Balanced Incomplete Block Design as Satisfiability

Steven Prestwich
Department of Computer Science
National University of Ireland at Cork
tel. +353 21 490 3165
s.prestwich@cs.ucc.ie

Abstract

Balanced incomplete block design generation is a standard combinatorial problem from design theory. Constraint programming has recently been applied to the problem using a mixture of binary and non-binary constraints, with special techniques for symmetry breaking. We describe a new binary constraint model and apply search algorithms indirectly via satisfiability encoding. The encoded problems turn out to be hard for current algorithms, and symmetry breaking sometimes makes them harder, but the results suggest a promising direct approach.

1 Introduction

Balanced incomplete block design (BIBD) generation is a standard combinatorial problem from design theory, originally used in the design of statistical experiments but since finding other applications such as cryptography. It is a special case of block design, which also includes such problems as latin squares. A constraint programming approach to BIBD generation was recently described by Meseguer & Torras [15]. They exploit symmetries to improve variable selection and domain pruning in backtracking search (forward checking with conflict-directed backjumping), reducing the time to find all solutions. This paper proposes a new constraint model for BIBD generation, and investigates the application of backtracking, local search and hybrid algorithms to the problem via satisfiability (SAT) encoding.

We are particularly interested in the use of *symmetry breaking* techniques to partially or wholly eliminate symmetries from constraint satisfaction problems (CSPs). Many CSPs contain symmetries. An example is the N-queens problem, which has 8 symmetries: each solution may be rotated through 90 degrees and reflected about an axis. Other problems have many more symmetries that make it very hard to find all solutions, including the special case where there are no solutions and we wish to prove unsatisfiability. Symmetry breaking techniques may be broadly divided into two camps. Probably the most popular is to add symmetry breaking constraints to the problem formulation, so that each equivalence class of solutions to the original problem corresponds to a single solution in the new problem. It is rarely possible to achieve this goal completely but results are often good, and symmetries may be detected in SAT encodings [3] or lifted (quantified) versions of CSPs [11]. A formal framework for this approach is given in [18]. The other approach is to detect and exploit symmetries dynamically during search. An ordering can be defined on solutions, and the search restricted to the first solutions under this ordering within each equivalence class [2]. Constraints may be posted at each branch point in the search tree [7]. The search may be guided towards subspaces with many non-symmetric states [15].

Symmetry breaking techniques are usually applied when generating all solutions or proving unsolvability, and a popular benchmark is the unsolvable pigeonhole problem. However, it is hard to find reported results for their effects on single-solution search, an exception being a remark in [10] that they do not always improve the performance of the Oz system. From conversations with other researchers this appears to be common knowledge, but it seems worth investigating further because symmetry breaking is widely used. For the common situation in which we simply wish to find a solution, any guidelines on when to use symmetry breaking would be valuable. For example, might symmetry breaking hinder the search for a solution? Does the answer depend on the type of algorithm used? This paper provides some tentative answers.

2 Balanced incomplete block design

A BIBD is defined as an arrangement of v distinct objects into b blocks such that each block contains exactly k distinct objects, each object occurs in exactly r different blocks, and every two distinct objects occur together in exactly λ blocks. Another way of defining a BIBD is in terms of its *incidence matrix*, which is a $v \times b$ binary matrix with exactly r ones per row, k ones per column, and with scalar product λ between any pair of distinct rows. A BIBD is therefore specified by its parameters (v, b, r, k, λ).

It can be proved that for a BIBD to exist its parameters must satisfy the conditions $rv = bk$, $\lambda(v-1) = r(k-1)$ and $b \geq v$, but these are not sufficient conditions. Constructive methods can be used to design BIBDs of special forms, but the general case is very challenging and there are surprisingly small open problems, the smallest being (22,33,12,8,4) [14]. One source of intractability is the large number of symmetries: given any solution, any two rows or columns may be exchanged to obtain another solution.

2.1 Binary CSP model

A wide variety of problems encountered in artificial intelligence can be expressed as CSPs. A CSP consists of a set of variables, and a set of constraints (relations) defined on subsets of the variables. Each variable has a domain of possible values. The problem is to assign a value to each variable from its domain without violating any constraints.

