Breaking Instance-Independent Symmetries in Exact Graph Coloring

Arathi Ramani[*] Fadi A. Aloul[†] Igor L. Markov[*] and Karem A. Sakallah[*]
[*] Department of EECS, University of Michigan, Ann Arbor, USA
{ramania, faloul, imarkov, karem}@umich.edu
[†] School of Computer Engineering, American University in Dubai, UAE

Abstract

Code optimization and high level synthesis can be posed as constraint satisfaction and optimization problems, such as graph coloring used in register allocation. Naturally-occurring instances of such problems are often small and can be solved optimally. A recent wave of improvements in algorithms for Boolean satisfiability (SAT) and 0-1 ILP suggests generic problem-reduction methods, rather than problem-specific heuristics, because (1) heuristics are easily upset by new constraints, (2) heuristics tend to ignore structure, and (3) many relevant problems are provably inapproximable. The NP-spec project offers a language to specify NP-problems and automatic reductions to SAT.

Problem reductions often lead to highly symmetric SAT instances, and symmetries are known to slow down SAT solvers. In this work, we compare several avenues for symmetry-breaking, in particular when certain kinds of symmetry are present in all generated instances. Our surprising conclusion is that instance-independent symmetries should often be processed together with instance-specific symmetries rather than earlier, at the specification level.

1 Introduction

Many techniques for code optimization and high-level synthesis operate with relatively few objects at a time. For example, *graph coloring* used for register allocation [7] is limited by small numbers of registers in embedded processors as well as by the number of local variables and virtual registers. Optimal solutions may be desirable in commercial and defense applications for competitive reasons, and can often be found. Several useful combinatorial problems are used in this context, e.g., *maximum independent set, graph-coloring* and *vertex cover*, but individual applications often imply additional constraints and non-trivial optimization functions. These extensions may upset heuristics for standard problems. Heuristics, particularly those based on local search, often fail to use structure in problem instances [17] and are inefficient when used with problem reductions. In contrast, exact solvers based on branch-and-bound and back-tracking tend to adapt to new constraints and can be applied through problem reduction. There is a growing literature on handling structure in optimal solvers [1, 4], and our work falls into this category as well.

1530-1591/04 $20.00 (c) 2004 IEEE

The NP-spec project offers a framework for formulating a wide range of combinatorial problems [5] and automatically reducing their instances to instances of *Boolean satisfiability*. This approach is attractive because it circumvents problem-specific solvers and leverages recent breakthroughs in Boolean satisfiability [15]. However, this approach remains unexplored in practice, possibly because the efficiency of problem-solving may be reduced when domain-specific structure is lost during problem reductions. This potential drawback is addressed by recent work on the detection of structure, particularly symmetry, in SAT and *0-1 Integer Linear Programming* (ILP) instances in order to accelerate exact solvers [1, 4]. Symmetry-detection via the *graph automorphism* problem [8, 1] may take time, but adding simple symmetry-breaking predicates as new constraints significantly speeds up exact SAT solvers [1]. This work can be viewed as a case study of symmetry-breaking in problem reductions, as we focus on *graph coloring* and its variants that can be reduced to Boolean satisfiability and 0-1 ILP. Our main goals are to (i) accelerate optimal solving of graph coloring instances, and (ii) compare different strategies for breaking instance-independent symmetries. There are two distinct sources of symmetries in graph-coloring instances: colors can be arbitrarily permuted (instance-independent symmetries), and some graphs may be invariant under some permutations (instance-dependent symmetries). Given that there may be many instance-specific symmetries, one may process all symmetries at once using techniques from [1, 3]. Alternatively, one may add symmetry-breaking predicates for instance-independent symmetries early, hoping to speed-up the processing of remaining symmetries. This type of symmetry-breaking has not been discussed in earlier work [1, 3], and in this paper we study its utility for the graph coloring problem.

