Constraints manuscript No.
(will be inserted by the editor)

The Future of Optimization Technology

Maria Garcia de la Banda · Peter J. Stuckey · Pascal Van Hentenryck · Mark Wallace

the date of receipt and acceptance should be inserted later

Abstract Technology for combinatorial optimization is rapidly changing, and as the size and scope of problems that can be solved steadily increases, the complexity of the underlying technology is growing. We foresee a huge demand for both the simplification of use of combinatorial optimization technology (so called "model and run" capabilities), as well as increasing need for the ability to quickly build complex hybrid solutions. These demands will place new emphasis on universal modeling languages and model transformation capabilities, as well as flexible and high level ways of specifying hybrid solutions. These changes put constraint programming in an ideal position since: constraint programming has the most high-level view of problems to begin with so we can ease modeling difficulties; and since constraint programmng is an integrative technology, we have already spent considerable effort in making different solving technologies work together seamlessly. In this position paper we outline some of the key challenges and important research directions we foresee for optimization technology,

1 Introduction

Optimization technology is increasingly pervasive in our society. It is used in many different applications, such as scheduling supply-chains, controlling our main infrastructures, mitigating natural disasters, organizing transportation systems, discovering motifs in genetic material or patterns in social networks, and validating programs. This trend is likely to continue and intensify given the stress on our natural resources and the need to preserve our environment.

As the need for optimization technology increases, is it essential to lower the barrier of entry to using this technology. Many optimization problems are simple to solve using modern technology, but the problem owners are unaware or unable to use the

Maria Garcia de la Banda · Mark Wallace
National ICT Australia and Monash University, Australia E-mail: {mbanda,mark.wallace}@infotech.monash.edu.au

Peter J. Stuckey · Pascal Van Hentenryck
National ICT Australia and the University of Melbourne, Australia
E-mail: {pstuckey,pvh}@unimelb.edu.au

optimization technology already available. It is not acceptable that a new optimization application requires completing a Ph.D. to build a solution. The technology needs to be much easier to use so that domain experts with low optimization background can take advantage of it. While problem modeling is a skill that needs to be learned, we need to make it far easier to learn and apply this skill. To this end we need universal modeling languages that can be used to "model and run", that is, once a problem is modeled correctly, the underlying optimization system automatically determines what is the most appropriate technology, without any further intervention from the modeler. Furthermore, we need to consider other avenues for using optimization technology, such as taking advantage of the widespread ability of people to build simulations as a method for accessing optimization technology (by automatically converting these simulations into optimization tools).

As optimization tackles increasingly more complex and large-scale applications, the demands placed on the underlying technology in terms of speed, scale, and expressive power generates fundamental computational challenges. The complexity of this new generation of applications places a significant cognitive burden even on expert practitioners. For these applications it is no longer sufficient to write high-level models that can be readily solved by a black-box solver: It is now necessary to discover which technology is most appropriate for each part of a problem. This is why hybridizations of major technologies, such as constraint programming, mathematical programming and local search, are becoming the de-facto standard for addressing the complexity of emerging applications. At the same time, the scope of optimization has broadened from a focus on routing and scheduling to other areas of application that are sometimes both more challenging and more rewarding. These include integrated resource planning; operational decision-making under uncertainty, with data being delivered continuously in real-time to the optimization engine; and data-intensive optimization where optimization technology is applied to very large data sets. This, in turn, places new requirements on the technology which now needs to find high-quality solutions under time constraints, exploit uncertainty, deal with large data sets, and be integrated in complex runtime environments.

2 Challenges

2.1 Modeling and Solving

It is clear from the above discussion that optimization technology needs to be easier to use for both non-experts and experts. We believe it is vital to create new technology that will dramatically simplify the generation of new complex optimization applications by supporting rapid prototyping, deep solver hybridization,[1] data-intensive optimization, and decision-making under uncertainty. In particular, optimization technology needs to address the following challenges:

1. The design of high-level modeling languages and development environments for optimization that capture the structure and execution behavior of optimization problems at a high level of abstraction, including its combinatorial substructures and stochastic information;

[1] That is, using tightly interacting different solving technologies together.

