The G12 Project: Mapping Solver Independent Models to Efficient Solutions

Peter J. Stuckey[1], Maria Garcia de la Banda[2], Michael Maher[3], Kim Marriott[2], John Slaney[4], Zoltan Somogyi[1], Mark Wallace[2], and Toby Walsh[3]

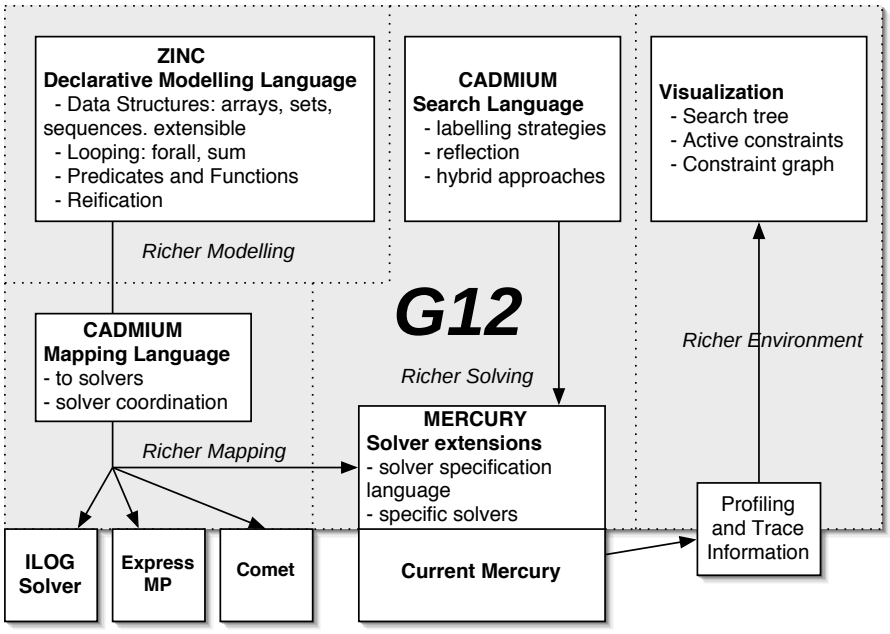
[1] NICTA Victoria Laboratory
Department of Computer Science and Software Engineering
University of Melbourne, 3010 Australia. {pjs,zs}@cs.mu.oz.au
[2] School of Comp. Sci. & Soft. Eng., Monash University, Australia.
{mbanda,marriott,mgw}@mail.csse.monash.edu.au
[3] NICTA Kensington Laboratory
University of New South Wales, 2052, Australia
{michael.maher,toby.walsh}@nicta.com.au
[4] NICTA Canberra Laboratory
Canberra ACT 2601, Australia john.slaney@nicta.com.au

Abstract. The G12 project recently started by National ICT Australia (NICTA) is an ambitious project to develop a software platform for solving large scale industrial combinatorial optimisation problems. The core design involves three languages: Zinc, Cadmium and Mercury (Group 12 of the periodic table). Zinc is a declarative modelling language for expressing problems, independent of any solving methodology. Cadmium is a mapping language for mapping Zinc models to underlying solvers and/or search strategies, including hybrid approaches. Finally, existing Mercury will be extended as a language for building extensible and hybridizable solvers. The same Zinc model, used with different Cadmium mappings, will allow us to experiment with different complete, local, or hybrid search approaches for the same problem. This talk will explain the G12 global design, the final G12 objectives, and our progress so far.

1 Introduction

The G12 project aims to build a powerful and easy-to-use open source constraint programming platform for solving large scale industrial combinatorial optimization (LSCO) problems. The research project is split into four related threads: building richer modelling languages, building richer solving capabilities, a richer control language mapping the problem model to the underlying solving capabilities, and a richer problem-solving environment.

The underlying implementation platform will be the Mercury system. On top of Mercury the project will build a generic modelling language, called Zinc, and a mapping language, called Cadmium, which takes a Zinc model and generates a Mercury program. We also plan that Zinc and Cadmium will combine to output programs for different constraint solving systems such as ILOG Solver [6], Xpress



MP [7] and Comet [2]. A diagram showing the four threads and how they interact with existing solvers and the current language Mercury is shown below.

2 Richer Modelling

The process of solving LSCO problems can be separated into creating the conceptual model, and an algorithm development process for mapping the conceptual model to a design model. This depends upon a language for writing conceptual models, and usually another language for writing design models.

In order to maintain clarity, flexibility, simplicity and correctness, we separate the conceptual modelling language Zinc from the mapping language Cadmium, which is both the design modelling language and the search language.

