In Proc. Computational Symposium on Graph Coloring and Generalizations, 2002

Completing Quasigroups or Latin Squares: A Structured Graph Coloring Problem *

Carla P. Gomes[1] and David Shmoys[1,2]

[1] Dept. of Comp. Science
Cornell University
Ithaca, NY 14853, USA
{gomes,shmoys}@cs.cornell.edu
[2] Dept. of Operations Research and Industrial Engineering
Cornell University
intr Ithaca, NY 14853, USA

Abstract. We introduce a graph coloring challenge benchmark based on the problem of completing Latin squares. We show how the hardness of the instances can be finely controlled by varying the fraction of precolored squares. We compare three complete (exact) solution strategies on this benchmark: (1) a Constraint Satisfaction (CSP) based approach, (2) a hybrid Linear Programming / CSP approach, and (3) a Boolean Satisfibility (SAT) approach. None of these methods uniformly dominates the others on this domain. The CSP and hybrid approaches lead to more compact encodings. The hybrid approach uses a randomized rounding LP based approximation that considerably boosts the performance of the CSP strategy on hard instances. However, the SAT-based approach can solve some of the hardest problem instances, provided the SAT encoding remains manageable. We will discuss the various tradeoffs between these approaches in detail.

1 Introduction

In recent years the Artificial Intelligence, Automated Reasoning, and Constraint Programming communities have become interested in the study of structures from finite algebra as a way of testing general purpose search and reasoning techniques. In particular, the study of Quasigroups or Latin squares has led to

* This research was partially funded by AFOSR, grant F49620-01-1-0076 (Intelligent Information Systems Institute) and F49620-01-1-0361 (MURI grant on Cooperative Control of Distributed Autonomous Vehicles in Adversarial Environments) and DARPA, F30602-00-2-0530 (Controlling Computational Cost: Structure, Phase Transitions and Randomization) and F30602-00-2-0558 (Configuring Wireless Transmission and Decentralized Data Processing for Generic Sensor Networks). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFOSR, DARPA, or the U.S. Government.

good progress in the area of Automated Reasoning. For example, SATO (Satisfiability Testing Optimized)[30], a model generator based on the Davis-Putnam-Logemann-Loveland method for propositional clauses [7, 8], has solved a number of difficult open existence problems for Quasigroups with specific algebraic constraints [10, 27, 31]. More recently, the problem of completing partial Quasigroups or Latin Squares has been proposed as a structured benchmark domain for the study of Constraint Satisfaction (CSP) and Boolean Satisfiability (SAT) methods. The domain provides a good alternative to other benchmarks, which are often based on *purely* random instances [12, 13]. Moreover, the underlying stucture of Latin squares is similar to that found in many real-world applications, such as scheduling and time-tabling, experimental design, error correcting codes, and wavelength routing in fiber optics networks [20, 19].

The study of backtrack search techniques on the Latin squares completion task has uncovered so-called "heavy-tailed" run time behavior of backstrack search in structured domains [14]. Such heavy-tailed behavior explains the high variance, and, in general, the unpredictability of the run time of backtrack search methods in many practical applications. Rapid restarting of the backtrack search can be used to eliminate heavy-tailed behavior and leads to more robust overall solution strategies [23, 14, 22].

In this paper, we first introduce the Quasigroup completion benchmark domain and discuss its structural properties. We then discuss three complete solution strategies for the Quasigroup completion problem: a Constraint Satisfaction (CSP) based approach, a hybrid Linear Programming / CSP approach, and a Boolean Satisfibility (SAT) approach. We provide different encodings for these approaches and details on the search strategies used with each approach. A common feature of our search strategies is the careful use of randomization and restarts to obtain better scaling and more robust solvers, while maintaining the completeness of backtrack search methods.

As we will see, none of the methods uniformly dominates the others on our benchmark. The CSP and hybrid approaches lead to more compact encodings. Our hybrid approach uses a randomized rounding LP based approximation that considerably boosts the performance of the CSP strategy on hard instances. However, the SAT-based approach can solve some of the hardest problem instances, provided the SAT encoding remains manageable. We will discuss the various tradeoffs between these approaches in detail.

2 Benchmark Definition and Problem Hardness

A quasigroup is an ordered pair $(Q, \cdot)$, where Q is a set and $\cdot$ is a binary operation on Q such that the equations $a \cdot x = b$ and $y \cdot a = b$ are uniquely solvable for every pair of elements a, b in Q. The *order* N of the quasigroup is the cardinality of the set Q. The best way to understand the structure of a quasigroup is to consider the N by N multiplication table as defined by its binary operation. The constraints on a quasigroup are such that its multiplication table defines a Latin square. This means that in each row of the table, each element of the set

Q occurs exactly once; similarly, in each column, each element occurs exactly once [9].

An *incomplete* or *partial* Latin square P is a partially filled N by N table such that no symbol occurs twice in a row or a column. The Quasigroup or Latin Square Completion Problem (QCP) is the problem of determining whether the remaining entries of the table can be filled in such a way that we obtain a complete Latin square (see Figure 1). (See www.cs.cornell.edu/gomes/demos for a java applet demonstrating the Quasigroup Completion Problem.)

