A Cooperative Multilevel Tabu Search Algorithm for the Covering Design Problem

Chaoying Dai, (Ben) Pak Ching Li, and Michel Toulouse

Department of Computer Science, University of Manitoba
{chaoying, lipakc, toulouse}@cs.umanitoba.ca

Abstract. This work describes an adaptation of multilevel search to the covering design problem. The search engine is a tabu search algorithm which explores several levels of overlapping search spaces of a $t-(v,k,\lambda)$ covering design problem. Tabu search finds "good" approximations of covering designs in each search space. Blocks from those approximate solutions are transferred to other levels, redefining the corresponding search spaces. The dynamics of cooperation among levels tends to regroup good approximate solutions into small search spaces. Tabu search has been quite effective at finding re-combinations of blocks in small search spaces which provide successful search directions in larger search spaces.

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

1 Introduction

A $t-(v,k,\lambda)$ *covering design* is a pair (X,B), where X is a set of size v, called *points* and B is a collection of k-subsets of X, called *blocks*, such that every t-subset of X is contained in at least λ blocks of B. 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 [12]. 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 [5].

The value $C_\lambda(v,k,t)$ can be determined using an exact search algorithm. Unfortunately, such algorithms are ineffective for all but a few set of parameters, due to the effects of combinatorial explosion. Therefore, search heuristics may be a viable option for improving upper bounds on $C_\lambda(v,k,t)$.

In this paper, we introduce a cooperative multilevel search heuristic method to improve upper bounds on $C_\lambda(v,k,t)$. Assume we are looking for a $t-(v,k,\lambda)$ covering design of b blocks. Let $\binom{X}{k}$ be the set of all k-subsets in X and let $\mathcal{S}_0 = \{S \subset \binom{X}{k} || |S| = b\}$ be the solution space for $t-(v,k,\lambda)$. Assume $S_1, S_2, \ldots, S_l$ are subsets of $\binom{X}{k}$ such that $S_l \subset S_{l-1} \subset \cdots \subset S_1 \subset S_0 = \binom{X}{k}$. Each subset S_i defines a search space $\mathcal{S}_i$ on $t-(v,k,\lambda)$ in the same way as $\mathcal{S}_0$. A tabu search algorithm explores independently each search space to seek sets of b blocks that

E. Talbi et al. (Eds.): EA 2005, LNCS 3871, pp. 119–130, 2006.

cover as many t-subsets as possible. The cooperative multilevel search strategy consists of substituting some blocks of the smallest search spaces by sets of good blocks discovered by tabu search. This exchange of blocks eventually brings combinations of good blocks in small search spaces where tabu search is quite effective finding covering for large number of t-subsets. We have tested our algorithm on covering design problems with tight gaps between lower and upper bounds [8, 9]. Those are the most difficult upper bounds to improve. We were able to find known upper bounds for all the problem instances tested and found new upper bounds for several of them.

The subsequent sections of this paper are organized as follows. Section 2 provides background information on the covering design problem. In section 3, we describe the multilevel paradigm. Section 4 summarizes the implementation of our cooperative multilevel algorithm. Section 5 reports experimental results and we conclude in Section 6.

2 Background

In this section we provide a short background on covering designs and search heuristics for covering designs.

2.1 Covering Designs

The study of covering designs began around the end of the 1930's. Turán (see [5]) 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 ($L_\lambda(v, k, t)$) [19] 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 $C_\lambda(v, k, t)$ [13].

In 1963, Erdős and Hanani [7] conjectured that for fixed values of t and k, where $t < k$.

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

This result was shown to be true in 1985 by Rödl [18], 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 [9]. 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 [14]. 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 $C_1(v, k, t)$. In 2003, Margot [11] used integer programming techniques, branch-and-cut and isomorphism rejection to design an algorithm for computing $C_1(v, k, t)$. However, such algorithms are effective for only a few set of parameters.

