



A Mathematical Programming Approach to the Construction of BIBDs

Journal:	<i>International Journal of Computer Mathematics</i>
Manuscript ID:	GCOM-2009-0553-B
Manuscript Type:	Original Article
Date Submitted by the Author:	02-Aug-2009
Complete List of Authors:	Yokoya, Daisuke; National Defense Academy, Computer Science Yamada, Takeo; National Defense Academy, Computer Science
Keywords:	balanced incomplete block design, mixed integer programming problem, branch-and-bound method, tabu search, solvers



International Journal of Computer Mathematics
Vol. 00, No. 00, January 2008, 1–16

A Mathematical Programming Approach to the Construction of BIBDs

Daisuke Yokoya^{a†} and Takeo Yamada^a

^a*Department of Computer Science, National Defense Academy,
Yokosuka, Kanagawa 239-8686, Japan;*

(Received 00 Month 200x; in final form 00 Month 200x)

BIBD (balanced incomplete block design) is instrumental in design of experiments. This is usually constructed using algebraic tools such as finite projective (or affine) algebra or difference sets. Recently, heuristic algorithms have been tried, including artificial neural networks, simulated annealing, and various local search methods. In this paper, we present a novel approach to construct BIBDs that makes use of MIP (mixed integer programming) solvers. Based on this, branch-and-bound and tabu search algorithms are given, and these are compared against the local search method presented by a previous researcher.

Keywords: BIBD (balanced incomplete block design); MIP (mixed integer programming problem); branch-and-bound method; tabu search

AMS Subject Classification: 90C10; 90C57; 05B05

1. Introduction

Let V be a set of v elements called *points* and B be a collection (i.e., multi-set) of b non-empty subsets of V called *blocks*. (V, B) is a *balanced incomplete block design* (BIBD [3, 8, 12, 22]), if the following conditions are satisfied.

- (i) Each point is contained in exactly r blocks.
- (ii) Each block contains exactly k points.
- (iii) Every pair of distinct points is contained in exactly λ blocks.

With positive integer parameters (v, b, r, k, λ) , this is denoted as $\text{BIBD}(v, b, r, k, \lambda)$. BIBD has its origin in statistical design of experiments [6], and recently it is also applied to coding theory [2, 7], failure diagnosis and group testing [10], and scheduling sports leagues [1] among others.

†Corresponding author. Email: g47090@nda.ac.jp

In matrix notation, BIBD is a $v \times b$ binary matrix $X = (x_{il})$ such that

$$\sum_{l=1}^b x_{il} = r, \quad i = 1, \dots, v, \quad (1)$$

$$\sum_{i=1}^v x_{il} = k, \quad l = 1, \dots, b, \quad (2)$$

$$\sum_{l=1}^b x_{il}x_{jl} = \lambda, \quad i, j = 1, \dots, v, \quad i < j, \quad (3)$$

$$x_{il} \in \{0, 1\}, \quad i = 1, \dots, v, \quad l = 1, \dots, b. \quad (4)$$

Form these relations, the following is easily shown [12].

PROPOSITION 1.1 *In BIBD(v, b, r, k, λ), parameters satisfy*

$$vr = bk, \quad (5)$$

$$r(k-1) = \lambda(v-1). \quad (6)$$

Then, b and r are determined by (v, k, λ) as

$$b = \frac{v(v-1)}{k(k-1)}\lambda, \quad (7)$$

$$r = \frac{v-1}{k-1}\lambda. \quad (8)$$

Thus, in literature BIBD(v, b, r, k, λ) is often referred to as (v, k, λ) -BIBD, or alternatively 2 -(v, k, λ) *design* [8, 25]. Given a set of integer parameters (v, b, r, k, λ) satisfying (5) and (6), we are concerned with the existence and construction of such a BIBD. For the limited case of $k = 3$ or $k = 4$, Hanani [13, 14] proved the following.

THEOREM 1.2 *Let v and λ be positive integers, and $k = 3$ or $k = 4$. Then, BIBD(v, b, r, k, λ) exists if and only if b and r determined by (7) and (8) are both integers.*

Triple system is the special case of BIBD with $k = 3$, and *Steiner triple system* is with $k = 3$ and $\lambda = 1$ [8]. These are denoted as TS(v, λ) and STS(v) respectively. For the Steiner triple system, the above result was shown by Kirkman [16] as early as in 1847, together with a construction method for STS(v). Hanani [14] proved similar results for $k = 5$ and $k = 6$ with some minor exceptions. However, except for these neither necessary and sufficient conditions nor construction methods of BIBDs are known in general.

Various approaches have been tried for the construction of BIBDs, including algebraic methods such as *finite projective* (or *affine*) *geometry* and *difference sets* [12, 22], as well as computational methods like *constraint programming* [20], *neural networks* [18] and *meta-heuristic* algorithms [5, 17]. Recently, by an exhaustive computer search over more than 90000 days in total Bilous et al. [4] proved that no $(22, 8, 4)$ -BIBD exists. Similarly, Houghten et al. [15] declared non-existence of $(46, 6, 1)$ -BIBD. On the other hand, Morales [19] discovered six new BIBDs by an

elaborate tabu search [11]. Extensive list of BIBDs, as well as unknown BIBDs up to date, are available in the handbook by Colbourn and Dinitz [8].

The purpose of this paper is to present a novel approach for the construction of BIBDs that makes use of *mixed integer programming* (MIP) solvers [9]. In Section 2, we formulate the problem as a non-linear MIP problem, and present a backtracking framework to solve the problem through a repeated use of MIP solvers. Based on this, we develop a branch-and-bound algorithm in Section 3 and a tabu search algorithm in Section 4, respectively. In Section 5, we compare these algorithms against the local search method proposed by Prestwich [21] on the same 86 instances given in his paper. Out of 86 instances we solved 78, 23 more than the local search algorithm [21] was able to solve.

