is applied recursively on set A_i to compute set A_{i+1} until $|A_l|$ entries of M are assigned value l.

The number of blocks assigned to 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 original problem instance, the number of levels $(l+1)$ and the cardinality of the smallest set A_l. The coarsening factor cf is computed as followed:

$$cf \quad = \quad \frac{\binom{v}{k} - (|A_l| \times (l+1))}{(l+1) \times \frac{l}{2}}. \tag{8}$$

The value cf expresses the difference between the number of entries in M which are assigned adjacent values i and $i+1$. The number of entries in M receiving the same value i is given by

$$\sum_{j=1}^{\binom{v}{k}} (M[j] = i) = |A_l| + (l-i) \times cf. \tag{9}$$

Definition 1 *A block s "belongs strictly" to A_i if $M[s] = i$.*

Each block s belongs strictly to exactly one set A_i. The number of blocks that belong strictly to set A_i is given by equation (9).

Since $A_{i+1} \subset A_i$, the blocks belonging strictly to sets $A_{i+1}, A_{i+2}, \ldots, A_l$ are also members of set A_i.

Definition 2 *A block s "belongs" to set A_i if $M[s] \geq i$.*

The number of blocks that belong to set A_i is given by

$$|A_i| = \sum_{j=1}^{\binom{v}{k}} (M[j] \geq i) = (l-1+1) \times |A_l| + \frac{(l-1)(l-i+1)}{2} \times cf. \tag{10}$$

With respect to the integer programming model in (2), a decision variable d_s is considered to be a *fixed decision variable* at level i when $M[s] < i$. A decision variable d_s such that $M[s] \geq i$ is a *free decision variable* at level i. Free decision variables at level i defined an instance of (2). The search space $\mathcal{S}_i$ at level i is $\mathcal{S}_i = \{S \subset A_i \text{ such that } |S| = b\}$. Given $A_{i+1} \subset A_i$,

recursive coarsening induces a hierarchy of increasingly smaller search spaces $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_l$ of $\mathcal{S}_0$ such that

$$\mathcal{S}_l \subset \mathcal{S}_{l-1} \subset \cdots \subset \mathcal{S}_1 \subset \mathcal{S}_0. \tag{11}$$

Given (11), each solution $S \in \mathcal{S}_i$ is also a solution of $\mathcal{S}_{i-1}, \mathcal{S}_{i-2}, \ldots, \mathcal{S}_0$. Each search space $\mathcal{S}_i$ overlaps completely with the solution space $\mathcal{S}_0$.

Recursive coarsening also induces a set of neighborhood structures $\mathcal{N}_i$ on $\mathcal{S}_0$. The neighborhood of solution S in $\mathcal{N}_i$ is defined as:

$$\mathcal{N}_i(S) \quad = \quad \begin{cases} \emptyset, & S \notin \mathcal{S}_i \\ S', & S' \in \mathcal{S}_i \text{ and } S' \in \mathcal{N}_0(S). \end{cases} \tag{12}$$

From (11) and (12), it follows 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

$$\mathcal{N}_i(S) \subseteq \mathcal{N}_{i-1}(S) \subseteq \cdots \subseteq \mathcal{N}_0(S). \tag{13}$$

4 Search and cooperation operators

This section introduces our tabu search heuristic for computing covering designs. We also describe the interpolation and re-coarsening operators of this multilevel cooperative search algorithm.

4.1 Tabu search

Our tabu search heuristic exhaustively evaluates the cost of all neighbors $S' \in \mathcal{N}_i(S)$ of the current solution S and selects the next current solution according to the pivoting rule of equation (5). Two tabu lists prevent short term cycling of the search steps. A first tabu list prohibits the reversal of recent moves. A move $(s' \rightarrow s)$ brings block s' into the current solution S and removes block s from S. The first tabu list prohibits the opposite move $(s \rightarrow s')$ by storing move $(s \rightarrow s')$ in the tabu list. In order to control situations where $(s' \rightarrow s)(s \rightarrow t) \cdots (t \rightarrow s')(s' \rightarrow s)$, the move $(s' \rightarrow s)$ is also stored in this first tabu list. A second tabu list disallows block s' from leaving S for a pre-defined number of iterations after entering S. This second tabu list has greatly improved the performance of our tabu search heuristic.

The length of the two tabu lists varies randomly in 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 interval $t-2$ to $t+2$.

The second tabu list imposes strong limitations on the exploration of the solution space, and therefore 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 move $s' \rightarrow s$, then the opposite move $s \rightarrow s'$ cannot occurred. On the other hand, our penalty function has many local optima which are a few moves away from each other. It is preferable to remember several previously visited solutions. Given the first tabu is less constraining on the search, we usually set t_1 which determines the interval of the first tabu list as twice the size of t_2, which determines the interval of the second tabu list.

Each tabu search step has to compute the cost of each candidate solution in the neighborhood $\mathcal{N}(S)$. To speed-up this computation, neighbors of each solution $S \in \mathcal{S}$ and respective cost are stored in tables. Computing the cost of $\mathcal{N}(S)$ is reduced to fetch $O(\mathcal{N}(S))$ entries from these tables [24]. The mapping function in [28] is used to rank the $\binom{v}{k}$ blocks and to index the tables.

The termination criterion for the tabu search heuristic is a pre-defined number of iterations without improving the best known solution. The termination criterion is an input parameter. Pseudo-code of our tabu search heuristic is provided in Figure 3.

```
tabu_search(initial_solution)
  E = S = initial_solution;
  while (tabu search stopping criterion not satisfied) do
    S = S' ∈ N(S); (S' is the best solution in the neighborhood of S)
    update tabu lists;
    if (c(S) ≤ c(E)) then
      E = S;
  return E;
```

Figure 3: Tabu search heuristic for covering designs

4.2 Re-coarsening operator

According to relation 11, the search space $\mathcal{S}_{i+1}$ is a sub-space of $\mathcal{S}_i$. Given that $\mathcal{S}_{i+1}$ is smaller than $\mathcal{S}_i$, we can draw the conclusion that $\mathcal{S}_{i+1}$ is easier than $\mathcal{S}_i$ to search. We also note that searching $\mathcal{S}_{i+1}$ is equivalent to explore regions of $\mathcal{S}_i$ which are overlapped by the search space $\mathcal{S}_{i+1}$. The purpose of the re-coarsening operator is to modify the original projection of P_i into P_{i+1} (denoted as $P_i \xrightarrow{p} P_{i+1}$) such that the induced search space $\mathcal{S}_{i+1}$ overlaps with the best solutions of $\mathcal{S}_i$. The end goal is to use the easiness of the search in $\mathcal{S}_{i+1}$ to discover the best solutions in $\mathcal{S}_i$. To obtain a better overlapping, the re-coarsening operator uses the cost function as a parameter in the projection $P_i \xrightarrow{p} P_{i+1}$. Figure 4 illustrates one possible implementation of a re-coarsening procedure. The cost function parameter in this projection is a set of decision variables which have the same state among the best solutions of P_i. In Figure 4, solutions E_{0-1}, E_{0-2} and E_{0-3} are best solutions of P_0. Decision variables which appear in the same state in all 3 solutions of P_0 are fixed in this same state in the definition of problem instance P_1, replacing the previous set of fixed decision variables. This re-definition of P_1 induces a new search space $\mathcal{S}_1$. The new search space $\mathcal{S}_1$ overlaps with the regions of the solution space $\mathcal{S}_0$ where the state of the decision variables is the same as the set of fixed decision variables in $\mathcal{S}_{i+1}$.