A surprising empirical observation is that, among the possibilities we considered, the best one was to ignore the generic nature of instance-independent symmetries. The most plausible explanation is that instance-independent symmetry-breaking predicates that we tried are too complicated as constraints and do not facilitate additional learning in the solver. In contrast, when all symmetries are detected at the instance level, symmetry-breaking predicates auto-

matically generated by known techniques [3, 4] are simpler.

The remaining part of the paper is structured as follows. Section 2 covers background on graph coloring, SAT and 0-1 ILP, as well as previous work on symmetry-breaking. Instance-independent symmetry-breaking predicates are discussed in Section 3. Section 4 presents our empirical results and Section 5 concludes the paper.

2 Background

Given an undirected graph $G(V,E)$, a **vertex coloring** of the graph is an assignment of a label (color) to each node such that the labels on adjacent nodes are different. A minimum coloring uses the smallest possible number of colors (*chromatic number*). The *decision version* of graph coloring ($K-$coloring) asks whether vertices in a graph can be colored using $\leq K$ colors for a given K.

A *clique* is a set of mutually adjacent vertices. Graph coloring is related to the **maximum clique** problem which seeks a clique of maximal size. Namely, the max-clique size is a lower bound on the chromatic number of the graph. Both problems are NP-hard for general graphs [12] and even finding near-optimal solutions with good approximation guarantees is NP-hard [11]. The inapproximability of graph coloring suggests that it may be more difficult to solve heuristically than, say, the Traveling Salesman Problems for which Polynomial-Time Approximation Schemes (PTAS) are known for Euclidean and Manhattan graphs. For this and a number of other reasons, we study optimal graph coloring, and many application-derived instances are solvable in reasonable time. Several applications are outlined below (for more details see [16]).

TIME-TABLING AND SCHEDULING problems often disallow performing certain tasks in parallel due to dependencies between computations. Scheduling with minimal hardware can often be formalized as a graph coloring problem.

REGISTER ALLOCATION: The register allocation problem seeks to assign variables to a limited number of hardware registers during program execution. Two variables cannot be assigned to the same register if they are "live" at the same time. Assigning more variables leads to faster execution as fewer variables need to be fetched from memory. To formalize this, one creates a graph where nodes represent variables and edges represent conflicts between variables. A coloring maps to a conflict-free assignment, and if the number of registers exceeds the chromatic number, a conflict-free register assignment exists [7].

Applications of graph coloring in circuit design and layout have included printed circuit board testing [12], circuit clustering, scheduling for signal flow graphs, and many others. Benchmarks from these applications are not publicly available, and therefore do not appear in this paper. However, all the symmetry-breaking techniques described here extend to instances from any application. The benchmarks we use here do include register allocation, $n-$queens, and several other applications discussed in more detail in Section 4. The literature on graph coloring includes generic algorithms and specialized algorithms for a particular application, such as Chaitin's register allocation algorithm [7]. Online surveys [16, 9] contain more details and examples.

Published algorithms for finding optimal graph colorings are mainly based on implicit enumeration. A recent optimal coloring algorithm expresses graph coloring using ILP-like constraints [14]. It relies on an auxiliary independent set formulation, where each independent set in a graph is represented by a variable. There can be prohibitively many variables but in practical cases this number may be reduced by *column generation*, a method that first tries to solve a linear relaxation using a subset of variables and then adds more where needed. This approach inherently breaks problem symmetries, and thus rules out the use of symmetry-breaking predicates (SBPs) as a way to speed up the search process. Our ILP construction differs considerably from the one described above, since it does not rely on an independent set formulation, but assigns colors to individual vertices by using indicator variables. The construction is described in more detail later in this section.