	1	2	3
2		4	1
1	4		2
3		1	

4	1	2	3
2	3	4	1
1	4	3	2
3	2	1	4

Fig. 1. Quasigroup Completion Problem of order 4, with 5 holes.

A Latin square has a natural representation as a constraint network, in which nodes correspond to the cells of the table and edges connect nodes in the same row or column. A Latin square of order N has N^2 nodes and $N^2(N-1)$ edges, corresponding to $2N$ cliques, a clique per row/column, each of size N. The constraint network of a Latin square has a so-called *small world* topology. In a small world network, nodes are highly clustered and, yet, the path length between two nodes in different clusters is relatively small [29, 28]. Figure 2 shows the constraint network of a Latin square of order 4.

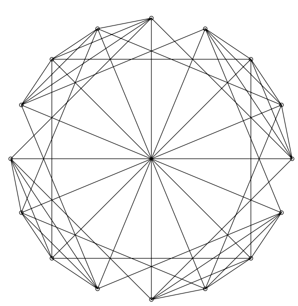
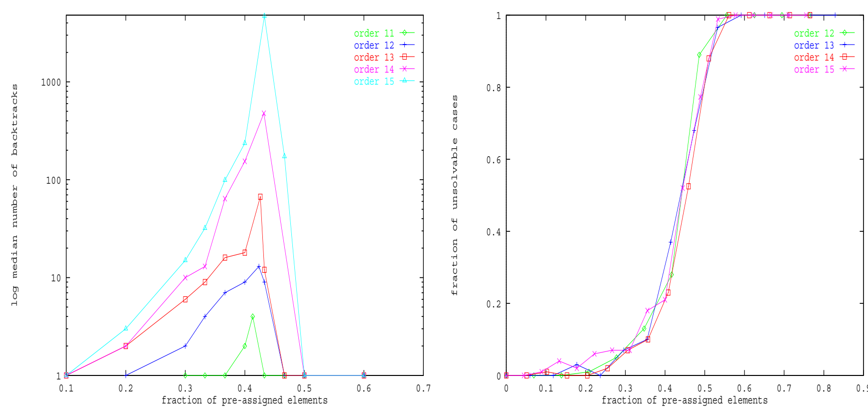


Fig. 2. Constraint network of a Latin Square ($N = 4$). The graph has N^2 nodes, and $N^2(N-1)$ edges. The edges form $2N$ cliques, each of size N [28].

The Quasigroup Completion Problem (QCP) is NP-complete [2, 6]. In previous work, we identified a phase transition phenomenon for QCP [12, 26]. At the phase transition, problem instances switch from being almost all solvable ("under-constrained") to being almost all unsolvable ("over-constrained"). The computationally hardest instances lie at the phase transition boundary. This phase transition allows us to tune the difficulty of our problem class by varying the percentage of pre-assigned values. The location of the phase transition for quasigroups of order up to around 15 occurs around 42% of preassigned colors. See [1] and [18] for details. Figure 3 shows the median computational cost and phase transition for the Quasigroup Completion Problem as functions of the percentage of pre-colored values, for quasigroups up to order 15. Each data point is generated using 1,000 problem instances.



(a) (b)

Fig. 3. (a) Cost profile, and (b) phase transition for the quasigroup completion problem (up to order 15).

In a variant of the Quasigroup Completion Problem, an initial complete quasigroup is randomly generated and a fraction of cells, randomly selected, is uncolored [1]. The initial complete quasigroup is generated using a Markov chain Monte Carlo shuffling process. The task again is to find a coloring for the empty cells that completes the Latin square. We refer to this problem as the "Quasigroup with Holes" (QWH) problem.[1] Interestingly, we can also finely tune the complexity of the QWH completion task by varying the fraction of the uncolored cells. In fact, QWH also exhibits an easy-hard-easy pattern in computational complexity, with the hardest instances coinciding with a phase transition in the *backbone* variables.[2] The backbone corresponds to the solution invariants, *i.e*, the cells that have the same colors assigned in all the solutions

[1] The code for this generator is available by contacting Carla Gomes (gomes@cs.cornell.edu).

[2] Note that since all the instances of QWH are guaranteed to be satisfiable, the problem does not exhibit a phase transition in solvability.

to a given instance (Figure 4) [1]. Given that the instances of QWH are guaranteed to be satisfiable, this benchmark is suitable for both complete (exact) and incomplete search methods.

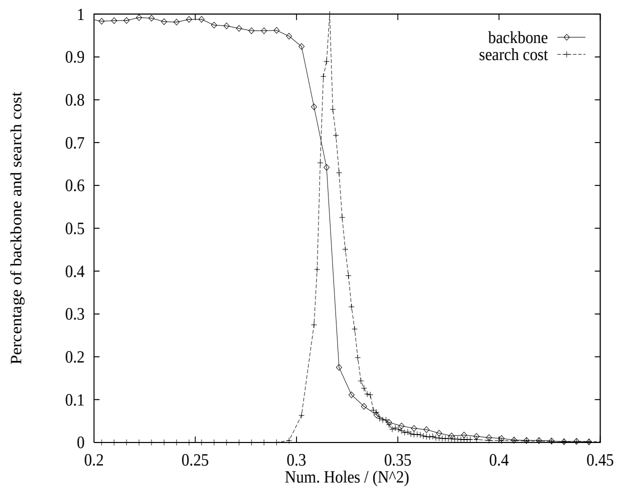


Fig. 4. "Easy-hard-easy" pattern of computational search cost (QWH) and backbone phase transition.