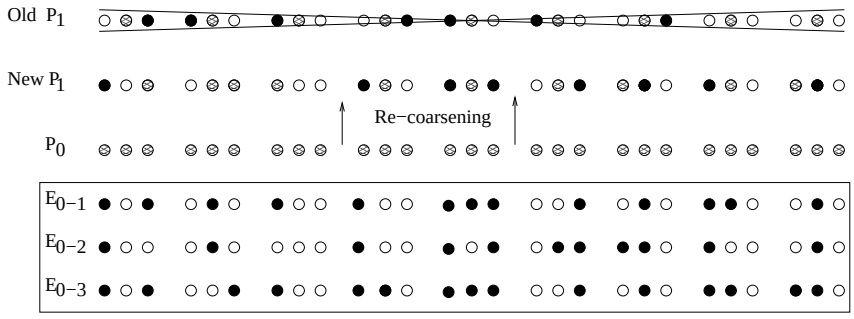

Figure 4: Illustration of the re-coarsening concept

The re-coarsening strategy in the present multilevel cooperative search algorithm has a slightly different objective. A best solution $E_i \in \mathcal{S}_i$ is used as a parameter in the projections $P_i \xrightarrow{p} P_{i+1}$, $P_{i+1} \xrightarrow{p} P_{i+2}, \ldots, P_{l-1} \xrightarrow{p} P_l$. This in turn means that each problem instance P_i is re-defined by the best solutions

$E_0, E_1, \ldots, E_{i-1}$ of problem instances P_0 to P_{i-1}. Obviously, re-coarsening makes $\mathcal{S}_i$ to overlap with regions of $\mathcal{S}_0$ to $\mathcal{S}_{i-1}$ identified by $E_0, E_1, \ldots, E_{i-1}$. Most importantly, this re-coarsening strategy intensifies the search at each level in the small sub-spaces of $\mathcal{S}_0$ identified by the set of best solutions in A_i. We now describe the implementation of this re-coarsening strategy.

Assume elite solution E_i is the current best solution of problem instance P_i. Re-coarsening proceeds in two steps: 1- the blocks of elite solution E_i are copied in set A_l; 2- blocks are removed from set A_l and copied in sets A_i to set A_{l-1} to keep constant the cardinality of these sets. For each block $s \in E_i$, the value of $M[s]$ ranges from i to l. In step 1, a block $s \in E_i$ is copied in A_l by setting the value of $M[s]$ to l. In step 2, for each block $s \in E_i$ such that $M[s] = g$, we select randomly a block $s' \in A_l$ and make $M[s'] = g$.

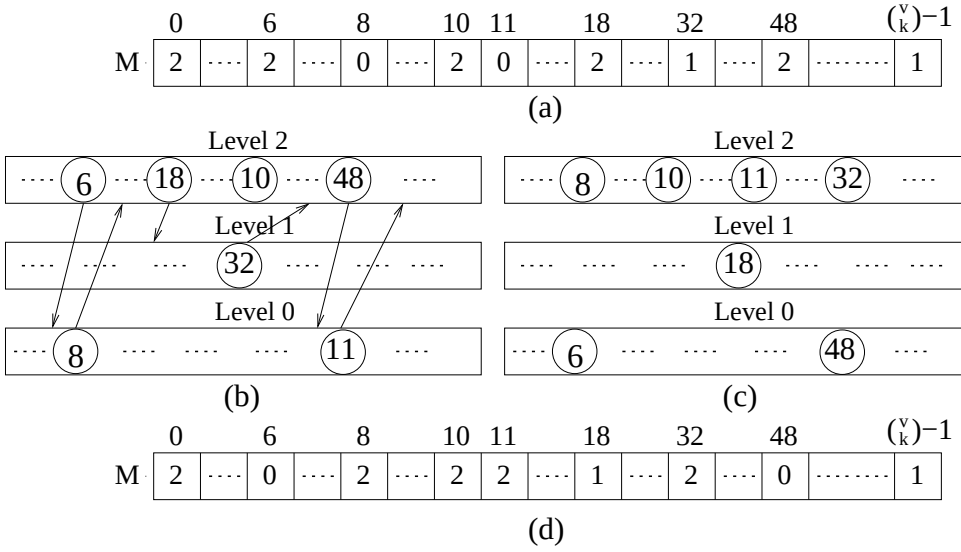

Figure 5: Example of a re-coarsening operation

Figure 5 illustrates a re-coarsening operation in which an elite solution E_0 from level 0 makes a re-coarsening of sets A_1 and A_2. Assume elite solution E_0 contains blocks 8, 10, 11 and 32, which all belong to set A_0. In E_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 (see Figures 5.a and 5.b). After copying E_0 into set A_2, all the blocks of E_0 belong strictly to A_2 (Figure 5.c). In order 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). Once this re-coarsening operation is completed, the search spaces $\mathcal{S}_1$ and $\mathcal{S}_2$ overlap with a subset of the solutions in $\mathcal{S}_0$ that contain blocks from E_0. Figure 6 gives the pseudo-code for this re-coarsening operator.

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 6: Re-coarsening operator

Note that blocks that belong to an elite solution are free decision variables. Re-coarsening operations create new free decision variables. According to relation (7), once a block $s \in E_i$ is copied in A_l (line 5 in Figure 6), and $s \notin A_j$, $i < j \leq l$, block s becomes a new member of sets A_j to A_l. The value of block s in M is changed from $M[s] = j - 1$ to $M[s] = l$. Given that $M[s] \geq i$ for all $i = 0, \ldots, l$, the corresponding decision variable d_s is a free decision variable for all levels, including levels $j, j+1, \ldots, l$. Re-coarsening intends to improve search in smaller search spaces by providing the same set of free decision variables as in the best solutions of larger search spaces.

To keep constant the cardinality of sets A_{j+1} to A_l, for each new block s entering A_l such that $M[s] = j - 1$, a block $s' \in A_l$ is selected randomly and the value of $M[s']$ is changed from l to $j - 1$ (lines 3 and 4 in Figure 6). In this way, block s' is removed from sets A_j to A_l. Given that $M[s'] = j - 1$, we have $M[s'] < i$ for levels i, $i = j, j+1, \ldots, l$. The corresponding free decision variable $d_{s'}$ at level l is changed into a fixed decision variable for levels j to l.

4.3 Interpolation operator

An interpolation operation uses a best solution S_i from the search space $\mathcal{S}_i$ to compute an initial point S_j for exploring search space $\mathcal{S}_j$, $j < i$. In general, there are more decision variables at level j in comparison with i. In order to compute an initial point in search space $\mathcal{S}_j$ from a solution in $\mathcal{S}_i$ which has fewer decision variables, we have to interpolate values for the decision variables of problem instance P_j that do not exist in problem instance P_i. Interpolation is dependent on the coarsening strategy. Given that our coarsening strategy is to fix to 0 the state of some decision variables, the interpolation operator simply copy the value of each decision variable in S_j from the corresponding fixed or free decision variable in S_i.

