Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)

Automated Modelling and Solving in Constraint Programming

Barry O'Sullivan
Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
b.osullivan@cs.ucc.ie

Abstract

Constraint programming can be divided very crudely into modeling and solving. Modeling defines the problem, in terms of variables that can take on different values, subject to restrictions (constraints) on which combinations of variables are allowed. Solving finds values for all the variables that simultaneously satisfy all the constraints. However, the impact of constraint programming has been constrained by a lack of "user-friendliness". Constraint programming has a major "declarative" aspect, in that a problem model can be handed off for solution to a variety of standard solving methods. These methods are embedded in algorithms, libraries, or specialized constraint programming languages. To fully exploit this declarative opportunity however, we must provide more assistance and automation in the modeling process, as well as in the design of application-specific problem solvers. Automated modelling and solving in constraint programming presents a major challenge for the artificial intelligence community. Artificial intelligence, and in particular machine learning, is a natural field in which to explore opportunities for moving more of the burden of constraint programming from the user to the machine. This paper presents technical challenges in the areas of constraint model acquisition, formulation and reformulation, synthesis of filtering algorithms for global constraints, and automated solving. We also present the metrics by which success and progress can be measured.

Introduction

Constraint programming provides powerful support for decision-making; it is able to search quickly through an enormous space of choices, and infer the implications of those choices. Constraint programming, is already widely used in industry. For example, constraint programming software from ILOG, the leading commercial purveyor of constraint technology, is embedded in products from leading companies like SAP and Oracle. However, the impact of constraint programming has itself been constrained by a lack of "user-friendliness". While user-friendliness could be regarded as the 21st century challenge for all of computer software, constraint programming presents particular challenges and opportunities. Jean-Francois Puget, Vice President of Optimization R&D at ILOG, delivered an invited talk at the 2004 Constraint Programming conference: *"Constraint Programming's Next Challenge: Simplicity of Use"*.

We present a set of challenges in the area of constraint programming. The *motivation* is to reduce the burden on constraint programmers, and to broaden the scope of problems that can be tackled alone by the domain experts, by shifting more of the modeling burden from the constraint programmer into constraint programming software tools, systems, and applications. While the constraint community has begun to develop approaches that address the usability issues in constraint programming, there is significant opportunity for the general artificial intelligence community to make contributions. The specific challenges posed are related to automated constraint model acquisition, automated formulation and reformulation, synthesis of filtering algorithms for global constraints, and automated solving. In many respects these challenges are at the core of artificial intelligence, since they capture many, if not all, the facets of AI research. For example, constraint model acquisition is related to concept learning, theory formation, knowledge acquisition, and programming-by-demonstration. Synthesizing filtering algorithms for global constraints is related to automated programming, classifier learning, and inductive logic programming. Finally, automated solving is closely related to automated theorem proving and problem solving.

Of course, modeling and solving are intimately connected, and we cannot study one without the other. The line between modeling and solving is a fuzzy one. For example, if symmetry in a problem makes solving difficult, we can break symmetry by adding additional constraints to the model, or by using an algorithm designed to cope with symmetry, which itself might effectively change the model dynamically during search. Ultimately, through the study of these challenges, we hope to shed light on the "deep structure" of usability and complexity challenges and the relationship between representational and computational responses to such challenges.

Evaluating Progress

Progress in constraint programming is usually evaluated empirically. Over the past number of years the constraints community has developed a large inventory of constraint satisfaction problems. For example, the CSPLib comprises almost 50 different problem classes. Each problem class is presented with a formal definition, pointers to research papers, and sample results; in some cases implementations in some well-known constraint solvers have been made available. The CSPLIB is available online[1].

The constraints community has recently begun to benchmark progress in the capabilities of constraint solving technology through the use of international competitions. Two specific competitions are well supported: the International CSP Solver Competition[2], and the Minizinc Challenge[3]. These provide a basis for benchmarking progress; the problems are available in a standardized machine readable format, and past competition results document the state-of-the-art in terms of solving capability for each of those constraint solvers that participated in previous competitions.