One can solve the decision version of graph coloring by reducing it to Boolean satisfiability, and the optimization version to 0-1 Integer Linear Programming (ILP). The Boolean satisfiability (SAT) problem involves finding an assignment to a set of *0-1 variables* that satisfies a set of constraints, called *clauses*, expressed in conjunctive normal form (CNF). A CNF formula on n binary variables, $x_1,\ldots,x_n$ consists of a conjunction of clauses,$\omega_1,\ldots,\omega_m$. A clause consists of a disjunction of *k literals*. A literal l is an occurrence of a Boolean variable or its complement. In addition to CNF constraints, a 0-1 ILP problem can include *pseudo-Boolean* (PB) constraints, which are linear inequalities with integer coefficients and can be normalized [2] to: $a_1x_1 + a_2x_2 + \ldots\ a_nx_n \leq b$ where $a_i, b \in Z^+$ and x_i are literals of Boolean variables.[1] In some cases a single PB constraint can replace an exponential number of CNF clauses [2]. Subject to given constraints, one may request the minimization of an objective function which must be a linear combination of x_j variables.

Reducing Graph Coloring to 0-1 ILP. We express an instance of the minimal graph coloring problem as a 0-1 ILP optimization problem, consisting of (i) CNF and PB constraints that model the graph (ii) An objective function to minimize the number of colors used.

Consider a graph $G(V,E)$. Let $n = |V|$ be the number of vertices in G. An instance of the $K-$coloring problem for G is formulated as follows.

- For each vertex v_i, indicator-variables $x_{i,1},\ldots,x_{i,K}$, denote possible color assignments to v_i. Variable $x_{i,j}$ indicates that vertex v_i is using color j.

[1] Using the relations $(Ax \geq b) \Leftrightarrow (-Ax \leq -b)$ and $\overline{x_i} = (1 - x_i)$, any arbitrary PB constraint can be expressed in normalized form with only positive coefficients.

- For each vertex v_i, a PB constraint of the form $x_{i,1} + \ldots + x_{i,K} = 1$ ensures that each vertex is colored with *exactly* one color.

- Each edge e_i in E connects two vertices: (v_m, v_n). For each edge e_i, we define CNF constraints of the form $\bigwedge_{j=1}^{K} (\overline{x_{m,j}} \vee \overline{x_{n,j}})$ to specify that no two vertices connected by an edge can be given the same color.

- To track unused colors, we define k new variables, $y_1, \ldots, y_k$. Variable y_i is true *if and only if* at least one vertex uses color i. This is expressed using the following CNF constraints: $\bigwedge_{j=1}^{K} (y_j \Leftrightarrow (\bigvee_{i=1}^{n} x_{i,j}))$.

- The optimization objective is to minimize the number of y_i variables set to true.

The total number of variables in the formula is $nK + K$. An interesting observation is that instance symmetries in graph coloring survive the above reduction to 0-1 ILP. Therefore we can apply known techniques for symmetry-detection in 0-1 ILP.

Detecting and breaking symmetries in 0-1 ILPs. Recent work [8, 1] showed that breaking symmetries in CNF formulas effectively prunes the search space and can lead to significant runtime speedups. The main idea is to detect symmetries in the CNF formula using graph automorphism. The formula is expressed as an undirected graph such that the symmetry group of the graph is isomorphic to the symmetry group of the CNF formula. Symmetries induce equivalence relations on the set of truth assignments of the CNF formula. All assignments in an equivalence class result in the same truth value for the formula (satisfying or not). Therefore, it is only necessary to consider at least one assignment from each such class. Both [8, 1] propose adding symmetry-breaking predicates (SBPs) that choose lexicographically smallest assignments (lex-leaders) from each equivalence class. We will refer to such SBPs as instance-dependent SBPs, since the symmetries are first detected and then broken. Aloul et. al. [3] describe efficient tautology-free SBP construction, whose size is linear in the number of problem variables.

In [4], symmetry detection and breaking was extended to optimization problems that include both CNF and PB constraints, and an objective function. As before, symmetries are detected by reduction to graph automorphism. A PB formula for an optimization problem is represented by an undirected graph. Graph symmetries are detected using graph automorphism tools such as Nauty [13] or Saucy [10]. The efficient symmetry-breaking predicates described in [3] are appended to the formula as CNF clauses. The empirical results in [4] show that the addition of symmetry-breaking predicates to PB formulas results in considerable search speedups for the specialized 0-1 ILP solver PBS [2].