4.4 Multilevel cooperative search algorithm

We are now ready to describe our multilevel cooperative search algorithm for the problem of finding improved upper bounds to $C_\lambda(v, k, t)$. This algorithm is summarized in Figure 7. The concurrent optimization of problem instances P_0 to P_l is achieved by a decomposition of the search into consecutive cycles. For each cycle, tabu search is run at each level together with the execution of the cooperation operators. We distinguish between two types of cycles according to the primary cooperation operator in action during the cycle: *search* cycles (Figure 8) and *interpolation* cycles (Figure 9). The rest of the pseudo-code in Figure 7 describes the other phases of the algorithm: Line 1 refers to the coarsening phase where problem instances $P_1, P_2, \ldots, P_l$ are generated. The **for** loop of line 2 controls the initialization of the search at each level. In line 3, an initial solution S_i for level i is computed by selecting randomly b blocks from A_i. In line 4, a first elite solution E_i for level i is obtained by executing tabu search until it satisfies its stopping criterion. At regular intervals, search at level l is restarted (line 12). Variable j in Figure 7 stands for the overall cycle index of the multilevel cooperative search procedure.

4.4.1 Search cycles

The purpose of search cycles is to optimize newly defined problem instances P_1 to P_l following re-coarsening operations. Three short consecutive search cycles are performed (line 7 in Figure 7). For each search cycle j, a tabu

Multilevel cooperative search algorithm( )
1. run the coarsening procedure;
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);
5. $j = 1$;
6. **while** (multilevel termination criterion not satisfied)
7. **if** $(j \bmod 4 \neq 0)$
8. **search_cycle**(j);
9. **if** $(j \bmod 4 = 0)$
10. **interpolation_cycle**(j);
11. **if** $((j \bmod 4 = 1) \& (j \neq 1))$
12. **restart_search**(j);
13. $j + +$;

Figure 7: Multilevel cooperative search for covering design

search procedure at each level i is initialized with elite solution E_i^{j-1} of level i from the previous cycle $j - 1$ (line 2). Once a search cycle is completed, the re-coarsening operation copies the blocks from elite solution E_i into A_l (line 7). (Note, search proceeds from level l to level 0 such that the re-coarsening operation at level i in the current cycle j does not re-defined problem instances P_{i+1} to P_l before the are explored during cycle j).

Search cycles explore more intensively regions of each search space $\mathcal{S}_i$ which are spawn by blocks belonging to elite solutions. This situation is created by the re-coarsening operator and the initialization of search cycles with elite solutions. This is particularly obvious at level l. Re-coarsening operations bring in set A_l during a same cycle elite solutions from levels 0 to $l - 1$. The size of A_l is usually small enough such that blocks in A_l belong mostly to elite solutions. Therefore, the search space $\mathcal{S}_l$ is spawn largely by elite solutions. Solutions in $\mathcal{S}_l$ combine blocks from different elite solutions, and search steps in $\mathcal{S}_l$ explore these re-combinations guided by the neighborhood structure $\mathcal{N}_l$ at level l.

Search at levels 0 to $l - 1$ goes potentially through three different phases. In the first phase (line 2, Figure 8), tabu search explores the whole search space $\mathcal{S}_i$ according to neighborhood structure $\mathcal{N}_i$. In the first phase, search is largely driven by elite solutions. Obviously, blocks which do not belong to

search_cycle$(j, exploration\_factor)$
1. **for** $(i = l; i \geq 0; i--)$ **do**
2. $E_i =$ **tabu_search**(E_i^{j-1});
3. **if** $(i \neq l)$ **then**
4. **if** $(c(E_i) \geq c(E_i^{j-1}))$ **then**
5. $E_i =$ restricted search at level i;
6. $E_i =$ **tabu_search**(E_i);
7. **else**
8. $E_i =$**tabu_search**(E_i);
9. **if** $(c(E_i) \leq c(E_i^{j-1}) + exploration\_factor)$ **then**
10. **re-coarsening**(E_i);
11. **else re-coarsening**(E_i^{j-1});

Figure 8: Search cycle

any elite solution are included in the current solution S_i in some search steps. However, these blocks tend not to stay in S_i, and most importantly, when improving solutions are found in this phase, they are largely recombinations of blocks from the elite solutions in A_i. There is a flip side to this intensification of search in regions spawn by elite solutions. To have re-coarsening, some blocks must not already belong to an elite solution. Otherwise, there is not enough re-coarsening to sustain a diversified exploration of the solution space. Consequently, after completing half of the iterations of the tabu search termination criterion, the cost of the current best solution E_i is compared with the cost of the initial solution E_j^{j-1} (line 4). If $c(E_i) < c(E_i^{j-1})$, an improving solution has been found, search continues in $\mathcal{S}_i$ until the tabu search termination criterion is satisfied at level i (line 8). If $c(E_i) \geq c(E_i^{j-1})$, we assume an improving solution is less likely to be found within the current cycle at level i. Then, search enters a second phase called the *restricted search* (line 5) where the objective is to find new blocks for re-coarsening.

During the second phase, search steps are based on the following pivoting rule:

$$S_i = \min\{c(S_i') | S_i' \in \mathcal{N}_i(S_i) \text{ and } M[s'] = i\}. \tag{14}$$

where $s' \in \overline{S_i}$, the set of blocks that differ by exactly one point with a block s in S_i and $M[s'] = i$ means that blocks s' belongs strictly to set

A_i. In a restricted search, blocks that belongs to sets A_{i+1} to A_l, including all the blocks from elite solutions, cannot enter in the current solution S_i. Blocks which enter S_i at each search step during a restricted search cannot be removed from S_i for the duration of the restricted search. The restricted search always starts with E_i^{j-1}, the best solution in cycle $j-1$. The number of search steps performed during the restricted search is smaller or equal to b.

Usually, restricted search increases substantially the cost of the current S_i. The restricted search is followed by a third phase where search explores the whole search space $\mathcal{S}_i$, any blocks that belong to A_i can enter the current solution S_i, including blocks from the elite solutions (line 6). The third phase always starts with the last solution visited during the restricted search. The goal of the third phase is to obtain a new elite solution E_i which, hopefully, improves the cost of the initial elite E_i^{j-1} at level i while in the same time, keeps a few blocks belonging strictly to A_i, ensuring proper re-coarsening of problem instances. If the best solution E_i found in the second or third phase is such that $c(E_i) \geq c(E_i^{j-1})$, a decision must be made about which solution between E_i and E_i^{j-1} is copied in A_l. The $exploration\_factor$ parameter guides this decision. E_i is the selected solution for re-coarsening if $c(E_i) \leq c(E_i^{j-1}) + exploration\_factor$ (line 9), otherwise, E_i^{j-1} is sent back in A_l (line 11), which provides no re-coarsening.

The design of the re-coarsening operator as well as the sequences of short consecutive search cycles aims to reduce cycling of the search in same permutation groups or in solutions equivalent in other ways under the cost function. Search focuses on small sets of blocks and re-combine intensively blocks from these sets. This is the opposite of a strategy that would have aimed at forming combinations of b blocks from a large set of different blocks. We also think this is an excellent search strategy for combinatorial design problems like covering design. Empirical investigations led us to conclude that for the covering design problem, it doesn't seem we need a lot of different blocks to find good solutions. Rather, our observations indicated that computational efforts should be spent on finding the right combinations among small set of blocks.