2. The definition of model transformations that take as inputs high-level models and transform them into hybrid algorithms, exploiting the specific features of the application at hand;
3. The design and implementation of efficient, highly parallel, optimization solvers that integrate constraint programming (CP), mixed integer programming (MIP), mixed integer nonlinear programming (MINLP), local search (LS), dynamic programming (DP), and data-intensive optimization.

2.2 Optimization Tools

We take the view that there are multiple classes of optimization users, whose needs must be accommodated differently. In particular, we need at least 3 different classes of tools:

1. A universal modeling language for optimization that targets the large audience of optimization modelers, building on the success of systems such as AMPL [10], OPL [27], and ZINC [16]. Capturing a concise and precise model of the problem is the first and most important step in any optimization solution. We need to make this modeling language easy to use, well documented, extensible, and sufficiently expressive for more expert users. This is the only tool set for the non expert and, hence, it is the most important for proselytizing optimization technology.
2. A programming language for optimization that targets an audience with advanced skills in both modeling and optimization, and for which application prototyping and development time are paramount. This will build on tools such as CHIP [25], ECLIPSE [31], and COMET [28], which integrate a programming language with an optimization system. Clearly, these tools are for use by optimization experts tackling the most challenging problems where new integrations and hybridizations are required.
3. A suite of optimization libraries that will serve as the back-ends for the dedicated modeling and programming languages and will give unparalleled control to optimization application developers. The kernel of these libraries should be as small and extensible as possible. The API must be carefully designed to allow both flexibility and ease of use.

The three optimization tools interact as illustrated in Figure 1. The optimization libraries provide the solving workhorse for the other components, and applications can make use of whichever component best matches the needs and abilities of the application developer.

The modeling and solving components are represented in the architecture as the "Conceptual Model", the "Executable" and the mapping between them called the "Model transformation". In fact this model transformation can be broken down into two stages, via a "Design Model" , which may be implemented as a program in an "Optimization programming language", or could be captured only implicitly in the transformation. The two stages are: "trans" mapping the Conceptual Model down to a program (Design Model); and "links" which simply links the program to underlying "Optimization libraries" at the Executable level.

Applications can be developed that making use of each of the 3 different tools:

– An application may invoke a high-level model written in the universal modeling language, and simply make use of the answer returned;

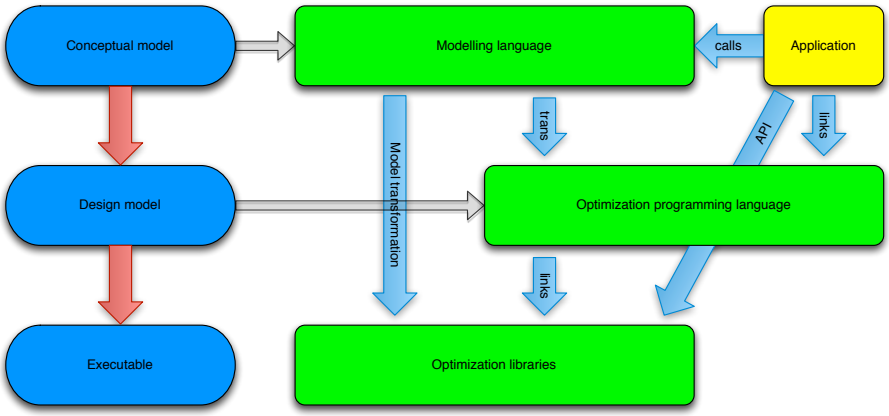

Fig. 1 The three components of optimization technology, and their interaction and relations to different modeling concepts.