The most direct CSP model for BIBD, as described in [15], would be to represent each matrix entry by a boolean variable. There are then three types of constraint: (i) v b-ary constraints for the r ones per row, (ii) b v-ary constraints for the k ones per column, and (iii) $v(v-1)/2$ $2b$-ary constraints for the λ matching ones in each pair of rows. These constraints are expensive to process and not very useful for propagation. This direct model is modified in [15] to avoid the most expensive constraints (iii), using $\lambda v(v-1)/2$ variables to denote the column number of each shared one between each pair of rows. However, this still leaves the fairly expensive constraints (i) and (ii).

There are standard techniques for transforming a non-binary constraint problem into a binary one (the dual graph [6] and hidden variable [5] methods). Alternatively, we can obtain a binary constraint model directly, via the same trick of using integer variables to represent positions. To do this we define variables denoting the positions of the ones *and zeroes* in each row and column, giving five sets of variables:

$$\begin{array}{ll} R1_{i,j} & (1 \leq i \leq v,\ 1 \leq j \leq r) \\ R0_{i,j} & (1 \leq i \leq v,\ 1 \leq j \leq b-r) \\ C1_{i,j} & (1 \leq i \leq b,\ 1 \leq j \leq k) \\ C0_{i,j} & (1 \leq i \leq b,\ 1 \leq j \leq v-k) \\ S1_{i,j,p} & (1 \leq i < j \leq v,\ 1 \leq p \leq \lambda) \end{array}$$

The $R1_{i,j}$ denote the positions in row i of the r ones, and the $R0_{i,j}$ those of the $b-r$ zeroes. Similarly the $C\{0,1\}_{i,j}$ denote the positions of the k ones and $v-k$ zeroes in column i. The $S1_{i,j,p}$ denote the shared positions of the λ ones in rows i and j. The $R\{0,1\}_{i,j}$ and $S1_{i,j,p}$ have domain $\{1 \dots b\}$ and the $C\{0,1\}_{i,j}$ have domain $\{1 \dots v\}$.

The constraints are as follows. No zero can be placed in the same place as a one

$$R1_{i,j} \neq R0_{i,j'} \quad C1_{i,j} \neq C0_{i,j'}$$

The row values must agree with the column values. That is, if a one (or zero) is placed in a given row (or column) position, then no zero (or one) can be placed in the corresponding column (or row) position.

$$R1_{i,j} \neq p \vee C0_{p,q} \neq i \quad R0_{i,j} \neq p \vee C1_{p,q} \neq i$$

Each $S1_{i,j,p}$ must agree with one of the $R1_{i,j'}$ and one of the $R1_{i',j}$, but to state this directly would involve non-binary constraints. Instead we state it indirectly as: no two rows can share a one at a given position if a zero is placed there in either row.

$$S1_{i,j,p} \neq R0_{i,q} \quad S1_{i,j,p} \neq R0_{j,q} \quad (i < j)$$

The positions of the ones and zeroes must be different in both rows and columns, to ensure that no location contains more than one entry. Because there are exactly b variables for each row and v for each column, this also ensures that every row and column position is assigned either a one or a zero. We consider three ways of enforcing this condition, which we shall refer to as three *levels* of symmetry breaking. At level 1 we use constraints

$$\begin{array}{r} R1_{i,j} < R1_{i,j+1} \quad R0_{i,j} < R0_{i,j+1} \quad C1_{i,j} < C1_{i,j+1} \quad C0_{i,j} < C0_{i,j+1} \\ S1_{i,j,p} < S1_{i,j,p+1} \quad (i < j) \end{array}$$

at level 2 we add implied constraints

$$\begin{array}{r} R1_{i,j} < R1_{i,j'} \quad R0_{i,j} < R0_{i,j'} \quad C1_{i,j} < C1_{i,j'} \quad C0_{i,j} < C0_{i,j'} \quad (j < j') \\ S1_{i,j,p} < S1_{i,j,q} \quad (i < j,\ p < q) \end{array}$$

and at level 3 we add further implied constraints

$$
\begin{array}{lll}
R1_{i,j} + d < R1_{i,j'} & R0_{i,j} + d < R0_{i,j'} & (j < j',\ d = j' - j - 1) \\
C1_{i,j} + d < C1_{i,j'} & C0_{i,j} + d < C0_{i,j'} & (j < j',\ d = j' - j - 1) \\
 & S1_{i,j,p} + d < S1_{i,j,q} & (i < j,\ p < q,\ d = q - p - 1)
\end{array}
$$