2.2 Search Heuristics for Covering Designs

Search heuristic methods are used to search for a $t-(v, k, \lambda)$ covering design which is smaller than the best known upper bound for $C_\lambda(v, k, t)$. These methods have worked well for small values of v, k, λ [15, 16]. For $\lambda = 1$, the covering design problem can be modeled as a combinatorial optimization problem in $\binom{v}{k}$ Boolean decision variables, one for each k-subset. A feasible solution is a Boolean vector where at most b variables are set to 1, where b is the size of the covering design we are looking for. The cost function optimized by the search heuristic is the number of t-subsets not covered at least one time by the current solution (a set B of b blocks). More precisely, let $\binom{X}{t}$ be the set of t-subsets and let $cover_y$ be the number of times the t-subset $y \in \binom{X}{t}$ is covered by the b blocks in B. Let $notcover_y = \max\{0, \lambda - cover_y\}$ denote the number of times the b blocks in B fails to cover the t-subset y. The cost of solution B is given by

$$cost(B) = \sum_{y \in \binom{X}{t}} notcover_y.$$

When $cost(B) = 0$, then all t-subsets are covered at least λ times, meaning we have discovered a $t-(v, k, \lambda)$ covering design with b blocks.

A natural mapping function to define neighborhoods for covering design problems consists of choosing m points among the k points of a block and replace these by m other points from the $v-k$ points not belonging to this block. Such a move can replace $1 \leq m \leq \min(k, v-k)$ points belonging to a same block. The neighborhood $\mathcal{N}(B)$ of solution B is a subset of $\mathcal{S}_0$ such that $I \in \mathcal{N}(B)$ if I has $b-1$ identical blocks with B and one block which differs by exactly m points. The size of the neighborhood $\mathcal{N}(B)$ is given by $|\mathcal{N}(B)| = b \times \binom{k}{m} \times \binom{v-k}{m}$. Since the size of neighborhoods based on swapping points increases rapidly in terms of m, typical move based heuristics for covering designs are based on neighborhood where $m = 1$.

3 The Multilevel Paradigm

Multilevel approaches have first been proposed in the field of numerical approximation [3]. Based on the original problem domain discretization, coarser discretizations (levels) are recursively constructed by increasing the grid spacing in comparison with the latest generated grid. In *nested iteration*, the simplest multilevel scheme [4], starting with the coarser grid, an approximation is computed and then interpolated on the next grid, which is less coarsened. The approximation is then refined using an iterative solver. The latest refined approximation is used as initial point of the relaxation in the original problem domain discretization. The nested iteration scheme helps improve convergence in the original domain discretization by providing a good initial point to the relaxation method. In the *V-cycle scheme*, a first approximation is computed on the original grid spacing. The residual error associated to the approximation is *projected* on the next coarser grid where the system of linear equations is solved for this residual error. One V-cycle consists of projections upward from less coarsened grids toward coarser grids. Then interpolations from coarser grids toward less coarsened grids refine the approximation. Projections change the problem definition by solving for a new residual error at each level. They also help to improve convergence of iterative solvers by focusing on the oscillatory component of the error function at each level.

The multilevel approach has been adapted recently to combinatorial optimization problems in combination with search algorithms. The basic framework of multilevel search is the following: Let $\mathcal{A}$ denote a given combinatorial optimization problem and A_0 a problem instance of $\mathcal{A}$. During the **coarsening phase**, a succession $A_1, \ldots, A_l$ of increasingly smaller problem instances of $\mathcal{A}$ is generated by reducing the number of decision variables in comparison with the definition of problem instance A_0. During the **initial search phase**, a feasible solution s_l is computed for the smallest problem instance A_l. During the **refinement phase**, the feasible solution s_l is used to *interpolate* values for the decision variables of problem instance A_{l-1}. This setting of decision variables in A_{l-1} is used as initial solution for a search algorithm which explores the search space of A_{l-1}. The optimization of the cost function for A_{l-1} using a search algorithm also improves (refines) the feasible solution s_l obtained from the problem instance A_l. The refinement phase consists of interpolating and refining feasible solutions until the values of the decision variables of s_0 can be interpolated from a feasible solution of problem instance A_1. This last interpolation provides for an initial solution for a search algorithm to optimize the cost function problem instance A_0.

