First-Solution Search with Symmetry Breaking and Implied Constraints

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

Abstract

Symmetry breaking and implied constraints can speed up both exhaustive search and the search for a single solution. We experiment with both types of constraint, using three search algorithms (backtracking, local and hybrid) to find single solutions for SAT encodings of three combinatorial problems (clique, set cover and balanced incomplete block design generation). Both show strong positive and negative effects, depending on the problem class and algorithm. However, symmetry breaking constraints consistently have a negative effect on local search. This suggests an opposite strategy when applying local search: *maximising* symmetry in the constraint model.

1 Introduction

An important issue in constraint modeling is the addition of constraints to speed up the search. One approach is the use of *implied constraints* that are implied by others in the model. These have been shown to speed up both backtracking [18] and local search [3, 9]. Another use of added constraints is *symmetry breaking*. These constraints are not implied by others, but prevent the generation of equivalent versions of each solution. Many constraint satisfaction problems (CSPs) contain symmetries, for example the N-queens problem has 8: each solution may be rotated through 90 degrees and reflected. 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. Symmetries may be detected in propositional satisfiability (SAT) encodings [4] or lifted (quantified) versions of CSPs [8], and good results are reported. A formal framework for this approach is given in [16]. 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 [1, 5]; and search may be guided towards subspaces with many non-symmetric states [12].

Implied constraints have been shown to speed up the search for the first solution [3, 9] but it is not clear that this will always be the case, even ignoring overheads introduced by the extra constraints. Symmetry breaking constraints are usually applied when exhaustive search is to be performed. Here the benefits are clear, though again the extra overheads sometimes outweigh the benefits. However, it is known that their effect on first-solution search are not necessarily beneficial. For example it is remarked in [5] that the more easily-found solutions may inadvertently be eliminated, and in [7] that symmetry breaking does not always improve performance in the Oz system.

This issue seems worth investigating further, and there are several obvious questions. Are symmetry breaking and implied constraints always beneficial when searching for one solution, modulo overheads? Does the answer depend on whether we use a backtracker or a local search algorithm? Does it depend on the problem? Does it ever make sense to add symmetry or remove implied constraints? Section 2 describes several experiments intended to answer these questions, and the results are discussed in Section 3.

2 Experiments

We experiment with three combinatorial problems: clique construction, set covering and balanced incomplete block design (BIBD) generation. All three can be modeled in a natural way using Boolean variables, but the natural models contain complicated constraints that are expensive to use for propagation. Because of these constraints the problems are also hard to encode as SAT problems, despite their use of Boolean variables. However, they can all be modeled using an alternative approach with integer variables, which eliminates the difficult constraints and also makes them easy to SAT-encode. Unfortunately the latter models introduce extra symmetries, and it is with these symmetries that we shall experiment. The implied constraints we add are implied by the symmetry breaking constraints. We shall ignore the possibility of combining the two models [20]: in this paper we are interested in the effects of adding constraints, not in solving specific problems in the most efficient way.

Our experiments are all performed on SAT encodings of the integer variable models, allowing a rapid comparison of several commonly available algorithms. CSPs can be encoded as SAT problems (and vice-versa) in more than one way, and we use the *direct encoding* [19]. To reduce bias caused by choosing a specific class of search algorithm, we apply three distinct SAT solvers:

- The SATZ backtracker [10], or more precisely SATZ-RAND [6] 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 this feature is not used in our experiments. However, its resolution-based preprocessor is used, and accounts for a significant portion of the execution times.
- The WSAT local search algorithm with the default (SKC) heuristic [17]. Local search is incomplete but often scales to larger problems. WSAT has a noise parameter expressed as a percentage, and for each class of problem we choose a fixed noise level that gives roughly optimal results overall.
- The CLS hybrid algorithm [13]. This is a backtracker with the local search characteristics of incompleteness and scalability. It also has a noise parameter expressed as a positive integer, and again we tune the parameter to each class of problem.

In the tables of results below "back." denotes backtracks, "flips" denotes changes to variable truth values by WSAT, "sec." denotes seconds of CPU time, and "—" denotes failure to find a solution in a time roughly two orders of magnitude greater than typically taken by another algorithms (except for the BIBD problems where it denotes one order of magnitude). All figures are means over 20 runs and times are obtained on a 300 MHz DEC Alphaserver 1000A 5/300 under Unix. The results are discussed in Section 3.