3 Instance-Independent SBPs

In the context of problem reductions from Section 2, we found that adding instance-dependent SBPs improves performance on many DIMACS graph coloring benchmarks. Empirical results for these experiments are reported in Section 4. The question addressed here is whether *instance-independent* SBPs, added during the reduction can provide even greater speedups, possibly by accelerating the detection of instance-dependent symmetries. To answer this question, we propose four provably-correct SBP constructions of varying relative strength, sophistication and completeness. Each of them is implemented and empirical results are reported in Section 4.

We use the following notation. Consider an instance of the $K-$coloring problem, which asks whether a graph $G(V,E)$ can be colored using $\leq K$ colors and further minimizes the number of colors. Assume the colors are numbered $1 \ldots K$. We denote a valid color assignment by $(n_1, n_2, \ldots, n_K)$, where n_i is the number of vertices colored with color i, and $|V| = \sum_{i=1}^{K} n_i$.

Null-Color Elimination (NU). Consider a $K-$coloring problem with colors $1 \ldots K$ for a graph $G(V,E)$. Assume that G can be minimally colored using $K-1$ colors. Consider an optimal solution where color i is not used: $(n_1, n_2, ..n_{i-1}, 0, n_{i+1}, \ldots, n_K)$. This assignment is equivalent to another assignment, $(n'_1, n'_2, ..n'_{j-1}, 0, n'_{j+1} ... n'_K)$, where $i \neq j$ and $n'_i = n_j$. For example, the assignment $(1,0,2,3)$ is equivalent to $(1,3,2,0)$, $(0,1,2,3)$, $(1,2,0,3)$. This is due to the existence of *null* colors, which create symmetries in an instance of $K-$coloring because any color can be swapped with a null color. We propose a construction that enforces an ordering on null colors: null colors may appear only at the *end* of a color assignment, after all non-null colors. In the example above, only one of the four symmetric assignments $(1,3,2,0)$ would be allowed.

Assume that under the original formulation, an optimal solution for graph $G(V,E)$ uses m colors, and with null-color elimination, there is a *different* optimal solution that uses m' colors, where $m \neq m'$. The only colors used in this solution are $1 \ldots m'$, since null colors cannot occur before non-null colors. Since our construction adds SBPs without changing the original constraints, any legal solution that satisfies the SBPs will satisfy all constraints in the original formulation. The solution to the original satisfies all constraints in the new formulation except the SBPs. If $m < m'$, we can re-order the solution so that all null colors are placed last. This will satisfy all SBPs and use m colors, where $m < m'$, violating the assumption that the m'-color solution was optimal. If $m' < m$, we already have a solution that satisfies all the original constraints and uses fewer colors, which again violates assumptions of optimality.

Cardinality-Based Color Ordering (CA). Null-color elimination is useful *only* in cases where null colors exist. For a $K-$coloring problem where all colors are needed, the construction breaks no symmetries. Even when null col-

ors exist, several symmetries go undetected. In the example used above, null-color elimination permits both $(1,2,3,0)$ and $(1,3,2,0)$, and also $(3,2,1,0)$, which are symmetric to each other. A solution to an instance of $K-$coloring is a partition of the vertices of the graph into independent sets. All the vertices in an independent set are given the same color. The previous construction places restrictions on null colors, but none on the ordering of non-null colors. A stronger construction would distinguish between the independent sets themselves. We propose an alternate construction, which assigns colors based on the cardinality of independent sets. This subsumes null-color elimination (null colors can be viewed as coloring sets of cardinality 0). The cardinality rule is implemented as follows: the largest independent set is assigned the color 1, the second-largest the color 2, etc. In the example above, *only* the assignment $(3,2,1,0)$ is valid.

Assume an optimal solution under this construction uses $m < K$ colors: $(n_1, n_2, \ldots, n_m)$, where $(n_1 \geq n_2 \ldots \geq n_m)$. Colors $> m$ are not used on any vertex, Assume there exists an optimal solution to the original formulation that uses m' colors: $(n'_1, n'_2, \ldots, n'_{m'})$, (where n'_1, etc. are not arranged in descending order). Without loss of generality, assume that $m' < m$. We can sort the numbers $n'_1, \ldots, n'_{m'}$ and reassign colors in descending order. We would have a solution with m' colors satisfying cardinality constraints. However, $m' < m$, which is not possible if the $m-$color solution was optimal. A similar argument applies when $m < m'$.