The CSP Solver Competition instances are readily available online[4] in a standardized XML format, for which parsers are also available in languages such as Java and C++. Minizinc Challenge problems are available in a two-file format: the first containing the generic problem description, and the other containing parameter sets for generating specific instances for benchmarking purposes. These instances are also available online. In total, several thousand benchmark problems are available to those who wish to take up one or more of the challenges presented in this paper.

Progress on the challenges can be measured in a variety of ways. Acquisition methods can be evaluated on the basis of the difficulty with which they can acquire specific problem classes or instances. For example, difficulty can be measured in terms of the number of solutions or non-solutions that need to be presented to an acquisition system in order to acquire the model. Reformulation and search based methods can be evaluated against the performances of those solvers that appeared in competition.

Of course theoretical evaluation is also possible; for example, generating bounds on the running time of a particular method. Such analysis is typically done from a *worst-case* point of view and is, therefore, often of limited use in practice. However, where theoretical results can be obtained that demonstrate improvements in worst-case behaviour, they should be reported. In so far as it is possible, average-case analysis should be performed. However, such results can be notoriously difficult to obtain. A compromise is to empirically study runtime distributions.

Automated Model Acquisition

In many application domains, one prefers to model a practical problem as a constraint satisfaction problem (CSP) and then use available constraint programming tools to solve it. However, the precise specification of a CSP is sometimes not available, and users often find difficult to articulate their constraints. In these situations we would like the computer to take an active role in learning the CSP from a *training set*, which is given, for instance, as a set of examples of its solutions and non-solutions. This kind of learning is called *constraint acquisition* (Bessiere et al. 2005). The motivations for constraint acquisition are many. For example, in order to solve partially defined constraints more efficiently, Lallouet and Legtchenko (2005) have proposed to complete their specification by using machine learning techniques and then solve them. Wilson et al. (2007) have interleaved constraint elicitation and constraint solving with the objective of minimizing the overall burden of the process.

Theoretically, generic methods from the *machine learning* field can be applied to learn an appropriate formulation of the target problem as a CSP. However, standard machine learning methods do not take the characteristics of CSPs into account, and are usually not sufficient to efficiently acquire constraints. A new class of learning methods with emphasis on the characteristics of CSPs has been studied in recent years to learn CSPs more efficiently. In 1992, by introducing the notion of constraint-directed generalization in constraint logic programs, Mizoguchi and Ohwada (1992) developed a technique to learn generalized spatial constraints given a set of input spatial constraints so that each input spatial constraint is satisfied by the generalization. O'Connell et al. (2002) presented an approach to interactive constraint acquisition based on machine learning techniques, in which they employ strategies that minimize the dialog length between the user and the computer. More recently, Coletta et al. (2003) devised a specialized instance of the well-known Candidate-Elimination algorithm (Mitchell 1982) for version space learning to acquire classical CSPs. In that version space-based approach constraints with different scopes of variables are acquired separately. One version space of candidate relations is maintained for each scope of variables. A candidate CSP is then the combination of the final sets of candidate relations over all scopes of variables. Constraint acquisition is performed by searching through the *hypothesis space* of candidate relations on a scope of variables and removing those that violate the examples in the training set. The partial order over the relations is simply defined as the natural subsumption order. Bessiere et al. (2004) further improved that version space-based algorithm by exploiting redundancy in CSPs. Recently, Bessiere et al. (2005) have proposed to reformulate the CSP learning problem as a SAT problem, benefitting from the use of efficient SAT techniques to assist in the acquisition process.

The above-mentioned constraint acquisition methods, though useful for finding CSPs that fully agree with the training set, are still limited to learning classical CSPs and provide no useful information in the case where a CSP that fully agrees with the training set does not exist in the hypothesis space. This deficiency motivated the development a new framework in which one can acquire CSPs that minimally differ, according to a predefined measure, from the training set (Vu and O'Sullivan 2008).

