Constraint Generation
via Automated Theory Formation

Simon Colton\textsuperscript{1} and Ian Miguel\textsuperscript{2}

\textsuperscript{1}Division of Informatics
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN, UK
simonco@dai.ed.ac.uk

\textsuperscript{2}Department of Computer Science
University of York
Heslington
York YO10 5DD, UK
ianm@cs.york.ac.uk

1 Introduction

Adding constraints to a basic CSP model can significantly reduce search, e.g. for Golomb rulers [6]. The generation process is usually performed by hand, although some recent work has focused on automatically generating symmetry breaking constraints [4] and (less so) on generating implied constraints [5]. We describe an approach to generating implied, symmetry breaking and specialisation constraints and apply this technique to quasigroup construction [10].

Given a problem class parameterised by size, we use a basic model to solve small instances with the Choco constraint programming language [7]. We then give these solutions to the HR automated theory formation program [1] which detects implied constraints (proved to follow from the specifications) and induced constraints (true of a subset of solutions). Interpreting HR’s results to reformulate the model can lead to a reduction in search on larger instances. It is often more efficient to run HR, interpret the results and solve the CSP, than to solve the problem with the basic model alone.

2 System Architecture

The HR program [1] [2] performs theory formation in domains of pure mathematics. When used in finite algebraic domains such as quasigroup theory, given some examples of the algebra, HR invents new concepts and makes and proves theorems using the Otter theorem prover [8]. Given a basic model of a family of quasigroup CSPs, we employ the following 5-stage approach:

[1] We use Choco to produces solutions for small instances.
[2] HR is employed to form a theory around the examples supplied by Choco.
[3] We interpret HR’s results as implied and induced constraints for the CSP.
[4] We remodel the problem using the additional constraints and see which, if any, reformulations increase efficiency for the small instances.
[5] We add any constraints which improve efficiency to the CSP model and look for solutions to larger problem instances.
We look for both concepts and theorems in HR’s output. Theorems can potentially be added as implied constraints to a basic CSP model. Any concept which specialises the notion of quasigroup can be used in two ways. Firstly, it can be used as a case split: we remodel the CSP twice to specify the quasigroups with and without the specialised property. Performing both searches covers the space, but splitting it in this fashion can introduce symmetry breaking constraints, thus reducing overall search. Secondly, if we are only interested in finding an example, rather than exhausting the search space, we can choose to look for solutions to the specialised CSP only (which will be solutions to the original problem).

3 Quasigroup Generation Experiments

Quasigroups are finite algebras where every element appears in every row and column, i.e. Latin squares. Quasigroups of every size exist, but for certain specialised classes of quasigroups, there are open questions about the existence of examples. Such classes include those termed QG3-QG7, which are quasigroups with these additional axioms: QG3: \((a * b) * (b * a) = a\), QG4: \((b * a) * (a * b) = a\), QG5: \((b * a) * b = a\), QG6: \((a * b) * b = a * (a * b)\), QG7: \((b * a) * b = a * (b * a)\). Constraint satisfaction approaches to existence questions have been very successful, e.g. size 12 QG3 quasigroups, settled by Slaney [10].

To find a quasigroup of size \(n\), we used \(n^2\) variables, \(x_{(ij)}\), with domain \(\{1, 2, \ldots, n\}\). The quasigroup constraint imposed an all-different on each row and column, and the constraints imposed by the quasigroup type were implemented via sets of implication constraints. For each quasigroup class, we ran Choco for increasing sizes until 10 million backtracks were reached. For small orders, Choco constructed all solutions of each size and HR removed isomorphic copies.

For each class, we ran HR with full functionality for 45 minutes with the examples from Choco. Then, for a further 15 minutes, we turned off the theorem proving abilities, so that HR performed a best first search for concepts only (using the coverage heuristic measure discussed in [3]). On average, after each theory was formed, there were around 150 prime implicates (implication theorems where no proper subset of the premises implies the goal) and 100 concepts of which around 10 were specialisations suitable for case splits. The reformulations we made for each class are summarised below.

For QG3, Choco produced 4 non-isomorphic quasigroups from which HR formed a theory. We noticed this prime implicate: \(a \in Q \rightarrow \exists b \in Q \text{ s.t. } b * b = a\), meaning that every element must appear on the diagonal of the multiplication table, i.e. an all-different constraint on the diagonal (constraint \(C_{3,1}\)). Next we noticed this theorem: \(\forall a, b, c \in Q \left( \exists c \in Q \text{ s.t. } a * c = c * a = b \rightarrow a * a = b \right)\).

Since \(a * c = b\) and \(a * a = b\) and it is a quasigroup, \(a = c\). Hence, for all elements \(a\), the only other element \(a\) commutes with itself, i.e. QG3 quasigroups are anti-Abelian: no pair of distinct elements commute, which we interpreted as constraint \(C_{3,2}\): \(\forall i, j \left( i \neq j \rightarrow x_{(i,j)} \neq x_{(j,i)} \right)\). HR also found this prime implicate: \(a * a = b \rightarrow b * b = a\), which highlights a symmetry on the diagonal, i.e. if \(x_{(a,a)} = b\), then \(x_{(b,b)} = a\), (constraint \(C_{3,3}\)). HR also made
Table 1. Quasigroups class 3 and 4 results. Dash: no solutions found after \(10^8\) backtrack results. Some specialisations, including quasigroups with symmetry of left identities, i.e. \(\forall a, b \quad (a * b = b \rightarrow b * a = a)\), interpreted as constraint \(C_{3.4}\). We used the specialisation constraints to specialise the model.