We also consider the effect of applying no symmetry breaking, referred to as level 0. This will obviously be counter-productive when searching for all solutions, or proving unsolvability, but its effect on finding the first solution is worth investigating. Instead of constraining the variables to be ordered, we merely constrain them to take different values.

$$
\begin{array}{lllll}
R1_{i,j} \neq R1_{i,j'} & R0_{i,j} \neq R0_{i,j'} & C1_{i,j} \neq C1_{i,j'} & C0_{i,j} \neq C0_{i,j'} & (j < j') \\
 & & & S1_{i,j,p} \neq S1_{i,j,q} & (i < j,\ p < q)
\end{array}
$$

Our model has a total of $\lambda v(v-1)/2 + 2vb$ integer variables, whereas Meseguer & Torras's model has $\lambda v(v-1)/2$ integer variables and vb boolean variables.

2.2 SAT model

Our experiments are all performed on propositional satisfiability models of these CSPs, which allows a rapid comparison of commonly available algorithms. SAT is the archetypal NP-hard problem. It has applications such as planning and VLSI design, has well-known algorithms and benchmarks, and has been the subject of a great deal of recent research. The SAT problem is to determine whether a boolean expression has a satisfying assignment. The expression is usually in conjunctive normal form: a conjunction of clauses $C_1 \wedge \ldots \wedge C_m$ where each clause C_i is a disjunction of literals $l_1 \vee \ldots \vee l_n$ and each literal is either a boolean variable v_i or its negation $\neg v_i$. A boolean variable takes values from the domain $\{T, F\}$.

The first systematic algorithm for SAT was the Davis-Putnam procedure in Loveland's form [4] which used static variable ordering; modern implementations use dynamic variable ordering schemes. The version we use in this paper is SATZ [13], or more precisely SATZ-RAND [8] because the original SATZ is deterministic and cannot be used to average performance over several runs. SATZ-RAND allows repeated restarts with slight randomisation of the heuristics, which improves its scalability at the cost of completeness, but we do not use this feature for our experiments. We also use the WSAT local search algorithm [19]. Local search is incomplete but often scales to larger problems. Pioneering SAT algorithms of this type were Selman, Levesque & Mitchell's GSAT [20] and Gu's algorithms [9], and WSAT is one of the fastest available implementations. Finally, we apply the CLS hybrid algorithm [16]. CLS is a Davis-Putnam-style algorithm with a randomised form of backtracking, which is incomplete but has been shown to scale comparably to WSAT on standard SAT benchmarks [17].

CSPs can be encoded as SAT problems and vice-versa in more than one way. We use the simple *direct encoding*. For each variable V with domain D we use SAT variables denoted by V^x $(x \in D)$. Each CSP variable must take a value from its domain, a fact expressed by clauses $V_i^{x_1} \vee \ldots \vee V_i^{x_n}$ $(x_j \in D)$. No CSP variable may take more than one value from its domain, expressed by clauses $\neg V_i^x \vee \neg V_i^y$ $(x, y \in D,\ x < y)$. To express constraints such as $V_i + d < V_j$ we enumerate the forbidden cases: $\neg V_i^x \vee \neg V_j^y$ $(x, y \in D,\ x + d \geq y)$.

problem	v	b	r	k	λ	S	V	C_0	C_1	C_2	C_3
bibd1	7	7	3	3	1	1	833	7028	8106	9674	10080
bibd2	6	10	5	3	2	1	1260	13650	16245	20625	21745
bibd3	7	14	6	3	2	4	2646	40418	48013	72366	79093
bibd4	9	12	4	3	1	1	2700	36756	41940	64728	71865
bibd5	8	14	7	4	3	4	3640	58520	70280	101444	109284

Figure 1: BIBD instances as SAT problems

3 Experimental results

BIBD instances yield rather large SAT problems with $b^2v + v^2b + \lambda bv(v-1)/2$ variables, for example the smallest unsolved instance (22,33,12,8,4) has $70,422$ boolean variables. We therefore limit our attention to the smallest instances listed in [14]. However, these are still interesting problems with few solutions (modulo isomorphism induced by the symmetries). Figure 1 shows the first few instances with their parameters, number of non-isomorphic solutions S, and number of SAT variables V, with the number of clauses at level i denoted by C_i.