4.4.2 Interpolation cycles

The pseudo-code of Figure 9 describes interpolation cycles. Two types of interpolation operations are used: interpolation between adjacent levels $i+1$,

interpolation_cycle(j)
1. **for** $(i = 0; i \leq l-1; i++)$ **do**
2. Interpolate S_i from E_{i+1}^{j-1};
3. $E_i =$ **tabu_search**(S_i);
4. **re-coarsening**(E_i);
5. $E_l =$ **tabu_search**(E_l^{j-1});
6. Interpolate S_0 from E_l;
7. $E_0 =$ **tabu_search**(S_0);
8. **re-coarsening**(E_0);

Figure 9: Interpolation cycle

i and interpolation between levels l, 0. The **for** loop of line 1 controls interpolations between adjacent levels. During these interpolations, tabu search is run from level 0 to level $l-1$. Search at level i is initialized with a solution S_i interpolated from the elite solution E_{i+1}^{j-1} of level $i+1$ (line 2). Once tabu search has completed a run at level i, $i < l$, a re-coarsening operation is performed using E_i which re-coarsen levels $i+1$ to l. Note that changing the coarsening of levels $i+1$ to l has no impact on elite solutions $E_{i+1}^{j-1}, E_{i+2}^{j-1}, \ldots, E_l^{j-1}$. Finally, at level l, tabu search computes a new elite solution E_l (line 5) using E_l^{j-1} from cycle $j-1$ as initial solution. At the end of each interpolation cycle, an interpolation operation is executed at level 0 using E_l (line 6), followed by tabu search (line 7) and a re-coarsening (line 8) based on the new elite solution E_0.

The interpolation between adjacent levels works as followed. An initial solution S_i for tabu search at level i is interpolated from an elite solution E_{i+1}^{j-1} from level $i+1$ by setting the decision variables of S_i in the same state as decision variables of E_{i+1}^{j-1}. Given S_i and E_{i+1}^{j-1} are a same solution, both solutions have same cost. According to relation (13), we have $\mathcal{N}(E_{i+1}^{j-1}) \subseteq \mathcal{N}(S_i)$. Consequently, the cost of the best neighbor in $\mathcal{N}(S_i)$ is not worst than the cost of the best neighbor in $\mathcal{N}(E_{i+1}^{j-1})$. Furthermore, search following interpolation usually improves the cost of the interpolated solution. Since $|A_i| > |A_{i+1}|$, there are blocks $s \in A_i$ such that $s \notin A_{i+1}$. The cost of E_{i+1}^{j-1} will be improved, if for at least one search step, there exists 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 for any other pair of blocks s, s' where $s' \in A_{i+1}$. If the coarsening factor

cf is sufficiently large (see Section 3), such block s' is likely to be found. Empirically, we observed that search at level i (line 3 in Figure 9) almost always yields an elite solution E_i at level i which cost $c(E_i)$ improves over the cost $c(S_i)$ of the interpolated solution S_i. We note finally that when $s' \notin A_{i+1}$, we also have $s' \notin A_l$ and consequently s' does not belong to any elite solution. In general, elite solutions E_i (line 4) have a good re-coarsening profile because they contain several blocks that do not belong to any previous elite solution.

For the interpolation between level l and level 0, an initial solution S_0 is interpolated from the best solution $E_l \in \mathcal{S}_l$. The large difference in the number of blocks between A_l and A_0 amplifies considerably the observations made in the previous paragraph regarding interpolations between adjacent levels. For example, the neighborhood $\mathcal{N}(S_0)$ of S_0 is much larger than $\mathcal{N}(E_l)$. The set $\overline{S_0}$ of blocks s' that differ by exactly one point from a block in S_0 is much larger than set $\overline{E_l^{j-1}}$ at level l. As a consequence, the cost $c(E_0)$ of the new elite solution at level 0 (line 7) improves considerably on the cost $c(S_0)$. Furthermore, E_0 has usually a very rich re-coarsening profile.

Interpolation operations in this multilevel cooperative search play two roles. As for multilevel search algorithms, interpolation in our multilevel cooperative search refines the search started a higher levels. But interpolation in our multilevel cooperative search is also a mean to re-initialize the exploration of search spaces. These re-initializations handle the problems created by plateaus of our penalty function for covering designs. From experimentations, we determined that after three consecutive search cycles, little cost improvements were made from re-coarsening problem instances or re-starting tabu search with different tabu lists. Interpolation operations extract search from these plateaus. Since re-initialization points are obtained from a complex optimization process, interpolation operations guide the search in regions of the solution space where cost improvements can be made with respect to solutions in the previous plateau. Interpolation operations between adjacent levels perform this function. Interpolation operations between level l and level 0 contribute even more remarkably to cost improvements. Though re-combinations of blocks at level l usually produce elite solutions E_l that have high cost $c(E_l)$ (because of the small number of blocks in A_l), these re-combinations contribute in some way to identify regions of the solution space $\mathcal{S}_0$ where cost improvements can be made. Experimentally, interpolations between level l and level 0 account the largest and the most frequent

overall cost improvements in our algorithm.

4.4.3 Restart search at level l

After three consecutive search cycles and one interpolation cycle, still search at level l has not been re-initialized. We now describe the operator that re-initializes tabu search at level l. This re-initialization uses a solution interpolated from a meta-set of blocks called the R set (for "restart" set). The set R is handled as a special level above level l, though the blocks in R do not necessary constitute a subset of A_l.

restart_search(j);
1. Build set R;
2. Compute an initial solution S_R by selecting randomly b blocks from R;
3. $E_R =$ **tabu_search**(S_R);

Figure 10: Restarting search at level l

Blocks in R are selected in the following way. Let

$$E^{opt_g} = \min\{c(E_0^g), c(E_1^g), \ldots, c(E_l^g)\}$$

where $E_0^g, E_1^g, \ldots, E_l^g$ are the elite solutions of levels 0 to l in cycle g. Let j be the index of the current search cycle. Three categories of blocks are placed in R in the following order:

1. $R = R \cup R_j$ where $R_j = E^{opt_{j-1}} \cup E^{opt_{j-2}} \cup \cdots \cup E^{opt_{j-history}}$, and $history$ is a variable which indicates the number of past cycles we probe in order to restart search at level l;

2. $R = R \cup R_r$ where R_r is a set of blocks randomly selected from the blocks belonging strictly to A_0;

3. If $|R| > |R_j \cup R_r|$ then selected randomly $|R| - |R_j \cup R_r|$ blocks from A_l.

The size of R and R_r are input parameters of the algorithm. Once R has been generated, a set S_R of b blocks is selected randomly from blocks in R. The set S_R is the initial solution of a tabu search which runs in the solution

space generated by the blocks in R. Finally, a new search cycle starts with tabu search at level l and an initial solution interpolated from E_R, the best solution computed by tabu search in R (line 3 in Figure 10). Since R is not necessary a subset of A_l, to obtain a feasible initial solution S_l from E_R, the blocks of E_R which do not belong to A_l are copied in A_l.