2.1 Clique construction

The maximum clique problem was one of the first problems shown to be NP-complete, and theoretical results indicate that even near-optimal solutions are hard to find. Its applications include computer vision, coding theory, tiling, fault diagnosis and the analysis of biological and archaeological data. The problem is defined as follows. A graph $G = (V, E)$ consists of a set V of vertices and a set E of edges between vertices. Two vertices connected by an edge are *adjacent*. A *clique* is a subset of V whose vertices are pairwise adjacent. A *maximum clique* is a clique of maximum cardinality.

seed	k	variables	clauses$_0$	clauses$_1$	clauses$_2$	clauses$_3$
1	7	350	27475	34075	53200	54880
2	8	400	36632	44157	70932	73606
3	8	400	36128	43653	70428	73102
4	7	350	27643	34243	53368	55048
5	8	400	35568	43093	69868	72542

Figure 1: Clique construction as SAT (with models 0–3)

The problem of constructing a clique of size k can be modeled as a CSP, the most direct way being as follows. Define Boolean variables $v_1, \ldots, v_n$, one for each vertex in G, the truth value of v_i denoting whether vertex i is in the clique. The constraints are $\neg v_i \vee \neg v_j$ ($i < j$, i and j are non-adjacent vertices) and

$$|\{v_i \in G : \ v_i = \text{true}\}| = k$$

However, the latter constraint is expensive to use. To avoid it we can instead use the following model. Define integer variables $v_1, \ldots, v_k$ each with domain $\{1, \ldots, n\}$. An assignment $v_i = j$ denotes that vertex j is in the clique. The constraints are as follows. No two non-adjacent vertices may be in the clique:

$$v_i \neq j \vee v_{i'} \neq j' \quad (i < i',\ j \text{ and } j' \text{ are non-adjacent vertices})$$

No vertex is adjacent to itself, so these constraints also ensure that no two variables can represent the same vertex. This model is highly symmetrical: a clique of size k has $k!$ representations, one for each permutation of values among the variables. These symmetries can be removed by adding constraints to order the values, which can be done in more than one way:

$$\begin{array}{rrl} 1. & v_i < v_{i+1} & \\ 2. & v_i < v_j & (i < j) \\ 3. & v_i + j - i - 1 < v_j & (i < j) \end{array}$$

We shall refer to these cases as models 1–3, and to the problem without symmetry breaking constraints as model 0. Model 2 contains constraints implied by those in model 1, and model 3 contains constraints implied by those in model 2.

Using J. Culberson's graph generator [1] we generate 5 random (IID) graphs with 50 vertices, density 0.5 and random seeds 1–5. The SAT encoding needs a value for k, so we apply a clique algorithm to each graph to find the maximum or near-maximum clique size. The problem parameters are shown in Figure 1 and the results in Figure 2. The notation cliquei_j denotes the problem with seed i and model j. WSAT uses a noise level of 50% and CLS 2. On these problems symmetry breaking degrades WSAT performance, but adding implied constraints helps slightly. CLS behaves similarly but is slower. The SATZ results are qualitatively different: symmetry breaking constraints improve performance, but implied constraints drastically degrade it.

2.2 Set covering

The set covering problem is also NP-hard and has many applications, including delivery, routing, scheduling and location problems, switching theory, line balancing and Boolean logic minimisation. Given a ground set of objects V and a set E of subsets of V, a *cover* for V is a subset C of E such that every member of V also occurs in some member of C. The (unicost) problem is to find a cover of minimum cardinality.

[1] http://web.cs.ualberta.ca/~joe/Coloring/index.html

	SATZ		WSAT		CLS	
problem	back.	sec.	flips	sec.	back.	sec.
clique1_0	360	1.48	236	0.046	268	0.338
clique2_0	11719	24.5	1032	0.13	2678	1.92
clique3_0	4332	9.25	384	0.063	626	0.656
clique4_0	641	1.89	294	0.047	637	0.525
clique5_0	5570	11.2	303	0.055	661	0.656
clique1_1	22	1.55	2146	0.21	9130	7.13
clique2_1	139	3.80	129127	12.3	652375	575
clique3_1	1.4	1.12	13428	1.30	118309	103
clique4_1	3.25	1.10	7317	0.66	21466	16.5
clique5_1	27	2.05	13150	1.26	43704	37.6
clique1_2	—	—	701	0.12	1349	1.88
clique2_2	—	—	14450	2.50	47620	64.5
clique3_2	—	—	1865	0.37	9459	13.1
clique4_2	—	—	1533	0.22	3263	4.02
clique5_2	—	—	2783	0.52	7694	10.8
clique1_3	—	—	667	0.12	1036	1.56
clique2_3	—	—	11911	2.16	50973	71.2
clique3_3	—	—	2149	0.44	14044	19.8
clique4_3	—	—	1078	0.18	3119	4.00
clique5_3	—	—	1593	0.35	7107	20.2