- An application may link to a complex hybrid solution written in the optimization programming language; and react to the solutions of partial solutions determined by the solution module; or
- An application may directly create a model for a single solver using the optimization libraries API.

The advantage of having the three optimization tools in a single system is that users can gradually extend their knowledge of the system starting from the most accessible modeling language level and beginning to use the lower level tools directly as they become more familiar with the system. There is also a symbiotic relationship between the components, that should be taken advantage of in their development. For instance, modeling abstractions developed initially for the modeling language may find their way into the optimization libraries, if for example solving technology can be developed that solves them more efficiently than by translation. Similarly, advances in the lower layers can be exposed in the modeling and programming languages, for example half-reification [9] initially introduced to improve the translation of complex terms inside solvers, can be lifted to modeling systems.

3 Research Directions

We see the need and scope for major scientific contributions to optimization technology in three main areas: Modeling, model transformations and solver technology. Obviously, these three areas are intimately connected and interdependent, but they often interact through well-defined interfaces.

3.1 Modeling

The goal of the modeling component is to design and implement a high-level language and its development environment. We need a fundamental shift from the current view

of (mathematical) modelling which is *design modelling* dictated by the facilities of the underlying solvers, to a problem-oriented *conceptual modelling* which captures the problem specification in a solver agnostic way. The primary goal of high-level modeling is to support *rapid* prototyping by both experts and non-experts. A secondary goal is to provide optimizers with an environment that allows them to debug and understand their models at a high level of abstraction, a facility which is strongly lacking in existing optimization tools. *The ultimate goal for modeling should be to develop comprehensive modeling concepts covering most of the uses of optimization in the field today, from deterministic to stochastic optimization, from simulation optimization to on-line optimization.* The next paragraphs outline important research directions for achieving this goal.

Search Fundamental to solving all combinatorial problems is search. One of the key advantages of CP technology over other optimization technology is the ability for the user to specify complex search procedures succinctly, that are executed efficiently. There has been substantial progress in the field for defining complex search languages, e.g. [20, 30, 29]. But there is more to be done. We believe that search languages should be as orthogonal as possible, following the ideas of *search combinators* [22]. Further we need to extend the complex search available for CP solvers to be usable for other types of complete solvers like SAT and MIP solvers, and make new kinds of hybrid search, e.g. combining autonomous search features like activity with programmed search.

Patterns for Hybridization: Complex optimization problems often require hybridizations of various techniques to obtain high-quality solutions in reasonable time. Traditionally, each hybrid algorithm is a unique, stand-alone piece of software that requires substantial implementation effort and integration with existing software components, as well as a deep understanding of the hybrid method. This makes building hybrid solutions very challenging. An aim of optimization technology developers should be to dramatically simplify this process, using a "plug-and-play" approach that allows modelers to make use of optimization techniques (or "hybridization patterns") without understanding and/or reimplementing all the components or the communication between them. We believe one can define common hybridization patterns, i.e., recipes for creating new solvers from commonly required components see e.g. [21] or [2]. But we need to go well beyond what is possible currently. A modeler should be able to explore many hybridizations quickly using concise definitions. For instance, an hybridization pattern may capture multi-scale modeling that arises in modeling a whole transportation network at the strategic, tactical, and operational levels.

Symmetry and Dominance Relations: Many industrial applications feature symmetries and dominance relations which, when not exploited by the underlying algorithms, may lead to significant degradation in performance. While it is unreasonable to expect that modelers will be experts in techniques to break symmetries and dominance relations, it is often the case that users are aware of these features and could communicate them in the model. A modeling language should make it possible to express symmetries and dominances or to reveal them through high-level constructs (e.g., [14]).

