Automated Modelling and Model Selection in Constraint Programming

Current Achievements and Future Directions

Ozgur Akgun[1], Alan M. Frisch[2], Christopher Jefferson[1], and Ian Miguel[1]

[1] University of St Andrews
[2] University of York

1 Introduction

In attacking the *modelling bottleneck* [13], we present current achievements in automated model generation and selection in constraint programming (CP). We also discuss promising future directions in automated model selection, which we believe are of key importance in enabling successful automated modelling in CP.

2 Automated model generation

The *modelling bottleneck* has been considered to be a significant limiting factor preventing widespread adoption of CP technology. For a problem of interest, formulating a correct CP model suitable for input to a constraint solver requires expertise and time; formulating an *effective* CP model requires significantly more so. In order to address this challenge, automated modelling has become a very active area of research in CP. A promising approach to automate aspects of CP modelling has been the refinement of abstract constraint specifications [2, 4] in languages such as ESRA [3], ESSENCE [5], $\mathcal{F}$ [7] or Zinc [10, 12].

Our approach in CONJURE [1, 2] is to start from a highly abstract *problem specification* and produce concrete CP models automatically. The problem specification language ESSENCE enables specifying problems concisely; the language is designed to capture problem structure at a level of abstraction that is above where most CP modelling decisions are made. For instance, having an array of decision variables and posting an `allDiff` constraint on this array is a common idiom in CP modelling. Many uses of this idiom can be replaced with a set decision variable. Using a set decision variable instead, the user gets access to a large set of predefined set operators — such as union, intersection, subset, set equality — and more importantly they don't commit to using a one dimensional array. CONJURE can *refine* this decision variable and all expressions involving it in multiple ways, possibly also including the one-dimensional array representation as an alternative.

In addition to sets, ESSENCE provides decision variables with other abstract domains: tuples, enumerations, functions, relations, multi-sets, partitions, and allows arbitrary nesting of these. It also provides a rich collection of operators for variables with abstract domains enabling concise specification of problem structure. While producing

a concrete CP model, CONJURE needs to make two kinds of modelling decisions, *representation selection* and *expression refinement*. In order to express the transformation rules used by CONJURE we have devised a domain-specific language. This enables us to write rules more easily, extend CONJURE's modelling capabilities without recompiling, and make rule authoring more accessible to CP researchers so they can encode new modelling *tricks* and improve CONJURE.

CONJURE contains at least one, but typically several, representation options for each abstract domain, and alternative translations for operators on abstract variables. Hence, it can typically generate several alternative formulations of a single problem specification. In this regard, CONJURE is similar to conventional compilers, which also have to choose between alternative transformations during compilation. Each decision in CONJURE is far more important though, since solution performance of different models for the same problem can be drastically different.

Previous work shows how CONJURE generates kernels of published models from the literature [2] and how it produces symmetry breaking constraints automatically [1].

Future directions

Automated modelling tools provide a powerful infrastructure and they can be improved in several directions: addition of alternative representations for abstract domains and alternative reformulations for problem expressions; automated generation of symmetry breaking and implied constraints.

3 Automated model selection

CP modelling is important; a single problem can be modelled in several different ways, and there is a vast variation in solution performance depending on the model chosen. Now that we are able to automatically generate alternative models, automating the model selection process naturally becomes the next big challenge. Equipped with a good way of differentiating between CP models, an automated modelling system can finally be very useful to both novice and expert users of CP technology.

The model selection problem in CP is very similar to the general algorithm selection problem [14]. A thorough survey of the literature about this problem with a strong emphasis to combinatorial search problems and CP can be found in [11].

There are several options to consider before we can attack this problem: to select a single model or a set of models; to work on problem classes or problem instances; if working on problem classes, how to analyse and explore the instance space. Leaving these questions unanswered, we first present the kinds of approaches CONJURE enables.

Post-CONJURE analysis Using CONJURE to generate all alternative models for a given problem, and using this set of concrete CP models as input for model selection. An advantage of this approach is the clear separation between model generation and selection. On the other hand, a major disadvantage is having to generate possibly thousands of models only to realise that most of them aren't very promising early in model selection.

Mid-CONJURE heuristics Using heuristics to choose promising transformations during CONJURE. A first iteration of this approach presents [1] promising results. In this work, CONJURE locally selects the transformation which generates the most compact domain/expression.

In addition, hybrid approaches can also be taken; i.e., multiple heuristics can be used to generate a smaller set of alternative models, and this set can be used as input to a more generic model selection procedure.

Selecting models for problem classes vs problem instances. In general, working on problem instances is easier for both automated model generation and for automated model selection. This shouldn't be surprising as problem classes are essentially parameterised problem instances, and they *describe* a set of problems rather than a single problem. For this very reason, selecting good models for a problem class is also more valuable: once selection is completed, findings can be used for all instances of the same class. This is also why we can afford more expensive analysis for model selection of problem classes, the cost will be amortised over all the instances.