2. Backtracking algorithm

Finding a matrix satisfying (1) – (4) is a kind of *constraint satisfaction problem* which can be transformed into the following nonlinear 0-1 programming problem [23].

$$P : \quad \text{Maximize} \quad \sum_{i=1}^v \left(\sum_{l=1}^b x_{il} \right) + \sum_{l=1}^b \left(\sum_{i=1}^v x_{il} \right) + \sum_{i=1}^{v-1} \sum_{j=i+1}^v \left(\sum_{l=1}^b x_{il}x_{jl} \right) \quad (9)$$

$$\text{subject to} \quad \sum_{l=1}^b x_{il} \leq r, \quad i = 1, \dots, v, \quad (10)$$

$$\sum_{i=1}^v x_{il} \leq k, \quad l = 1, \dots, b, \quad (11)$$

$$\sum_{l=1}^b x_{il}x_{jl} \leq \lambda, \quad i, j = 1, \dots, v, \quad i < j, \quad (12)$$

$$x_{il} \in \{0, 1\}, \quad i = 1, \dots, v, \quad l = 1, \dots, b. \quad (13)$$

Note that equalities (1) – (3) are relaxed to inequalities (10) – (12), and the objective function is simply the sum of the left-hand sides of these inequalities. Then, the following is trivial.

THEOREM 2.1

- (i) P is always feasible.
- (ii) For an arbitrary feasible solution $X = (x_{ij})$ to P , let $z(X)$ denote its objective value. Then, we have

$$z(X) \leq vr + bk + \frac{v(v-1)}{2}\lambda. \quad (14)$$

- (iii) If (14) is satisfied with equality, (10) – (12) are also satisfied with equality, and thus X gives a $\text{BIBD}(v, b, r, k, \lambda)$.

Therefore, if equality holds in (14) with respect to an optimal solution X^* to P , we are done. Otherwise, if $z(X^*) < vr + bk + v(v-1)\lambda/2$ no BIBD exists. However, since P is a *nonlinear* 0-1 programming problems, it is difficult to get an optimal

solution. Thus, we propose an *incremental* approach that determines rows of X one by one.

Now, suppose that the first j rows of X is known as $X_j = (\bar{x}_{il})$, $i = 1, \dots, j$. From (1) – (4), this is a binary matrix that satisfies the followings.

$$\sum_{l=1}^b \bar{x}_{il} = r, \quad i = 1, \dots, j, \quad (15)$$

$$\sum_{i=1}^j \bar{x}_{il} \leq k, \quad l = 1, \dots, b, \quad (16)$$

$$\sum_{l=1}^b \bar{x}_{il} \bar{x}_{i'l} = \lambda, \quad i, i' = 1, \dots, j, \quad i < i'. \quad (17)$$

Then, the $(j + 1)$ th row vector $\mathbf{x} = (x_l)$ must satisfy

$$\sum_{l=1}^b x_l = r, \quad (18)$$

$$x_l \leq k - \sum_{i=1}^j \bar{x}_{il}, \quad l = 1, \dots, b, \quad (19)$$

$$\sum_{l=1}^b \bar{x}_{il} x_l = \lambda, \quad i = 1, \dots, j. \quad (20)$$

If such an \mathbf{x} is found, we can augment X_j to a $(j + 1) \times b$ matrix

$$X_{j+1} = \begin{pmatrix} X_j \\ \mathbf{x} \end{pmatrix}. \quad (21)$$

To obtain a binary vector \mathbf{x} satisfying (18) – (20), we formulate the following optimization problem.

$$P_j(X_j) : \quad \text{Maximize} \quad \sum_{l=1}^b x_l + \sum_{l=1}^b \left(\sum_{i=1}^j \bar{x}_{il} \right) x_l \quad (22)$$

$$\text{subject to} \quad \sum_{l=1}^b x_l \leq r, \quad (23)$$

$$x_l \leq k - \sum_{i=1}^j \bar{x}_{il}, \quad l = 1, \dots, b, \quad (24)$$

$$\sum_{l=1}^b \bar{x}_{il} x_l \leq \lambda, \quad i = 1, \dots, j, \quad (25)$$

$$x_l \in \{0, 1\}, \quad l = 1, \dots, b. \quad (26)$$

Contrary to nonlinear P , $P_j(X_j)$ is a *linear* 0-1 programming problem which can be solved (in many cases) using free or commercial MIP solvers. Let the optimal

1 objective value to $P_j(X_j)$ be $z_j^*(X_j)$. Then, since the objective function (22) is the
 2 sum of the left-hand sides of (23) and (25), any optimal solution to $P_j(X_j)$ gives a
 3 binary \mathbf{x} satisfying (18) – (20) if and only if
 4

$$5 \quad z_j^*(X_j) = r + j\lambda. \quad (27)$$

6 Without loss of generality the first and second rows of X_j can be assumed to be
 7

$$8 \quad X_2 = \begin{pmatrix} \overbrace{11 \cdots 11 \cdots 1}^r 0 \cdots 0 & 0 \cdots 0 \\ \underbrace{11 \cdots 1}_\lambda 0 \cdots 0 & \underbrace{1 \cdots 1}_{r-\lambda} 0 \cdots 0 \end{pmatrix}, \quad (28)$$

9 and a BIBD is obtained if we can augment this through (21) to a $v \times b$ matrix X_v .
 10 On the other hand, if

$$11 \quad z_j^*(X_j) < r + j\lambda \quad (29)$$

12 no BIBD exists as an extension of X_j , and thus subproblem $P_j(X_j)$ is *terminated*.