The Challenges. While there have been significant advances in the area of automated acquisition of constraint models, there remain many challenges. Amongst these are:

1. All of the approaches summarized above rely heavily on

[1]`http://www.csplib.org`

[2]`http://cpai.ucc.ie/08/`

[3]`http://www.g12.cs.mu.oz.au/minizinc/challenge2009/challenge.html`

[4]`http://www.cril.univ-artois.fr/~lecoutre/benchmarks.html`

supervision from an expert who either labels examples of solutions and non-solutions of a target CSP, or reacts to questions posed by the acquisition system. Developing approaches that support forms of unsupervised CSP acquisition would be particularly important, especially in data-rich domains. Techniques from the field of datamining are promising in meeting this challenge.

2. Even more importantly, the techniques presented above are suitable for learning specific instances of a general problem class, but they are not capable of acquiring a description of the problem class itself. Techniques from the field of inductive logic programming are promising in meeting this challenge.

Automated Model Reformulation

Given a specification of the problem to be solved, there are many different ways to formulate the problem as a CSP. A major aim in reformulation is to ensure that the resulting CSP can be solved as efficiently as possible. Recently a large number of papers have been published that study the reformulation of specifications of CSPs (Frisch et al. 2005b; 2005a). Using such formal approaches to reformulation, one can define sets of reformulation rules which facilitate the generation of many alternative CSP formulations (Frisch et al. 2007). However, little or nothing is said about the relative merits of each formulation until its runtime behaviour is empirically analysed. The TAILOR system attempts to automate this by regarding model reformulation as a compilation-like process (Gent, Miguel, and Rendl 2007).

An extremely innovative piece of work on automated reformulation is by Charnley et al. (2006), which builds upon earlier work (Colton and Miguel 2001). Their objective is to automatically detect implied constraints, i.e. constraints that are logically implied by the problem formulation but can also help in solving the problem more efficiently. The concern of Charnley et al.'s methods is to generate candidate implied constraints through the use of mathematical theory formation software. The output of the theory formation step is a set of conjectures (candidate constraints), which are then proved to be implied constraints by using a theorem prover. Those theorems, implied constraints, that are discovered are evaluated for efficiency by simulation and added to the reformulated model only if they improve search. Related work for reformulating global constraints has also been reported by (Bessière, Coletta, and Petit 2007).

The Challenges. Automated reformulation is an extremely challenging task. Two specific challenges whose resolution could significantly impact of general purposes problem reformulation are:

1. Generating candidate sets of variables (model viewpoints) and implied constraints for a problem is extremely difficult. However, many problems have significant structure that can be exploited. Techniques from the fields of machine learning, datamining, and discrete mathematics have the potential to identify interesting candidate implied constraints. For example, implied constraints could be mined from the problem formulation using association rule mining techniques.

2. Once a potential reformulation has been identified, it is difficult to predict whether it is more efficient than an earlier candidate without simulation. This can introduce a major bottleneck in the modelling process. A key challenge here is to develop heuristic or learning-based approaches to reliably predict the runtime of a reformulated CSP model.

Synthesis of Filtering Algorithms

Global constraints provide constraint programmers with an expressive notation for modelling problems. Ideally, as well as being expressive, a global constraint should be filtered efficiently. Filtering is the process by which values from the domains of variables that cannot participate in the solution of a constraint are removed from their respective domains. The classic global constraint is the ALLDIFFERENT (Régin 1994) which ensures that a set of variables all take unique values. As well as being expressive, this constraint can be efficiently filtered, using a combination of algorithms from matching theory.

However, the major challenge one faces when designing a new global constraint is that its filtering algorithm can be difficult to design. Some researchers have noticed that many global constraints can be expressed in terms of a set of lower level constraints (Bessiere et al. 2009). Alternatively, some have shown that filtering algorithms can be configured from a set of filtering rules (Abdennadher and Rigotti 2004). A major challenge is automating this process.