The coarsening phase is critical to multilevel algorithms. It reduces the size of the problem instance and, more importantly, it determines which regions of the solution space will be explored during the initial search and refinement phases. Coarsening strategies were first proposed in the context of applications of the multilevel paradigm to the graph partitioning problem [1, 10]. These strategies are based on clustering decision variables. Starting from the original graph instance $G_0 = (V_0, E_0)$, pairs of adjacent vertices x, y are selected

randomly and merged together to become a single vertex xy in the coarsened graph $G_1 = (V_1, E_1)$. Edge $\{x, y\} \in G_0$ is removed, edges $\{u, x\}$ or $\{u, y\}$ in G_0 are replaced by edge $\{u, xy\}$ in G_1. The coarsening of G_0 yields a graph G_1 where $|V_1| \approx |\frac{V_0}{2}|$ and $E_1 \subset E_0$.

Let $GPP(G_x)$ be the graph partitioning problem for the graph instance G_x and $\mathcal{S}_x$ the solution space of $GPP(G_x)$. The number of decision variables in $GPP(G_1)$ is about half of $GPP(G_0)$. Nonetheless, a feasible solution of $GPP(G_1)$ can be interpolated in the solution space of $GPP(G_0)$ by expanding vertex $xy \in V_1$ into vertices $x, y \in V_0$ and placing x, y in the same partition as xy. Consequently, $\mathcal{S}_1 \subset \mathcal{S}_0$, i.e., any feasible solution to $GPP(G_1)$ is also a feasible solution to $GPP(G_0)$. The coarsening phase recursively applies the above coarsening strategy to the latest coarsened graph and outputs a succession of increasingly coarsened graph $G_1, G_2, \cdots, G_l$ which satisfy the condition $\mathcal{S}_l \subset \mathcal{S}_{l-1} \subset \cdots \subset \mathcal{S}_1 \subset \mathcal{S}_0$.

For several combinatorial optimization problems, coarsening by clustering decision variables is hardly applicable. In [6], the authors proposed a coarsening strategy by fixing the state of some decision variables. A decision variable is fixed if its value cannot be changed by the solution process. Let $x_1, x_2, \ldots, x_n$ be the set of decision variables of a problem instance A_0. By fixing some decision variables of A_0, a new problem instance A_1 is defined where $\mathcal{S}_1 \subset \mathcal{S}_0$. Any solution to A_1 can be trivially interpolated in the solution space of A_0. Fixing recursively the state of some decision variables has the effect of coarsening the original problem instance A_0 into problem instances with fewer decision variables. Furthermore, the strict inclusion condition $\mathcal{S}_l \subset \mathcal{S}_{l-1} \subset \cdots \subset \mathcal{S}_1 \subset \mathcal{S}_0$ is also satisfied.

In most multilevel algorithms applied to combinatorial optimization problems, the refinement phase is reminiscent of the nested iteration scheme in multigrid approximation. Recently, a multi-cycle refinement phase has been proposed [6, 17] in the context of parallel cooperative search algorithms. In multi-cycle refinement, projection operators transformed the problem instance searched at each level by changing its coarsening. Modification to the coarsening of levels define new regions of the solution space that can be explored by search heuristics. This allows for a new sequence of interpolations and searches, closing one cycle. There are several possible variations in the multi-cycle refinement phase, we propose a new one in this paper.

4 Multilevel Tabu Search Algorithm for the Covering Design Problem

In this section, we introduce the design of our multilevel algorithm for the covering design problem. We describe our strategy to coarsen covering design problem instances as well as the projection operator and re-coarsening strategies applied to transform subsequently the initial coarsening. Next, we describe the tabu search algorithm which is used to explore the search space defined by each level of coarsening. Finally, we describe a variation of the multi-cycle refinement phase adapted for the covering design problem.

4.1 The Coarsening Phase

Our coarsening procedure defines search spaces (levels) by fixing recursively subsets of decision variables. The decision variables for the covering design problem correspond to the $\binom{v}{k}$ blocks of a given problem instance. Each block is assigned exclusively a level during the coarsening phase. An integer array (*multilevel*) of dimension $\binom{v}{k}$ expresses this assignment. Entry j of the array takes a value i in the range 0 to l to indicate that the block j is assigned to level i. At the initialization, all the $\binom{v}{k}$ blocks are assigned to level 0. Then, through random selection, blocks assigned to level 0 are re-assigned to level 1. This procedure is repeated for each level i, re-assigning randomly blocks from level $i-1$ to level i.