13 *Example 2.2* For BIBD(12, 22, 11, 6, 5), except for trivial inequalities and 0-1 con-
 14 ditions, $P_2(X_2)$ is

$$15 \quad \begin{aligned} & \text{Maximize } (3333322222 \ 2222222111 \ 11)\mathbf{x} \\ & \text{subject to } \begin{pmatrix} 1111111111 & 1111111111 & 11 \\ 1111111111 & 1000000000 & 00 \\ 1111100000 & 0111111000 & 00 \end{pmatrix} \mathbf{x} \leq \begin{pmatrix} 11 \\ 5 \\ 5 \end{pmatrix}. \end{aligned}$$

16 Solving this we have $\mathbf{x}^* = (0111100010 \ 0000100111 \ 11)$ with $z_2^*(X_2) = 21 = r + \lambda j$,
 17 and

$$18 \quad X_3 = \begin{pmatrix} 1111111111 & 1000000000 & 00 \\ 1111100000 & 0111111000 & 00 \\ 0111100010 & 0000100111 & 11 \end{pmatrix}.$$

19 Continuing this for $j = 3, \dots, 8$, we obtain

$$20 \quad X_8 = \begin{pmatrix} 1111111111 & 1000000000 & 00 \\ 1111100000 & 0111111000 & 00 \\ 0111100010 & 0000100111 & 11 \\ 0001100101 & 1011001100 & 11 \\ 0001111001 & 0110010111 & 00 \\ 1100010101 & 0100101110 & 01 \\ 1100001011 & 0001011101 & 10 \\ 1001001100 & 1001110011 & 01 \end{pmatrix}. \quad (30)$$

21 We stop here since $z_8^*(X_8) = 50 < r + \lambda j = 51$, and restart with the 8th row of
 22 X_8 (denoted as \mathbf{x}_8) changed to an optimal solution of $P_7(X_7)$ other than \mathbf{x}_8 . If no
 23 such solutions exist, we *backtrack*.

24 Before stating a backtracking algorithm [24], we note that optimal solution to
 25 $P_j(X_j)$ may not be unique even if (27) is satisfied. If this is the case, we do not know
 26 exactly which optimal solution to adopt in (21). In the backtracking algorithm, we
 27

list up all the optimal solutions of $P_j(X_j)$ and define its *children* by appending each of these solution vectors to X_j . Thus, we have a tree of subproblems rooted at $P_2(X_2)$, and traversing all the subproblems obtain a BIBD, or prove that no BIBD exists.

To list up all the optimal solutions of $P_j(X_j)$, we introduce the following *auxiliary* problem.

$$P_j(X_j, F, R) : \quad \text{Maximize} \quad (22)$$

$$\text{subject to} \quad (23) - (26),$$

$$x_l = 1, \quad l \in F, \quad (31)$$

$$x_l = 0, \quad l \in R. \quad (32)$$

Here F and R are the disjoint subsets of variables which are fixed either at 1 and 0, respectively. We obtain an optimal solution $\mathbf{x}^*(X_j, F, R)$ and the corresponding objective value $z_j^*(X_j, F, R)$ by solving $P_j(X_j, F, R)$ with some MIP solvers. In the following algorithm, this part is written as a function FIND_A_SOLUTION(X_j, F, R). Then, the algorithm to list up all the optimal solutions satisfying (27) is given as the following recursive procedure.

Algorithm FIND_ALL_SOLUTIONS

Input: j, X_j, F, R .

Output: All the optimal solutions of $P_j(X_j, F, R)$.

Step 1. Call FIND_A_SOLUTION(X_j, F, R), and obtain an optimal $\mathbf{x}^*(X_j, F, R)$ and the corresponding $z_j^*(X_j, F, R)$.

Step 2. If $z_j^*(X_j, F, R) < r + j\lambda$, then return.

Step 3. Output $\mathbf{x}^*(X_j, F, R)$ as an optimal solution.

Step 4. Let $\{u_1, u_2, \dots, u_p\}$ be the set of indices such that the corresponding variables of $\mathbf{x}^*(X_j, F, R)$ are all 1, and $u_i \notin F$ ($i = 1, \dots, p$).

Step 5. For $s = 1, 2, \dots, p$, do

- Let $F' := F \cup \{u_1, u_2, \dots, u_{s-1}\}$ and $R' := R \cup \{u_s\}$.
- Call FIND_ALL_SOLUTIONS(j, X_j, F', R') recursively.

Example 2.3 Applying the above algorithm to $P_7(X_7)$ with $F = R = \emptyset$, in Step 1 we obtain $\mathbf{x}^*(X_7, \emptyset, \emptyset) = \mathbf{x}_8$. This is the 8th row of X_8 in (30), and Step 3 outputs this as a solution. From the indices of non-zero elements of \mathbf{x}_8 we have $(u_1, u_2, \dots, u_{11}) = (1, 4, 7, \dots, 22)$. Then, in Step 5 auxiliary problems are generated by adding constraints $x_{u_1} = \dots = x_{u_{s-1}} = 1$ and $x_{u_s} = 0$ to $P_7(X_7)$ as $P_7^{(s)} := P_7(X_7, \{u_1, \dots, u_{s-1}\}, \{u_s\})$, and the feasible region of $P_7(X_7, \emptyset, \emptyset)$ is divided into mutually disjoint feasible regions of $P_7^{(s)}$ ($s = 0, 1, \dots, 11$). Here for $s = 0$ we mean $P_7^{(0)} := P_7(X_7, \{u_1, u_2, \dots, u_{11}\}, \emptyset)$, and from (23) \mathbf{x}_8 is the unique optimal solution to this problem. Then, by solving $P_7^{(1)} - P_7^{(11)}$ recursively, we obtain all the optimal solutions to $P_7(X_7, \emptyset, \emptyset)$, together with \mathbf{x}_8 .

3. A branch-and-bound method

We put FIND_ALL_SOLUTIONS into the backtracking framework and construct a branch-and-bound algorithm [23] to find a BIBD in the following way. First of all, we define $\bar{P}_j(X_j, F, R)$ as the *continuous relaxation* of $P_j(X_j, F, R)$, and $\bar{z}_j(X_j, F, R)$ denotes the optimal objective value to this problem. Then, we can

1 terminate subproblem $P_j(X_j, F, R)$ if

$$2 \bar{z}_j(X_j, F, R) < r + j\lambda. \quad (33)$$

3
4
5
6
7
8
9
10 We can check if (33) is satisfied by solving a *linear programming* (LP) prob-
11 lem. If (33) is not met, we solve the original integer programming (IP) problem
12 $P_j(X_j, F, R)$ to see if (27) is satisfied. Then, the algorithm is given as follows.

