Constraint Programming Letters 1 (2007) 7-13 Submitted 5/2007; Published 11/2007

Constraint Programming - the Paradigm to Watch

Mark Wallace MARK.WALLACE@INFOTECH.MONASH.EDU.AU
Faculty of Information Technology
Monash University
Clayton, Vic 3800, AUSTRALIA

Editor: Pascal Van Hentenryck

1. The Big Picture

Computer hardware has improved much more than computer software over the last 50 years. Moore's law for its rate of performance improvement is an astounding testimony to the success of research and development in computer technology. This stunning progress has been complemented by the internet revolution, resulting from a marriage of computation and communication, which fundamentally changes the role of computation - and communication - in our society.

Over the last fifty years there has also been a huge investment in software research and development, and this has yielded some significant benefits. Indeed the internet revolution was made possible by advances in both hardware and software.

However the early visions of software researchers for provably correct programs; computers that can perceive, think, and communicate like people; and the automatic compilation of high-level specifications into computer programs, have not been realised. Like many other ventures, software research started out with high expectations, and when the goals proved more challenging than expected some disillusionment set in for a period. Now I believe this period is over, and we have positive and realistic objectives and expectations for software research.

It is striking how much opportunity has now opened up for software advances to make a massive impact. We have:

- Unimaginable computing power
- Masses of up-to-date data
- Huge numbers of users online almost permanently

The old software challenges remain - and we will not lose sight of them! - but many new challenges have arisen:

- Supporting information retrieval, maintenance and communication for communities of people of different sizes from 1 to a billion
- Modelling and optimising the behaviour and plans of organisations, or communities, in relation to other organisations with whom they interact

- Supporting negotiation, cooperation, commitments and payments between organisations

The ubiquity of computation in all manner of devices from watches to vehicle braking systems, telephones, heart pacemakers, automated share trading mechanisms and power stations, means that we depend absolutely on correctly functioning hardware and software for our lives and the functioning of our society. At the same time the increasing performance of computer chips has finally encountered limits and as a result increased computing power is being achieved by multiplying the processors instead of miniaturising them.

Software research over the years has continually developed new paradigms, formalisms and languages to express them. It has been said that all software research reduces to different compiler projects. Most software research focuses on functionality that is implemented in some underlying programming language, rather than the language itself, yet the language constrains much of what can be implemented on top. New paradigms - like the web - require new languages, such as XML.

The constraint programming paradigm has been curiously underexploited. CP arrived at a time when the disillusion with software research was at its peak. Research investment and focus was on making the best of what we had rather than exploring newly available programming paradigms. CP nevertheless prospered in certain areas - in particularly industrial combinatorial optimisation problems. Yet CP offers scope for exploitation that has hardly been noticed let alone harnessed.

CP is the ideal paradigm for encoding correct programs that must run efficiently on multiprocessor hardware. The embedding of highly efficient specialist constraint solvers within a declarative host programming language offers an ideal platform for building correct, efficient software that can run on multiprocessor hardware without modifications to the language.

The facility to build in one or many different solvers into a CP system provides exactly the flexibility needed to serve for information representation and retrieval, system modelling and optimisation and even negotiation and contracts. For example the article by Hassan Ait Kaci in this volume, describes how CP can provide a formally correct and efficient operational base for the Semantic Web.

CP is ideal for modelling because of its support for encapsulation (a predicate in a declarative language is a perfect example of encapsulation), and modularity (also an immediate consequence of its declarative semantics). It also supports optimisation through its constraints solvers: indeed CP has been used as a basis for integrating optimisation techniques from Operations Research, Artificial Intelligence and Mathematical Programming. *CP is a programming paradigm whose time has come - no other paradigm is so well suited to meet so many of today's software challenges in the context of today's computer architectures.*

2. CP in Theory

The great deficiency of Prolog - and other versions of declarative programming - is that for almost every problem, the algorithm Prolog uses to solve it is exponential.

With this background it is extraordinary that the first successes of CP, implemented as an extension to Prolog, was in *efficiently* solving complex problems, such as car-sequencing (Dincbas et al., 1988b) and cutting-stock (Dincbas et al., 1988a).

While Prolog wilfully ignored the issue of worst-case complexity, the theory of CP has increasingly focussed on this very issue. A typical result is the design of a new algorithm for constraint propagation showing how much inference is achieved at what cost in worst-case complexity. Of course the program from which the constraint is invoked almost always suffers from exponential complexity! However if a constraint performs more inference with a smaller complexity bound, then the performance of the program using this constraint is likely also to improve. Indeed if the search behaviour is not changed but merely pruned, then the program performance is guaranteed to be better. This kind of theoretical result is relevant but not revolutionary.