Lowest Index Color Ordering (LI). Cardinality-based ordering also does not completely break symmetries for the case where different independent sets have the same cardinality. Consider a graph G where $V = \{v_1, \ldots, v_8\}$, and an optimal solution, satisfying cardinality-based ordering, that partitions V into 4 independent sets: $S_1 = \{v_4, v_6, v_7\}$, $S_2 = \{v_1, v_5\}$, $S_3 = \{v_3, v_8\}$, $S_4 = \{v_2\}$. A solution that assigns colors 2 and 3 to S_2 and S_3 is symmetric to one that assigns colors 2 and 3 to S_3 and S_2. Both are legal under cardinality-based ordering. To improve upon cardinality-based ordering, we propose a set of predicates to enforce the *lowest-index ordering*. This requires that the lowest vertex index colored with color i be greater than the lowest vertex index colored with $i+1$. Lowest-index ordering is complete and breaks all instance-independent symmetries. Independent sets in a partition are disjoint and each set has a *unique* lowest-index vertex. An assignment of sets to colors based on smallest vertex index is unique. In the above example, the only permissible assignment is: color 1 to S_1, 2 to S_3, 3 to S_4, and 4 to S_2. Since the LI ordering completely breaks symmetries between independent sets, it subsumes earlier constructions. The proofs of correctness and optimality outlined above extend to this construction as well.

Selective Coloring (SC). In addition to the preceding constructions, we also propose a simple "heuristic" construction to break as many symmetries between vertices as possible while adding very few additional constraints. To impact as many vertices as possible, we find the vertex v_l with the largest degree of all vertices in the graph. We then color v_l with color 1. This is achieved by simply adding the unary clause $x_{l,1}$. We search v_l's neighbors to find the vertex $v_{l'}$ with the highest degree out of all vertices adjacent to v_l. We color $v_{l'}$ with color 2, by adding the unary clause $x_{l',2}$. This construction has the effect of simplifying color assignment for all vertices adjacent to v_l and $v_{l'}$. No vertex adjacent to v_l can be colored color 1, and no vertex adjacent to $v_{l'}$ can be colored color 2. Moreover, all vertices in an independent set with v_l ($v_{l'}$) *must* be colored color 1 (color 2). If v_l and $v_{l'}$ have sufficiently large degree, this construction can restrict many vertex assignments. We refer to this construction as *selective coloring*.

The extent to which selective coloring breaks symmetries is instance-dependent. It fails to completely break symmetries for almost all graphs. However, it is a simple construction, adding just two constraints as unary clauses. These are easily resolved in pre-processing by most SAT solvers, so any symmetry-breaking achieved by this construction has virtually no overhead.

Instance	#V	#E	K
anna	138	986	11
david	87	812	11
DSJC125.1	125	1472	5
DSJC125.9	125	13922	>20
games120	120	1276	9
huck	74	602	11
jean	80	508	10
miles250	128	774	8
mulsol.i.2	188	3885	>20
mulsol.i.4	185	3946	>20
myciel3	11	20	4
myciel4	23	71	5
myciel5	47	236	6
queen5_5	25	320	5
queen6_6	36	580	7
queen7_7	49	952	7
queen8_12	96	2736	12
zeroin.i.1	211	4100	>20
zeroin.i.2	211	3541	>20
zeroin.i.3	206	3540	>20

Table 1: DIMACS graph coloring benchmarks

4 Empirical Results

Here we discuss our experiments and present empirical results on 20 medium-sized instances from the DIMACS graph coloring benchmark suite. The benchmarks include random graphs (`DSJ`), "book" graphs, where edges represent interaction between characters in a book (`anna, david, huck, jean`), mileage graphs representing distances between cities (`miles`), college football game graphs (`games`), $n-$queens graphs (`queen`), register allocation (`mulsol, zeroin`), and triangle-free graphs based on the Mycielski transformation (`myciel`). Table 1 gives the name, size (number of vertices and edges) and the chromatic number for each benchmark. We use a maximum value of $K = 20$ for $K-$coloring, therefore for benchmarks with chromatic number > 20, we do not find the exact value.

