The Impact of Balancing on Problem Hardness in a Highly Structured Domain *

Carlos Ansótegui[1], **Ramón Béjar**[2], **César Fernàndez**[2], **Carla Gomes**[3], **Carles Mateu**[2]
[1] carlos@iiia.csic.es, IIIA-CSIC, Barcelona, SPAIN
[2] {ramon,cesar,carlesm}@diei.udl.es, Dept. of Computer Science, Universitat de Lleida, SPAIN
[3] gomes@cs.cornell.edu, Dept. of Computer Science, Cornell University, USA

Abstract

Random problem distributions have played a key role in the study and design of algorithms for constraint satisfaction and Boolean satisfiability, as well as in our understanding of problem hardness, beyond standard worst-case complexity. We consider random problem distributions from a highly structured problem domain that generalizes the Quasigroup Completion problem (QCP) and Quasigroup with Holes (QWH), a widely used domain that captures the structure underlying a range of real-world applications. Our problem domain is also a generalization of the well-known Sudoku puzzle: we consider Sudoku instances of *arbitrary* order, with the additional generalization that the block regions can have rectangular shape, in addition to the standard square shape. We evaluate the computational hardness of Generalized Sudoku instances, for different parameter settings. Our experimental hardness results show that we can generate instances that are considerably harder than QCP/QWH instances of the same size. More interestingly, we show the impact of different balancing strategies on problem hardness. We also provide insights into backbone variables in Generalized Sudoku instances and how they correlate to problem hardness.

Introduction

In recent years we have seen a tremendous development of both complete and incomplete search methods for constraint satisfaction (CSP) and Boolean satisfiability (SAT) problems. An important factor in the development of new search methods is the availability of good sets of benchmarks problems used to evaluate and fine-tune the algorithms. Random problem distributions have played a key role in the study and design of algorithms, as well as in our understanding of problem hardness, beyond standard worst-case complexity. Problem distributions, such as random k-SAT, have traditionally been used to study the typical case complexity of combinatorial search problems. However, real-world problem instances have much more structure than that found in random SAT instances. In order to study the impact of structure on problem hardness, Gomes and Selman introduced the Quasigroup Completion completion problem (QCP) as a benchmark problem for evaluating combinatorial search methods: the structure underlying this domain can be found in a range of real-world applications, such as timetabling, routing, and design of statistical experiments (Gomes & Selman 1997). QCP (and its variants) has become a widely used benchmark domain and it has led researchers to the discovery of interesting phenomena that characterize combinatorial search and consequently to the design of new search and inference strategies for such domains. Examples include, the so-called heavy-tailed phenomena, randomization and restarts strategies, e.g., (Gomes, Selman, & Crato 1997; Gomes *et al.* 2004; Refalo 2004; Hulubei & O'Sullivan 2006), the design of efficient global constraints, e.g., (Shaw, Stergiou, & Walsh 1998; Régin & Gomes 2004), and tradeoffs in different CSP and SAT based representations (see e.g. (Dotú, del Val, & Cebrián 2003; Ansótegui *et al.* 2004)).

We consider random problem distributions from a highly structured problem domain that generalizes QCP and Quasigroup with Holes (QWH). Our problem domain also generalizes the popular Sudoku puzzle: In Sudoku the goal is to complete a partially filled 9x9 matrix, with 9 symbols, without repeating a symbol in a row, column, and in each of its 9 3x3 sub-matrix regions. Like QCP and QWH, Sudoku's structure is that of a Latin square, with the additional constraint that all 9 symbols must appear in each of its 9 subregions of size 3x3. Although some Sudoku instances can be challenging for humans, they can be easily solved by current state-of-the art CSP or SAT solvers, even when only using polytime inference methods, without search (Simonis 2005; Lynce & Ouaknine 2006). So, our goal is to produce challenging Generalized Sudoku Problem (GSP) instances for CSP and SAT based search algorithms.