Stochastic Optimization: In the last decade, significant progress has been realized in pushing the frontier of stochastic optimization. Simultaneously, there has been an increasing number of applications requiring optimization under uncertainty. Ideally one

should be able to start from an existing deterministic model, and extend this to a stochastic model without reformulating constraints, by providing stochastic input data, and relating different decisions to different stages. While this is possible in some systems [1] modelling these problems is still too low level, and tied to a the solving mechanism. A modeling language should make it possible to express optimization problems under uncertainty concisely independent of the solving mechanism. This means being able to specify random variables, common distributions, a variety of uncertainty models such as Markov and graphical models, as well as a variety of sampling procedures (e.g., Latin-Square sampling and importance sampling). The modeling language also needs to make it possible to express various types of objective functions to minimize expected value or various measures of risk and to perform robust optimization. Support for multi-stage (e.g. bi-level) models must also be included given their prominence in numerous emerging applications in energy and infrastructures.

Dynamic Optimization: As optimization is increasingly moving from strategic planning to operational decision making under uncertainty, there is a need to support dynamic optimization applications, i.e., applications where the data, variables, and/or constraints evolve over time. In many such applications, a common model is extended over time by adding constraints expressing what was previously the unknown future or is applied to different scenarios capturing some future events. There is a need to find the proper modeling support for dynamic optimization, to develop a theory of incremental optimization supporting addition and deletion of constraints and variables, and to build efficient incremental solvers to support dynamic optimization better.

Debugging and Explanation: There is a tremendous lack of integrated development environments for optimization. As a result, modelers or programmers receive little or no help to debug their models/programs, to analyze performance, and to understand the behavior of their applications and the consequences of their modeling choices. This severely restricts access to optimization, since optimization tends to have rather complex control and data flows, due to non-deterministic search, constraint propagation, randomization, and sophisticated building blocks such as cuts, nogoods, and global constraints, whose behavior may be hard to understand.

We need to remedy this situation by building a sophisticated integrated development environment (IDE) for optimization systems. While the IDE itself can be (and indeed should be) produced using off the shelf systems such as Eclipse, the tools that are integrated in the IDE are crucial to helping develop the next generation of optimization practitioners. A key aspect of the research is to communicate some of the solver behavior in terms that are understandable to modelers, i.e., in terms of the original model. Another important feature will be a set of tools to "debug" the correctness of the models (e.g., identifying sets of infeasible constraints e.g [15], a feature already offered by systems like AIMMS [1]), to explore the performance of a given optimization technique (e.g., capturing how the search space is explored and what is performed at every node), to compare the performance of several solving techniques (e.g., by comparing profiling data of each technique on an array of benchmarks), and to automatically tune an optimization model. Developing these tools in a scalable, meaningful and, as much as possible, solver-independent way is very challenging, but even modest progress in this direction may make a significant difference on the practice in the field.

Simulation: Much "optimization" performed in practice is through simulation. An organization can build a simulation of their operations, and then simulate the effect of varying some configurations and/or business strategies on some scenarios in order to adjust their processes accordingly. Most organizations find it far easier to generate a simulation of their organization than to model it as a suite of optimization problems. Optimization technology developers need to integrate simulation and simulation optimization for use in modeling to cover this fundamental use of optimization and smooth the path from simulation to optimization (see e.g. [5, 11]). Simultaneously, many simulation tools are in need of optimization to address some of their challenges as they move closer to optimization. While the integration of optimization and simulation should provide a fertile ground to make significant contributions to the field, at present it is not heavily explored.

3.2 Model Analyses and Transformations

The availability of high-level "conceptual" models opens up some intriguing perspectives for optimization software. By capturing the structure of the application, conceptual models convey substantial information to the modeling system, which can then exploit it to derive "design" models that can be efficiently solved. In this section, we review some of these opportunities focusing on model analyses and transformations.

3.2.1 Model Analyses

Given a high-level conceptual model of an optimization problem, it is important for an optimization system to perform a static or runtime analysis to discover as much information regarding the model as it is needed to apply the appropriate model transformations. While the use of analysis techniques has been extensively explored in other programming paradigms, this has not yet been the case for constraint programming. This following paragraphs define a number of examples where model analyses can be used.