The Challenges. Given the work cited above, the major challenge in this area is the automatic design of global constraint filtering algorithms. We put forward the following challenges:

1. Assuming a rule grammar, or primitive constraint language, design a filtering algorithm by searching through the space of possible 'programs' in the grammar, evaluating their quality against the specification of the constraint. This can be formulated as beam search. Methods suitable for implementing large-scale beam searches are genetic programming, and local search.

2. A disadvantage of using global constraints in a model is that they can be very expensive to propagate. We therefore propose the challenge of predicting the cost (time complexity) and effectiveness (number of pruned values) due to propagating a specific constraint to ensure that global constraints are used in a beneficial manner.

Automated Solving

It is recognized within the field of constraint programming that different solvers are better at solving different problem instances, even within the same problem class (Gomes and Selman 2001). It has been shown in other areas, such as satisfiability testing (Xu et al. 2007) and integer linear programming (Leyton-Brown, Nudelman, and Shoham 2002), that the best on-average solver can be out-performed by carefully exploiting a portfolio of possibly poorer on-average solvers. Selecting from a portfolio usually relies on a machine learning technique based on feature data extracted from constraint satisfaction problems.

The SATZILLA[5] system builds runtime prediction models using linear regression techniques based on structural features computed from instances of the Boolean satisfiability problem. Given an unseen instance of the satisfiability problem, SATZILLA selects the solver from its portfolio that it predicts to have the fastest running time on the instance. In the International SAT Competition 2007, SATZILLA won two of the categories, and came second and third in two others. The AQME system is a portfolio approach to solving quantified Boolean formulae, i.e. SAT instances with some universally quantified variables (Pulina and Tacchella 2007). AQME is built on the Weka data-mining library[6]. Three versions of AQME have competed in the International Competitive Quantified Boolean Formula Evaluation[7]: a version using decision trees to select which solver to use, a version using logistic regression, and another using 1-nearest neighbour. Like SATZILLA, AQME selects one solver to run for a given unseen formula. Streeter et al. (Streeter, Golovin, and Smith 2007) build upon the work of Sayag et al. (Sayag, Fine, and Mansour 2006), by using optimization techniques to produce a schedule of solvers that should be tried in a specific order, for specific amounts of time, in order to maximize the probability of solving the given instance.

CPHYDRA (O'Mahony et al. 2008) is a solver portfolio for solving CSPs, specifically designed to compete in the 2008 CSP Solver Competition[8]. The task in that competition was to solve as many problem instances as possible, given 30 minutes per instance. There were five categories of problem instances: binary CSPs with extensional constraints, binary CSPs with intensional constraints, CSPs with global constraints, non-binary CSPs with extensional constraints, and non-binary CSPs with intensional constraints. CPHYDRA uses case-based reasoning to decide how much of the 30 minute allocation per instance to give to each solver in its portfolio. The competition entry of CPHYDRA comprised three solvers: `Abscon`, `Choco` and `Mistral`. The results of the competition are summarized in Table 1. For each solver, and each category, we give the percentage of instances solved. We also give the overall percentage based on the set of instances from all categories that were solved by at least one solver in the competition. CPHYDRA dominated its constituent solvers in every category in the solver competition. Moreover, it performed better than all other solvers in every category with the (surprising) exception of `Sugar+PicoSat` in the "Global" constraint category. This is clear evidence that straightforward learning capability can been extremely successful in the context of automated search.

