Algebraic Constraint Programming*

Ian P. Gent†

1 Summary

I aim to introduce Algebraic Constraint Programming as a new means of solving hard search problems such as scheduling and optimisation. This will be done by exploiting and integrating the computational power of two mature technologies, Constraint Programming and Computational Algebra.

To achieve my aim, my objectives are as follows:

1. To exploit algebraic structure in constraint programs, enabling constraint programs to be written much more simply and effectively.
2. To introduce algebraic algorithms for constraint propagation.

In the rest of this document, I first explain some background material and a motivating example for the research, and then outline the research I propose.

2 Background

Combinatorial Search and Constraint Programming Combinatorial search is arguably the most fundamental aspect of Artificial Intelligence (AI). Some of the most basic research questions in search remain open, the most obvious example being the question P=NP? The research area is extremely active. At the same time, the existing technology of search in AI has become very important commercially, through the area known as Constraint Programming (CP). CP relies on efficient 'propagation' algorithms, deducing derived consequences of current facts. Software packages such as Eclipse from IC-Parc and Solver from ILOG are used widely on problems such as workforce management at BT and BA, resulting in savings of many millions for the companies concerned.

Computational Algebra There is now a wealth of research into computational techniques for problems in group theory and other algebraic systems. Built on this are remarkably powerful computer programs. For example, the system GAP contains millions of lines of code, some of it peer-reviewed in a style similar to journal papers [2]. A flexible interpreter allows deep questions to be answered easily, while GAP also serves as a programming language with access to primitives performing complex operations on groups.

A motivating example: All-different The 'All-different' constraint occurs very frequently in constraint programs. For example, one states that exam times for an individual student's courses must all be different. Régin's algorithm makes constraint propagation very efficient [6], but this is only available to a user if it has been implemented specially in the constraint language being used.[1] On n variables, the constraint can be decomposed into $n(n-1)/2$ separate not-equal constraints, without changing the set of solutions. But this has many disadvantages. At the same time as enlarging the space and run time requirements, code has to be written to express all $O(n^2)$ constraints, and less propagation can be performed than in the specialised algorithm. Yet the constraint can be *presented* very efficiently algebraically. It is simply the constraint $x \neq y$, acted on by the symmetric group on all the variables to be made different.

3 Proposed Research

The aim of Algebraic Constraint Programming is to allow algebraic structure in constraints to be used as the basis for a constraint modelling language, and then to implement specialised algebraic algorithms for constraint

*Generously supported by a Royal Society of Edinburgh SEELD/RSE Support Research Fellowship, for which I am very grateful.

†School of Computer Science, University of St Andrews, St Andrews, Fife, KY16 9SS. Email ipg@dcs.st-and.ac.uk.

[1] For example, it is available in ILOG Solver but not in Eclipse, because Régin works for ILOG!

propagation.

Algebraic Structure in Constraint Programs There have been efforts to introduce new modelling languages to simplify the construction of logic programs: for example the language OPL provides a modelling front-end to ILOG Solver [5]. A language such as OPL does simplify the statement of constraints like All-different, but with two problems. First, the result in the target constraint programming language is the explicit presentation of each of the $O(n^2)$ constraints, with the disadvantages mentioned above. Second, it provides little help expressing more complicated combinations of constraint. For these, the power of computational algebra is necessary, as for example in the Alien Tiles puzzle [3]. In this example, we wrote a GAP program to calculate the 729 necessary constraints, and wrote these out as an ILOG Solver program. These constraints would have been very hard to write by hand, and to automate their construction in any non-algebraic language would have meant reinventing many wheels already turning in GAP. While we retained the disadvantage of writing out constraints explicitly, we demonstrated the expressive power of GAP as a high level constraint language. In this section of my research, I will first provide an interface between the algebraic language and the constraint programming system, and second provide constraint constructs in GAP to exploit this interface. This will greatly simplify the coding of algebraic constraint programs. A particular advantage will be the easier exploitation of symmetry in constraint programs, based on the method I introduced with Barbara Smith [4]. This research is now funded by EPSRC and is laying some groundwork for the initial integration of constraint programming and computational algebra.

Algebraic Propagation Algorithms The most exciting aspect of my research will be to develop novel constraint propagation algorithms exploiting the algebraic structure of constraints. The best algorithm for general constraint propagation is GAC-schema [1]. This algorithm allows special purpose algorithms to be incorporated for specialised sets of constraints. This is ideal for the integration of algebraic forms of reasoning. This has the huge advantage that it allows a set of constraints, as expressed in the algebraic modelling language, to be treated as one object. The key step in GAC-schema is the calculation of a 'supporting tuple' for a variable's value, a set of values of the other variables in a constraint allowing the given value. When expressed as one object in an algebra, the calculation of a supporting tuple becomes a question on the existence of an object in the algebra. Algebraic propagation algorithms will therefore be based on calculating these tuples directly from the algebra in GAP. In fact, it will not be necessary to pass constraints from the algebraic model to the constraint system at all, eliminating many of the disadvantages listed earlier. Instead, we can make calls to GAP as part of the GAC-schema algorithm when new supporting tuples are needed, propagating results to other constraints not expressed algebraically. This will significantly speed search, and lead to improved propagation. The result will be a fully integrated Algebraic Constraint Programming language.

The research directions I have proposed above are clear cut and achievable. However, the research also has risks that I would not wish to minimize. I would highlight one particular risk. The new propagation algorithms might not be able to perform significantly more efficiently than standard techniques. Indeed, Régin's algorithm for All-different is so good that it is unlikely to be beatable. My belief is that we can obtain general algorithms for many cases that will greatly improve on existing technology where no account is made of algebraic structure. This will significantly improve constraint programming because the number of constraints where special algorithms are available is a tiny fraction of the constraints used every day in constraint programming.

References

[1] C. Bessière and J.C. Régin. Arc consistency for general constraint networks: Preliminary results. In *Proceedings IJCAI-97*, pages 398–404, 1997.

[2] The GAP Group, Aachen, St Andrews. *GAP – Groups, Algorithms, and Programming, Version 4*, 1998. (`http://www-gap.dcs.st-and.ac.uk/~gap`).

[3] I.P. Gent, S.A. Linton, and B.M. Smith. Symmetry breaking in the alien tiles puzzle. Technical Report APES-22-2000, APES Research Group, 2000. Available from http://www.dcs.st-and.ac.uk/~apes/apesreports.html.

[4] I.P. Gent and B.M. Smith. Symmetry breaking in constraint programming. In W. Horn, editor, *Proceedings of ECAI-2000*, pages 599–603. IOS Press, 2000.

[5] P. Van Hentenryck. *The OPL Optimization Programming Language*. MIT Press, 1999.

[6] J-C Regin. A filtering algorithm for constraints of difference in csps. In *Proceedings AAAI'94*, pages 362–367, 1994.