Re-initialization of level l induces two divergent actions: a more intensive exploration of previously visited regions of the solution space and diversification from the currently explored regions of the solution space. Intensification is based on elite solutions from previous cycles and diversification is achieved by selecting blocks randomly. The degree of intensification and diversification is expressed in terms of the ratio of each type of blocks in the set R.

5 Experimentations

In this section, we describe the tests that have been performed and report computational results. Next, based on the analysis of the computational results, we propose some future work to improve the current heuristic.

Most search heuristics have several free control parameters often referred as *search parameters*. So first, we briefly describe some search parameters for the implementation of the multilevel cooperative search heuristic that has been used to run tests. We also give the typical settings of these parameters in our tests.

5.1 Search parameters

Search parameters adapt a search heuristic to specific characteristics of problem instances such as size, solution space structure (large plateaus, ridges, narrow valleys, number of local optima), neighborhood structure, etc. Search parameters can also be used to emphasize specific goals such as rapid convergence or on the contrary slow convergence for more difficult problem instances. For complex heuristics like multilevel cooperative search, which involves several "internal" heuristics (tabu search, local search, cooperation operators), there are also search parameters controlling the internal heuristics. Setting search parameters is an important and time consuming task. The setting often makes the difference in the performance of the heuristic.

Tests with our multilevel cooperative search algorithm have been realized with an implementation which had a total of 16 search parameters. Table 1

	Parameters	*Typical range*
1	Stopping criterion for multilevel cooperative search	$600 - 1000$
2	Number of levels (l)	$3 - 6$
3	Size of A_l	$b \times (l+3) - 3000$
4	Number of random blocks r_l in A_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	# of TS iterations in phase 2 of search cycles	$40 - 60$
9	Exploration factor	$0 - 20$
10	Size of the restart set R	$120 - 1200$
11	Number of random blocks in R (R_r)	$20 - 100$
12	Depth of the tabu search exploration in R	$2 - 60$

Table 1: Search parameters and range of their values

lists 12 of these search parameters. The complete list of search parameters is given in [11]. The first row of Table 1 refers to the stopping criteria of the multilevel cooperative search. Two stopping criteria have been used: 1- maximum wall clock time; 2- maximum number of cycles performed by the multilevel cooperative search procedure. A third implicit stopping criterion is the discovery of a new upper bound for $C(v,k,t)$. The computation ends once the procedure reaches any of these stopping criteria. In Table 1, the range refers to the maximum number of cycles.

The parameter in the second row determines the number of levels for the multilevel cooperative search. The parameter in row 3 specifies the size of the set A_l. The value of this parameter should be at least $b \times (l+3)$ to store blocks from elite solutions $E_0, E_1, \ldots, E_l$. For large problem instances, where the ratio $\frac{b \times (l+3)}{\binom{v}{k}}$ tends to get very small, the neighborhood structure $\mathcal{N}_l$ could potentially become disconnected, or neighborhoods too sparsely populated to allow for efficient search. Therefore, the size of A_l has to be larger than $b \times (l+3)$. We have considered two ways to increase the size of A_l for larger problem instances: 1- keep in A_l blocks from elite solutions of previous cycles; 2- add in A_l blocks selected randomly from blocks that belong strictly to A_0. Each of these two approaches by itself has pitfalls. The first approach tends to drag search for too long in same regions of the

solution space. The second approach is a re-coarsening using blocks selected randomly, this dilutes the guidances provided by elite solutions. In practice, we combine both approaches. The parameter in row 4 specifies the number of blocks r_l selected randomly from blocks belonging strictly to A_0. These blocks are copied in A_l at every 4 cycles, when the operator **restart_search** (Section 4.4.3) is executed.

Parameters in rows 5, 6 and 7 control tabu search. Parameter 5 specifies the value of the stopping criterion for tabu search in search and interpolation cycles (see Section 4.1). Parameters 6 and 7 specify the pivot values around which the length of the tabu lists are selected randomly in a range pre-specified in the code of the algorithm.

The parameters in rows 8 and 9 control the behavior of the second phase (restricted search phase) of tabu search in search cycles. Parameter 8 specifies the number of tabu iterations performed during the restricted search. This number is expressed in percentage of b. Parameter 9 specifies the value of the exploration factor. It is preferable to set parameter 9 with small values such to initiate search in each cycle from good initial solutions and to maintain an overlapping of the search spaces with good regions of $\mathcal{S}_0$. Exploration can be achieved by setting search parameters 2, 3, 4 and 11 with large values. Particularly, a large value for search parameter 4 brings new blocks in A_l, which favors a high re-coarsening of problem instances P_1 to P_{l-1}. A high value for parameter 11 also creates more exploration because these new solutions are interpolated to lower levels.

In some search cycles, the second phase of tabu search is performed unconditionally and in the third phase, and a certain number of blocks introduced in S_i during the second phase are prohibited from leaving S_i. Search parameters controlling this implementation of search cycles are described in [11].

Parameters in rows 10, 11 and 12 control the operator **restart_search**. Parameter 10 specifies the size of R. Parameter 11 specifies the number of blocks that belong strictly to A_0 and which are selected randomly from A_0 and copied in R. 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 without improving the current best solution.

The settings of search parameters in Table 1 is based on tests realized with a few small and medium size problem instances ($t = 3, 4, 5$ and $v \leq 15$). These are problem instances where improved upper bounds had been found.

No attempts has been made to conduct a systematic calibration study of the search parameters for this algorithm.

5.2 Tests

We have conducted tests on problem instances from the La Jolla Covering Repository Tables [10] of upper bounds for $v \leq 32$, $k \leq 16$, $t \leq 8$ and $\lambda = 1$. La Jolla Covering Repository Tables first appeared in [18] and are since then constantly updated on the web page in [10]. Almost all of the best known upper bounds for $C_1(v, k, t)$ (referred as $C(v, k, t)$ from now on) are published in those Tables.

We have selected 158 covering design problem instances in La Jolla Tables for which $C(v, k, t)$ was unknown as of Spring 2005. Two criteria have been used to select test instances. One criterion is the computer memory requirements for storing tables of neighbors (Section 4.1). Memory requirements exceeded the memory capacity of several of our computers for $v > 20$. Consequently, we have tested problem instances where $v \leq 20$. The second criterion is related to the size of b. The bulk of the computation for our algorithm is spent computing neighborhoods in tabu search. The size of the neighborhood for a solution S is $|\mathcal{N}(S)| = b \times k \times (v - k)$. Among the three covering design parameters which impact the size of neighborhoods, b is the one with the broader variations, from 4 to 18497 in instances where $v \leq 20$. These variations have a strong impact on the computational time. We have mostly selected test instances where $b < 200$, which represents about 80% of the problem instances for which $v \leq 20$.

The tests have been realized using 41 dedicated sequential computers from the following set of 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. All tests have been run under the same following two stopping criteria: a maximum of 600 to 1000 cycles or 168 hours of wall clock time. Computation ended whenever one of these two criteria was satisfied. These criteria have been applied uniformly across all computer architectures, though we tried to schedule larger problem instances on computer architectures with faster clock rate.