13
14
15 **Algorithm BAB.BIBD**

16 Input: j, X_j, F, R .

17 Step 1. If $j = v$, X is a BIBD. Output this and stop.

18 Step 2. i) Solve an LP problem $\bar{P}_j(X, F, R)$.

19 a) If $\bar{z}_j(X_j, F, R) < r + j\lambda$ then terminate this subproblem and
20 return.

21 b) Otherwise, if the solution $\mathbf{x}^*(X, F, R)$ is a 0-1 vector go to
22 Step 3.

23 ii) Solve an IP problem $P_j(X_j, F, R)$.

24 If $z^*(X, F, R) < r + j\lambda$, terminate this subproblem and return.

25 Step 3. Append $\mathbf{x}^*(X, F, R)$ to X_j and update this to a matrix X_{j+1} with $j+1$
26 rows.

27 Call BAB.BIBD($j+1, X_{j+1}, \emptyset, \emptyset$) recursively.

28 Step 4. Let $\{u_1, u_2, \dots, u_p\}$ be the set of indices such that the corresponding
29 components of $\mathbf{x}^*(X_j, F, R)$ are all 1, and $u_i \notin F$ ($i = 1, \dots, p$).

30 Step 5. For $s = 1, 2, \dots, p$, do

31 • Let $F' := F \cup \{u_1, u_2, \dots, u_{s-1}\}$ and $R' := R \cup \{u_s\}$.

32 • Call BAB.BIBD(j, X_j, F', R') recursively.

33 Step 6. No BIBDs exist. Stop.

34
35
36
37 Some remarks are in order on the branch-and-bound *strategies*. First of all, linear
38 relaxation is not mandatory, i.e., we may skip part (i) of Step 2. In the standard
39 case, we solve an LP problem $\bar{P}_j(X, F, R)$ first, and then solve $P_j(X, F, R)$ if nec-
40 essary, as described in the algorithm. This is referred to as the *bounding strategy*
41 *LP*. If we skip part (i) in Step 2 and go directly to part (ii), we call this bounding
42 strategy *IP*.

43
44 Next, recursive calls to subproblems can be carried out in the *reverse* order. For
45 the problem $P_7(X_7, \emptyset, \emptyset)$ in Example 2, the original algorithm calls subproblems
46 in the order of $P_7^{(1)} \rightarrow \dots \rightarrow P_7^{(11)}$. We call this the *forward branching strategy*
47 (*FWD*). In the *backward* strategy (*BWD*), these subproblems are called reversely as
48 $P_7^{(11)} \rightarrow \dots \rightarrow P_7^{(1)}$. We denote these strategies explicitly as BAB.BIBD(*bounding*
49 *strategy, branching strategy*), such as BAB.BIBD(*LP, FWD*).

50
51
52
53
54 *Example 3.1* Figure 1 shows the behavior of the algorithm under bounding strategy
55 IP and branching strategy FWD, i.e., BAB.BIBD(IP, FWD). As in Example 1
56 subproblem P_8 is terminated, and by solving $P_7^{(1)}$ we obtain an alternative solution
57 $\mathbf{x}'_8 = (0010101100 \ 01110011)$ that replaces the 8th row of X_8 in Example 1. In
58 Figure 1 this is P'_8 . Restarting from this point, the process stops again at P_{10} . Here
59
60

we switch to P'_{10} , and finally obtain an BIBD as

$$X_{12} = \begin{pmatrix} 1111111111 & 1000000000 & 00 \\ 1111100000 & 0111111000 & 00 \\ 0111100010 & 0000100111 & 11 \\ 0001100101 & 1011001100 & 11 \\ 0001111001 & 0110010111 & 00 \\ 1100010101 & 0100101110 & 01 \\ 1100001011 & 0001011101 & 10 \\ 0010010110 & 1010111101 & 00 \\ 1010000011 & 1110010010 & 11 \\ 0001011010 & 1101101010 & 10 \\ 1000101100 & 1001110011 & 01 \\ 0110011100 & 0111000001 & 11 \end{pmatrix} \quad (34)$$

after solving 13 IP problems.

Example 3.2 If we take bounding strategy LP instead of IP in the previous example, the algorithm BAB_BIBD(LP, FWD) produces the tree of subproblems as depicted in Figure 2. Here, shadowed nodes show the subproblems where both of LP and IP problems are solved, while at the remaining nodes we solve only LP problems. Thus, we obtain a BIBD

$$X_{12} = \begin{pmatrix} 1111111111 & 1000000000 & 00 \\ 1111100000 & 0111111000 & 00 \\ 1011100001 & 0000001111 & 11 \\ 0001100110 & 1011010100 & 11 \\ 0000111001 & 1111001001 & 10 \\ 0011011000 & 1100110010 & 11 \\ 0100100110 & 1100101011 & 01 \\ 1010001110 & 0011100011 & 10 \\ 0101001101 & 0101010111 & 00 \\ 1100010011 & 0101100100 & 11 \\ 1000010101 & 1010111110 & 00 \\ 0110011010 & 0010011101 & 01 \end{pmatrix} \quad (35)$$

after solving 12 LP and 7 IP problems. Note that the obtained BIBD (35) is different from the one shown in (34), as early as at $j = 3$.

Example 3.3 Figure 3 shows the behavior of the backward branching strategy, i.e., BAB_BIBD(LP, BWD). The algorithm proceeds exactly the same way as in the previous example until again P_9 is terminated (compare Figures 2 and 3). Then, auxiliary subproblems $P_8^{(1)} - P_8^{(8)}$ are generated and examined as in the previous example but in the reverse order. The first 7 subproblems are terminated due to

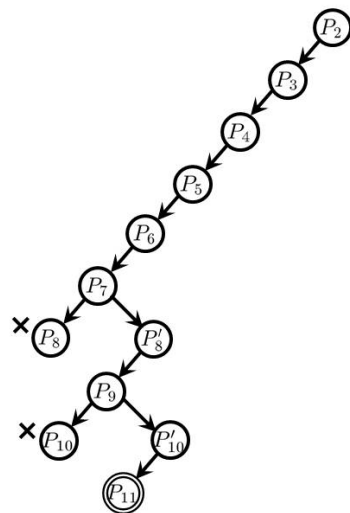


Figure 1. Behavior of BAB_BIBD(IP, FWD) for BIBD(12,22,11,6,5).