Automatic Recovery of Combinatorial Structures: A modeling language should aim to convey problem structure to the solver as explicitly as possible. However, non-experts may not always use the appropriate abstractions or may not recognize them as such in their models. Mixed integer programming (MIP) solvers uses a pre-processing steps which, among many other functionalities, tries to infer some implicit structure from the model (e.g., recognizing knapsack cuts by combining constraints [24]). A universal modeling system has a similar goal but for a much richer modeling language and a larger set of underlying solving technologies. A focus should be on automatically determine global substructure in the form of implied global constraints (as initiated by [3, 4]). Adding these global constraints to the model (with or without removing the constraints that imply them) can dramatically improve solving. More importantly it may make life considerably easier for other model analyses and transformations listed below. For example, current methods for automatic derivation of search procedures [8] strongly make use of global constraints. A better understanding of the global substructure can also help to transform a model written for one solver technology for another. So automatic derivation of implied constraints is an enabling analysis for many other analyses.

Automatic Recognition of Special Cases An analysis can detect when a model is linear, or when, for example, bounds consistency is the same as domain consistency. There are often several design choices for some parts of every model (e.g., time-indexed models or time-independent models in scheduling with MIP) and a long-term goal should be to automate this selection using model analyses. Preliminary research on scheduling models has shown the feasibility of this approach on a dedicated domain [18]: we need to generalize this approach to much broader classes of models.

Automatic Derivation of Symmetries and Dominance Relationships: Industrial applications often feature symmetries, conditional symmetries, and/or dominance relationships. As mentioned before, modelers should have the opportunity to express symmetries and dominances directly. But the modeling system should also be able to derive these relationships from the problem variables and constraints. Preliminary work has indicated the feasibility of this approach [17,26]. Once these symmetries and dominances are identified, the model can add symmetry-breaking constraints or implement dedicated search procedures dynamically breaking symmetries and dominances.

3.2.2 Model Transformations

Given a high-level conceptual model, it is possible to systematically derive design models that exploit the specific features of the application at hand (where the features might have been explicitly given by the user or automatically inferred by an analyzer). A language for capturing and expressing such transformations, like Cadmium [7], provides a uniform approach to recording and using transformations. This can provide a good starting point for building a library of reusable transformation components which can then be composed naturally to express complex model transformations. The following are three of the main research directions worth pursuing.

Automatic Derivation of Design Models: A drawback of most existing approaches to combinatorial optimization is the necessity of choosing a solver technology before expressing a model. This early decision can be wrong and thus lead to significant development effort which later has to be discarded entirely. Solver-independent conceptual models make it possible to delay this decision. However, they must be able to compile these high-level models into design models (e.g., CP or MIP models) and to find the "best" way to map high-level features into design models.

Automatic Derivation of Search Procedures: Although modelers must be able to specify search procedures if desired, it is likely that most modelers will choose not to use this feature and simply express the variables, constraints, and objective of their application. However, the high-level nature of the model makes it possible to derive dedicated search procedures automatically. The idea is to analyze the model constraints and objectives to derive dedicated heuristics that should outperform general-purpose search heuristics. Preliminary evidence on reasonably simple models has demonstrated the potential of this approach, which is however in its infancy [8].

Automatic Hybridization: Given a universal modeling language with a large variety of underlying optimization solvers, there is significant scope for developing hybrid optimization solutions. Model analyses, algorithm portfolios, or automatic model tuning may be used to find the best hybridization for a class of problems or instances.