A Generalized Sudoku instance has an arbitrary order n with block regions that can have rectangular or square shape and it can have more than one solution or it can even be unsatisfiable. We show that our Generalized Sudoku problem (GSP) is NP-complete. We also provide a method for generating GSP instances that are guaranteed to have solutions. We refer to this variant as Generalized Sudoku with Holes (GSWH) problem, inspired by the QWH problem (Achlioptas *et al.* 2000; Barták 2005). GSWH instances are generated by "punching" holes into a complete GS instance. A key question concerning GSWH is the impact of different hole punching strategies on problem hardness. We present a method that produces highly balanced GSWH instances: The idea is to balance the number of holes between the regions of the Sudoku, in addition to balancing the number of holes between rows and columns. Our results show that our GSWH instances are harder to solve than QWH instances;

*Research partially supported by projects TIN2004-07933-C03-03, TIN2005-09312-C03-01 and TIC2003-00950 funded by the *Ministerio de Educación y Ciencia*.

furthermore, we also show that hardness increases as a function of the squareness of the block regions. In other words, GS instances with square block regions are harder than GS instances with rectangular block regions, for the same order. Concerning hole patterns, our highly balanced method of punching holes produces instances substantially harder than balanced QWH instances (Kautz *et al.* 2001). An open question, raised by our results, is to what extent our method produces harder instances because it increases the "bandwidth" of the corresponding bipartite graph associated with the hole pattern. Finally, we also show an interesting correlation across different constrainedness regions between an approximation of the number of so-called backbone variables and the complexity of GSWH.

Generalized Sudoku Problems

A valid complete Generalized Sudoku (GS) instance of order s on s symbols, is an $s \times s$ matrix, with each of its s^2 cells filled with one of the s symbols, such that no symbol is repeated in a row, column, or block region. A block region is a set of s pre-defined cells; block regions don't overlap, and therefore there are exactly s block regions in a GS instance of order s. In the case of square region blocks, each block has $\sqrt{s} \times \sqrt{s}$ cells ($\sqrt{s}$ has to be an integer); in the case of rectangular regions, each block has $n \times l$ cells (n columns, l rows), always with $n \cdot l = s$. One can also consider block regions of arbitrary shape. These generalizations provide us with a range of interesting problems, starting from quasigroup with holes (QWH) (Achlioptas *et al.* 2000) (when the block regions correspond to single rows) to standard generalized Sudokus (when the regions are squares). In this paper we consider GS instances with rectangular and square block regions.

The Generalized Sudoku problem (GSP) is defined as follows: Given a partially filled Generalized Sudoku instance of order s can we fill the remaining cells of the $s \times s$ matrix such that we obtain a valid complete Generalized Sudoku instance? GSP is NP-complete, given that it is a generalization of Sudoku, known to be NP-complete (Yato & Seta 2002). To show the NP-completeness of GSP, when the regions are strictly of rectangular shape, with dimensions $n \times l = s$, we use a reduction from completing a partial Latin square (LS) of side l that can be seen as a generalization of the one in (Yato & Seta 2002). The idea is that now we use a GS of side $n \cdot l$ with set of symbols (a, b) with $0 \leq a < l, 0 \leq b < n$; the set of symbols associated with the LS is still the subset $(a, 0)$. To embed the LS into the GSP instance, we fill position $S(i, j)$ of the GSP instance with the symbol:

$$(\, i + \lfloor j/n \rfloor \; mod \, l, \; j + \lfloor i/l \rfloor \; mod \, n) \qquad (1)$$

Observe that in this (totally filled) GSP instance the set of symbols placed in the positions:

$$B = \{(i, j) \mid 0 \leq i < l, \; j \; mod \, n = 0\} \qquad (2)$$

is the subset $(a, 0)$ and that the positions in B form a LS of side l with that subset of symbols. Then, we embed the partial LS of side l by mapping symbol i of the LS to symbol $(i, 0)$ of the GSP and changing the contents of the set of positions B to embed the partial LS in this way:

$$S(i,j) = \begin{cases} (LS(i, j/n), 0) & (i, j) \in B, LS(i, j/n) \neq \perp \\ \perp & (i, j) \in B, LS(i, j/n) = \perp \end{cases} \qquad (3)$$