Challenges. Automated search is one of the most important aspects of constraint programming. Novice constraint programmers are often unable to specify their own search strategies for complex problems. Therefore, the challenges in this area are:

1. Use machine learning and knowledge discovery techniques to analyse the key aspects of problem structure and generate advice to novice users on how they should set about solving particular problems.
2. Develop automated search systems that go beyond the capabilities of both SATZILLA and CPHYDRA by focusing on other solver objectives such as maximizing the robustness of search time, minimizing the worst-case search time, etc.
3. Develop tools to support the automated integration of systematic and non-systematic constraint programming methods with operations research techniques to solve complex optimization problems using hybrid methods.

Conclusions

We have presented a set of challenges from four of the most important aspects of the constraint programming process: problem acquisition, reformulation, global constraint development, and automated solving. We have given an overview of the literature related to the automation of these phases.

It is hoped that the artificial intelligence community will rise to these challenges and make significant contributions to making constraint process easier to use. The constraint programming community have built sufficient research infrastructure in the form of benchmark problems to ensure that progress against these challenges can be monitored.

Acknowledgements. This work was supported by Science Foundation Ireland (Grant Number 05/IN/I886).

References

Abdennadher, S., and Rigotti, C. 2004. Automatic generation of rule-based constraint solvers over finite domains. *ACM Trans. Comput. Log.* 5(2):177–205.

Bessiere, C.; Coletta, R.; Freuder, E. C.; and O'Sullivan, B. 2004. Leveraging the learning power of examples in automated constraint acquisition. In *Proceedings of CP 2004*, LNCS 3258, 123–137.

Bessiere, C.; Coletta, R.; Koriche, F.; and O'Sullivan, B. 2005. A SAT-based version space algorithm for acquiring constraint satisfaction problems. In *Proceedings of ECML 2005*, 23–34.

Bessiere, C.; Hebrard, E.; Hnich, B.; Kiziltan, Z.; and Walsh, T. 2009. Range and roots: Two common patterns for specifying and propagating counting and occurrence constraints. *Artif. Intell.* 173(11):1054–1078.

Bessière, C.; Coletta, R.; and Petit, T. 2007. Learning implied global constraints. In Veloso (2007), 44–49.

Bessiere, C., ed. 2007. *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, volume 4741 of *Lecture Notes in Computer Science*. Springer.

Charnley, J.; Colton, S.; and Miguel, I. 2006. Automatic generation of implied constraints. In Brewka, G.; Corade-

[5]`http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/`

[6]`http://www.cs.waikato.ac.nz/ml/weka/`

[7]`http://www.qbflib.org/index_eval.php`

[8]`http://www.cril.univ-artois.fr/CPAI08/`

Table 1: Results summary of the 2008 CSP Solver Competition showing that CPHYDRA solved more instances that any other solver using a simple case-based reasoning approach to managing its portfolio.

Category	Binary Ext.	Binary Int.	Global	N-ary Ext.	N-ary Int.	Total
CPHYDRA	**92%**	**94%**	84%	**97%**	**86%**	**90.74%**
`Mistral`	89%	82%	80%	94%	80%	86.30%
`Choco`	89%	82%	69%	73%	78%	78.60%
`Abscon`	88%	81%	37%	90%	74%	75.19%
`Sugar+MiniSat`	76%	76%	81%	61%	74%	73.24%
`Sugar+PicoSat`	71%	77%	**85%**	57%	73%	72.12%
`Casper`	70%	N/A	79%	64%	84%	69.47%
`Bprolog`	80%	59%	69%	53%	70%	66.07%
`Concrete`	80%	65%	N/A	90%	69%	63.26%
`Minion`	67%	N/A	43%	67%	N/A	57.67%
`Galac`	78%	N/A	N/A	54%	N/A	40.93%
`SAT4J CSP`	68%	49%	13%	36%	26%	39.35%
`mddc-solv`	N/A	N/A	N/A	95%	N/A	23.28%

schi, S.; Perini, A.; and Traverso, P., eds., *ECAI*, volume 141, 73–77. IOS Press.

Coletta, R.; Bessiere, C.; O'Sullivan, B.; Freuder, E. C.; O'Connell, S.; and Quinqueton, J. 2003. Constraint acquisition as semi-automatic modeling. In *Proceedings of AI-2003*, 111–124. Cambridge, UK: Springer.