To solve instances of 0-1 ILP, we used the academic 0-1 ILP solvers PBS [2] and Galena [6], and the commercial

SBP Type	CNF Stats			Sym. Stats (SAUCY)		
	#V	#CL	# PB	#S	#G	Time
no SBPs	437K	777505	3193	1.1e+168	994	185
NU	437K	777885	3193	5.0e+149	614	49
CA	437K	777505	3630	5.0e+149	614	49
LI	870K	4019980	3193	2.0e+01	0	84
SC	437K	777545	3193	3.0e+164	941	167
NU+SC	437K	777925	3193	5.0e+148	597	47

Table 2: CNF formula sizes, symmetry detection results and runtimes, totaled for 20 benchmarks from Table 1, with $K = 20$. NU = null-color elimination; CA = cardinality-based; LI = lowest-index; SC = selective coloring. For the LI SBPs, one instance of the "do-nothing" symmetry is counted in each case, giving a total of 20 symmetries and 0 generators.

ILP solver CPLEX version 7.0. PBS is implemented in C++ and compiled using g++. Galena binaries were provided by the authors. PBS was run using the VSIDS decision heuristic option [15]. Galena was run using default options: linear search with CARD (cardinality reduction) learning. Experiments with PBS and CPLEX run on Sun-Blade-1000 workstations with 2GB RAM, CPUs clocked at 750MHz and the Solaris operating system. Galena binaries run on Linux-based Intel Xeon workstations with 1GB RAM and CPUs clocked at 2GHz. Time-out limits for all solvers are set at 1000 seconds. We use the symmetry-breaking flow from [4] to detect and break symmetries in our original ILP formulation from Section 2. This flow uses the tool Shatter [3], which uses the SAUCY [10] graph automorphism program and the efficient SBP construction from [3]. We also check for unbroken symmetries in formulations produced by each of the instance-independent constructions described in Section 3. Table 2 shows symmetry detection results and runtimes. The numbers reported in the table are sums of individual results for all 20 benchmarks used. The first column in the table indicates the type of construction: we use **no SBPs** for the basic formulation, **NU** for null-color elimination, **CA** for cardinality-based ordering, **LI** for lowest-index ordering, and **SC** for selective coloring (the row shows NU and SC in combination). The next three columns show the number of variables, CNF clauses, and PB constraints in the problems. The last three columns show the number of symmetries, number of symmetry generators, and symmetry detection runtimes for SAUCY. The top row is separated because it shows number of symmetries *without addition of any of the instance-independent SBP constructions*. Henceforth, we will refer to instance-dependent SBPs as *external*, because they are added to an instance *after* symmetries are detected and are not part of the problem formulation.

Table 3 shows the effect of symmetry-breaking on runtimes for PBS, CPLEX, and Galena. The first column in the table specifies the construction type, followed by the number of instances solved for the construction and the total runtime for each solver, with and without the addition of instance-dependent SBPs. For each solver, the best performance among all configurations (largest number of instances solved and corresponding runtime) is boldfaced. We observe the following trends:

1. All benchmarks possess large numbers of symmetries. Different instance-independent SBPs achieve varying degrees of completeness: the lowest-index ordering (LI) is complete and breaks all symmetries, while the selective coloring (SC) SBP breaks the fewest symmetries.

2. On most SBP-free instances, the solvers PBS and Galena perform very poorly, but CPLEX performs well, solving 14 out of 20 instances within the time limit.

3. Both PBS and Galena benefit considerably from instance-dependent symmetry-breaking. When *only* instance-dependent SBPs are used, both solvers solve all 20 instances. However, CPLEX is hampered by addition of SBPs, and solves only 7 instances in this case.