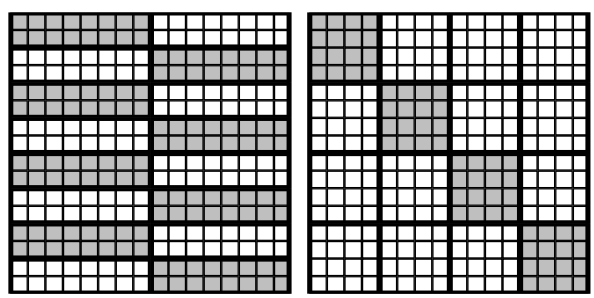
So, the (partially filled) GSP instance with rectangular block regions will have solution if and only if the partial LS has solution.

Generating complete Generalized Sudokus

To generate a GS instance, we follow the approach of building a Markov chain whose set of states includes, as a subset, the set of complete Generalized Sudokus. We use the second Markov chain defined in (Jacobson & Matthews 1996) that considers only "proper" Latin squares as states. Because any GS is also a Latin square of the same order, this chain obviously includes a subchain with all the possible GSs. So, for any pair of complete GSs there exists a sequence of proper moves, of the type mentioned in Theorem 6 of (Jacobson & Matthews 1996), that transforms one into the other.

However, if we simply use the Markov chain by making (proper) moves uniformly at random, starting from an initial complete GS, most of the time we will reach Latin squares that are not valid GSs. To cope with this problem, we select the move that minimizes the number of "violated" cells, i.e. cells with a color that appears at least once more in the same region of the cell. To escape local minima, we select the move uniformly at random every certain number of moves[1]. So, to generate a random GS, we start from an initial GS[2], and we perform the moves described above until we have generated a certain number of valid GSs. Observe that this method does not necessarily generate a GS instance uniformly at random, because we do not always select the next move uniformly at random. However, as we will see in the experimental results, the method provides us with very hard computational instances of the GSWH problem, once we punch holes in the appropriate manner.

Balanced Hole Patterns



8x2 Sudoku 4x4 Sudoku

Figure 1: Examples of Singly balanced hole patterns for Sudoku problems, that allow decomposition in smaller (and easier) problems. Empty cells are in gray.

Kautz et al. (Kautz *et al.* 2001) introduced a method for balancing the hole pattern of a QWH instance, such that the

[1] In our experiments, this number has been fixed to 20, because it works reasonable well with all the orders of Generalized Sudokus that we have tested.

[2] Note that we can trivially generate a Generalized Sudoku of arbitrary order, with arbitrary rectangular or square regions, using equation 1.

number of holes in each row and column is equal. We refer to such a method as ”Singly balanced”. It turns out that this way of balancing does not lead to the hardest Generalized Sudoku instances. To see why, consider the following hole pattern for a Sudoku problem with regions of dimensions $n \times l$ and such that $n \ mod \ l = 0$. We punch holes in all the cells of a same region, in such a way that in every region row (n in total) the number of regions with holes is 1 and in every region column (l in total) this number is n/l. This creates an instance with the same number of holes (n) in every row and column, but that can be decomposed in l smaller independent subproblems that only involve n/l regions each one. Observe that for (standard) Sudoku instances, $n/l = 1$, so we get a set of n trivial subproblems. Figure 1 shows two examples of this hole pattern, one for 8x2 Sudokus and the other for 4x4 Sudokus.

So, the distribution of holes between different regions can make a difference in the difficulty of the problem. So one would like to balance the holes *also* between different regions. Even if the method of (Kautz *et al.* 2001) will tend to distribute the holes approximately uniformly between the regions, it will not *always* ensure that we have exactly the same number of holes in every region when the total number of holes is a multiple of the size of the Sudoku. For that reason, we propose a method for ensuring both balance conditions: same number of holes in every row and column and same number of holes in each block region.