Both QCP and QWH problem instances are considerably harder when the distribution of the holes is *balanced*, *i.e*, when the numbered of uncolored cellls is (approximately) the same accross the different rows and columns [18].

3 Problem Encodings

We studied several solution strategies for our benchmark problems. In particular, we considered a Constraint Satisfaction (CSP) based approach, a hybrid Linear Programming / CSP based strategy, and a Boolean Satisfiability (SAT) based approach. For each of these approaches, we have a choice of problem encodings. In this section, we briefly describe the encodings we considered.

3.1 CSP Formulation

Given a partial latin square of order n, PLS, the latin square completion problem can be expressed as a CSP [12]:

$$x_{i,j} \in \{1, \ldots, n\} \quad \forall i, j$$

$$x_{i,j} = k \quad \forall i, j \text{ such that } PLS_{ij} = k$$

$$\texttt{alldiff}\ (x_{i,1}, x_{i,2}, \ldots, x_{i,n}\} \quad \forall i = 1, 2, \ldots, n$$

$$\texttt{alldiff}\ (x_{1,j}, x_{2,j}, \ldots, x_{n,j}\} \quad \forall j = 1, 2, \ldots, n$$

The `alldiff` constraint states that all the variables involved in the constraint have to have different values [25]. The problem of enforcing an `alldiff` constraint corresponds to solving a matching on a bipartite graph, and therefore it can be propagated efficiently. This constraint provides a powerful pruning and value propagation mechanisms throughout the search for a solution. The variables $x_{i,j}$ denote the color assigned to cell i, j. The model has N^2 variables. In some variants of this model we include redundant variables and corresponding `alldiff` constraints, namely the dual variables $x'_{i,j}$ denoting the column in which color i appears in row j, and similarly, $x''_{i,j}$ denoting the row in which color i appears in column j. Depending on the number of redundant variables considered, our CSP model includes N^2, $2N^2$, or $3N^2$ variables and corresponding `alldiff` constraints, $2N$, $4N$, or $6N$ constraints.

3.2 IP Formulation — Assignment Formulation

Given a partial latin square of order n, PLS, the latin square completion problem can be expressed as an integer program [19]:

$$\max \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} x_{i,j,k}$$

subject to

$$\sum_{i=1}^{n} x_{i,j,k} \leq 1, \quad \forall j, k$$

$$\sum_{j=1}^{n} x_{i,j,k} \leq 1, \quad \forall i, k$$

$$\sum_{k=1}^{n} x_{i,j,k} \leq 1, \quad \forall i, j$$

$$x_{i,j,k} = 1 \quad \forall i, j, k \text{ such that } PLS_{ij} = k$$

$$x_{i,j,k} \in \{0, 1\} \quad \forall i, j, k$$

$$i, j, k = 1, 2, \ldots, n$$

If PLS is completable, the optimal value of this integer program is the number of holes in PLS. Kumar et al. [19] considered the design of approximation algorithms for this optimization variant of the problem based on first solving the linear programming relaxation of this integer programming formulation; that is, the conditions $x_{i,j,k} \in \{0, 1\}$ above are replaced by $x_{i,j,k} \geq 0$. Their algorithm repeatedly solves this linear programming relaxation, focuses on the variable closest to 1 (among those not set to 1 by the PLS conditions), and sets that variable to 1; this iterates until all variables are set. This algorithm is shown to be a 1/3-approximation algorithm; that is, if PLS is completable, then it manages to find an extension that fills at least $h/3$ holes. Kumar et al. also provide a more sophisticated algorithm in which the colors are considered in turn; in the iteration corresponding to color k, the algorithm finds the extension (of at most n cells) for which the linear programming relaxation places the greatest total weight. This algorithm is shown to be a 1/2-approximation algorithm; that is, if PLS is completable, then the algorithm computes an extension that fills at least $h/2$ holes. In the experimental evaluation of their algorithms, Kumar et al. solve problems up to order 9.

The assignment formulation can be used as the basis for a randomized rounding procedure in a variety of ways. Let x^* denote an optimal solution to the linear programming relaxation of this integer program. For any randomized procedure in which the probability that cell (i,j) is colored k is equal to x^*_{ijk}, then we know that, in expectation, each row i has at most one element of each color k, each column j has at most one element of each color k, and each cell (i,j) is assigned at most one color k. However, having these each hold "in expectation" is quite different than expecting that all of them will hold simultaneously, which is extremely unlikely.