3.3 Solver Technology

Over the last decade, as problems have become increasingly complex and the time available for running a solution algorithm has been reduced for some classes of applications, hybrid methods have emerged as the technology of choice. To maximize the ease of building hybrid optimization solutions we need a fully integrated optimization solver that smoothly hybridizes the main optimization technologies: constraint programming (CP), Boolean satisfiability (SAT), mixed integer programming (MIP), mixed integer non-linear programming (MINLP), local search (LS), and dynamic programming (DP). An August 2011 OR/MS article [12] eloquently articulated that CP has become a must-have tool in any O.R. practitioner's toolkit and hybridization is becoming the future of optimization. CP is ideally positioned to become the kernel of the new generation of optimization technologies because, since its inception, it has been an integration framework. Constraints in CP are treated separately and communicate with each other, and it is not a large step to move from this to treating models or submodels separately and communicating between them. In addition, CP begins with the most high-level modeling viewpoint, and hence is well placed to define new higher level hybrid modeling facilities.

The key challenge in designing an architecture for such an integrated system is to achieve the performance of a dedicated hybridization, while maintaining the compositionality of the architecture. This means that users only pay for the technologies they use and that no significant overhead is introduced by the architecture.

An ideal optimization system should be implemented as a suite of low-level libraries, itself consisting of multiple components for each of the major technologies. The glue between the various components, and their integration, should be a CP-based architecture now consisting of a number of collaborating constraint stores. We now examine a number of areas where significant progress is necessary to meet this overall vision.

From SAT to CP Explanations and No-Goods: A key advance in modern optimization is the advent of rapid methods to explain the reasons for failure and to record them as constraints, often called no-goods in artificial intelligence and Benders cuts in operations research. Nogoods have been vital to the success of SAT/SMT solvers and form the basis of lazy clause generation [19], a hybridization of CP and SAT, which is currently the state of the art for classes of hard scheduling problems [23]. The challenge now is to extend no-good generation to combinatorial/global constraints which are the cornerstone of the CP modeling methodology. These constraints are often difficult to explain succinctly and long explanations can suffer from limited re-usability. What is needed is the right way to explain the behavior of global constraints, yielding shorter nogoods that can be exploited during search.

Mixed Integer Nonlinear Programming: Many of the important problems facing our society, in particular in energy and infrastructure management, involves mixed nonlinear integer programming problems. They arise, for instance, in modeling electrical power systems and gas infrastructures. Nonlinear optimization problems are very hard to solve: they are generally tackled by globally convergent methods which produce local minima from a wide range of starting points. Unfortunately, these methods do not produce (dual) lower bounds which makes it difficult to evaluate the quality of solutions; they also do not produce any guarantee on the quality of local minima. However, there has been significant progress in MIP and convex optimization in recent

years, which creates opportunities to develop global optimization solvers exploiting both the discrete and continuous aspects of MINLP problems. New universal solver technology must include MINLP capabilities. We need an MINLP component which automatically and dynamically convexifies or linearizes non-convex constraints, making use of the combinatorial substructure, which are then communicated to a central linear or convex optimization solver. An MINLP component must also include traditional globally convergent methods and be fully integrated with the CP-based architecture to allow for flexible search and the exploitation of combinatorial substructures arising in our application areas.

Dynamic Programming: When applicable, dynamic programming is a highly effective methodology to solve optimization problems. This is typically the case when there is enough structure in the applications and when parts of the problems are loosely connected. Dynamic programming algorithms are often programmed from scratch and no framework is available to express dynamic programs naturally. An optimization technology platform should meet the following three objectives:

1. Provide a framework for expressing recursive equations that will then be compiled into efficient low-level program;
2. Derive automatically recursive equations from certain classes of high-level models;
3. Automate the incremental updates of dynamic programs so that they can be easily included in search, e.g., to compute lower bounds.

Alternatively, we can try to take advantage of dynamic programming approaches in CP search using dominance relations (see [6]).

Data intensive Optimization: One fundamental change in the optimization space is the availability of massive amounts of data. This has led to new classes of optimization applications such as data-mining, (constraint-based) clustering and pattern discovery and recognition. These applications arise in many areas including computational biology, social networks and forecasting. Moreover, some of these areas are evolving from very dedicated algorithms, specialized to a given task, to more general frameworks. Indeed, some data-mining projects have shown that constraint programming is a flexible and efficient tool for addressing some traditional data-mining tasks [13].