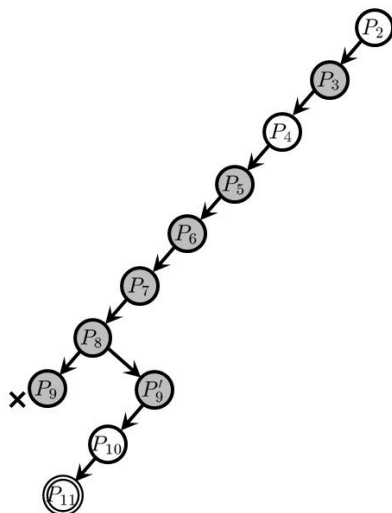


Figure 2. Behavior of BAB_BIBD(LP, FWD) for BIBD(12,22,11,6,5).

(33), we restart from $P_8^{(8)}$, and at P_{11} obtain a BIBD

$$\begin{pmatrix}
 1111111111 & 1000000000 & 00 \\
 1111100000 & 0111111000 & 00 \\
 1011100001 & 0000001111 & 11 \\
 0001100110 & 1011010100 & 11 \\
 0000111001 & 1111001001 & 10 \\
 0011011000 & 1100110010 & 11 \\
 0100100110 & 1100101011 & 01 \\
 1010001110 & 0011100011 & 10 \\
 1100010101 & 0110100100 & 11 \\
 0110011010 & 0010011101 & 01 \\
 0101001101 & 0101010111 & 00 \\
 1000010011 & 1001111110 & 00
 \end{pmatrix} \tag{36}$$

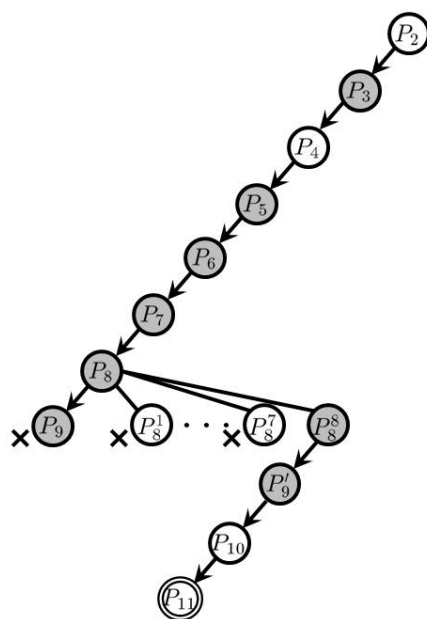


Figure 3. The behavior of BAB_BIBD(LP, BWD) for BIBD(12,22,11,6,5).

after solving 18 LP and 8 IP problems. Note that this BIBD is different again either from (34) or (35), although the first 8 rows of (35) and (36) are identical.

Now, we evaluate the performance of the branch-and-bound algorithm for triple systems $TS(v, \lambda)$. Table 1 shows the result of BAB_BIBD(LP, FWD) on an DELL Precision 670 computer (CPU: Xeon 3.8GHz \times 2) with CPLEX 10.100 [9] to solve LP and IP problems. Here shown are the number of subproblems generated (#Subp), the number of LP and IP problems solved (#LP, #IP) and the CPU time in seconds. We solved three instances for each value of $\lambda \leq 30$, within $v \leq 200$ and $b \leq 3000$.

From this table, we see that in almost all instances $\#Subp = v - 1$, which means that backtracking seldom occurs in triple systems. Next, #IP is relatively small, i.e., in most cases we obtain BIBD by solving approximately v LP and some additional IP problems. This implies that we frequently obtain 0-1 solutions by solving only LP problems, and thus bounding strategy LP is well suited to this case. To sum up, the branch-and-bound method works effectively for triple systems.

4. Tabu search

Branch-and-bound method is often inefficient for problems where backtracking occurs frequently. For example, in solving BIBD(10,15,6,4,2) by BAB_BIBD(LP, FWD), we had to solve 1,207,105 LP and 343,523 IP problems and the CPU time was 2068 seconds on the same computer used in Section 3. We were unable to solve BIBD(16,16,6,6,2) within 3600 seconds, while the local search algorithm by Prestwich [21] solved this within one second on a slower computer. The difficulty with the branch-and-bound method is the fact that once a 'bad' row vector x_i has been included in the binary matrix X_j with $i < j$, it takes very long time until we backtrack to X_{i-1} and thus eliminate the bad vector.

Therefore, we propose a method that makes a guess of a bad row vector included

Table 1. Result of BAB.BIBD(LP, FWD) for triple systems TS(v, λ).