4. Adding *only* instance-independent SBPs improves performance for PBS and Galena over the SBP-free version. The best performance is seen for the NU+SC construction. For CPLEX, the performance is largely unaffected (except for the LI construction, where it is noticeably worse). In general, the LI and CA constructions produce the worst performance out of instance-independent SBPs.

5. Adding instance-independent SBPs alone does not solve as many instances as adding instance-dependent SBPs to the SBP-free formulation. The best performance seen with instance-independent SBPs is 12 (PBS) and 13 (Galena) instances respectively, for the NU+SC construction.

6. For the cases where instance-dependent (external) SBPs were added on top of instance-independent constructions, the best performance for PBS and Galena was still obtained using the NU and SC constructions. For the SC construction with external SBPs added, both solvers solved all 20 instances faster than it took with *only* external SBPs.

7. PBS and Galena exhibit the same performance trends with respect to the constructions used (Galena solves more instances because it is executed on a 4.5x faster machine with the same timeout limit as PBS). This indicates that the variations in performance are due to the different SBPs, not due to differing solver implementations. Both solvers are independent implementations based on the same algorithmic framework (the Davis-Logemann-Loveland backtrack search procedure).

8. Adding external SBPs to any construction usually adversely affects the performance of CPLEX. A similar effect has been observed for CPLEX in [4]. Since the CPLEX algorithms and implementation are not available in the public domain, it is difficult to account for this effect. However, PBS and Galena with symmetry-breaking significantly outperform CPLEX without symmetry-breaking.

Overall, the results suggest that for graph coloring, adding instance-independent SBPs alone is not as good as adding instance dependent SBPs alone, and the best results are achieved using a combination of both types. This is true even when symmetry detection runtimes are taken into consideration. This result is somewhat surprising, and may perhaps be attributed to the complexity of instance-independent SBPs we use.

SBP Type	PBS, SunBlade1000, 750MHz				CPLEX, SunBlade1000, 750MHz				Galena, P4 Xeon, 2GHz			
	Original		w/inst.-dep. SBPs		Original		w/inst.-dep. SBPs		Original		w/inst.-dep. SBPs	
	Time	#solved	Time	#solved	Time	#solved	Time	#solved	Time	#solved	Time	#solved
no SBPs	20000	0	**647**	**20**	6371	14	13805	7	18978	2	**794**	**20**
NU	10719	10	10897	10	5949	15	**6555**	**15**	11339	9	10091	11
CA	20000	0	19717	1	10904	11	10900	10	14134	7	13349	7
LI	18141	2	18141	2	16673	4	16681	4	15827	5	15825	5
SC	17216	3	**177**	**20**	5323	15	12748	8	16061	4	**274**	**20**
NU+SC	**8293**	**12**	8263	12	**4546**	**16**	6419	14	**8594**	**13**	7771	13

Table 3: Runtimes before and after SBPs are added for all constructions using PBS, CPLEX, and Galena; PBS and CPLEX are run a SunBlade 1000 @750MHz, Galena on a Intel P4 Xeon @2GHz. Timeouts for all solvers were set at 1000s. We observe a speedup of 4.5x for the P4 Xeon compared with the SunBlade. This is not a comparison of solvers. We wish to solve ILP formulations with equal optimal values using different solvers to weed out solver-specific issues. Best results for a given solver are shown in boldface.

5 Conclusions

Our work shows that problem reduction to 0-1 ILP is a viable method for optimally solving combinatorial problems without investing into specialized solvers. This approach is likely to be even more successful as the efficiency of 0-1 ILP solvers improves in the future, and as they are able to better handle problem structure. In particular, problem reductions may produce highly-structured instances making the ability to automatically detect and exploit structure very important. In the case of graph coloring we demonstrate that the generic symmetry-breaking flow from [4] significantly improves empirical results in conjunction with ILP solvers PBS [2] and Galena [6]. These two solvers significantly outperform the commercial solver CPLEX 7.0, which is known not to benefit from symmetry-breaking.