Figure 2: Clique construction results

We can model the problem of finding a cover of size k with a Boolean variable v_i for each subset $e_i \in E$, denoting the membership or non-membership of e_i in the cover C. We then require constraints to ensure that each member of V is covered by some member of C: for each $x \in V$ where $x \in e_{i_1} \cap \ldots \cap e_{i_m}$ add a constraint $v_{i_1} \vee \ldots \vee v_{i_m}$. We also require a constraint to force the size of the cover to be k:

$$|\{v_i \in C : \; v_i = \text{true}\}| = k$$

Again this is an expensive constraint, so we use instead a model with k integer variables v_i, each with domain $\{1, \ldots, |E|\}$. An assignment $v_i = j$ denotes that subset e_j is used in the cover. We need constraints to ensure that each element of V is covered by at least one element of E. For each $x \in V$ where $x \in e_{i_1} \cap \ldots \cap e_{i_m}$:

$$(v_1 = i_1 \;\vee\; \ldots \;\vee\; v_k = i_1) \;\vee\; \ldots \;\vee\; (v_1 = i_m \;\vee\; \ldots \;\vee\; v_k = i_m)$$

We add constraints $v_i \neq v_j$ $(1 \leq i < j \leq k)$ to prevent the same subset from being used more than once, which would give a cover of size less than k. This is not strictly necessary for set covering because a smaller cover is at least as useful as a larger one, but we add the constraints so that symmetry breaking and implied constraints can be used.

We generate five random problems each with a ground set V of 20 elements, and a set E of 40 subsets of V each containing 3 randomly-chosen elements. The cover size k of each problem is increased until it iss solvable by at least one algorithm, and in each case this turns out to be $k = 7$. The covering constraints yield very long clauses, so we apply a simple transformation to break these up into clauses of no more than (an arbitrarily-chosen) 30 literals. Figure 3 shows the problem parameters and Figure 4 the results. The notation $\text{cover}i_j$ denotes the problem with seed i and model j. WSAT uses noise level 10% and CLS 6. The results are very different to those for the clique problems. SATZ performs well with implied constraints but poorly without, with and without symmetry breaking. WSAT and

seed	k	variables	clauses_0	clauses_1	clauses_2	clauses_3
1	7	324	6371	11051	22751	24116
2	7	332	6379	11059	22759	24124
3	7	322	6369	11049	22749	24114
4	7	332	6379	11059	22759	24124
5	7	328	6375	11055	22755	24120

Figure 3: Set covering as SAT (with models 0–3)

	SATZ		WSAT		CLS	
problem	back.	sec.	flips	sec.	back.	sec.
cover1_0	—	—	59949	1.29	41296	3.55
cover2_0	—	—	131499	2.81	29475	2.49
cover3_0	—	—	29244	0.624	18254	1.63
cover4_0	—	—	64869	1.38	35438	2.92
cover5_0	—	—	33100	0.708	20071	1.72
cover1_1	—	—	77493	2.33	33703	4.64
cover2_1	—	—	142498	4.13	26957	3.57
cover3_1	—	—	27581	0.822	15570	2.26
cover4_1	—	—	48828	1.48	31084	4.09
cover5_1	—	—	35977	1.16	16001	2.17
cover1_2	895	6.73	59939	3.28	188328	46.0
cover2_2	495	5.26	87025	5.32	107205	23.7
cover3_2	223	2.93	50991	2.82	69562	17.5
cover4_2	579	5.44	29598	1.72	98698	21.2
cover5_2	908	7.22	27615	1.51	20981	4.99
cover1_3	822	6.43	57771	3.45	120490	32.3
cover2_3	449	4.83	105213	6.54	60257	14.3
cover3_3	201	2.77	19861	1.17	117322	31.8
cover4_3	366	4.38	43942	2.66	84472	19.2
cover5_3	531	5.38	19630	1.15	23838	5.98

Figure 4: Set covering results

CLS perform well in all cases but are slowed down by symmetry breaking constraints, and implied constraints significantly increase the number of backtracks taken by CLS.

2.3 Balanced incomplete block design generation

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. A BIBD is specified by five parameters (v, b, r, k, λ) that describe its incidence matrix, a $v \times b$ binary matrix with exactly r ones per row, k ones per column, and scalar product λ between any two distinct rows. 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) [11].

The most direct CSP model for BIBD, as described in [12], represents each matrix entry by a Boolean variable. There are 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, and the model is modified in [12] to avoid

problem	v	b	r	k	λ	S	variables	clauses_0	clauses_1	clauses_2	clauses_3
1	7	7	3	3	1	1	833	7028	8106	9674	10080
2	6	10	5	3	2	1	1260	13650	16245	20625	21745
3	7	14	6	3	2	4	2646	40418	48013	72366	79093
4	9	12	4	3	1	1	2700	36756	41940	64728	71865
5	8	14	7	4	3	4	3640	58520	70280	101444	109284

Figure 5: BIBD generation as SAT (with models 0–3)

the most expensive constraints (iii), using $\lambda v(v-1)/2$ integer variables to denote the column number of each shared one between each pair of rows. This still leaves the fairly expensive constraints (i) and (ii), but they can be eliminated by further use of integer variables, leaving only binary constraints. For space reasons we omit details of the binary model which is fully described in [14]; the key is to define variables denoting the positions of the ones *and zeroes* in each row and column.