3.3 IP Formulation — Packing Formulation

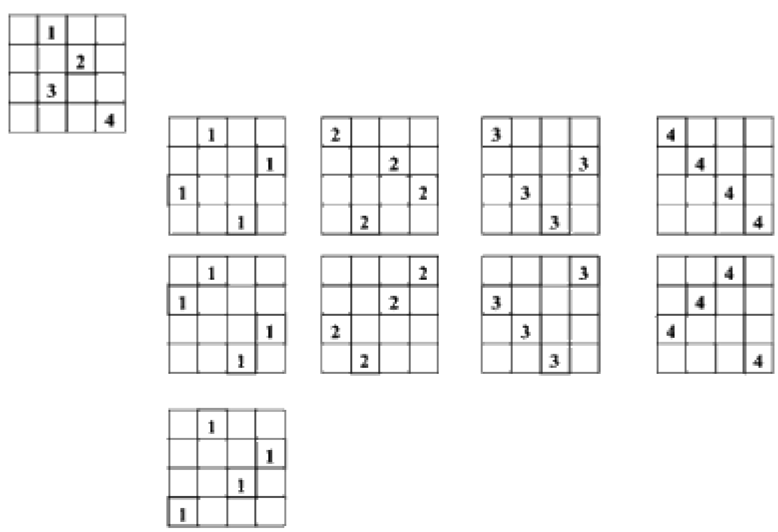


Fig. 5. Families of compatible matchings for the partial latin square in the left upper corner. For example, the family of compatible matchings for symbol 1 has three compatible matchings.

Alternate integer programming formulations of this problem can also be considered. The *packing formulation* is one such formulation for which the linear programming relaxation produces stronger lower bounds [15, 11]. For the given PLS input, consider one color k. If PLS is completable, then there must be an extension of this solution with respect to this one color; that is, there is a set of cells (i, j) that can each be colored k so that there is exactly one cell colored k in every row and column. We shall call one such collection of cells a *compatible matching* for k. Furthermore, any subset of a compatible matching shall be called a *compatible partial matching*; let $\mathcal{M}_k$ denote the family of all compatible partial matchings.

With this notation in mind, then we can generate the following integer programming formulation by introducing one variable $y_{k,M}$ for each compatible partial matching M in $\mathcal{M}_k$:

$$\max \sum_{k=1}^{n} \sum_{M \in \mathcal{M}_k} |M| y_{k,M}$$

subject to

$$\sum_{M \in \mathcal{M}_k} y_{k,M} = 1, \quad \forall k$$

$$\sum_{k=1}^{n} \sum_{M \in \mathcal{M}_k : (i,j) \in M} y_{k,M} \leq 1, \quad \forall i, j$$

$$y_{k,M} \in \{0, 1\} \quad \forall k, M.$$

Once again, we can consider the linear programming relaxation of this formulation, in which the binary constraints are relaxed to be merely non-negativity constraints. It is significant to note that, for any feasible solution y to this linear programming relaxation, one can generate a corresponding feasible solution x to the assignment formulation, by simply computing $x_{i,j,k} = \sum_{M \in \mathcal{M}_k} y_{k,M}$. This construction implies that the value of the linear programming relaxation of the assignment formulation (which provides an upper bound on the desired integer programming formulation) is at least the bound implied by the LP relaxation of the packing formulation; that is, the packing formulation provides a tighter upper bound. However, note that the size of this formulation is exponential in n. In spite of this difficulty, one may apply column generation techniques (see, e.g., the textbook by [5]) to compute an optimal solution relatively efficiently.

In contrast to the situation for the assignment formulation, there is a straighforward theoretical justification for the randomized rounding of the fractional optimal solution, as we proposed in [11]. Rather than the generic randomized rounding mentioned above, instead, for each color k choose some compatible partial matching M with probability $y_{k,M}$ (so that some matching is therefore selected for each color). These selections are done as independent random events. This independence implies there might be some cell (i, j) included in the matching selected for two distinct colors. However, the constraints in the linear program imply that the expected number of matchings in which a cell is included is at most one. In fact, if PLS is completable, and hence the linear programming relaxation satisfies the inequality constraints with equality (and hence $|M| = n$ whenever $y_{k,M} > 0$), then we can show that the expected number of cells not covered by a matching is at most h/e, where h is the number of holes in PLS; that is, at least $(1 - 1/e)h$ holes can expected to be filled by this technique [11].

3.4 SAT Based Formulations

Although quasigroup completion problems are most naturally represented as a CSP using multi-valued variables, encoding the problems using only Boolean variables in clausal form turns out to be surprisingly effective [1, 18].

In our SAT encoding, each Boolean variable x_{ijk} denotes that a color k is assigned to cell i, j, where $i, j, k = 1, 2, \ldots, n$; n is the order. There are n^3 variables. The most basic encoding, which we call the "minimal encoding," includes clauses that represent the following constraints:

1. Some color must be assigned to each cell (clauses of length n): $\forall_{ij}(x_{ij1} \wedge x_{ij2} \wedge \cdots \wedge x_{ijn})$;
2. No color is repeated in the same row (sets of negative binary clauses): $\forall_{ik}(\neg x_{i1k} \wedge \neg x_{i2k})(\neg x_{i1k} \wedge \neg x_{i3k}) \ldots (\neg x_{i1k} \wedge \neg x_{ink})$;
3. No color is repeated in the same column (sets of negative binary clauses): $\forall_{jk}(\neg x_{1jk} \wedge \neg x_{2jk})(\neg x_{1jk} \wedge \neg x_{3jk}) \ldots (\neg x_{1jk} \wedge \neg x_{njk})$.