Figure 2 shows the results of experiments on these problems using SATZ, WSAT and CLS. For all problems WSAT used the default (SKC) heuristic and noise level 10%, and CLS used noise level 3, parameter settings that give roughly optimal results. The notation bibdx_i denotes problem bibd$x$ with symmetry breaking level i. All figures are means over 20 runs. We were unable to solve in a reasonable time the bibd5 problem with any algorithm or encoding, showing the surprising hardness of these problems. Table entries "—" indicate that the algorithm failed to terminate even once, in a time at least an order of magnitude greater than other times for the same problem. All experiments were performed on a 300 MHz DEC Alphaserver 1000A 5/300 under Unix.

We now attempt to answer the questions raised in Section 1 regarding the effects of symmetry breaking. SATZ is improved by symmetry breaking, and implied constraints help further (though level 2 seems slightly better than level 3). WSAT is worse with symmetry breaking, but with level 3 implied constraints appears to scale better. CLS is much worse with symmetry breaking, and only slightly improved by implied constraints. In summary, the effects of symmetry breaking on single-solution search can be strongly positive or negative, depending on both the search algorithm and the constraints used to enforce it.

Another aim of the experiment was to determine what type of algorithm might best be applied directly to the CSP model of BIBD generation, though the SAT results will not necessarily carry across. SATZ scales poorly on these problems with all levels of symmetry breaking. WSAT is the most robust algorithm tested, solving more instances than either SATZ or CLS. Moreover, in terms of CPU time the best WSAT results (with level 3) are a few times faster than the best CLS results (with level 0), and with levels 1 and 2 WSAT is much faster. However, WSAT has a more mature and optimised implementation than CLS, and in terms of search steps the best CLS results beat the best WSAT results. Thus a promising approach for large instances is CLS-style hybrid search without symmetry breaking, and

problem/ level	SATZ back.	SATZ sec	WSAT flips	WSAT sec	CLS back.	CLS sec
$bibd1_0$	—	—	51412	0.88	43961	6.63
$bibd2_0$	—	—	121259	2.53	69704	15.1
$bibd3_0$	—	—	2028247	65.0	620790	264
$bibd4_0$	—	—	6382697	163	313356	140
$bibd1_1$	57.1	0.90	589010	6.44	4569058	745
$bibd2_1$	14523	32.6	—	—	—	—
$bibd3_1$	—	—	—	—	—	—
$bibd4_1$	—	—	—	—	—	—
$bibd1_2$	16.2	0.82	107279	1.89	347079	61.0
$bibd2_2$	313	1.60	633589	16.0	1967046	575
$bibd3_2$	—	—	11138495	499	—	—
$bibd4_2$	—	—	18387011	644	—	—
$bibd1_3$	459	1.57	113879	1.34	248697	42.3
$bibd2_3$	553	2.22	400273	8.48	1531026	392
$bibd3_3$	—	—	3319906	129	—	—
$bibd4_3$	—	—	3681702	115	—	—

Figure 2: Experimental results

the next step in this work is an implementation. Note that an alternative hybrid approach is that of [8], in which a systematic backtracker is frequently restarted with slightly randomised heuristics. SATZ-RAND uses this method and we applied it to the instance $bibd1_0$ using several parameter settings (the parameters control the restart interval and heuristic randomisation), but it was unable to solve it in a reasonable time.

4 Conclusion

Given a new constraint problem, it is in general unknown whether SAT encoding will make it easier or harder to solve. For example SAT-based planning has out-performed direct approaches [12], though recent direct approaches are more competitive [1]. We have shown that even small BIBD instances yield hard SAT problems, at least under one constraint model. We propose them as new benchmarks that challenge SAT algorithms to emulate the performance of direct approaches.

We used a new binary constraint model for BIBD generation. Binary constraints have been studied in more detail and allow cheaper propagation than non-binary constraints, and we hope that the new model will give good results when used directly rather than via SAT encoding. This will be tested in future work using the most promising approach: a hybrid search algorithm with no symmetry breaking.

Acknowledgements

Thanks to Toby Walsh for suggesting BIBD generation as an interesting problem.