The number of blocks assigns to each level is decided by the *coarsening factor* cf. The value of this coarsening factor is a function of the total number of blocks $\binom{v}{k}$, the number of levels $l+1$ and $|L_l|$ the number of blocks required at top level l. It is computed as follows:

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

The value cf expresses the difference between the number of blocks assigned to two adjacent levels i and $i+1$. Assume L_i represents the set of blocks assigned to level i. The number of blocks $|L_i|$ that must be assigned to level i is given by the following formula:

$$|L_i| = |L_l| + (l-i) \times cf$$

Therefore, the number of blocks $|L_{l-1}|$ at level $l-1$ is $|L_{l-1}| = |L_l| + cf$, the number of blocks at level $l-2$ is $|L_{l-2}| = |L_l| + 2 \times cf$, etc. The number of blocks at level 0 is $|L_0| = |L_l| + (l \times cf)$. The sum of blocks assigned to all the levels must be $\sum_{i=0}^{l} |L_i| = \binom{v}{k}$.

A block is considered to be fixed for level i if it is assigned to a level lower than i. Therefore, the number of decision variables at level i is $S_i = \sum_{k=i}^{l} |L_k|$, the set of blocks assigned to levels greater or equal to i. The search space at level i is constituted by all the possible combinations of b blocks in the set $\mathcal{S}_i = L_i \cup L_{i+1} \cup \cdots \cup L_l$. The search space $\mathcal{S}_0$ at level 0 corresponds to all combinations of b blocks in the set $\binom{X}{k} = \cup_{i=0}^{l} L_i$. Note that the set of decision variables $S_i = L_i \cup L_{i+1} \cup \cdots \cup L_l$ of level i is a strict subset of $S_{i-1} = L_{i-1} \cup L_i \cup \cdots \cup L_l$ of level $i-1$. Consequently, the strict inclusion condition $\mathcal{S}_l \subset \mathcal{S}_{l-1} \subset \cdots \subset \mathcal{S}_1 \subset \mathcal{S}_0$ among search spaces is satisfied by this coarsening procedure.

4.2 Projection and Re-coarsening

The projection operator copies from level i to level l the blocks B of the best covering design approximation at level i. This operator is implemented by re-assigning blocks in B to level l, as shown in the **while** loop of Fig. 1.

According to our coarsening procedure, each block in B is assigned to a level greater or equal to i. Line 1 obtains the current level assignment of block s from

projection(B)
  while ($B \neq \emptyset$) **do**
1. $s = s \in B$; $B = B \setminus s$; $j = multilevel[s]$;
    if ($j \neq l$) **then**
2. $multilevel[s] = l$;
3. $u =$ randomly select a free block assigned to level l; $multilevel[u] = j$;

Fig. 1. The projection procedure

the array *multilevel* (the array which stores the assignment of each block to a specific level). Line 2 re-assigns to level l those blocks of B not already assigned to level l.

Each time a block s is re-assigned by the projection operator from level j to level l, it removes one block from level j and adds one block to level l. Given the way levels are defined in our coarsening procedure, assigning a new block s to level l is equivalent to adding the decision variable s to sets $S_{j+1}, S_{j+2}, \ldots, S_l$ such that $S_{j+1} = S_{j+1} \cup s$, $S_{j+2} = S_{j+2} \cup s$, $\ldots$, $S_l = S_l \cup s$. The operation of line 2 is in fact a re-coarsening of levels $j+1$ to l, modifying the search space of all these levels. In order to keep the number of blocks constant at each level, line 3 re-assigns a block from level l to level j. Line 3 changes the coarsening of levels $j+1$ to l: $S_{j+1} = S_{j+1} \setminus u, S_{j+2} = S_{j+2} \setminus u, \ldots, S_l = S_l \setminus u$.

Re-coarsening is designed to re-focus the search space of each level toward better regions of the solution space. To achieve this purpose, the re-coarsening of each level is biased by the cost function through the projection of the best solutions to level l. In order to have a chance to influence the multilevel search, a block entering the search space of level i through projection must not exit this level before performing a search of the corresponding level. To enforce this condition, the blocks of solutions that have been projected to level l must stay assigned to level l for the duration of a search. The *free* blocks in line 3 of the projection procedure are blocks that do not belong to any of the best solutions recently projected to level l. In this manner, blocks that seem to contribute to find good solutions are kept at level l. Blocks at level l are re-combined together by the tabu search procedure or re-combined in the same manner with blocks from any of the other levels. On the other hand, blocks not belonging to any of the current best solutions are sent back to a lower level j through the last operation of line 3, they then become excluded from combining with blocks belonging to levels $j+1$ to l.