Results for the 158 test instances are summarized in Tables 2 to 5. In those tables, the first column (Instances) identifies the problem instances by listing the values of the parameters for $C(v, k, t)$. The second column (LB) gives the lower bound for the corresponding $C(v, k, t)$. The third column

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

Table 2: Tests for $t = 3$

(UB) gives the best known upper bound of $C(v, k, t)$ in Spring 2005. The fourth column (b) is the size of the covering design we tested for $C(v, k, t)$. Finally, the fifth column ($Cost$) gives the number of t-subsets not covered by our best solution. 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 $C(v, k, t)$ in the corresponding row. As can be seen from these tables, we have improved the upper bound of 29 instances.

Tables 2 and 3 report results for tests performed on cases where $t = 3$ and $t = 4$. Except for instance $C(17, 8, 4)$, all tests reported in Tables 2 and 3 have been limited to 600 cycles. All tests completed well inside the limit of 168 hours. New upper bounds have been found for five problem instances. For many cases, the non zero cost displayed in Tables 2 and 3 have been obtained very soon in the computation. We suspect the upper bound is optimal for some of these cases. At the opposite, the cases for which we have found a new upper bound were among a small set of cases where convergence is relatively slow. Instance $C(17, 8, 4)$ was given more time because convergence was particularly slow for this case. The new upper bound has been found after 1177 cycles.

Test results reported in Table 4 have completed 1000 cycles except for

Instances	*LB*	*UB*	*b*	*Cost*	*Instances*	*LB*	*UB*	*b*	*Cost*
C(12,6,4)	40	41	40	6	C(17,10,4)	19	23	22	4
C(12,7,4)	20	24	23	1	C(17,11,4)	13	16	15	0
C(13,6,4)	59	66	65	2	C(18,7,4)	111	130	129	5
C(13,7,4)	28	30	29	3	C(18,8,4)	57	66	65	2
C(14,6,4)	75	80	79	8	C(18,9,4)	34	38	37	6
C(14,7,4)	40	44	43	2	C(18,10,4)	24	26	25	8
C(14,8,4)	23	24	23	9	C(18,11,4)	18	19	18	18
C(15,6,4)	93	117	116	3	C(19,7,4)	131	153	152	104
C(15,7,4)	52	57	56	1	C(19,8,4)	74	84	83	3
C(15,8,4)	29	30	29	11	C(19,9,4)	45	48	47	3
C(15,9,4)	19	20	19	9	C(19,10,4)	27	32	31	13
C(15,10,4)	12	14	13	6	C(19,11,4)	19	23	22	10
C(16,6,4)	144	152	151	54	C(19,12,4)	15	17	16	22
C(16,7,4)	69	76	75	1	C(20,8,4)	83	93	92	2
C(16,9,4)	24	26	25	6	C(20,9,4)	54	64	63	0
C(16,10,4)	16	18	17	3	C(20,10,4)	30	36	35	18
C(17,7,4)	85	99	98	6	C(20,11,4)	24	28	27	14
C(17,8,4)	49	54	53	0	C(20,12,4)	15	20	19	40
C(17,9,4)	27	28	27	9	C(20,13,4)	14	16	15	9

Table 3: Tests for $t = 4$

$C(19, 13, 5)$ and for $v = 20$ and $k > 11$. New upper bounds have been found for 11 cases. We observe that several new upper bounds have been found for large cases such as $v = 17$ and $v = 19$. The algorithm fails for smaller (and likely easier) cases such as $v = 14$ and $v = 15$, even though the gap between bounds is important for many of these small cases. Two possible explanations come in mind. Problem instances for $v = 14$ and $v = 15$ are small enough to be in the same category as some problems in Tables 2 and 3. Existing methods which are efficient to compute upper bounds for problems in Tables 2 and 3 work as well for instances with $v = 14$ and $v = 15$ in Table 4. A second explanation is that we haven't yet found proper settings of the search parameters for these instances. The space of search parameter settings is huge and has only be partially explored. Furthermore, results displayed in Table 4 for cases where $v = 14$ and $v = 15$ have been obtained late in the computation. Given that convergence is slow, it is likely that different search parameter settings and possibly more time will lead to improve these upper bounds as well.

Instances	*LB*	*UB*	*b*	*Cost*	*Instances*	*LB*	*UB*	*b*	*Cost*
C(11,6,5)	96	100	99	1	C(17,9,5)	58	80	79	0
C(11,7,5)	33	34	33	3	C(17,10,5)	41	49	48	0
C(12,7,5)	55	59	58	2	C(17,11,5)	25	32	30	0
C(13,7,5)	75	78	77	34	C(18,9,5)	98	113	112	2
C(13,8,5)	33	43	42	0	C(18,10,5)	49	54	53	12
C(14,7,5)	118	138	137	1	C(18,11,5)	32	42	41	6
C(14,8,5)	49	55	54	5	C(18,12,5)	20	24	23	123
C(14,9,5)	28	32	31	4	C(19,10,5)	65	86	83	0
C(15,7,5)	161	189	188	1	C(19,11,5)	42	50	49	0
C(15,8,5)	75	89	88	2	C(19,12,5)	29	38	37	0
C(15,9,5)	39	42	41	15	C(19,13,5)	18	21	20	139
C(15,10,5)	24	27	26	1	C(20,10,5)	90	106	99	0
C(16,8,5)	104	117	116	12	C(20,11,5)	50	65	64	7
C(16,9,5)	52	61	60	0	C(20,12,5)	32	42	41	16
C(16,10,5)	31	37	35	0	C(20,13,5)	24	33	32	4
C(16,11,5)	18	22	21	22	C(20,14,5)	16	18	17	248
C(17,8,5)	147	188	185	0					

Table 4: Tests for $t = 5$

Several results reported in Table 5 are based on tests completed after exhausting 168 hours of wall clock time. These didn't complete 1000 cycles. New upper bound have been found for 13 problem instances. Most of the new upper bounds have been found for values of v between 13 and 16. Likely, the setting of the search parameters is poorly adapted for the larger problem instances in Table 5. Obviously, more time is required for the computation.

Table 6 reports computational times for the 29 instances where we improved the best known upper bound. In this table, the second column displays the best known upper bound previously to our improvements. The third column reports the value of the best new upper bound we have found. The fourth column displays the number of tabu search iterations executed before finding the upper bound reported in the previous column. Finally, the last column reports the CPU time in seconds for each test. Most new upper bounds have been found quite soon in the computation, the longest time been around 147 CPU hours for the case $C(17, 8, 4)$. For many cases, particularly larger problem instances, we have used our leverage in setting search parameters to stress rapid convergence. For such cases, search parameters 4, 9, 10 and 11 have been set to the highest value in the ranges of