We are particularly interested in comparing strategies for breaking symmetries that are present in every ILP instance produced by problem reduction (instance-independent symmetries). Such symmetries may be known even before the first instances of the original problem are delivered (i.e., symmetries may be detected at the specification level), and one has the option to use them during problem reduction. Intuitively, this may prevent discovering these symmetries in every instance and thus improve the overall CPU time. However, this does not happen, apparently because most instance-independent SBPs are rather complicated. It is well known that the syntactic structure of CNF and PB constraints may dramatically affect the efficiency of SAT and ILP solvers. Shorter clauses and PB constraints are much preferable as they are easier to resolve against other constraints and are more useful in recursive learning.

In the context of generic combinatorial problems defined in the NP-spec language [5], our empirical data suggest that new theoretical breakthroughs are required to make use of instance-independent symmetries during problem reductions to SAT or 0-1 ILP. At our current level of understanding, the simple strategy of processing instance-independent and instance-dependent symmetries together produces smallest runtimes for graph coloring benchmarks.

Acknowledgments. This work was funded in part by NSF ITR Grant #0205288. Also, we thank Donald Chai and Andreas Kuehlmann from UC Berkeley for providing us with binaries of the Galena solver.

References

[1] F. A. Aloul et. al, "Solving Difficult SAT Instances In The Presence of Symmetry", *IEEE Trans. on CAD*, vol. 22(9), 1117-1137, 2003.

[2] F. A. Aloul et. al, "Generic ILP versus Specialized 0-1 ILP: An Update", *in Proc. Intl. Conf. on CAD*, 450-457, 2002.

[3] F. A. Aloul, I. L. Markov, K. A. Sakallah, "Shatter: Efficient Symmetry-Breaking for Boolean Satisfiability", *in Proc. Intl. Joint. Conf. on AI*, 271-282, 2003.

[4] F. A. Aloul et. al, "Symmetry-Breaking for Pseudo-Boolean Formulas", *in Intl. Workshop on Symmetry in Constraint Satisfaction Problems (SymCon)*, 1-12, 2003.

[5] M. Cadoli et. al, "NP-SPEC: An Executable Specification Language for Solving All Problems in NP", *Proc. Practical Aspects of Declarative Languages*, 16-30, 1999.

[6] D. Chai, A. Kuehlmann, "A Fast Pseudo-Boolean Constraint Solver", *in Proc. Design Autom. Conf.*, 830-835, 2003.

[7] G. J. Chaitin et. al,"Register allocation via coloring", *in Computer Languages*, 6:47-57, 1981.

[8] J. Crawford et. al, "Symmetry-breaking predicates for search problems", *in Proc. of the Intl. Conf. on Principles of Knowledge Representation and Reasoning*, 148-159, 1996.

[9] J. Culberson, "Graph coloring page", `http://web.cs.ualberta.ca/~joe/Coloring/index.html`

[10] P. Darga, "SAUCY: Graph Automorphism Tool", `http://www.eecs.umich.edu/~pdarga/pub/auto/saucy.html`

[11] U. Feige et. al, "Approximating clique is almost NP-complete",*IEEE Symp. Foundations Comp. Sci.*, 2-12, 1991.

[12] M. R. Garey, D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", 1979.

[13] B. McKay, "NAUTY User s Guide, Version 1.5", *TR-CS-90-02*, Dep. of Comp. Sci., Australian Nat. Univ., 1990.

[14] A. Mehrotra, M. A. Trick, "A column generation approach for graph coloring", *in INFORMS Journal on Computing*, 8(4):344-354, 1996.

[15] M. Moskewicz et. al, "Chaff: Engineering an Efficient SAT Solver", *Proc. Design Autom. Conf.*, 530-535, 2001,

[16] M. Trick, "Network Resources for Coloring a Graph", `http://mat.gsia.cmu.edu/COLOR/color.html`

[17] S. Prestwich, "Supersymmetric Modelling for Local Search", *SymCon '02*, 21-28, September 2002; `http://user.it.uu.se/~pierref/astra/SymCon02/`