<i>v</i>	<i>b</i>	<i>r</i>	<i>k</i>	<i>λ</i>	#Subp	#LP	#IP	CPUseC
39	247	19	3	1	40	39	5	0.48
79	1027	39	3	1	80	79	7	5.47
133	2926	66	3	1	132	131	8	42.88
30	290	29	3	2	29	28	1	0.35
60	1180	59	3	2	59	58	7	4.28
94	2914	93	3	2	93	92	10	24.77
19	171	27	3	3	18	17	0	0.11
47	1081	69	3	3	46	45	1	2.72
77	2926	114	3	3	76	75	7	18.82
19	228	36	3	4	18	17	1	0.15
45	1320	88	3	4	44	43	3	3.15
67	2948	132	3	4	66	65	7	15.81
15	175	35	3	5	14	13	0	0.07
33	880	80	3	5	32	31	0	1.29
57	2660	140	3	5	56	55	1	10.52
14	182	39	3	6	13	12	1	0.07
40	1560	117	3	6	39	38	2	3.33
55	2970	162	3	6	54	53	3	11.82
9	108	36	3	9	8	7	0	0.01
29	1218	126	3	9	28	27	2	1.63
45	2970	198	3	9	44	43	2	8.92
9	144	48	3	12	8	7	0	0.02
25	1200	144	3	12	24	23	3	1.31
39	2964	228	3	12	38	37	2	7.66
11	275	75	3	15	10	9	0	0.06
23	1265	165	3	15	22	21	1	1.24
35	2975	255	3	15	34	33	1	6.64
10	270	81	3	18	9	8	0	0.06
20	1140	171	3	18	19	18	1	0.89
32	2976	279	3	18	31	30	5	6.12
11	385	105	3	21	10	9	0	0.11
19	1197	189	3	21	18	17	1	0.88
29	2842	294	3	21	28	27	4	5.13
10	360	108	3	24	9	8	1	0.09
18	1224	204	3	24	17	16	1	0.83
27	2808	312	3	24	26	25	1	4.41
9	324	108	3	27	8	7	0	0.06
17	1224	216	3	27	16	15	0	0.78
25	2700	324	3	27	24	23	0	3.81
9	360	120	3	30	8	7	0	0.06
17	1360	240	3	30	16	15	1	0.90
25	3000	360	3	30	24	23	3	4.51

in X_j and remove it at an earlier stage. The idea is as follows. When we solve $P_j(X_j)$ and obtain an optimal solution $\mathbf{x}^* = (x_l^*)$ with $z_j^*(X_j) = r + j\lambda$, we move forward by adding \mathbf{x}^* to X_j as we did in the branch-and-bound algorithm. On the other hand, if $z_j^*(X_j) < r + j\lambda$ we have either

$$\sum_{l=1}^b \bar{x}_{il} x_l^* < \lambda \tag{37}$$

for some row i , or

$$\sum_{l=1}^b x_l^* < r. \tag{38}$$

In case of (37), we consider \mathbf{x}_i as a bad row and eliminate it from X_j . If no such row exists, we have (38). Then, we pick up a row vector in X_j at random and remove it as well. In either of these cases, after eliminating such a row vector from

X_j we restart with a reduced matrix with $j - 1$ rows.

To prevent a removed vector from returning to X_j in the subsequent few steps, we introduce a *tabu list* \mathcal{T} to keep the eliminated vectors ‘frozen’ for a certain period of steps. The size of \mathcal{T} is specified by a parameter TL (tabu length), and vectors in the tabu list are forbidden to be an optimal solution to $P_j(X_j)$. To realize this, let $\mathcal{T} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(p)}\}$ be the current tabu list, and consider the following 0-1 programming problem.

$$\begin{aligned} P_j(X_j, \mathcal{T}) : \quad & \text{Maximize} \quad (22) \\ & \text{subject to} \quad (23) - (26), \\ & \mathbf{x}^{(s)} \cdot \mathbf{x} \leq r - 1, \quad s = 1, \dots, p. \end{aligned} \quad (39)$$

Let $z_j^*(X_j, \mathcal{T})$ be the optimal objective value to this problem. Due to (39) no vectors in the tabu list satisfy (23) with equality. Then, if $z_j^*(X_j, \mathcal{T}) = r + \lambda j$ holds, the optimal $\mathbf{x}_j^*(X_j, \mathcal{T})$ is not tabu, and thus we can expand X_j by appending this vector to X_j . Otherwise, we choose a row vector as stated before and eliminate it from X_j . The tabu search algorithm [11], with the tabu length explicitly shown as $TABU_BIBD(TL)$, is as follows.

Algorithm TABU_BIBD

- Step 1. Let $j := 2$, X_2 be given as (28), and $\mathcal{T} := \emptyset$.
- Step 2. If $j = v$, then X_j is a BIBD. Output this and stop.
- Step 3. Solve an IP problem $P_j(X_j, \mathcal{T})$. If $z_j^*(X_j, \mathcal{T}) = r + j\lambda$, go to Step 4. Otherwise go to Step 5.
- Step 4. Append the solution to X_j and augment it to X_{j+1} . Go back to Step 2.
- Step 5. If there exists a row i satisfying (37), go to Step 6. Otherwise pick up a row vector from X_j at random and go to Step 6.
- Step 6. Eliminate the selected vector from X_j , let $j \leftarrow j - 1$, and add the eliminated vector to the tabu list \mathcal{T} . (If $|\mathcal{T}| > TL$, the oldest vector is removed from \mathcal{T})
- Step 7. If terminating condition is satisfied, then print “Tabu search failed,” and stop. Otherwise go back to Step 2.

Example 4.1 In applying the tabu search algorithm to $BIBD(16,16,6,6,2)$ where BAB_BIBD failed to find a solution within 3600 CPU seconds, the tabu search produced a correct BIBD in 0.02 seconds as follows. Initially, the tabu list is initialized as $\mathcal{T} = \emptyset$. The algorithm behaves exactly the same way as $BAB_BIBD(IP, FWD)$ until at $j = 8$ we obtain

$$X_8 = \begin{pmatrix} 1111110000 & 000000 \\ 1100001111 & 000000 \\ 1100000000 & 001111 \\ 0100010100 & 110010 \\ 0010010110 & 000101 \\ 1001000010 & 110100 \\ 0001011010 & 001010 \\ 0001100101 & 000110 \end{pmatrix}.$$

Here we have an optimal solution $\mathbf{x}^* = (001000100 \ 010011)$ with $z_8^*(X_8) = 21 <$

Table 2. Result of computation for Prestwich's 86 instances (Part 1).