It is clear however that optimization technology has to evolve to meet the demands of some of these applications. Some of them feature few variables but extremely large domains (e.g., large DNA or RNA sequences), others feature huge number of variables and constraints (e.g., mining documents), while others deal with extremely large graphs that must be clustered with a variety of constraints on the clusters. We need to design the data structures and optimization algorithms appropriate for data-intensive optimization.

Large-Scale Parallel Optimization: The last couple of years have seen a fundamental change in hardware evolution. Processors are not doubling speed every 18 months or so. Rather, the industry has moved from single core desktop, to multi-core computers, large clusters, and clouds of computing resources.

Optimization software has benefited tremendously from advances in processor speed, making it possible to solve increasingly complex applications or problems which must be solved with time constraints. However, improvements in hardware will no longer translate into speedups for optimization software: It will become necessary to

exploit parallel computing resources which may be challenging given the nature of optimization algorithms.

Recent research has demonstrated that parallel computing on multi-core architectures can bring significant benefits and that communication starts becoming a significant issue when dealing with more than 1,000 processors [32]. One of the goals for parallel optimization will be to build a novel architecture to scale to larger clouds, which may fundamentally change the scope of highly complex problems and data-intensive optimizations. Another goal should be to study how to parallelize hybrid and decomposition algorithms, since they require more synchronization and communication.

4 Conclusion

In our opinion the future for constraint programming is very bright. The principle reason is that CP concentrates on a high level view of problem structure, and is able to take advantage of that structure. As we gain new insights into modeling, model analysis, model transformations, global constraint solving, and hybridization, the benefits of this new optimization technology can be utilized without expert knowledge. As we tackle more and more complex optimization problems, the combination of optimization algorithms and programming language concepts that is constraint programming, will become more and more essential.

Acknowledgments We would like to thank the anonymous reviewers for their comments which helped substantially improve this article. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Center of Excellence program.

References

1. Aimms modelling system. http://business.aimms.com.
2. I. D. Aron, J. N. Hooker, and T. H. Yunes. Simpl: A system for integrating optimization techniques. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, First International Conference, CPAIOR 2004*, volume 3011 of *Lecture Notes in Computer Science*, pages 21–36. Springer, 2004.
3. N. Beldiceanu and H. Simonis. A constraint seeker: Finding and ranking global constraints from examples. In J. H.-M. Lee, editor, *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference*, volume 6876 of *Lecture Notes in Computer Science*, pages 12–26. Springer, 2011.
4. N. Beldiceanu and H. Simonis. A model seeker: Extracting global constraint models from positive examples. In M. Milano, editor, *Proceedings of the 18th International Conference of Principles and Practice of Constraint Programming, CP 2012,*, volume 7514 of *Lecture Notes in Computer Science*, pages 141–157. Springer, 2012.
5. A. Brodsky and H. Nash. CoJava: optimization modeling by nondeterministic simulation, in constraint programming. In *Principles and Practice of Constraint Programming (CP)*, pages 91–107, 2006.
6. G. Chu, M. Garcia de la Banda, and P. Stuckey. Automatically exploiting subproblem equivalence in constraint programming. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 6140 of *LNCS*, pages 71–86. Springer, 2010.
7. G. Duck, L. De Koninck, and P. Stuckey. Cadmium: An implementation of ACD term rewriting. In M. G. de la Banda and E. Pontelli, editors, *Proceedings of the 24th International Conference on Logic Programming*, LNCS, pages 531–545. Springer, 2008.