4.3 The Tabu Search Procedure

The search space of each level is explored using a tabu search procedure. This tabu search procedure uses two tabu lists. A first tabu list prohibits moves that undo swaps of blocks $x \rightarrow y$ by entering in the tabu list the move $y \rightarrow x$. A second tabu list disallows a block from leaving the current solution B for a certain number of tabu iterations after entering B. The size of the tabu list varies randomly in a pre-defined range for each call to the tabu procedure. We found that variations in the length of the tabu lists is helpful to diversify the

exploration of search spaces when projection fails to re-coarsen some of the levels. The termination criterion for this tabu search procedure is a pre-defined number of iterations without improving the best known solution. Our tabu search procedure is described in Fig. 2.

tabu_search(*initial_solution*)
  $best = initial\_solution$; $B = initial\_solution$;
  while (termination criterion not satisfied) **do**
    $B = V \in \mathcal{N}(B) \wedge V$ not tabu; (V is the best solution in the neighborhood of B and V is not in any of the two tabu lists)
    update tabu lists;
    if ($cost(B) \leq cost(best)$) **then**
      $best = B$;
    return *best*;

Fig. 2. The tabu search procedure

4.4 The Multi-cycle Refinement Phase

This multilevel algorithm is based on a multi-cycle refinement phase. Refinement cycles are divided into two categories: interpolation cycles and search cycles.

Interpolation Cycles. Interpolation cycles are initiated at level 0 as described in Fig. 3. An interpolation operation at level $i \neq l$ uses the best solution of level $i+1$ to restart the tabu search procedure at level i (line 1). At level l, the search is restarted from the current best solution at level l (line 3). An interpolation cycle ends by a restart of the search at level 0 using the current best solution at level l (line 4). Each time the tabu search procedure has completed the search initiated from the interpolated solution, the best solution in the search sequence is projected to level l (lines 2 and 4). In an interpolation cycle, information move downward through the interpolation operations and upward through the projection operations performed at levels 0 to $l-1$.

Interpolation_cycle()
    for ($i = 0; i \leq l-1; i++$) **do**
1. $best_i = tabu\_search(best_{i+1})$;
2. $projection(best_i)$;
    B_l = best solution of level l from the previous cycle;
3. $best_l = tabu\_search(B_l)$;
4. $best_0\_tmp = tabu\_search(best_l)$; $projection(best_0\_tmp)$;
    if ($cost(best_0\_tmp) \leq cost(best_0)$ **then** $best_0 = best_0\_tmp$;

Fig. 3. The interpolation cycle

Search Cycle. Search cycles run a tabu search procedure at each level, starting at level l toward level 0. Search cycles have a dual purpose. One is to discover improving solutions once re-coarsening has modified the search space of each level. The second purpose is to diversify the exploration of the solution space $\mathcal{S}_0$.

During a search cycle, the tabu search procedure at level i starts with the current best solution at this level. If the search fails to improve the current best solution, the exploration of the search space is then restricted to blocks in L_i, the blocks assigned to level i (line 3 in Fig. 4). In the search space defined uniquely by blocks of L_i, tabu search cannot access the blocks of the best solutions, which are assigned to level l. Constrained to blocks in L_i, the search usually enters a sequence of uphill moves where blocks enter B that would not have been included if all candidate neighbors had been considered. Then, search is re-opened to the whole search space of level i (line 5). The last search sequence at level i is initiated from the last solution visited in the restricted search space (line 6). This is usually a poor solution, consequently, the last search sequence is a sequence of downhill moves, replacing blocks in B with other blocks improving the cost of B. The solution that is projected at the end of the search at level i may or may not have a better cost than the best solution in the previous cycle. However, because of the uphill and downhill search moves, level i is likely to project to level l a more diversified set of blocks than if the search has been performed uniquely in the search space of level i. This speed-up the re-coarsening of each level, which in turn diversifies the exploration of the solution space $\mathcal{S}_0$.