While the ultimate complexity hypothesis $P \neq NP$ is unlikely to be solved by someone in the CP community in the next ten years, the CP paradigm does engage and exploit a useful line of theory delineating broader and broader classes of tractable and intractable constraints and problems involving them (Cohen and Jeavons, 2006). For brevity we shall call large scale real-life problems from industry, government, health, education, environment etc. **LSCO** problems ("large scale combinatorial optimisation" problems). While there are LSCO problems that fall into the tractable (polynomially solvable) classes these are typically only subproblems of a larger NP-hard problem. However even though LSCO problems are almost all NP-hard, the study of tractable problem classes is important.

The real key to solving LSCO problems is to separate the core NP-hard problem from the tractable remainder of the problem. When the optimal solution to the core is found, the problem of finding an optimal solution to the full problem can be solved polynomially.

Methods for eliciting the core from an arbitrary problem are of great practical importance. Finding a core with fewer variables can dramatically reduce the time required to find an optimal solution. Naturally if $P = NP$ then the core of every NP-complete problem is empty: in other words we cannot find the minimal core unless we can determine whether $P = NP$. In general we can only find a "hard" part of the problem which is a superset of the core (or more correctly *a* core). However even non-optimal techniques for decomposing an arbitrary problem into its "hard" and tractable parts would be very important contributions. There is some similarity with research into finding the *backbone* (Parkes, 1997) of a problem - except that problems with backbones may themselves have no tractable part. The backbone is a property of a problem instance rather than a property of a problem class.

A second interesting and problematic notion is that of a search heuristic. Naturally for NP-hard problems no heuristic can be guaranteed to work for all problem instances, as formally established by the "no free lunch" theorems (Wolpert and McReady, 1997). There are classes of problems for which certain search heuristics appear to work well (such as the earliest available variable choice heuristic for scheduling problems). The CP research community has performed little analysis of the boundaries of the classes of problems for which specialised heuristics perform well. The analysis could be experimental - by generating random problems from the class and applying/not applying the heuristic - or theoretical by deriving the expected search tree size for problems of that class with and without the heuristic. The interesting research questions ask what most simple problem class benefits from the heuristic; what, if any, additional constraints destroy the benefits of the heuristic; how it is impacted by cost functions; and how robust the heuristic is to other changes in the problem class.

3. CP Platforms

CP is first and foremost a programming paradigm, and therefore the key research contribution in CP is an implementation of the paradigm. CP only has a future if there are successful working CP systems.

The first generation of CP implementations were dogged by secrecy. The platforms were seen as so innovative and valuable that the organisations funding them insisted on maintaining a lead by keeping implementation details hidden. At the same time the functionality of these systems was being continually enhanced driven by application demand and ongoing research. These enhancements included new types of solvers, new search facilities and new global constraints, as well as new ways of combining solvers and search, and facilities for interfacing to other software as well as to end users.

The result was that prototypes implementing new advances in CP were often standalone, and the major platforms had other application-specific enhancements which were inaccessible to the community. Many researchers built their prototypes from scratch, used them for experimenting with a new research idea, but then had no way to maintain the new feature and integrate it with other features designed and implemented by other researchers.

Standardisation has obvious advantages for the user of a technology, and obvious disadvantages for its ongoing research and development. However with the second generation of CP systems, it has become more feasible to build an architecture that locks in stable facilities, and allows room for enhancements in the research directions that we have now learnt to be fruitful.

In the next ten years it is crucial that researchers contribute their ideas and implementations to CP platforms that:

- are maintained and made available to other researchers in the future
- include both the full range of "standard" facilities, as well as other new features introduced by other researchers

Naturally this approach requires open access to these platforms, and requires researchers to adopt these platforms for their research and experiments, rather than resorting to C++ or Java, in order to avoid "wasting" time on learning how to use a CP platform, that is always changing as more facilities are added. The free availability of the ECLiPSe CP platform (Apt and Wallace, 2007) is an exciting step in this direction. Other open source implementations have been made available before (Diaz and Codognet, 2001) (Laburthe, 2000), but none with the wide range of facilities and the programming environment of ECLiPSe.

One important step in this direction is standardisation on problem modelling facilities: most obviously by establishing a standard problem modelling language. Once all problems are expressed using the same language, then future platforms will be built to interface to this language. The benefits of a shared modelling language include better communication between researchers from different groups, more and better comparison of research results, and an easier way "in" for new researchers and users of CP technology. This author is involved in one proposal for such a standard modelling language, called Zinc (Garcia de la Banda et al., 2006).

4. Users and Applications of CP

CP is well-positioned to become a mainstream programming language, with the advent of multi-processor desktops and laptops, with more and more processors per machine. Meanwhile the role of CP in expressing correct programs and especially for reasoning about program behaviour is an extremely important and still somewhat underdeveloped strength (but see Delzanno and Podelski (1999); Podelski (2004)). The facility to deal with partial information is at the heart of program analysis, and the facilities for constraint propagation and solving precisely meet the needs of program analysis.

The application area of CP that has been strongly taken up in industry is optimisation. CP is probably the technology of choice for short-term scheduling (Baptiste et al., 2001) and for configuration (Subbarayan, 2005). It has many successes in the areas of logistics[1], transportation[2] and rostering [3]. It has been used to build tools used by the leading researchers in bio-informatics (Konagurthu et al., 2006).