Our new ”doubly balanced” method is again based on a Markov chain, where every state is a hole pattern H with h holes that satisfies both balance conditions. In order to go from one hole pattern to another we use a “switch” (Kannan, Tetali, & Vempala 1997). This kind of move was introduced to build a Markov chain where every state is a regular bipartite graph (with vertices with a given degree), the Markov chain is connected, and it includes all possible such graphs. A hole pattern with the same number of holes in each row and column can be seen as a regular bipartite graph. A switch is defined by two entries $(i, j), (i', j')$ of the GSP instance, such that there is a hole in (i, j) and (i', j') but not in (i, j') and (i', j). If we change both holes to positions (i, j') and (i', j) the row and column hole number does not change. So, we use this Markov chain, but we restrict the moves to those moves that also maintain the number of holes present in each region, i.e., moves such that the regions of (i, j) and (i', j') are either in the same region row or region column, or both. Such moves always exist, even for a hole pattern with only one hole in each region. So, we have the following code for generating a hole pattern H with $q = h/(n \cdot l)$ holes in each row, column and region, using a GS $S(i, j)$ to create the initial pattern considering each symbol as a hole, and then performing s moves trough the Markov chain:

$H = \{ \ (i,j) \mid S(i,j) \in [1,q] \ \}$

Repeat s times:

$T = \{ \ switch((i,j),(i',j')) \ of \ H \mid$
$\lfloor i/l \rfloor = \lfloor i'/l \rfloor \ \vee \ \lfloor j/n \rfloor = \lfloor j'/n \rfloor \ \}$

pick a uniformly random $switch((i,j),(i',j'))$ from T

$H = (H - \{(i,j),(i',j')\}) \ \cup \ \{(i,j'),(i',j)\}$

We also considered a different method for punching holes: This method is based on the rectangular model presented also in (Kautz *et al.* 2001). The rectangular model selects a set of columns (or rows) and punches holes in all the cells of these columns: in the case of QWH, this method produces tractable instances (Kautz *et al.* 2001); they can be solved using an algorithm based on bipartite graph matching. An open question is whether a similar hole pattern in the case of GSP instances also corresponds to a tractable class. Figure 2 shows an example of a solvable 3x3 GSP instance, with a rectangular hole pattern. The initial assignment is indicated by the grayed cells. After two rounds of the bipartite graph matching algorithm, a possible outcome corresponds to the first two columns in the way shown; this configuration cannot be completed into a valid Sudoku (even though there exists a valid completion of the initial partially filled instance). So, the rectangular hole pattern could still provide hard instances for Sudoku. For that reason, we propose a rectangular model for Generalized Sudoku, that distributes a set of c hole columns between the different region columns of the Generalized Sudoku, in a uniform way. The uniform distribution of the hole columns tries to minimize the clustering of holes.

5	1	3		6	4			8
8	4	6		9	7			2
7	2	9		3	1			5
6	3	4	x	7	5			9
2	5	7		1	8			3
9	8	1	x	4	2			6
3	7	5		8	6			1
1	6	8		2	9			4
4	9	2		5	3			7

Figure 2: Grayed cells represent the initial assignment. The cells marked with $\times$ cannot be completed after the two first columns are completed in the way shown

Encodings and Solvers

In this work we consider the best performing encodings for the QCP analyzed in (Ansótegui *et al.* 2004), and we extend them with the suitable representation of the additional alldiff constraints for the regions in the Generalized Sudoku Problem (GSP). We consider a GSP instance on s symbols.

The SAT encoding extends the SAT 3-dimensional (3D) encoding proposed in (Kautz *et al.* 2001) for the QCP. The encoding uses s Boolean variables per cell; each variable represents a symbol assigned to a cell, and the total number of variables is s^3. The clauses corresponding to the 3D encoding represent the following constraints:

1. at least one symbol must be assigned to each cell (alo-cell);
2. no two symbols are assigned to the same cell (amo-cell);
3. each symbol must appear at least once in each row (alo-row);
4. no symbol is repeated in the same row (amo-row);
5. each symbol must appear at least once in each column (alo-column);
6. no symbol is repeated in the same column (amo-column).

Finally, for each region i the clauses we add represent the following constraints:

7. each symbol must appear at least once in each region i (alo-region$_i$).
8. no symbol is repeated in the same region i (amo-region$_i$).