Search_cycle()
  for $(i = l; i \geq 0; i--)$ **do**
    $B_i = best_i$; ($best_i$ is the current best solution at level i)
1. search space = any combination of b blocks in $L_i \cup L_{i+1} \cup \cdots \cup L_l$;
    $best_i = tabu\_search(best_i)$;
2. **if** $(cost(best_i) \geq cost(B_i))$ **then**
3.   search space = any combination of b blocks in L_i;
4. $best_i$ = last solution of $tabu\_search(best_i)$;
5.   search space = any combination of b blocks in $L_i \cup L_{i+1} \cup \cdots \cup L_l$;
6. $best_i = tabu\_search(best_i)$;
    if $(i \neq l)$ **then** $projection(best_i)$;

Fig. 4. Search procedure for refinement cycles

The Initial Search Phase. To compute the initial state of the multi-cycle refinement phase, we run a pre-defined number of p search cycles (the value of parameter is determined empirically). In the first search cycle, each tabu search procedure is started from a randomly generated solution. The random initial solution at level i is computed by selecting b blocks in the set of blocks $S_i = L_i \cup L_{i+1} \cup \cdots \cup L_l$, as described in Fig. 5. Searches in the first cycle from initial random solutions are likely to generate significant re-coarsenings at all levels above level 0. The **for** loop of line 3 launch a sequence of $p - 1$ search cycles in order to explore the new search spaces created by the re-coarsenings.

Refinement Phase. The entire multi-cycle refinement sequence is summarized in Fig. 6. Beyond the initial search phase, the refinement phase is decomposed into sequences of p cycles: first cycle is an interpolation cycle and it is followed by $p - 1$ cycles.

```
Initialization_sequence()
1.   for (i = l; i ≤ 0; i − −) do
       B_i = ∅;
       for (j = 1; j ≤ b; j + +) do
         block = a randomly selected block in S_i;
         B_i = B_i ∪ block;
       best_i = tabu_search(B_i);
2.     if (i ≠ l) then projection(best_i);
3.   for (j = 2; j ≤ p; j + +) do
       Search_cycle(j);
```

Fig. 5. The initial search phase

```
Multi-cycle_refinement_phase()
  Initialization_sequence();
  while (not found solution or number of cycles smaller than limit) do
    Interpolation_cycle();
    for (j = 1; j ≤ p − 1; j + +) do
      Search_cycle()
```

Fig. 6. Multi-cycle refinement phase

5 Experimentation

Several tests have been performed during the development and validation phases of this algorithm, we report the results in Table 1 below. The column "$t - (v, k, \lambda)$" describes the parameters of the covering design problem while the column "$\#\ of\ runs$" reports how many time we have run our algorithm on each problem. A large number of runs (such as 50 for 3-(14,5,1)) indicates that the corresponding problem has been used as a test problem during the development phase. The column "b" indicates the size of the covering design we have tested. All values of b are one block less than the best known upper bounds, except for some of the problems for which we have been able to improve the best known upper bounds. (Our tests are based on the best known covering design upper bounds as published on the web site [8] in Spring 2005). The columns "$Cost$" reports, for all runs, the solution with the smallest number of t-subsets not covered. For example, a cost of 2 indicates the best set of b blocks failed to cover 2 t-subsets. A cost of 0 indicates that we have improved the best known upper bound. In this case, on the corresponding row under columns b, we report the previous best known upper bound in () beside our new upper bound.

For runs where new upper bounds have been found, the total number of cycles executed varies between 7 and 175. For these runs, the range in computational time varies between 20 minutes to 15 hours on a 500 MHz sequential computer (many factors impact the computational time requirements of a cycle, among them the size of the covering design parameters). A run is aborted once 1000 cycles have been executed without discovering a new upper bound. For the tests reported in Table 1, the computational time requirements vary between 1 hour up to 168 hours (1 week) for runs that didn't improve the best known upper