Instances	*LB*	*UB*	*b*	*Cost*	*Instances*	*LB*	*UB*	*b*	*Cost*
C(12,7,6)	165	176	175	2	C(17,10,6)	89	119	118	4
C(12,8,6)	50	51	50	6	C(17,11,6)	48	61	60	13
C(13,8,6)	90	100	99	0	C(17,12,6)	26	36	35	31
C(13,9,6)	38	40	39	0	C(18,11,6)	68	89	88	2
C(14,8,6)	132	151	150	2	C(18,12,6)	38	48	47	55
C(14,9,6)	52	74	70	0	C(18,13,6)	24	28	27	134
C(14,10,6)	27	29	28	32	C(19,11,6)	85	102	101	12
C(15,9,6)	82	100	99	0	C(19,12,6)	51	76	75	8
C(15,10,6)	42	54	52	0	C(19,13,6)	30	42	41	103
C(16,9,6)	134	172	170	0	C(19,14,6)	21	22	21	362
C(16,10,6)	63	77	76	4	C(20,12,6)	70	93	92	0
C(16,11,6)	35	44	43	11	C(20,13,6)	45	66	65	90
C(17,9,6)	197	268	267	172	C(20,14,6)	26	32	31	458
C(13,8,7)	269	297	296	0	C(17,12,7)	50	64	63	4
C(13,9,7)	73	79	78	6	C(17,13,7)	25	26	25	296
C(14,9,7)	140	166	165	7	C(18,12,7)	72	101	100	15
C(14,10,7)	54	57	56	0	C(18,13,7)	36	50	49	168
C(15,10,7)	78	118	117	0	C(19,13,7)	56	84	83	70
C(15,11,7)	37	42	41	49	C(19,14,7)	33	42	41	229
C(16,10,7)	132	167	165	0	C(20,13,7)	79	136	135	210
C(16,11,7)	62	88	85	0	C(20,14,7)	43	60	59	595
C(17,11,7)	98	127	126	35	C(20,15,7)	28	34	33	531
C(14,10,8)	103	119	117	0	C(17,13,8)	37	42	41	396
C(15,11,8)	74	80	79	5	C(18,13,8)	70	103	102	48
C(16,11,8)	114	191	190	5	C(18,14,8)	33	34	33	839
C(16,12,8)	50	59	58	39	C(19,14,8)	49	75	74	246
C(17,12,8)	88	138	137	18	C(20,15,8)	44	57	56	746

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

Instances	*Previous Upper Bound*	*Improved Upper Bound*	*# of Iterations*	*CPU Time*
C(13,8,1)	297	296	675823	48331
C(13,8,5)	43	42	1006263	16780
C(13,8,6)	100	99	18833	1639
C(13,9,6)	40	39	3962341	199081
C(14,9,6)	74	70	920688	20621
C(14,10,7)	57	56	12150	891
C(14,10,8)	119	117	1977944	144094
C(15,9,6)	100	99	260927	70032
C(15,10,6)	54	52	266206	74306
C(15,10,7)	118	117	154952	41464
C(16,9,5)	61	60	148508	20486
C(16,9,6)	172	170	362941	177913
C(16,10,5)	37	35	336183	48867
C(16,10,7)	167	165	303568	36961
C(16,11,7)	88	85	185564	24160
C(17,8,4)	54	53	41396983	529789
C(17,8,5)	188	185	300071	82930
C(17,9,5)	80	79	2680690	138870
C(17,10,5)	49	48	899131	278101
C(17,11,4)	16	15	712030	7617
C(17,11,5)	32	30	415428	56489
C(19,6,3)	63	62	301834	8853
C(19,10,5)	86	83	714778	84262
C(19,11,5)	50	49	1813365	191949
C(19,12,5)	38	37	4164742	490706
C(20,6,3)	72	71	15469025	351503
C(20,9,4)	64	63	6815489	445288
C(20,10,5)	106	99	2379806	353505
C(20,12,6)	93	92	250548	330744

Table 6: Computational times on successful problem instances

Instances	b	*First run*		*Second run*		*Third run*	
		Iter	*Time*	*Iter*	*Time*	*Iter*	*Time*
3-(12,5,1)	29	19226	13	9271	6	7903	5
4-(13,6,1)	66	58003	212	33264	121	13725	45
4-(17,11,1)	16	171268	2024	386865	4431	15050	209
5-(13,8,1)	43	402150	3004	957527	7084	1251202	9185
5-(19,11,1)	50	175252	18732	89050	10144	18187	2469
5-(19,12,1)	37	4164742	490706	1377491	154184	1385207	166421
6-(13,9,1)	40	592921	6106	157792	1651	1803051	18480
7-(14,10,1)	55	493573	12661	906425	23060	597453	15408

Table 7: Comparison of different runs with same parameter setting

Table 1. When the exploration factor is high, search cycles initiate uphill search steps immediately when cost improving search steps cannot be made. Multilevel cooperative search explores rather superficially each visited region of the solution space. If new upper bounds can be found in this way, they are usually found soon during the computation.

Several steps in our algorithm are based on random decisions. They include the initial coarsening, initial solutions to tabu search, the length of tabu lists and the random blocks placed in sets A_l and R. In Table 7, we report the number of cycles and the CPU time of three independent runs performed on 8 different problem instances. Between problem instances (i.e., problem instances from different rows in Table 7), we use a different search parameter setting. However, inside each problem instance (problem instance on the same row), the runs are performed with the same search parameter setting. Results in Table 7 indicate that random choices do not compromise the results reported in Table 6. Each run is successful in finding the upper bound displayed in the second column of Table 7. Though there are important computational time variations between runs, they all find a covering well inside 300 cycles. The results reported in Table 7 are in agreement with our other tests. Upper bounds reported in Table 6, given a specific setting of the search parameters, can generally be reproduced within a small multiplicative factor of the computational time in Table 6.

We do not compare directly in this paper the performance of our algorithm with other algorithms. The problem instances in the La Jolla Tables have been extensively tested with different methods. However, we will like to emphasize the performance of tabu search through a few comparison tests.

Instances	*Multilevel*		*Simulated Annealing*		*Tabu Search*	
	CPU Time	*Iterations*	*Iterations*	*Cost*	*Iterations*	*Cost*
5-(13,8,1)	839	50313	330061196	4	39824	5
5-(16,9,1)	20486	148508	4820470893	2	185693	7
5-(16,10,1)	48867	366183	5727645000	11	315167	4
5-(17,11,1)	56489	415428	3178988604	27	225053	24
6-(13,8,1)	1639	18833	1032767980	14	53867	11
6-(14,9,1)	20621	920688	6007147920	12	215842	8
6-(16,9,1)	177913	362941	46526949504	185	726651	11
7-(14,10,1)	891	12150	205696966	10	13098	9
7-(16,11,1)	24160	185564	2184703573	32	39775	0

Table 8: Comparison between different search heuristics

In Table 8, we compare our multilevel cooperative search algorithm with a simulated annealing algorithm and the tabu search procedure used inside our multilevel cooperative search algorithm. The results regarding simulated algorithm have been obtained using the simulated annealing code for covering design available in the paper [24]. Table 8 compares the three methods on problem instances where the multilevel cooperative search algorithm was able to improve the best known upper bound. Tabu search and simulated annealing procedures are given the same amount of CPU time as used by the multilevel cooperative search algorithm to identify the new upper bounds. We report the number of iterations performed by each procedure as well as the number of t-subsets that have not been covered. Note that unlike tabu search, one iteration of simulated annealing evaluates only one neighbor. This explains why the number of iterations is far larger for simulated annealing in comparison with the two other search procedures. Results in Table 8 show that the tabu search procedure is very competitive. The performances of tabu search have been achieved with little effort in the design of the tabu search algorithm. Given the relatively simple design space of our tabu search algorithm, the results are impressive, which advocates favorably for using tabu search for combinatorial design problems.