Colton, S., and Miguel, I. 2001. Constraint generation via automated theory formation. In Walsh, T., ed., *CP*, volume 2239 of *Lecture Notes in Computer Science*, 575–579. Springer.

Frisch, A. M.; Hnich, B.; Miguel, I.; Smith, B. M.; and Walsh, T. 2005a. Transforming and refining abstract constraint specifications. In Zucker, J.-D., and Saitta, L., eds., *SARA*, volume 3607 of *Lecture Notes in Computer Science*, 76–91. Springer.

Frisch, A. M.; Jefferson, C.; Hernández, B. M.; and Miguel, I. 2005b. The rules of constraint modelling. In Kaelbling, L. P., and Saffiotti, A., eds., *IJCAI*, 109–116.

Frisch, A. M.; Grum, M.; Jefferson, C.; Hernández, B. M.; and Miguel, I. 2007. The design of essence: A constraint language for specifying combinatorial problems. In Veloso (2007), 80–87.

Gent, I. P.; Miguel, I.; and Rendl, A. 2007. Tailoring solver-independent constraint models: A case study with essence' and minion. In Miguel, I., and Ruml, W., eds., *SARA*, volume 4612 of *Lecture Notes in Computer Science*, 184–199. Springer.

Gomes, C. P., and Selman, B. 2001. Algorithm portfolios. *Artif. Intell.* 126(1-2):43–62.

Lallouet, A., and Legtchenko, A. 2005. Consistency for partially defined constraints. In *Proceedings of ICTAI 2005*, 118–125.

Leyton-Brown, K.; Nudelman, E.; and Shoham, Y. 2002. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In Hentenryck, P. V., ed., *CP*, volume 2470 of *Lecture Notes in Computer Science*, 556–572. Springer.

Mitchell, T. M. 1982. Generalization as search. *Artificial Intelligence* 18(2):203–226.

Mizoguchi, F., and Ohwada, H. 1992. Constraint-directed generalization for learning spatial relations. In *Proceedings of the International Workshop on Inductive Logic Programming*, volume ICOT TM-1182.

O'Connell, S.; O'Sullivan, B.; and Freuder, E. 2002. Strategies for interactive constraint acquisition. In *Proceedings of the CP-2002 Workshop on User-Interaction in Constraint Satisfaction*, 62–76.

O'Mahony, E.; Hebrard, E.; Holland, A.; Nugent, C.; and O'Sullivan, B. 2008. Using case-based reasoning in an algorithm portfolio for constraint solving. In *Proceedings of Artificial Intelligence and Cognitive Science (AICS 2008)*.

Pulina, L., and Tacchella, A. 2007. A multi-engine solver for quantified boolean formulas. In Bessiere (2007), 574–589.

Régin, J.-C. 1994. A filtering algorithm for constraints of difference in csps. In *AAAI*, 362–367.

Sayag, T.; Fine, S.; and Mansour, Y. 2006. Combining multiple heuristics. In Durand, B., and Thomas, W., eds., *STACS*, volume 3884 of *Lecture Notes in Computer Science*, 242–253. Springer.

Streeter, M. J.; Golovin, D.; and Smith, S. F. 2007. Combining multiple heuristics online. In *AAAI*, 1197–1203.

Veloso, M. M., ed. 2007. *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007.*

Vu, X.-H., and O'Sullivan, B. 2008. A unifying framework for generalized constraint acquisition. *International Journal on Artificial Intelligence Tools* 17(5):803–833.

Wilson, N.; Grimes, D.; and Freuder, E. C. 2007. A cost-based model and algorithms for interleaving solving and elicitation of csps. In Bessiere (2007), 666–680.

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2007. : The design and analysis of an algorithm portfolio for sat. In Bessiere (2007), 712–727.