Table 1. Experimental results

$t-(v,k,\lambda)$	*Cost*	*b*	*# of runs*	$t-(v,k,\lambda)$	*Cost*	*b*	*# of runs*
3-(12,5,1)	2	28	5	3-(13,5,1)	1	33	10
3-(14,5,1)	1	42	50	3-(15,5,1)	2	55	5
3-(16,5,1)	2	64	5	3-(17,6,1)	1	43	50
3-(19,6,1)	5	62	3	3-(20,6,1)	30	71	8
4-(13,6,1)	2	65	5	4-(14,6,1)	8	79	5
4-(15,6,1)	32	116	5	4-(14,7,1)	2	43	5
4-(15,7,1)	7	56	5	4-(16,7,1)	1	75	2
4-(17,7,1)	53	98	3	4-(17,8,1)	4	53	5
4-(16,9,1)	6	25	5	4-(17,10,1)	5	22	5
5-(11,6,1)	6	99	3	5-(12,7,1)	5	58	2
5-(13,8,1)	1	42	8	5-(14,7,1)	10	137	2
5-(14,8,1)	6	54	4	5-(15,8,1)	7	88	8
5-(16,9,1)	0	61(62)	6	5-(16,10,1)	0	36(37)	8
6-(13,8,1)	7	99	2	6-(14,9,1)	0	72(75)	8
6-(15,9,1)	1	99	3	6-(15,10,1)	0	53(55)	7
6-(16,10,1)	4	76	8	6-(16,11,1)	11	43	8
6-(17,12,1)	31	35	8	7-(13,9,1)	7	78	8
7-(14,10,1)	0	56(57)	8	5-(17,10,1)	2	48	7

bound. Finally, in terms of comparison, we have ran extensive tests against simulated annealing [15] for all the problems reported in Table 1, none was able to improve the best known upper bound.

6 Conclusion

The general strategy of cooperative multilevel algorithms is to solve several problems and use the solutions to define a new set of problems. This paper has described an exploratory application of this approach to covering designs. Blocks of successful approximate solutions discovered by a tabu search procedure are substituted to some blocks of an existing problem description, yielding a new problem definition. The key observation here is that the new problem definition hold at its core a successful combination of blocks. By making the problem small enough such that it holds only successful combination of blocks, we create conditions to obtain successful search directions from the re-combinations of blocks in the smaller problem. Furthermore, under the strict inclusion condition, blocks of the smaller problem are included in the definition of all the other problems. This provide for individual blocks to be tested inside good combinations of blocks, which often provides small increments in the definition of new successful combination of blocks. Overall, this multilevel strategy has already delivered interesting numerical results and seems to hold the potential to deliver more for covering designs and other problems in the field of combinatorial designs.

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. Multi-level adaptive solutions to boundary value problems. *Mathematics of Computation*, 31:333–390, 1977.
4. W.L. Briggs, V.E. Henson, and S.F. McCormick. *A Multigrid Tutorial*. SIAM, 1999.
5. C.J. Colbourn and J.H. Dinitz, editors. *The CRC Handbook of Combinatorial Designs*. CRC Press, 1996.
6. 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.
7. P. Erdős and H. Hanani. On a limit theorem in combinatorial analysis. *Publicationes Mathematicae Debrecen*, 10:10–13, 1963.
8. C.J. Gordon. Web site of covering bounds. *http://www.ccrwest.org/cover.html*.
9. C.J. Gordon, O. Patashnik, and G. Kuperberg. New constructions for covering designs. *Journal of Combinatorial Designs*, 3(4):269–284, 1995.
10. B. Hendrickson and R. Leland. The Chaco User's Guide: Version 2.0. Report SAND95-2344, Sandia National Laboratories, 1995.
11. F. Margot. Small covering designs by branch-and-cut. *Mathematical Programming*, 94:207–220, 2003.
12. 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.
13. W.H. Mills. Covering designs I: coverings by a small number of subsets. *Ars Combinatoria*, 8:199–315, August 1979.
14. K. J. Nurmela. Constructing combinatorial designs by local search. Technical report, Helsinki University of Technology, November 1993.
15. K. J. Nurmela and P. R. J. Östergård. Constructing covering designs by simulated annealing. Technical report, Helsinki University of Technology, January 1993.
16. K. J. Nurmela and P. R. J. Östergård. New coverings of t-sets with (t+1)-sets. *Journal of Combinatorial Designs*, 7:217–226, 1999.
17. 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.
18. V. Rödl. On a packing and covering problem. *European Journal of Combinatorics*, 5:69–78, 1985.
19. J. Schönheim. On coverings. *Pacific Journal of Mathematics*, 14:1405–1411, 1964.