v	b	r	k	λ	vb	Prestwich	BAB	TABU
8	14	7	4	3	112	0.00	0.00	0.00
11	11	5	5	2	121	0.04	0.01	0.01
10	15	6	4	2	150	0.04	2068.03	0.02
9	18	8	4	3	162	0.05	0.01	0.00
13	13	4	4	1	169	0.01	0.00	0.00
10	18	9	5	4	180	0.12	0.01	0.01
8	28	14	4	6	224	0.06	0.00	0.00
15	15	7	7	3	225	0.61	0.01	0.01
11	22	10	5	4	242	1.20	0.01	0.01
16	16	6	6	2	256	0.54	—	0.02
12	22	11	6	5	264	1.23	0.02	0.02
10	30	12	4	4	300	0.12	0.05	0.01
16	20	5	4	1	320	0.16	0.01	0.01
9	36	16	4	6	324	0.21	0.01	0.00
8	42	21	4	9	336	0.25	0.01	0.00
13	26	8	4	2	338	0.50	0.04	0.06
13	26	12	6	5	338	7.93	17.54	0.08
10	36	18	5	8	360	1.08	0.03	0.03
19	19	9	9	4	361	9.73	0.21	0.05
11	33	15	5	6	363	1.79	0.04	0.01
14	26	13	7	6	364	—	0.38	0.88
16	24	9	6	3	384	9.80	0.03	0.05
12	33	11	4	3	396	0.33	—	0.22
21	21	5	5	1	441	0.22	0.31	0.02
8	56	28	4	12	448	0.12	0.01	0.01
10	45	18	4	6	450	0.08	0.02	0.01
15	30	14	7	6	450	—	10.13	0.95
16	30	15	8	7	480	—	237.50	0.44
11	44	20	5	8	484	1.13	0.04	0.01
9	54	24	4	9	486	0.21	0.01	0.00

$r + \lambda j = 22$. The first row conflicts against \mathbf{x}^* , since $\mathbf{x}_1 \cdot \mathbf{x}^* = 1 < \lambda = 2$. We remove \mathbf{x}_1 from X_8 and solve $P_7(X'_7, \mathcal{T})$ with the reduced matrix X'_7 . Again we have a conflicting row \mathbf{x}_4 and eliminating this obtain a matrix X'_6 with 6 rows. After this, 10 forward moves are repeated and at this point we obtain a BIBD(16,16,6,6,2) as

```

(1100001111 000000)
1100000000 001111
0100010100 110010
1001000010 110100
0001011010 001010
0001100101 000110
0101101000 100001
0110110010 000100
0111000001 011000
1011010100 000001
1000110001 101000
0000100110 011001
0000011001 010101
0010001100 101100
0010000011 100011
1010101000 010010)

```

Prestwich [21] proposed a local search algorithm, and tried 86 instances with v and b satisfying $vb \leq 1000$. Tables 2 – 4 compare his result against our branch-and-bound and tabu search methods with MIP solvers. In these tables CPU time in seconds is shown for each method. The column of 'Prestwich' is the result of his method on a DEC Alphaserver 100A 5/300 computer (300MHz), while 'BAB'

Table 3. Result of computation for Prestwich's 86 instances (Part 2).

v	b	r	k	λ	vb	Prestwich	BAB	TABU
13	39	12	4	3	507	1.22	0.03	0.02
13	39	15	5	5	507	11.20	0.05	0.01
16	32	12	6	4	512	—	1985.33	3.57
15	35	14	6	5	525	—	3.29	0.05
12	44	22	6	10	528	27.90	0.04	0.03
23	23	11	11	5	529	—	—	0.07
10	54	27	5	12	540	0.98	0.04	0.02
8	70	35	4	15	560	0.13	0.01	0.00
17	34	16	8	7	578	—	—	7.37
10	60	24	4	8	600	0.75	0.01	0.01
11	55	20	4	6	605	0.80	0.03	0.01
11	55	25	5	10	605	4.45	0.02	0.06
18	34	17	9	8	612	—	—	30.91
25	25	9	9	3	625	—	—	0.24
15	42	14	5	4	630	42.90	0.14	0.05
21	30	10	7	3	630	—	—	61.42
16	40	10	4	2	640	4.22	0.05	0.08
16	40	15	6	5	640	—	2.72	0.84
9	72	32	4	12	648	0.54	0.01	0.01
15	45	21	7	9	675	—	0.06	0.06
13	52	16	4	4	676	3.76	0.03	0.01
13	52	24	6	10	676	49.70	0.04	0.02
10	72	36	5	16	720	1.33	0.02	0.01
19	38	18	9	8	722	—	—	—
11	66	30	5	12	726	1.52	0.04	0.01
22	33	12	8	4	726 ^a	—	—	—
14	52	26	7	12	728	—	0.11	0.07
27	27	13	13	6	729	—	—	0.54
21	35	15	9	6	735	—	—	—
10	75	30	4	10	750	0.92	0.02	0.02

^aNo solution exists. See [4].

is by the branch-and-bound algorithm BAB_BIBD(LP , FWD), and 'TABU' is by TABU_BIBD(10) on a faster DELL Precision 670 machine (3.8GHz \times 2). Computation was truncated (and shown in tables by —) at 2 million backtracks in 'Prestwich' and 3600 CPU seconds in our algorithms.

From these tables we see that the branch-and-bound method solves more instances than the Prestwich's method. However, in some instances it took much longer time, or even unable to solve, while the same instances were solved easily by the Prestwich's local search algorithm. By the tabu search method of this paper, we were able to solve all the instances that were solved by Prestwich. The tabu search algorithm solved some additional instances that neither Prestwich's nor the branch-and-bound algorithms were able to solve.

Table 5 gives a summary of the numbers of instances out of 86 solved by Prestwich's and our methods. The branch-and-bound method solved approximately 10 more instances than the Prestwich's, irrespective to bounding and branching strategies. Tabu search was able to solve 10 additional instances, irrespective to tabu length.

5. Conclusion

We have presented branch-and-bound and tabu search algorithms to find BIBDs that make use of MIP solvers. The developed methods were successful for triple systems, and solved more instances than the previous algorithm based on local search method. However, with these algorithms we were unable to find new BIBDs. To solve problems with $v \geq 30$, some new bounding conditions are required, and thus make the algorithm more efficient.