The same SAT encoding is considered in (Lynce & Ouaknine 2006).

The CSP encoding extends the "bichannelling model" of (Dotú, del Val, & Cebrián 2003). It consists of:

- A set of *primal variables* $X = \{x_{ij} \mid 1 \leq i \leq s,\ 1 \leq j \leq s\}$; the value of x_{ij} is the symbol assigned to the cell in the ith row and jth column.
- Two sets of *dual variables*: $R = \{r_{ik} \mid 1 \leq i \leq s,\ 1 \leq k \leq s\}$, where the value of r_{ik} is the column j where symbol k occurs in row i; and $C = \{c_{jk} \mid 1 \leq j \leq s,\ 0 \leq k \leq s\}$ where the value of c_{jk} represents the row i where symbol k occurs in column j.

The domain of all variables is $\{1, \ldots, s\}$, where these values represent respectively symbols, columns, and rows. Variables of different types are linked by channeling constraints:

- *Row channeling constraints* link the primal variables with the row dual variables: $x_{ij} = k \Leftrightarrow r_{ik} = j$.
- *Column channeling constraints* link the primal variables with the column dual variables: $x_{ij} = k \Leftrightarrow c_{jk} = i$.

The concrete CSP encoding we use in our experimental investigation is a Binary CSP represented with nogoods.

Finally, for each region we add the nogoods representing the alldiff constraint over the set of primal variables involved in each region of the Sudoku problem.

For the experimental investigation we considered four state-of-the-art SAT solvers: Satz (Li & Anbulagan 1997), zChaff (Moskewicz *et al.* 2001), MiniSAT (Eén & Sörensson 2003), and Siege[3].

We also considered a CSP solver, a variation of the MAC solver by Régin and Bessière (Bessière & Régin 1996) (MAC-LAH) proposed in (Ansótegui *et al.* 2004) that incorporates the technique of failed literals and the Satz's heuristic in terms of a CSP approach.

Complexity Patterns

Singly Balanced

We consider first the complexity of solving GSWH instances generated with the Singly Balanced method. Our first set of results shows the complexity of Solving GSWH instances with different region shapes, comparing it with the complexity of solving QWH instances. Figure 3 shows the results for GSWH with regions 15x2, 10x3 and 6x5 (size 30) and QWH also of size 30 (the encoding used for QWH is the 3D). We employ 100 instances per point and MiniSAT solver with 5,000 seconds cutoff. We observe similar complexity patterns for all problems, the difference being that as the shape of the block region gets closer to a square, the peak in complexity increases. So, for the same size, the easiest instances are from QWH and the hardest ones are the ones from GSWH, with regions that are almost square.[4] Observe

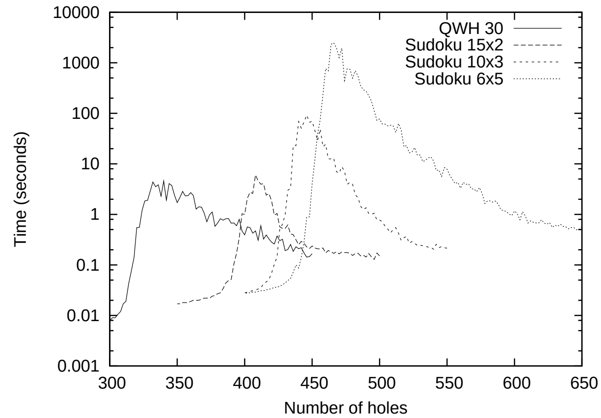


Figure 3: Empirical complexity patterns for singly balanced GSWH instances with different region shapes, but same size

that the difference between QWH and GSWH with square regions is about three orders of magnitude for this size. A possible partial explanation of this fact is the following. For a GSWH instance with regions $n \times l$ and l fixed, when we fix a given cell, the larger n, the more cells in a same region become constrained in each of the regions that intersect with the row of the fixed cell. So, when fixing a cell, for a large n some regions will become more constrained than others, and this may be an advantage for simplifying the problem where look-ahead heuristics can take advantage of. By contrast, for square regions, fixing a cell constrains the same number of cells (if the current hole pattern is balanced) in all the regions that intersect with the row or the column of the fixed cell. So, again it seems that balance is making a difference in the complexity of this problem.