As shown in Table 1, using combinations of constraints \(C_{3.1}\) to \(C_{3.4}\), we re-formulated the problem in 8 additional ways. We tested whether the re-formulations reduced (a) the number of backtrack (b) the number of nodes and (c) the CPU time to solve the CSPs. In order to test the relative effectiveness of the re-formulations with different search strategies, we ran Choco with both a lexigraphic column-wise variable ordering beginning in the top left-hand corner, and the smallest-domain first heuristic. Results are presented in Table 1. For QG4, HR found a similar theory to that for QG3 and all the same theorems held. As we found no better results, we used the same re-formulations for QG4 as for QG3, with the results also presented in Table 1. For QG4, in re-formulation R_{a.9}, we also used specialisation constraint \(C_{4.5}\), idempotent quasigroups: \(\forall a \quad (a * a = a)\).

The implied constraints are clearly beneficial to the solver. Choco did not solve any instance above order 6 using the basic model, but with the implied
constraints, Choco solved instances of orders 7 and 8 for both QG3 and QG4. Variable ordering is also important when using the implied constraints, because, while $R_{3,2}$ and $R_{4,2}$ (anti-Abelian) were the least effective reformulations using the lexicographic ordering, they were the most effective when using the smallest domain heuristic. The heuristic forces Choco to jump around the quasigroup table, using the extra pruning given by the anti-Abelian constraint.

None of the reformulated models containing implied constraints only solved the order 9 problem within the specified limits. However, some of the induced models did solve this problem quickly. For QG3, reformulation $R_{3,8}$ (symmetry of left identities) allowed an instance of order 9 to be found in 20 seconds. Similarly, reformulation $R_{4,9}$ (idempotency) found an instance of QG4, size 9. This shows the value of induced constraints: searching for specific quasigroup types reduces the effort required so that a solution is obtained relatively easily.

As discussed in [3], for classes QG5-QG7 we did less analysis of HR’s output, making one reformulation for each. For QG5, we used this result from HR: $\forall a, b \in Q \ (a * b = a \leftrightarrow b * a = a)$ to reformulate the problem. This significantly outperformed the basic model by all measures, finding an instance of order 9 which the basic model could not. When the basic model did solve the problem, it was much slower than the reformulated model. This trend increased with problem size, and easily justified the time spent on reformulation. The smallest domain heuristic was always beneficial to this model, taking advantage of its extra pruning power, but was of limited value to the basic model.

For QG6 and 7, HR re-discovered the theorem stated in [10] that both quasigroup types are idempotent (i.e. $\forall a, (a * a = a)$). We added this constraint to produce two reformulations (see [3]). Using the smallest domain heuristic with the basic model, QG6 and QG7 were solvable up to orders 9 and 10 respectively, matching the abilities of the reformulated idempotent models. As with QG5, however, the decrease in search offered by the reformulated models was significant and increased with problem size. For both QG6 and QG7, the smallest domain heuristic made a substantial saving, suggesting that the structure of these problem classes is such that the solver must be allowed to focus on the most constrained areas of the quasigroup table to be most efficient.

4 Conclusions and Further Work

A more complete account of this work with additional applications to group theory and Balanced Incomplete Block Designs is presented in [3]. We have demonstrated that HR can find implied and induced constraints for CSPs and that reformulating the model to include these additional constraints gives a clear improvement in efficiency, even considering the time taken to run HR, interpret the results and re-formulate the CSP. The implied constraints produced a consistent, significant, speedup, yet only with both implied and induced constraints were we able to find solutions to the larger problems.

So far, our approach has been interactive, whereby we interpret HR’s results and use them to reformulate the CSP. We intend to automate the interaction be-
between HR and the solver, eventually using them in a cycle whereby the examples
found by solver feed HR's theory formation, which in turn generates constraints
to improve the solver's performance. This may be problematic, as some implied
constraints may not improve the search at all, and combining implied constraints
may reduce efficiency, because one constraint subsumes another. It is therefore
likely that the pruning phase will be important for a fully automated approach.

The question of how to reformulate CSPs automatically in general needs
much more research. The system we have described could be applied to other
problem classes, such as tournament scheduling [9], to shed further light on au-
tomating this process. We hope to have added to the evidence that reformulating
CSPs, in particular by adding implied and induced constraints, can dramatically
increase efficiency, and to have shown that automating certain aspects of this
process is certainly possible and a worthy area for future research.

Acknowledgments

The first author is also affiliated to the Department of Computer Science, Uni-
versity of York. We thank Toby Walsh and Alan Bundy for their continued input.
This work is supported by EPSRC grants GR/M98012 and GR/N16129.

References

2. S Colton, A Bundy, and T Walsh. HR: Automatic concept formation in pure
   http://www.dcs.st-and.ac.uk/~apes/apesreports.html.
5. A Frisch, I Miguel, and T Walsh. Extensions to proof planning for generating
   implied constraints. In Proceedings of the 9th Symposium on the Integration of
   the Golomb ruler problem. In Proceedings of the 3rd International Workshop on
   Integration of AI and OR Techniques (CPAIOR-01), 2001.
7. F Laburthe and the OCRE group. Choco: implementing a CP kernel. In Pro-
   ceedings of the CP00 Post Conference Workshop on Techniques for Implementing
10. J Slaney, M Fujita, and M Stickel. Automated reasoning and exhaustive search:
    Quasigroup existence problems. Computers and Mathematics with Applications,