Selecting for the whole class vs subdivisions of the instance space. Selecting effective models for a problem class can be tempting. However, we know different models can be better for different instances — in an extreme example a problem class can be composed of two subproblems and a parameter value can be controlling which subproblem to actually solve. In such a case, the choice of an effective model highly depends on the value of the given parameter.

Selecting a single model vs multiple models. Even if we limit ourselves to working on a single subdivision of the instance space or to a single instance, trying to select a single effective model can result in eliminating promising models prematurely due to possible shortcomings of the learning technique. In contrast, selecting a set of promising models can lead us to the notion of *model portfolios*, analogous to algorithm portfolios [6, 8].

In [1], we present *racing* as a general technique when a user provides a representative set of instances for a problem class. We are also investigating how to automatically generate instance data from a given problem specification in [9], to use when this data is not available.

Future directions

Better metrics to compare models are needed. For instance level analysis, solution time is the ultimate metric, however it is potentially very expensive, we need proxies to this. For class level analysis, instance level metrics can be augmented with standard sampling and aggregation methods if a representative subset of instance data is available. Without such data we are left with class level symbolic analysis to compare models.

An important property of a model comparison metric is whether it can be used to compare partial models or not. If a metric has this property, a best-so-far model can be used as a *lower bound* and we can employ branch-and-bound to prune early during automated model generation, and only generate *good* models.

Another direction is to explore better heuristics to guide CONJURE and evaluate their relative performances. Simply, each such heuristic can be used independently to

generate a portfolio of models. A more sophisticated method will be to use a hyper-heuristic to guide which one of the smaller heuristics should be used during model generation; dynamically switching heuristics for different parts of the model.

4 Conclusion

We present the current achievements in automated modelling and model selection in CP followed by a summary of promising future directions. These two areas of research are closely related, automated modelling requires good model selection to be successful; and model selection needs a diverse set of models to select from. The findings of model selection are likely to be fed back to automated modelling systems to improve quality and the signal-to-noise ratio in the generated models. We expect this feedback loop to be fruitful and help making CP technology more accessible and more powerful.

Acknowledgements

This research is supported by UK EPSRC grant no EP/K015745/1.

References

1. Akgun, O., Frisch, A.M., Gent, I.P., Hussain, B.S., Jefferson, C., Kotthoff, L., Miguel, I., Nightingale, P.: Automated symmetry breaking and model selection in CONJURE. In: Principles and Practice of Constraint Programming - CP 2013 (2013)
2. Akgun, O., Miguel, I., Jefferson, C., Frisch, A.M., Hnich, B.: Extensible automated constraint modelling. In: AAAI-11: Twenty-Fifth Conference on Artificial Intelligence (2011)
3. Flener, P., Pearson, J., Ågren, M.: Introducing ESRA, a relational language for modelling combinatorial problems. In: LOPSTR 2003. pp. 214–232 (2003)
4. Frisch, A.M., Jefferson, C., Hernandez, B.M., Miguel, I.: The rules of constraint modelling. In: Proc. of the IJCAI 2005. pp. 109–116 (2005)
5. Frisch, A.M., Harvey, W., Jefferson, C., Martínez-Hernández, B., Miguel, I.: ESSENCE: A constraint language for specifying combinatorial problems. Constraints 13(3) pp. 268–306 (2008)
6. Gomes, C.P., Selman, B.: Algorithm portfolios. Artificial Intelligence 126(1-2), 43–62 (2001)
7. Hnich, B.: Thesis: Function variables for constraint programming. AI Commun 16(2), 131–132 (2003)
8. Huberman, B.A., Lukose, R.M., Hogg, T.: An economics approach to hard computational problems. Science 275(5296), 51–54 (1997)
9. Hussain, B.S., Miguel, I., Gent, I.P.: Parameter generation with Conjure. In: Doctoral Program - Principles and Practice of Constraint Programming - CP 2013 (2013)
10. Koninck, L.D., Brand, S., Stuckey, P.J.: Data independent type reduction for zinc. In: ModRef10 (2010)
11. Kotthoff, L.: On algorithm selection, with an application to combinatorial search problems. Ph.D. thesis, University of St Andrews (2012)
12. Marriott, K., Nethercote, N., Rafeh, R., Stuckey, P.J., de la Banda, M.G., Wallace, M.: The design of the zinc modelling language. Constraints 13(3) (2008)
13. Puget, J.F.: Constraint programming next challenge: Simplicity of use. In: Principles and Practice of Constraint Programming - CP 2004. pp. 5–8 (2004)
14. Rice, J.R.: The Algorithm Selection Problem. Advances in Computers 15, 65–118 (1976)