We observe the same qualitative behavior when using different SAT algorithms. The main difference is in the magnitude of the peak of the complexity curve. Figure 4 shows a plot with the performance of different algorithms in the critically constrained area for different GSWH problems. The plot shows, for different sizes and different algorithms, the percentage of solved instances from a test-set of 200 instances, when using a cutoff of 10^4 seconds. For small sizes, all algorithms solve almost all the instances. But as the size increases, the solver MiniSAT clearly outperforms the other solvers.

Doubly Balanced

Next, we consider the Doubly balanced method for punching holes. When using this method, the typical hardness of the GSWH instances seems to be very similar to the singly balanced method, for small sizes; however, as we increase the size of the instances, we see a dramatically difference in computational hardness. This is probably due to the fact that the singly balanced method tends to distribute the holes uniformly between regions in such a way that the difference with respect to the doubly balanced method is not significant for small instances. This can be quantified by looking at the percentage of solved instances from a test-set with 500 instances, for both methods, when working with a cutoff of 5.000 seconds. Table 1 shows these values. The values are almost the same for 5x5 Sudokus, but for the instances of

[3] Available at http://www.cs.sfu.ca/~loryan/personal

[4] Since 30 is not a perfect square, we cannot have perfectly square regions.

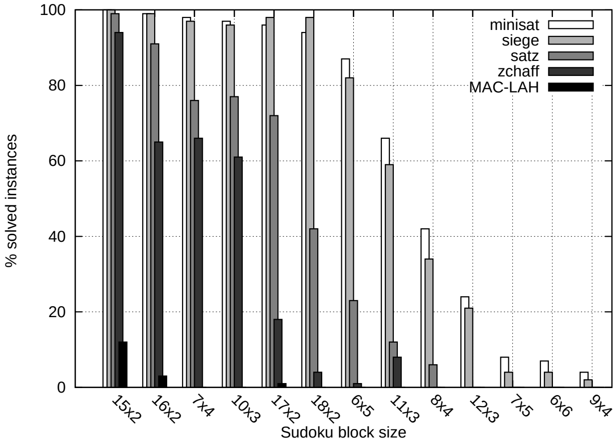


Figure 4: Empirical complexity patterns for singly balanced GSWH instances with different region shapes

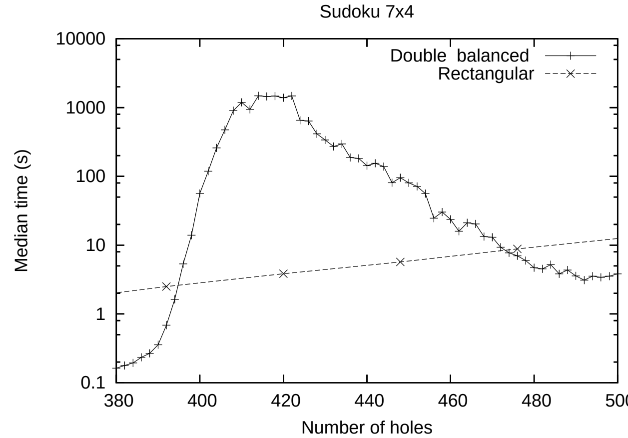


Figure 5: Comparison of the hardness of instances generated using the doubly balanced method versus the rectangular model.

region	holes	singly balanced % solved	doubly balanced % solved
5x5	344	98.8	98.8
7x4	414	71.6	67.4
17x2	504	48.6	37.8
18x2	572	33.8	24.9
6x5	480	18.6	11.6

Table 1: Comparison of percentage of solved GSWH instances generated with both methods for putting holes, for instances in the peak of hardness