The best starting point for a universal conceptual modelling language is a purely declarative modelling language. Such a language allows the modeller to give a high-level specification of the constraint problem in terms natural to the problem itself. In order to do so it must include data structures that naturally arise in modelling such as arrays and sets, as well as be extensible in order to incorporate new problem specific structures such as jobs and tasks. We need natural constructs for specifying large constraints and large conjunctions of constraints. In order to encapsulate common problem structure we need to be able to specify predicates and functions in the modelling language for reuse.

The modeller needs to be able to specify requirements for robust, as well as optimal, solutions. *Robust* solutions are less sensitive to change in parameters, and reflect the reality that real solutions often need to be repaired when they

are put into practice. It must be possible for the modelling language to specify the required type of robustness.

There are many challenges in the design of the Zinc language. For example, how can we make the language suitable for both an operations researcher experienced in using restricted mathematical modelling languages such as AMPL [1], as well as computer scientists used to the flexibility and power of programming languages. OPL [5] is the closest current language to how we envisage Zinc.

3 Richer Mapping

In order to make use of a conceptual model we must have some way of compiling it, that is mapping it to a design model. One advantage of separating of the conceptual modelling language from the design model is the ability to then rapidly experiment with different design models for the same conceptual model.

We wish to provide transparent and flexible ways of specifying how a conceptual model is mapped to a design model. Experience in developing solutions to industrial constraint problems has shown that we will often need to use two or more solving technologies to tackle a hard constraint problem. Various constraints will be treated by one solver, while other constraints will be treated by another. Some constraints may be treated by two or more solvers. When we are using multiple solvers we not only need to specify which constraints are sent to each solver, and how they are mapped to that solver, but how the solvers will interact. This must be supported by Cadmium.

G12 will not only need to provide a modelling interface to distinct solving methods from mixed integer programming (MIP), constraint programming (CP) and local search, but will also need to provide a modelling and mapping interface to methods for integrating these techniques. The design models for such an integrated scheme may involve combinations of algorithms from all three areas. The Cadmium language in which the design models are expressed must therefore subsume the expressive power of all the above languages. Much more is required however, since the interaction between local search and *branch-and-infer* search open a huge space of possible hybridisations.

4 Richer Solving

Constraint programming systems typically employ tree search to complement constraint propagation. Moreover the search is depth first and alternative search choices are only explored after backtracking to the relevant choice point. By contrast MIP search typically explores the search tree in a best-first fashion, which requires a multitude of *open* nodes to be recorded, ready for expansion at a later time. Recently systems like Mozart [4] have incorporated the open nodes approach in CP. With G12 we shall pursue the convergence of CP and MIP search by reducing the cost of jumping between open nodes, and maintaining flexibility between the many different tree search strategies.

However local search techniques are playing an increasingly important role in CP. The Comet CP system [2] supports a wide range of local search techniques,

with constraint handlers adapted to the local search paradigm. The final addition to the arsenal of search methods offered by G12 will be population-based search methods, such as genetic algorithms. These methods explore a whole population of solutions concurrently, and then combine the results from the population to focus the search on promising areas of the search space.

To date no system has enabled the user to specify the problem in terms of an algorithm-independent conceptual model, and have the computer map this into, say, an ant colony optimisation algorithm. The challenge for Cadmium is to make this mapping straightforward and concise, yet precise and flexible.

Another important research direction for richer solving will be developing algorithms for returning more robust solutions, more diverse solutions, or finding similar solutions to previous solutions.

5 Richer Environment

The key to solving complex industrial application problems is rapid applications development, with close end-user involvement. To support rapid application development, a rich solution development environment is essential.

The first stage in developing an application is constructing a correct Zinc and Cadmium model. This is much easier for the application programmer if solutions are graphically realized in a way that they can readily understand. The second and more time consuming phase is performance debugging in which we study the behaviour of the algorithms at runtime and understand exactly what is going on. Interaction with a running algorithm is necessary to detect its weaknesses, and to understand and build on its strengths. To support close end-user involvement, the problem solving behaviour must be made meaningful and transparent to the end-user. This requires that the algorithm behaviour be mapped back onto the problem model, so that the user can understand the behaviour in terms of the original application.

6 Conclusion

The G12 project aims, using the separation of the conceptual model from the design model, to provide a software framework where many, perhaps all, optimizations approaches can be experimented with efficiently. By allowing this exploration we hope to get closer to the ultimate goal of simply specifying the problem and letting the G12 system determine the best way to solve it.

References

1. AMPL: `www.ilog.com/products/ampl/`
2. Comet: `www.cs.brown.edu/people/pvh/comet1.html`
3. Mercury: `www.cs.mu.oz.au/mercury/`
4. Mozart: `www.mozart-oz.org`
5. OPL Studio: `www.ilog.com/products/oplstudio/`
6. ILOG SOLVER: `www.ilog.com/products/solver/`
7. Xpress MP: `www.dashoptimization.com`