Table 4. Result of computation for Prestwich's 86 instances (Part 3).

v	b	r	k	λ	vb	Prestwich	BAB	TABU
25	30	6	5	1	750	14.30	0.04	0.03
20	38	19	10	9	760	—	—	—
16	48	15	5	4	768	43.20	0.11	0.15
16	48	18	6	6	768	—	0.38	0.37
12	66	22	4	6	792	1.38	0.05	0.02
12	66	33	6	15	792	12.80	0.04	0.03
9	90	40	4	15	810	0.69	0.02	0.02
13	65	20	4	5	845	1.40	0.04	0.01
11	77	35	5	14	847	5.11	0.03	0.02
21	42	10	5	2	882	—	—	798.86
21	42	12	6	3	882	—	—	—
21	42	20	10	9	882	—	—	—
16	56	21	6	7	896	—	0.12	0.11
10	90	36	4	12	900	0.99	0.04	0.01
15	60	28	7	12	900	—	0.09	0.05
18	51	17	6	5	918	—	0.78	1.34
22	42	21	11	10	924	—	—	—
15	63	21	5	6	945	30.70	0.09	0.20
16	60	15	4	3	960	6.31	0.05	0.02
16	60	30	8	14	960	—	0.34	1.12
31	31	6	6	1	961	2.04	0.07	0.04
31	31	10	10	3	961	—	—	1.70
31	31	15	15	7	961	—	—	0.10
11	88	40	5	16	968	4.07	0.03	0.04
22	44	14	7	4	968	—	—	—
25	40	16	10	6	1000	—	—	—

Table 5. Summary of computation.

Methods	Solved	Unsolved
Prestwich	55	31
BAB.BIBD(LP, FWD)	66	20
BAB.BIBD(IP, FWD)	63	23
BAB.BIBD(LP, BWD)	64	22
BAB.BIBD(IP, BWD)	60	26
TABU.BIBD(5)	77	9
TABU.BIBD(10)	77	9
TABU.BIBD(20)	78	8

References

- [1] I. Anderson, *Combinatorial Designs and Tournaments*, Oxford University Press, New York, 2006.
- [2] T. Beth, D. Jungnickel, and H. Lenz, *Design theory, 2nd ed.*, Cambridge University Press, Cambridge, 1999.
- [3] N.L. Biggs, E.K. Lloyd, and R.J. Wilson, *The history of combinatorics*, in R.L. Graham, M. Grötschel and L. Lovász (eds.), *Handbook of Combinatorics, Vol. II*, Elsevier, Amsterdam, 1995, pp. 2163-2198.
- [4] R. Bilous, C.W.H. Lam, L.H. Thiel, B.P.C. Li, G.H.J. van Rees, S.P. Radziszowski, W.H. Holzmann, and H. Kharaghani, *There is no 2-(22, 8, 4) block design*, *Journal of Combinatorial Designs*, 15(2006), pp. 262-267.
- [5] P. Bofill, R. Guimera, and C. Torras, *Comparison of simulated annealing and mean field annealing as applied to the generation of block designs*, *Neural Networks*, 16(2003), pp. 1421-1428.
- [6] G.E. Box, W.G. Hunter, J.S. Hunter, and W.G. Hunter, *Statistics for Experiments: Design, Innovation, and Discovering, 2nd ed.*, John Wiley & Sons, Hoboken, 2005.
- [7] E. Brouwer, *Block designs*, in R. Graham, M. Grötschel, and L. Lovász (Editors), *Handbook of Combinatorics, Vol. I*, Elsevier, Amsterdam, 1995, pp. 693-771.
- [8] C.J. Colbourn, and J.H. Dinitz, *Handbook of Combinatorial Designs 2nd ed.*, Chapman & Hall/CRC, New York, 2007.
- [9] CPLEX Ver. 11.0, ILOG, <http://www.ilog.com/products/cplex/>, 2008.
- [10] D.-Z. Du, and F.K. Hwang, *Combinatorial Group Testing and its Applications, 2nd ed.*, World Scientific Publ., Singapore, 2000.
- [11] F. Glover, and M. Laguna, *Tabu Search*, Kluwer, Norwell, 1997.
- [12] M. Hall, Jr., *Combinatorial Theory, 2nd ed.*, John Wiley & Sons, New York, 1998.
- [13] H. Hanani, *The existence and construction of balanced incomplete block designs*, *Annals of Mathematical Statistics*, 32(1961), pp. 361-386.
- [14] H. Hanani, *Balanced incomplete block designs and related designs*, *Discrete Mathematics*, 11(1975), pp. 255-369.

- 1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
- [15] S.K. Houghten, L.H. Thiel, J. Janssen, and C.W.H. Lamet, *There is no (46,6,1) block design*, Journal of Combinatorial Designs, 9(2001), pp. 60-71.
- [16] T. Kirkman, *On a problem in combinations*, Cambridge and Dublin Math. Journal 2, (1847), pp. 191-204.
- [17] D. L. Kreher, and D. R. Stinson, *Combinatorial Algorithms: Generation, Enumeration and Search*, CRC Press, Boca Raton, 1999.
- [18] T. Kurokawa, and Y. Takefuji, *Neural network parallel computing for BIBD problems*, IEEE Trans on Circuits and Systems II, 39(1992), pp. 243-247.
- [19] L.B. Morales, *Constructing difference families through an optimization approach: Six new BIBDs*, Journal of Combinatorial Designs, 8(2000), pp. 261-273.
- [20] S. Prestwich, *Balanced incomplete block design as satisfiability*, Irish Conf. on AI and Cognitive Science 12(2001), 189-198.
- [21] S. Prestwich, *A local search algorithm for balanced incomplete block designs*, Lecture Notes in Artificial Intelligence, Cork Constraint Computation Centre, University Collage Cork, 2002.
- [22] V.N. Sachkov, *Combinatorial Methods in Discrete Mathematics*, Cambridge University Press, Cambridge, 1977.
- [23] A. Schrijver, *Theory of Linear and Integer Programming*, John Wiley & Sons, Amsterdam, 1998.
- [24] R. Sedgewick, *Algorithms in C, 3rd ed.*, Addison-Wesley, Massachusetts, 1998.
- [25] D. R. Stinson, *Combinatorial Designs: Construction and Analysis*, Springer-Verlag, New York, 2004.