One source of intractability in BIBD generation is the large number of symmetries: given any solution, any two rows or columns may be exchanged to obtain another solution. We shall ignore these intrinsic symmetries and consider only the extra symmetries introduced by our new model. These are of the same kind as those in our clique and covering models, and we apply the same symmetry breaking and implied constraints. We obtain rather large SAT problems with $b^2v + v^2b + \lambda bv(v-1)/2$ variables (for example instance (22,33,12,8,4) has $70,422$ variables). We therefore limit our attention to the smallest instances listed in [11], but these are still interesting problems with few solutions (modulo isomorphism induced by the symmetries).

Figure 5 shows the first five instances with their parameters, number of non-isomorphic solutions S, and number of SAT variables and clauses. Figure 6 shows results for the four smallest problems — none of the algorithms are able to solve problem 5 in a comparable time. WSAT uses noise level 10% and CLS 3. The notation $\text{bibd}i_j$ denotes problem i with model j. The results are different from those for cliques and covers. Symmetry breaking constraints have a strong negative effect on WSAT and CLS, but a strong positive effect on SATZ. Implied constraints help all three algorithms.

3 Discussion

The results may be summarised as follows. Firstly, the effect of symmetry breaking constraints on WSAT and CLS are weakly or strongly negative, but on SATZ strongly positive (or unknown in the case of set covering). Secondly, the effect of implied constraints on WSAT are strongly positive or neutral, on CLS weakly positive or negative, and on SATZ strongly positive or negative. The results are consistent within each problem type, showing that the effects are not random but depend on properties of the SAT problems. Many more experiments are needed to draw firm conclusions, but these results show that adding either type of constraint can have a very unpredictable effect on first-solution search.

One trend emerges, though it may be contradicted by future results: symmetry breaking constraints slow down local and (one form of) hybrid search. This suggests the following novel approach to solving large problems: choose a model that *maximises* the number of solution symmetries, then apply local or hybrid search. We might call such models *supersymmetric* to indicate that they contain symmetries in addition to those occurring naturally in the problem. Most of the shortest execution times in this paper were achieved by applying local search to such models. We have also obtained good results by applying hybrid search to supersymmetric models for Golomb rulers and maximum cliques [15], and low-autocorrelation binary sequences [13] (though the term *supersymmetric* was not used and comparisons were not made with non-symmetric models).

We propose the following interpretation. Symmetry breaking reduces the density of solutions in the search space so (other factors being equal) it should have a negative effect on the search for the first

problem	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 6: BIBD generation results

solution. (This is not incompatible with a positive effect on *exhaustive* search.) In contrast, implied constraints have no effect on solution density but prevent some assignments from being explored, so (other factors being equal) they should have a positive effect. The WSAT results largely conform to these expectations. With SATZ and CLS the contrary effects might be caused by interactions between the new constraints and dynamic variable ordering heuristics. In future work we will test this interpretation by comparing versions of the algorithms with random variable orderings.

References

[1] R. Backofen and S. Will. Excluding Symmetries in Constraint-Based Search. *Proceedings of the Fifth International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 1713, Springer-Verlag 1999, pp. 73–87.

[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] B. Cha and K. Iwama. Adding New Clauses for Faster Local Search. *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, American Association for Artificial Intelligence 1996, pp. 332–337.

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

[9] K. Kask, R. Dechter. GSAT and Local Consistency. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann 1995, pp. 616–622.

[10] 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.

[11] 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.

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

[13] 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.

[14] S. D. Prestwich. Balanced Incomplete Block Design as Satisfiability. *Proceedings of the Twelfth Irish Conference on Artificial Intelligence and Cognitive Science*, Maynooth 2001 (to appear).

[15] S. D. Prestwich. Trading Completeness for Scalability: Hybrid Search for Cliques and Rulers. *Proceedings of the Third International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, Ashford, Kent, England, pp. 159–174.

[16] 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.

[17] 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.

[18] B. Smith, K. Stergiou, T. Walsh. Modelling the Golomb Ruler Problem. Research Report 1999.12, University of Leeds, England, June 1999. (Presented at the IJCAI'99 Workshop on Non-binary Constraints.)

[19] T. Walsh. SAT v CSP. *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 1894, Singapore, 2000, pp. 441–456.

[20] T. Walsh. Permutation Problems and Channelling Constraints. *Proceedings of the IJCAI-2001 workshop on Modelling and Solving Problems with Constraints,* 2001.