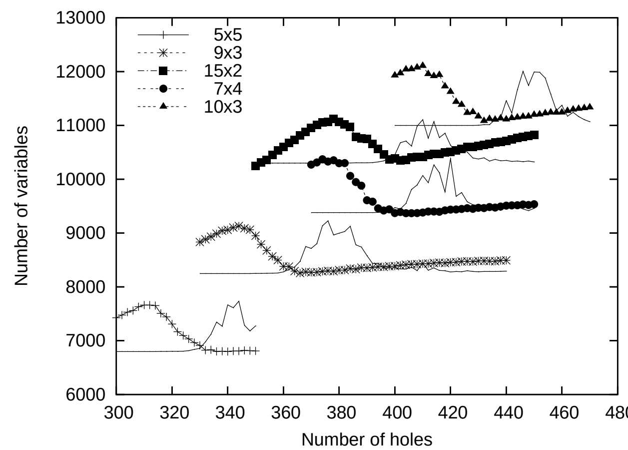


Figure 6: Look-ahead backbone and normalized complexity patterns for Sudokus with different regions

higher orders, we observe an increase in the hardness of the instances. This is reflected in the decrease in the number of solved instances, for a given cutoff, when using the doubly balanced method, in comparison to the singly balanced one.

So, our doubly balanced method generates harder instances than the ones produced by balanced QWH, they are also guaranteed to be satisfiable, and therefore they constitute a good benchmark for the evaluation of local and systematic search methods.

Rectangular model

For the rectangular model, in which holes are punched across entire rows (or columns), our empirical results with Satz do not seem to show a clear exponential scaling cost as the size is increased. Figure 5 shows the results of solving 7x4 GSWH instances with the rectangular and the doubly balanced model methods. We observe that the complexity of solving the 7x4 problem is much higher for the balanced model. Also, the fact that for Satz, with the problems tested so far, the hardest instances of the rectangular model are the ones obtained with the maximum number of holes (an empty Sudoku) seems to indicate that this pattern is not inherently hard. A key question is whether this pattern corresponds to a tractable class.

Backbone

In this section we discuss results about the correlation between the backbone of the satisfiable GS instances and computational hardness. It is known that computing the exact backbone is an intractable problem so we consider only an approximation of the full backbone. In particular, we use the look-ahead backbone provided by Satz. This is the set of variables that Satz discovers to have a unique value by checking all possible assignments with unit propagation and fixing every discovered backbone variable, until no new backbone variable is discovered.

In our approximation of the backbone, we consider the fraction of fixed variables by Satz over the number of variables of the (satisfiable) encoded instance. Our satisfiable instances are obtained after preprocessing our instances for discarding the cells that are discovered to have a unique possible value due to the initial partial assignment and the propagation of the Sudoku constraints. Figure 6 shows the evolution of the backbone together with the complexity of solving instances, for different region shapes, but normalized so that the value at the peak of hardness coincides with the maximum number of look-ahead backbone variables. We observe that this approximated backbone fraction starts to increase until it reaches a point where it decreases abruptly and then it again starts to increase, but this time more slowly. The point where it reaches the minimum value is around the value where the hardness starts to increase towards its peak. So, this point of "suddenly" decrease in the backbone fraction can be used as a sign for the beginning of the hard region of the problem. It is remarkable that even though the backbone is hard to approximate (Kilby *et al.* 2005), in this problem this approximated backbone provides a valuable information.

We have obtained an approximated location of this minimum point through a doubly exponential regression model, using the location of this minimum value for all possible Sudoku problems with different region forms ($n \times l$), from size 26 to 49. The model obtained is:

$$holes = e^{0.537} * n^{1.57} * l^{1.72}$$

The coefficient of regression (R^2) is 0.989, thus indicating that the model is quite good. We have also obtained an analogous regression model, but for the location of the hardness peak. For this model we used data obtained experimentally with Satz, but only for problems with sizes ranging from 26 to 30. The model obtained is:

$$holes = e^{-0.217} * n^{1.8} * l^{1.97}$$

Again, we obtain a high value for R ($R^2 = 0.997$). Observe that for the hardest problems ($n = l$), the relative difference between these two points is only $O(n^{1.14})$, much smaller than the whole range of possible holes (n^4). So, as n increases, the width of the hard part of the phase transition seems to decrease, in a normalized scale.