8. S. Elsayed and L. Michel. Synthesis of search algorithms from high-level CP models. In J. Lee, editor, *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming*, volume 6876 of *LNCS*, pages 256–270. Springer, 2011.
9. T. Feydy, Z. Somogyi, and P. Stuckey. Half-reification and flattening. In J. Lee, editor, *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming*, volume 6876 of *LNCS*, pages 286–301. Springer, 2011.
10. R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming.* Duxbury Press, 2002.
11. K. Francis, S. Brand, and P. Stuckey. Optimization modelling for software developers. In M. Milano, editor, *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming*, page to appear. Springer, 2012.
12. H. Ganu. Constraint programming. In *ORMS Today*, pages 44–47. August 2011.
13. T. Guns, S. Nijssen, and L. D. Raedt. Itemset mining: A constraint programming perspective. *Artificial Intelligence*, 175(12–13):1951–1983, 2011.
14. W. Harvey and T. Kelsey. Symmetry group expression for CSPs. In *Proceedings of SymCon03: Third International Workshop on Symmetry in Constraint Satisfaction Problems*, pages 86–96, 2003.
15. U. Junker. Quickxplain: Preferred explanations and relaxations for over-constrained problems. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence*, pages 167–172. AAAI Press / The MIT Press, 2004.
16. K. Marriott, N. Nethercote, R. Rafeh, P. Stuckey, M. Garcia de la Banda, and M. Wallace. The design of the Zinc modelling language. *Constraints*, 13(3):229–267, 2008.
17. C. Mears, M. G. de la Banda, and M. Wallace. On implementing symmetry detection. *Constraints*, 14(4):443–477, 2009.
18. J.-N. Monette, Y. Deville, and P. Van Hentenryck. *Aeon: Synthesizing Scheduling Algorithms from High-Level Models*, pages 43–59. Operations Research/Computer Science Interfaces. Springer, 2009.
19. O. Ohrimenko, P. Stuckey, and M. Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, 2009.
20. L. Perron. Search procedures and parallelism in constraint programming. In J. Jaffar, editor, *Fifth International Conference on Principles and Practice of Constraint Programming*, volume 1713 of *LNCS*, pages 346–360. Springer, 1999.
21. J. Puchinger, P. Stuckey, M. Wallace, and S. Brand. Dantzig-wolfe decomposition and branch-and-price solving in G12. *Constraints*, 16(1):77–99, 2011.
22. T. Schrijvers, G. Tack, P. Wuille, H. Samulowitz, and P. Stuckey. Search combinators. In J. Lee, editor, *Seventeenth International Conference on Principles and Practice of Constraint Programming*, volume 6876 of *LNCS*, pages 774–788. Springer, 2011.
23. A. Schutt, T. Feydy, P. Stuckey, and M. Wallace. Explaining the cumulative propagator. *Constraints*, 16(3):250–282, 2011.
24. M. Trick. Formulations and reformulations in integer programming. In *Proceedings of the Second International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'05)*, 2005.
25. P. Van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, 1989.
26. P. Van Hentenryck, P. Flener, J. Pearson, and M. Agren. Compositional derivation of symmetries for constraint satisfaction. In *Proceedings of the 6th International Symposium on Abstraction, Reformulation and Approximation, (SARA 2005)*, pages 234–247, 2005.
27. P. Van Hentenryck, I. Lustig, L. Michel, and J.-F. Puget. *The OPL Optimization Programming Language.* MIT Press, 1999.
28. P. Van Hentenryck and L. Michel. *Constraint-Based Local Search.* MIT Press, 2005.
29. P. Van Hentenryck and L. Michel. *Constraint-Based Local Search.* MIT Press, 2005.
30. P. Van Hentenryck, L. Perron, and J.-F. Puget. Search and strategies in OPL. *ACM TOCL*, 1(2):285–315, 2000.
31. M. Wallace, S. Novello, and J. Schimpf. Eclipse: A platform for constraint logic programming. Technical report, IC-Parc Imperial College, London,, 1997.
32. F. Xie and A. J. Davenport. Massively parallel constraint programming for supercomputers: Challenges and initial results. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 6140 of *LNCS*, pages 334–338. Springer, 2010.