Constraint (1) becomes a clause of length n for each cell, and (2) and (3) become sets of negative binary clauses. The total number of clauses is $O(n^4)$.

The binary representation of a Latin square can be viewed as a cube, where the dimensions are the row, column, and color. This view reveals an alternative way of stating the Latin square property: any set of variables determined by holding two of the dimensions fixed must contain exactly one true variable. The "extended encoding" captures this condition by also including the following constraints:

1. Each color much appear at least once in each row (clauses of length n): $\forall_{ik}(x_{i1k} \wedge x_{i2k} \wedge \cdots \wedge x_{ink})$;
2. Each color much appear at least once in each column (clauses of length n): $\forall_{jk}(x_{1jk} \wedge x_{2jk} \wedge \cdots \wedge x_{njk})$;
3. No two colors are assigned to the same cell (sets of negative binary clauses): $\forall_{ij}(\neg x_{ij1} \wedge \neg x_{ij2})(\neg x_{ij1} \wedge \neg x_{ij3}) \ldots (\neg x_{ij1} \wedge \neg x_{ijn})$;

As before, the total size of the extended encoding is $O(n^4)$.

4 Search Strategies

We use variants of randomized backtrack search methods, drawing on recent results that provide evidence of the effectiveness of randomization and restart strategies to combat the heavy-tailed nature of combinatorial search. By using restart strategies we take advantage of any significant probability mass early on in the distribution, reducing the variance in run time and the probability of failure of the search procedure, resulting in a more robust overall search method [14, 13]. Note that we still maintain the completeness of our randomized backtrack style algorithms, by a combination of nogood learning and, in the case where we don't perform learning, by increasing the cutoff for the restart strategies.

Our most competitive results for complete methods were obtained with the following strategies:

4.1 CSP-based strategy

Our CSP models are implemented in Ilog/Solver [17]. We randomized Ilog's backtrack search procedure, maintaing its completeness. We use a variant of the Brelaz rule for variable selection [3, 12]. Value selection is performed randomly. For the propagation of the `alldiff` constraint we use the extended version provided by Ilog [17, 25].

4.2 Hybrid LP/CSP approach

Our hybrid LP/CSP approach draws on recent results on approximation algorithms with theoretical guarantees, based on LP relaxations and randomized rounding techniques (see e.g., [4, 24]). A central feature of our hybrid algorithm is the fact that it maintains two different formulations of the quasigroup completion problem: the CSP formulation and the relaxation of the LP formulation described above combined with randomized rounding.[3] The hybrid nature of the algorithm results from the combination of strategies for variable and value assignment, and propagation, based on the two underlying models.

The algorithm is initialized by populating the CSP model and propagating constraints over this model, as described above. The updated domain values are then used to populate the LP model. We solve the LP model using Ilog/Cplex Barrier [16]. The LP model provides valuable search guidance and pruning information for the CSP search. However, since solving the LP model is relatively expensive compared to the inference steps in the CSP model, we have to carefully manage the time spent on solving the LP model. The LP effort is controlled by two parameters: At the top of the backtrack search tree, variable and value selection are based on the LP rounding. After each value assignment based on the LP randomized rounding, full propagation is performed on the CSP model. The percentage of variables set by the LP is controlled by the parameter %LP. (So, with % LP = 0, we have a pure CSP strategy.) After this initial phase, variable and value settings are based purely on the CSP model. Note that deeper down in the search tree, the LP formulation continues to provide information on variable

[3] In the experiments reported here, we use the assignment formulation for the LP. We are currently performing an empirical evaluation of our hybrid approach using the packing formulation, which has stronger theoretical bounds, but requires a column generation approach.

settings. However, we have found that this information can be computed more efficiently through the CSP model.

Ideally, in order to increase the accuracy of the variable assignments based on LP-rounding, one would like to update and re-solve the LP model after each variable setting. However, in practice, this is too expensive. We therefore introduce a parameter, *interleave-LP*, which determines the frequency with which the LP model is updated and re-solved. In our experiments, we found that updating the LP model after every five variable settings (*interleave-LP*= 5) is a good compromise.

In our LP rounding strategy, we first rank the variables according to their LP values (*i.e.*, variables with LP values closest to 1 are ranked near the top). We then select the highest ranked variable and set its value to 1 (*i.e.*, set the color of the corresponding cell) with a probability p given by its LP value. With probability $1 - p$, we randomly select a color for the cell from the colors still allowed according to the CSP model. After each variable setting, we perform CSP propagation. The CSP propagation will set some of the variables on our ranked variable list. We then consider the next highest ranked variable that is not yet assigned. A total of *interleave-LP* variables is assigned this way, before we update and re-solve the LP.

Backtracking can occur as a result of an inconsistency detected either by the CSP model or the LP relaxation. It is interesting to note that backtracking based on inconsistencies detected by the LP occurs rather frequently at the top of our search tree. This means that the LP does indeed uncover global information not easily obtained via CSP propagation, which is a more local inference process. Of course, as noted before, lower down in the search tree, using the LP for pruning becomes ineffective since CSP propagation with only a few additional bactracks can uncover the same information.