References

[1] R. I. Brafman, H. H. Hoos. To Encode or Not to Encode — I: Linear Planning. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann 1999, pp. 988–993.

[2] C. A. Brown, L. Finkenstein, P. W. Purdom Jr. Backtrack Searching in the Presence of Symmetry. T. Mora (ed.), Applied Algebra, Algebraic Algorithms and Error-Correcting Codes. *Lecture Notes in Computer Science* vol. 357, Springer-Verlag 1988, pp. 99–110.

[3] M. Crawford, M. Ginsberg, E. Luks, A. Roy. Symmetry Breaking Predicates for Search Problems. *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, 1996, pp. 148–159.

[4] M. Davis, G. Logemann, D. Loveland. A Machine Program for Theorem Proving. *Communications of the ACM* vol. 5, 1962, pp. 394–397.

[5] R. Dechter. On the Expressiveness of Networks with Hidden Variables. *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, Mass., 1990, pp. 556–562.

[6] R. Dechter, J. Pearl. Tree Clustering for Constraint Networks. *Artificial Intelligence* vol. 38, 1989, pp. 353–366.

[7] I. P. Gent, B. Smith. Symmetry Breaking During Search in Constraint Programming. *Proceedings of the Fourteenth European Conference on Artificial Intelligence*, 2000, pp. 599–603.

[8] C. Gomes, B. Selman, H, Kautz. Boosting Combinatorial Search Through Randomization. *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference*, AAAI Press / The MIT Press 1998, pp. 431–437.

[9] J. Gu. Efficient Local Search for Very Large-Scale Satisfiability Problems. *SIGART Bulletin* vol. 3, no. 1, January 1992, pp. 8–12.

[10] M. Henz. Constraint Programming — An Oz Perspective. *Tutorial at the Fifth Pacific Rim International Conferences on Artificial Intelligence*, 1998, NUS, Singapore, November 1998.

[11] D. Joslin, A. Roy. Exploiting Symmetry in Lifted CSPs. *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, American Association for Artificial Intelligence 1997, pp 197–203.

[12] H. Kautz, B. Selman. Pushing the Envelope: Planning, Propositional Logic and Stochastic Search. *Proceedings of the Thirteenth National Conference on Artificial Intelligence* vol. 2, MIT Press, 1996, pp. 1194–1201.

[13] C. M. Li, Anbulagan. Look-Ahead Versus Look-Back for Satisfiability Problems. *Proceedings of the Third International Conference on Principles and Practice of Constraint*

Programming, Lecture Notes in Computer Science vol. 1330, Springer-Verlag 1997, pp. 341–355.

[14] R. Mathon, A. Rosa. Tables of Parameters of BIBDs with $r \leq 41$ Including Existence, Enumeration, and Resolvability Results, *Annals of Discrete Mathematics* vol. 26, 1985, pp. 275–308.

[15] P. Meseguer, C. Torras. Exploiting Symmetries Within Constraint Satisfaction Search. *Artificial Intelligence* vol. 129 no. 1–2, 2001, pp. 133–163.

[16] S. D. Prestwich. Stochastic Local Search in Constrained Spaces. *Proceedings of Practical Applications of Constraint Technology and Logic Programming*, Practical Applications Company 2000, pp. 27–39.

[17] S. D. Prestwich. A Hybrid Search Architecture Applied to Hard Random 3-SAT and Low-Autocorrelation Binary Sequences. *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 1894, Springer-Verlag 2000, pp. 337–352.

[18] J.-F. Puget. On the Satisfiability of Symmetrical Constrained Satisfaction Problems. J. Komorowski, Z. W. Ras (eds.), Methodologies for Intelligent Systems, *Proceedings of the International Symposium on Methodologies for Intelligent Systems, Lecture Notes in Computer Science* vol. 689, Springer-Verlag 1993, pp. 350–361.

[19] B. Selman, H. Kautz, B. Cohen. Noise Strategies for Improving Local Search. *Proceedings of the Twelfth National Conference on Artificial Intelligence*, AAAI Press 1994, pp. 337–343.

[20] B. Selman, H. Levesque, D. Mitchell. A New Method for Solving Hard Satisfiability Problems. *Proceedings of the Tenth National Conference on Artificial Intelligence*, MIT Press 1992, pp. 440–446.