Much of this work could fall under the old name of "operations research". However the advent of CP has moved the field on from using a tractable approximation of the real problem and solving the approximation, to modelling the real problem in all its complexity and - as far as practicable - solving it.

The mathematical restriction to "convexity" has in the CP approach been ignored, rather than overcome. However the resulting freedom of thought and expression has lead to new ways of solving complex problems. New ways of integrating different subproblem solvers, partly based just on the ease of expression that comes from CP, have made it possible to solve more complex problems involving more different kinds of resources than was previously practical. The new direction is a move from optimising one department to the whole company, or from one company to the whole supply chain.

Orthogonally there has always been a separation between planning and control. Plans are optimised using sophisticated algorithms and precise (but actually incorrect) estimates of the data. These plans cannot be carried out, due to delays and other problems on the day, and so operational controllers perform naive dispatching to keep things going.

The benefits from integrating planning and control will be enormous, and CP researchers are starting to explore this exciting challenge (Van Hentenryck and Bent, 2006).

The technology enabling these advances is not just CP. The CP paradigm makes it possible to integrate different approaches into a single integrated algorithm and system (Wallace, 2007). However crucial technology comes from the different approaches themselves: mathematical programming, specialised algorithms from operations research, stochastic search methods and metaheuristics, artificial intelligence, constraint propagation, advanced techniques for propositional satisfiability, and theorem proving.

Perhaps the greatest excitement that I hope will be generated by CP is the bringing together of researchers from these diverse areas. A common problem modelling language and a common problem solving framework - CP - provides a conduit for the communication and - more importantly - the understanding of ideas from other communities. With this shared basis for expression and experimentation, people from these different areas can

1. `www.ilog.com/industries/logistics/index.cfm`
2. `www.carmensystems.com/research_development/research_reports.htm`
3. `www.friartuck.net/customer/case-studies/nuh.htm`

discuss, compete and ultimately combine the best of their techniques. Thus CP can underpin the establishment of a brand new research community addressing large scale industrial combinatorial optimisation problems exploiting ideas and techniques from all these backgrounds.

References

K.R. Apt and M. G. Wallace. *Constraint Logic Programming Using ECLiPSe.* Cambridge Univerity Press, 2007. ISBN 0-521-86628-6.

Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-Based Scheduling.* Kluwer Academic Publishers, Norwell, MA, USA, 2001. ISBN 0792374088.

D. Cohen and P. Jeavons. The complexity of constraint languages. In *Handbook of Constraint programming*, chapter 6. Elsevier, 2006.

Giorgio Delzanno and Andreas Podelski. Model checking in CLP. *Lecture Notes in Computer Science*, 1579:223–239, 1999.

Daniel Diaz and Philippe Codognet. Design and implementation of the GNU prolog system. *Journal of Functional and Logic Programming*, 2001(6), 2001.

M. Dincbas, H. Simonis, and P. van Hentenryck. Solving a Cutting-Stock Problem in Constraint Logic Programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Fifth International Conference on Logic Programming*, pages 42–58, Seattle, WA, August 1988a. MIT Press.

M. Dincbas, H. Simonis, and P. van Hentenryck. Solving the Car Sequencing Problem in Constraint Logic Programming. In *European Conference on Artificial Intelligence (ECAI-88)*, Munich, W. Germany, August 1988b.

M. Garcia de la Banda, K. Marriott, R. Rafeh, and M. Wallace. The modelling language zinc. In *Proc. CP06*, pages 700–705. Springer-Verlag, 2006.

A.S. Konagurthu, J.C. Whisstock, P.J. Stuckey, and A.M. Lesk. MUSTANG: A multiple structural alignment algorithm. *Proteins: Structure, Function, and Bioinformatics*, 64 (3):559–574, 2006.

F. Laburthe. CHOCO: implementing a cp kernel. In *CP'00 Post Conference Workshop on Techniques for Implementing Constraint programming Systems - TRICS*, Singapore, 2000.

Andrew J. Parkes. Clustering at the phase transition. In *AAAI/IAAI*, pages 340–345, 1997.

A. Podelski. Constraints in program analysis and verification. In *Principles and Practice of Constraint Programming - CP 2004*, pages 1–4, 2004.

S. Subbarayan. Integrating CSP decomposition techniques and BDDs for compiling configuration problems. In *Proc. CP-AI-OR*, volume 3524 of *LNCS*, Brussels, 2005.

P. Van Hentenryck and R. Bent. *Online Stochastic Combinatorial Optimization*. MIT Press, 2006. ISBN 978-0-262-22080-4.

M.G. Wallace. Hybrid algorithms in constraint programming. In *Recent Advances in Constraints*, volume 4651 of *"LNCS"*, pages 1–32. Springer, 2007.

D.H. Wolpert and W.G McReady. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.