In this setting, we are effectively using the LP values as heuristic guidance, using a randomized rounding approach inspired by the rounding schemes used in approximation algorithms. As we discussed in Section 3.3, for the packing formulation of the LP, we have a clear theoretical basis for such a rounding scheme. For the assignment formulation, the theoretical justification is less immediate — nevertheless, our empirical results show that this scheme leads to a clear practical payoff.

4.3 SAT-based strategy

We adopted the *extended* encoding described above for our SAT model. Even though this encoding is not as compact as the minimal encoding, its redundant clauses increase propagation dramatically and allow us to solve much larger problems, considerably faster. Our search procedure called Satz-rand is a randomized version of the complete Satz solver [21, 14]. Satz implements the backtracking Davis-Putnam algorithm augmented with 1-step lookahead.

As mentioned above, backtrack search methods are characterized by long tails, often heavy-tails. That is, a backtrack search is quite likely to encounter extremely long runs. To avoid getting stuck in such unproductive runs, we use a cutoff parameter in our randomized backtrack search procedures. This parameter defines the number of backtracks after which the search is restarted, at the beginning of the search tree, with a different random seed. Note that in order to maintain the completeness of the algorithm we just have to increase the cutoff. In the limit, we run the algorithm without a cutoff.

5 Empirical Results

In this section we summarize the performance of our different strategies on the benchmark instances of type qg and qwh (GOM). In the appendix we provide detailed experimental data.

Overall, the performance of Satz-rand, our complete randomized backtrack search SAT solver, is quite surprising; Satz-rand outperforms other approaches on critically and medium constrained instances, as long as the size of the encoding remains manageable. Interestingly, SAT based methods perform quite poorly on the *minimal* SAT encoding, without any redundancy. The explanation for the effectiveness of the extended encoding lies in the formidable amount of relatively inexpensive propagation, unit propagation performed in linear time, which is enhanced by the redundant constraints. Nevertheless, as mentioned above, SAT based strategies are out of reach for large instances or, perform poorly, given the size of the encodings. Recall that the size of an instance is dependent on the order of the quasigroup but also on the number of holes. For example, an instance of order 30 with 100% of holes, or an instance of order 60 with 100 % holes, are too large for a SAT based encoding. An instance of order 20 with 100% of holes can be solved with a SAT based approach but, given the size of the encoding, the performance is poor, significantly worth than when using a CSP based approach.

Pure CSP strategies can handle larger instances in the under-constrained area better than SAT based approaches, given the compacteness of CSP encodings. However, and similarly to SAT encodings, the performance of CSP approaches increases considerably with a less compact representation that triggers additional propagation. We obtained good results with one set of redundant variables and corresponding constraints, *i.e*, a total of $2N^2$ variables and $4N$ alldif constraints. Nevertheless, as mentioned above, for critically and medium constrained instances, the performance of pure CSP based approaches is not as competitive as SAT based approaches. The additional propagation provided by the `alldiff` constraint does not counterbalance its relative high cost, in comparison to the low cost of unit propagation used in SAT based approaches. For such instances, the hybrid CSP/LP randomized rounding backtrack search approach considerably outperforms the pure CSP strategy. The LP rounding at the top of the tree provides a more accurate heuristic than the variant of the Brelaz rule used in the pure CSP strategy, reducing considerably the amount of search. Our hybrid CSP/LP approach solves instances that cannot be solved by the pure CSP strategy and, in some cases, not even by our SAT based approach. For example, the critically constrained instances of order 40 could not be solved by the pure CSP strategy, while the hybrid strategy solved them. The hybrid strategy was the only complete method that could solve the instance qwhdec.order50.holes825.bal.1.col, a very hard instance of order 50, critically constrained and fully balanced.

6 Conclusions

We described a structured graph coloring benchmark based on the completion of Latin squares and showed how we can finely control the hardness of the benchmark instances. We considered three complete solution methods for our benchmark: a CSP approach, a hybrid LP/CSP strategy, and a SAT-based approach. None of the methods uniformly dominates the others on our benchmark.

SAT-based approaches are surprisingly effective on critically-constrained instances. However, the limited expressiveness of the SAT formulation leads to relatively large encodings, which renders solving instances of order over 40 practically infeasible.

CSP-based approaches provide more compact encodings. The CSP formulation is particularly effective on under-constrained instances, where the strong propagation and pruning due to the `alldiff` constraints can practically eliminate search. However, on critically constrained instances, the size of the backtrack search tree increases considerably, and computing `alldiff` constraint throughout the backtrack search becomes prohibitive.

In the critically constrained area, the success of the SAT approach with its relatively inexpensive propagation and redundant problem encoding suggests as a promising research direction for CSP methods, the development of a faster, but restricted, versions of the `alldiff` constraint, thereby trading pruning power for a decrease in computational cost. Alternatively, one could limit the frequency with which the `alldiff` constraint is invoked during the backtrack search.

Another promising research direction is to consider stronger heuristics for value and variable selection. As a step forward in this direction, we showed that LP rounding provides a powerful search heuristic, boosting the performance of pure CSP based strategies considerably. We are currently experimenting with the packing formulation, and, given the strong theoretical guarantees of our approximation method, we hope to further improve our results. Overall, we see a real potential in the use of fast approximation methods to increase pruning and propagation for speeding up combinatorial search on hard structured problem domains.