Conclusions

In this paper we show how different strategies for generating Generalized Sudoku instances, based on the shape of the block regions and the balance of the initial empty cells, can dramatically impact the computational hardness of the instances. Our Generalized Sudoku problem generator produces instances that are several orders of magnitude harder than other structured instances of comparable size. We believe our Generalized Sudoku generator should be of use in the future development of systematic and stochastic local search style CSP and SAT methods.

References

Achlioptas, D.; Gomes, C.; Kautz, H.; and Selman, B. 2000. Generating satisfiable problem instances. In *Proc. of AAAI-00*, 193–200.

Ansótegui, C.; del Val, A.; Dotú, I.; Fernández, C.; and Manyà, F. 2004. Modelling choices in quasigroup completion: Sat vs csp. In *Proc. of AAAI-04*.

Barták, R. 2005. On generators of random quasigroup problems. In *Proc. of ERCIM 05 Workshop*, 264–278.

Bessière, C., and Régin, J.-C. 1996. Mac and combined heuristics: Two reasons to forsake fc (and cbj?) on hard problems. In *CP*, 61–75.

Dotú, I.; del Val, A.; and Cebrián, M. 2003. Redundant modeling for the quasigroup completion problem. In *CP03*.

Eén, N., and Sörensson, N. 2003. An extensible sat-solver. In *Proceedings of SAT 2003*.

Gomes, C., and Selman, B. 1997. Problem structure in the presence of perturbations. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, 221–227. New Providence, RI: AAAI Press.

Gomes, C.; Fernández, C.; Selman, B.; and Bessière, C. 2004. Statistical regimes across constrainedness regions. In *Proceedings CP'04*.

Gomes, C. P.; Selman, B.; and Crato, N. 1997. Heavy-tailed distributions in combinatorial search. In *Proceedings of the Third International Conference of Constraint Programming (CP-97)*. Linz, Austria.: Springer-Verlag.

Hulubei, T., and O'Sullivan, B. 2006. The impact of search heuristics on heavy-tailed behaviour. *Constraints* 11(2).

Jacobson, M. T., and Matthews, P. 1996. Generating uniformly distributed random latin squares. *Journal of Combinatorial Design* 4:405–437.

Kannan, R.; Tetali, P.; and Vempala, S. 1997. Simple markov-chain algorithms for generating bipartite graphs and tournaments. In *Proc. of the eighth annual ACM-SIAM Symposium on Discrete Algorithms*, 193–200.

Kautz, H.; Ruan, Y.; Achlioptas, D.; Gomes, C.; Selman, B.; ; and Stickel, M. 2001. Balance and filtering in structured satisfiable problems. In *Proc. of IJCAI-01*, 193–200.

Kilby, P.; Slaney, J.; Thiebaux, S.; and Walsh, T. 2005. Backbones and backdoors in satisfiability. In *Proc. of AAAI-05*, 193–200.

Li, C. M., and Anbulagan. 1997. Look-ahead versus look-back for satisfiability problems. In *Proc CP'97*, 341–355.

Lynce, I., and Ouaknine, J. 2006. Sudoku as a sat problem. In *Proc. of Ninth International Symposium on Artificial Intelligence and Mathematics*.

Moskewicz, M.; Madigan, C.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an efficient sat solver. In *Proceedings of 39th Design Automation Conference*.

Refalo, P. 2004. Impact-based search strategies for constraint programming. In *Proceedings CP'04*.

Régin, J. C., and Gomes, C. 2004. The cardinality matrix constraint. In *Proceedings CP'04*.

Shaw, P.; Stergiou, K.; and Walsh, T. 1998. Arc consistency and quasigroup completion. In *Proceedings of the ECAI-98 workshop on non-binary constraints*.

Simonis, H. 2005. Sudoku as a constraint problem. In *Proc. of Fourth International Workshop on Modelling and Reformulating Constraint Satisfaction Problems, CP 2005*, 13–27.

Yato, T., and Seta, T. 2002. Complexity and completness of finding another solution and its application to puzzles. In *Proc. of National Meeting of the Information Processing Society of Japan (IPSJ)*.