5.3 Summary & future work

We have survey the cost of all the elite solutions found by the multilevel cooperative search heuristic for each tested problem instance in Tables 2 to

5. For several problem instances, an elite solution with the cost displayed in Tables 2 to 5 has been found many times. For other problem instances, mainly those where $Cost$ is closed to zero, very few elite solutions have been found having the cost displayed in Tables 2 to 5. The second situation indicates a lack of intensification of the exploration in the neighborhood of these solutions. The operator **restart search** (section 4.4.3), where elite solutions from previous cycles are placed in the set R, addresses in part this problem. We can also set the search parameters such to keep the same elite solutions in A_l for several cycles, but convergence will be substantially slower. A more general solution to this problem will be to adapt the setting of some search parameters to the conditions of the search. For example, the setting of search parameters such as r_l (parameter 4 in Table 1), exploration factor (parameter 9), the size of R (parameter 10) and R_r (parameter 11) should be made dependent on the variations in the cost of the overall best elite solution. Other more specific changes in the control of the heuristic will also be made to ensure greater responsiveness from re-coarsening to all good solutions.

Results from the larger problem instances in Table 5 reveal that the current sequential implementation should be revised to address large problem instances. We have considered three approaches to speed-up computation. One is the parallelization of the neighborhood evaluation. This parallelization has been realized, details can be found in [11]. Preliminary results in Table 9 show that speed-up is closed to linear for an appropriate number of processors with respect to the size of each problem instance. The second

Instances	*# of processors*	*# of Iterations*	*Clock Time*	*Speed Up*
4-(15,6,1)	1	124767	2157	1
	4	131120	1366	1.660
	8	127606	1347	1.638
	16	130320	2258	0.998
5-(16,7,1)	1	246992	16769	1
	4	248590	5101	3.309
	8	254210	3519	4.905
	16	262441	5144	3.464
6-(17,9,1)	1	333266	86909	1
	4	327796	22397	3.817
	8	336163	12878	6.807
	16	337759	14959	5.888

Table 9: Measures of speed-ups for computing neighborhoods

approach to speed-up computation consists to have the concurrent optimization of problem instances P_0 to P_l performed in parallel by running each tabu search procedure on a different processor. This is a well known parallelization strategy for cooperative search algorithms [9]. We should note that shared data structures will be required to implement the re-coarsening operator. Lock accesses to this shared data structure will have a small negative impact on speed-up. Finally, to speed-up computation for problem instances where $v > 20$, we consider performing, at the lowest levels, a partial evaluation of the neighborhood based on a random selection of neighbors. This technique is well known and is often used to evaluate neighborhoods with a large set of neighbors or with neighbors that are costly to compute.

We noticed in Tables 2 to 5 the cost of some problem instances is surprisingly high considering the relatively large gap between the lower bound and the upper bound. This is the case for problem instances $C(16, 6, 4)$, $C(19, 7, 4)$ and $C(18, 12, 6)$ among others. This happens when the search heuristic cannot collect "good" elite solutions from problem instances at the lower levels. We then have a poor re-coarsening of the problem instances at higher levels. We expect this situation to be more frequent for $t - (v, k, \lambda)$ covering designs where v is large or when the upper bound is large (i.e., b is large). The situation with large b can be addressed by coarsening the vector b as well. This will be a new coarsening procedure where coarsened problem instances have decision variables fixed in non-zero states. This will help to control the cost of evaluating neighborhoods due to large variations in the size of b. For large v, it is almost inevitable that any search heuristic will be challenged while dealing with the combined difficulty of a large search space and poor guidance from the cost function. There are a few ways to partially get around this problem. Search cycles at the lowest levels can be eliminated. The gathering of information for re-coarsening operators can be based rather on multiple interpolations between level l and the lowest levels. Another way, similar to [25], has search cycles at the lowest levels implemented as a multilevel search. For example, search cycle j at level 0 can use sets A_0 to A_l to perform a multilevel search where the interpolated solution at level $l - 1$ is obtained from elite solution E_0^{j-1}.

6 Conclusion

We have introduced a new 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. This heuristic have been run on 158 covering design problem instances from the La Jolla Tables [10] for which $C_1(v,k,t)$ was unknown in Spring 2005. The algorithm was given a maximum of one week of computation on standard desktop computers to find new upper bounds for each tested problem instance. Improved upper bounds have been found for 29 problem instances, many of which exhibit a tight gap with the lower bound.

The proposed heuristic is an application of the multilevel cooperative search approach to the problem of finding covering designs. The standard re-coarsening and interpolation operators of multilevel cooperative search have been adapted for a better optimization by local search of the cost function. The numerical results indicate the proposed approach is competitive with all the other methods existing for computing upper bounds on $C_\lambda(v,k,t)$. We have also suggested a few trivial changes in the current design of the heuristic to speed-up computation and to tackle larger problem instances.

Key features of this heuristic for covering designs, such as problem dimensionality reduction through recursive projections, and refinement of the projection operators through re-coarsening and interpolations, can be the foundation of similar search heuristics for other combinatorial design problems. If such attempts are made, it will be interesting to investigate the relation between dimensionality reduction through recursive projections and isomorphic solutions in the solution space of combinatorial design problems. Dimensionality reduction has first been exploited in multilevel search methods to speed-up computation of search heuristics in large problem instances. Unfortunately, a great deal of cost related information is lost during projections, trading optimization for speed-up. Permutation groups and other equivalent solutions under the cost function might help to minimize this lost of information. It would seem that less cost related information will be lost during projection where this information is repeated several times in the solution space of the original problem and where information destruction prevails among repeated states. If this is the case, it might be possible to derive projection operators for multilevel search methods and multilevel cooperative search methods that will address efficiently the challenges pose to local search methods by combinatorial 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: Partice & Experience*, 6(2):111–117, 1994.

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

[3] A. Brandt. Multilevel computations: Review and Recent Developments. In S.F. McCormick, editor, *Multigrid Methods: Theory, Applications, and Supercomputing, Proc. of 3rd Copper Mountain Conf. Multigrid Methods*, volume 110 of *Lecture Notes in Pure and Applied Mathematics*, pages 35–62. Marcel Dekker.

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

[5] T. Bui and C. Jones. A Heuristic for Reducing Fill in Sparse Matrix Factorization. In Linda R. Petzold Richard F. Sincovec, David E. Keyes, Michael R. Leuze and Daniel A. Reed, editors, *Proc. of the 6th SIAM Conf. on Parallel Processing for Scientific Computing*, pages 445–452. SIAM Press, 1993.

[6] 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.

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

[8] T.G. Crainic, Y. Li, and M. Toulouse. A Simple Cooperative Multilevel Algorithm for the Capacitated Multicommodity Network Design. *Computer & Operations Research*, Accepted for publication.

[9] T.G. Crainic, M. Toulouse, and M. Gendreau. Towards a Taxonomy of Parallel Tabu Search Algorithms. *INFORMS Journal of Computing*, 9(1):61–72, 1997.

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

[11] C. Dai. A multilevel cooperative parallel tabu search algorithm for the covering design problem. Master's thesis, University of Manitoba, 2006.

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

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

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

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

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

[17] D. Gordon. http://www.ccrwest.org/cover.html.

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

[19] B. Hendrickson and R. Leland. The Chaco User's Guide: Version 2.0. Report SAND95-2344, Sandia National Laboratories, 1995.

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

[21] 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.

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

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

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

[25] 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.

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

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

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

[29] 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, Aug. 1999.

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