References

1. D. Achlioptas, C. Gomes, H. Kautz, and B. Selman. Generating Satisfiable Instances. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-00)*, New Providence, RI, 2000. AAAI Press.

2. L. Anderson. Completing partial latin squares. *Mathematisk Fysiske Meddelelser*, 41:23–69, 1985.
3. D. Brelaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.
4. F. Chudak and D. Shmoys. Improved approximation algorithms for the uncapacitated facility location problem. In *Submitted for publication*, 1999. Preliminary version of this paper (with the same title) appeared in proceedings of the Sixth Conference on Integer Programming and Combinatorial Optimization.
5. V. Chvatal. *Linear Programming*. W.H.Freeman Company, 1983.
6. C. Colbourn. Embedding partial steiner triple systems is np-complete. *J. Combin. Theory*, A(35):100–105, 1983.
7. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1979.
8. M. Davis and H. Putman. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.
9. J. Denes and A. Keedwell. *Latin squares and their applications*. Akademiai Kiado, Budapest, and English Universities Press, London, 1974.
10. M. Fujita, J. Slaney, and F. Bennett. Automatic generation of some results in finite algebra. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 52–57, France, 1993. AAAI Pess.
11. C. Gomes, R. Regis, and D. Shmoys. An improvement performance guarantee for the partial latin square problem. Manuscript in preparation, 2002.
12. C. Gomes and B. Selman. Problem Structure in the Presence of Perturbations. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 221–227, New Providence, RI, 1997. AAAI Press.
13. C. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. of Automated Reasoning*, 24(1–2):67–100, 2000.
14. C. Gomes, B. Selman, and H. Kautz. Boosting Combinatorial Search Through Randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 431–438, New Providence, RI, 1998. AAAI Press.
15. C. P. Gomes and D. Schmoys. The promise of LP to boost CSP techniques for combinatorial problems. In N. Jussien and F. Laburthe, editors, *Proceedings of the Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'02)*, pages 291–305, Le Croisic, France, Mar., 25–27 2002.
16. Ilog. Ilog cplex 7.1. user's manual., 2001.
17. Ilog. Ilog solver 5.1. user's manual., 2001.
18. H. Kautz, Y. Ruan, D. Achlioptas, C. Gomes, and B. Selman. Balance and Filtering in Structured Satisfiable Problems. In *Proceedings of the Seventeenth Inernational Joint Conference on Artificial Intelligence (IJCAI-01)*, Seattle, WA, 2001.
19. S. R. Kumar, A. Russell, and R. Sundaram. Approximating latin square extensions. *Algorithmica*, 24:128–138, 1999.
20. C. Laywine and G. Mullen. *Discrete Mathematics using Latin Squares*. Wiley-Interscience Series in Discrete mathematics and Optimization, 1998.
21. C. M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 366–371. AAAI Pess, 1997.
22. I. Lynce and J. Marques-Silva. Building State-of-the-Art SAT Solvers. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, 2002.

23. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. "chaff: Engineering an efficient SAT solver". In *Proc. of Design Automation Conference*, pages 530–535, 2001.
24. R. Motwani, J. Naor, and P. Raghavan. Randomized approximation algorithms in combinatorial optimization. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, 1997.
25. J. C. Regin. A filtering algorithm for constraints of difference in csp. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 362–367, Seattle, WA, 1994. AAAI Press.
26. P. Shaw, K. Stergiou, and T. Walsh. Arc consistency and quasigroup completion. In *Proceedings of ECAI-98, workshop on binary constraints*, 1998.
27. J. Slaney, M. Fujita, and M. Stickel. Automated reasoning and exhaustive search: Quasigroup existence problems. *Computers and Math. with Applications*, 29:115–132, 1995.
28. T. Walsh. Search in a Small World. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, 1999.
29. D. J. Watts and S. Strogatz. Collective dynamics of *smallworld* networks. *Nature*, 393:440–442, 1998.
30. H. Zhang. Sato: An efficient propositional prover. In *Proc. CADE-97*, 1997.
31. H. Zhang. Specifying latin square problems in propositional logic. In *Automated Reasoning and its applications*. MIT Press, 1997.

Appendix

In this appendix, we include results for the different instances in the graph coloring repository of the form qg.orderX, qwhdec.orderX.holesY.1, and qwhdec.orderX.holesY.bal (GOM). Instances of the form qg.orderX correspond to empty QWH instances, *i.e.*, instances with 100% holes. For example, qg.order30, corresponds to an empty qwh instance of order 30, *i.e.* and instance with 900 holes. Instances of the form qwhdec.orderX.holesY correspond to QWH instances of order X with Y holes. For example, qwhdec.order5.holes10.1 corresponds to a QWH instance of order 5 with 10 holes. Instances of the form qwhdec.orderX.holesY.bal correspond to QWH *balanced* instances of order X with Y holes. For example, qwhdec.order5.holes10.1.bal corresponds to a balanced QWH instance of order 5 with 10 holes, with 2 holes per row/column. Balanced instances are considerably harder than instances in which holes are randomly placed [18].

Instance			Level	Region	Best of CSP vs. Hybrid			SAT	
Name	Order	Holes			Back.	Secs	Strategy	Back.	Secs
qwhdec.order5.holes10.1	5	10	1	U	0	0.01	CSP	0	1
qwhdec.order18.holes120.1	18	120	1	U	0	0.24	CSP	N.A.	N.A.
qg.order30	30	900	1	U	0	4.23	CSP	N.A.	N.A.
qwhdec.order30.holes316.1	30	316	1	U	100	2.64	CSP	35	1.010
qwhdec.order30.holes320.1	30	320	1	U	5	2.18	CSP	3	0.88
qg.order40	40	1600	2	U	0	16.37	CSP	N.A.	N.A.
qg.order60	60	3600	2	U	20	125.86	CSP	N.A.	N.A.
qg.order100	100	10000	2	U	17	1447.03	CSP	N.A.	N.A.
qwhdec.order35.holes405.1	35	405	2	M	see table 2	table 2	Hybrid	31329	336.38
qwhdec.order40.holes528.1	40	528	2	M	table 3	table 3	Hybrid	16464	208.55
qwhdec.order60.holes1440.1	60	1440	2	M	N.A.	N.A.	N.A.	219	121.75
qwhdec.order60.holes1620.1	60	1620	2	M	9022	2219.81	Hybrid	N.A.	N.A.
qwhdec.order70.holes2940.1	70	2940	2		3226	156.4	CSP	N.A.	N.A.
qwhdec.order70.holes2450.1	70	2450	2	M	34495	6625.14	Hybrid	N.A.	N.A.
qwhdec.order33.holes381.bal.1	33	381	3	C	N.A.	N.A.	N.A.	28385	99.36
qwhdec.order50.holes825.bal.1	50	825	4	C (Bal)	table 4	table 4	Hybrid	N.A.	N.A.
qwhdec.order50.holes750.bal.1	50	750	5	C (Bal)	N.A.	N.A.	N.A.	N.A.	N.A.
qwhdec.order60.holes1080.bal.1	60	1080	5	C (Bal)	N.A.	N.A.	N.A.	N.A.	N.A.
qwhdec.order60.holes1152.bal.1	60	1152	5	C (Bal)	N.A.	N.A.	N.A.	N.A.	N.A.

Table 1. Preview of the results for the QG and QWH instances. (Level - computational hardness, 1 - 5, 1 is easy; 5 is very hard; Region: U - under-constrained; M - medium constrained; C - critically-constrained; Bal - balanced instance; Back. - number of backtracks; Secs - time in seconds. NA - not applicable; could not be solved by the strategy.)

% LP	Cutoff	Num. Runs	% Succ. Runs	Median Backtracks	Median Time
0	10^6	100	6%	474049	1312.58
1	10^6	100	42%	589438	1992.08
5	10^6	100	90%	188582	615.16
10	10^6	100	100%	26209	114.35
15	10^6	100	100%	22615	116.29
20	10^6	100	100%	17203	112.64
25	10^6	100	100%	21489	158.07
30	10^6	100	100%	24139	179.37
50	10^6	100	100%	19325	262.67
75	10^6	100	100%	17458	379.68

Table 2. Hybrid CSP/LP search on qwhdec.order35.holes405.1, instance of order 35 with 405 holes.

% LP	Cutoff	Num. Runs	% Succ. Runs	Median Backtracks	Median Time
0	10^5	100	0%	N.A.	N.A.
10	10^5	100	1%	48387	245.54
25	10^5	100	37%	17382	215.69
50	10^5	100	47%	21643	422.59
0	10^6	100	0%	N.A.	N.A.
10	10^6	100	8%	355362	1488.08
25	10^6	100	64%	123739	574.68
50	10^6	100	65.3%	128306	757.55

Table 3. Hybrid CSP/LP search on instance qwhdec.order40.holes528.1, instance of order 40 with 528 holes. (% LP - percentage of assignments at the top of the tree dictated by the LP rounding. Cutoff - cutoff used per run)

Strategy	Cutoff	Num. Runs	Succ. Run Backtracks	Succ. Run Time
CSP	50000	100	N.A.	N.A.
CSP	250000	100	N.A.	N.A.
CSP	500000	100	N.A.	N.A.
CSP	1000000	100	N.A.	N.A.
25% LP	50000	100	N.A.	N.A.
25% LP	250000	100	N.A.	N.A.
25% LP	500000	100	N.A.	N.A.
25% LP	1000000	100	N.A.	N.A.
50% LP	50000	100		
			23420	629.89
50% LP	250000	100		
			130613	990.8
			165859	1034.39
50% LP	50000	100		
			43620	653.52
50% LP	1000000	100	N.A.	N.A.
SAT	50000	100	N.A.	N.A.
SAT	250000	100	N.A.	N.A.
SAT	500000	100	N.A.	N.A.
SAT	1000000	100	N.A.	N.A.

Table 4. Different search strategies for a balanced instance of order 50, with 825 holes. Note that this instance is only solved with the hybrid CSP/LP stratagy, with a 50% level of variable assignments based on the LP rounding. The pure CSP strategy and the SAT approach could not solve it.