Chapter 1

Introduction

Francesca Rossi, Peter van Beek, Toby Walsh

Constraint programming is a powerful paradigm for solving combinatorial search problems that draws on a wide range of techniques from artificial intelligence, computer science, databases, programming languages, and operations research. Constraint programming is currently applied with success to many domains, such as scheduling, planning, vehicle routing, configuration, networks, and bioinformatics. The basic idea in constraint programming is that the user states the constraints and a general purpose constraint solver is used to solve them. Constraints are just relations, and a constraint satisfaction problem (CSP) states which relations should hold among the given decision variables. For example, in scheduling activities in a company, the decision variables might be the starting times and the durations of the activities and the resources needed to perform them, and the constraints might be on the availability of the resources and on their use for a limited number of activities at a time.

Constraint solvers take a real-world problem like this, represented in terms of decision variables and constraints, and find an assignment to all the variables that satisfies the constraints. Constraint solvers search the solution space either systematically, as with backtracking or branch and bound algorithms, or use forms of local search which may be incomplete. Systematic method often interleave search and inference, where inference consists of propagating the information contained in one constraint to the neighboring constraints. Such inference (usually called constraint propagation) is useful since it may reduce the parts of the search space that need to be visited.

While defining a set of constraints may seem a simple way to model a real-world problem, finding a good model that works well with a chosen solver is not always easy. A poorly chosen model may be very hard to solve. Thus much care must be devoted to choosing a good model and also to devising solvers that can exploit the features of the chosen model.

From this description it may seem that constraint programming is "programming" in the sense of "mathematical programming": the user declaratively states the constraints on the feasible solutions for a set of decision variables, and an underlying solver solves the constraints. However, constraint programming is also "programming" in the sense of "computer programming": the user needs to program a strategy to search for a solution.

Without this, the solving process would be very inefficient. This is very natural to do in logic-based programming languages, such as constraint logic programming, but it can also be done in other programming paradigms.

1.1 Purpose of the Handbook

The aim of this handbook is to capture the full breadth and depth of the constraint programming field and to be encyclopedic in its scope and coverage. While there are excellent books on constraint programming (see, for example, [1, 2, 3, 4, 5, 6, 7, 8]), such books necessarily focus on the main notions and techniques and cannot cover also extensions, applications, and languages. The handbook gives a reasonably complete coverage of all these lines of work, based on constraint programming, so that a reader can have a rather precise idea of the whole field and its potential. Of course each line of work is dealt with in a survey-like style, where some details may be neglected in favor of broader coverage. However, the extensive bibliography of each chapter will help the interested readers to find suitable sources for the missing details. Each chapter of the handbook is intended to be a self-contained survey of a topic, and is written by one or more authors who are leading researchers in the area.

The intended audience of the handbook is researchers, graduate students, upper-year undergraduates, and practitioners who wish to learn about the state-of-the-art in constraint programming. No prior knowledge about the field is necessary to be able to read the chapters and gather useful knowledge. Researchers from other fields should find in this handbook an effective way to learn about constraint programming and to possibly use some of the constraint programming concepts and techniques in their own work, thus providing a means for a fruitful cross-fertilization among different research areas.

1.2 Structure and Content

The handbook is organized in two parts. The first part covers the basic foundations of constraint programming, including the history, the notion of constraint propagation, basic search methods, global constraints, tractability and computational complexity, and important issues in modeling a problem as a constraint problem. The second part covers constraint languages and solver, several useful extensions to the basic framework (such as interval constraints, structured domains, and distributed CSPs), and successful application areas for constraint programming.

Part I: Foundations

In Chapter 2, Eugene C. Freuder and Alan K. Mackworth survey the emergence of constraint satisfaction as a new paradigm within artificial intelligence and computer science. Covering the two decades from 1965 to 1985, Freuder and Mackworth trace the development of two streams of work, which they call the language stream and the algorithm stream. The focus of the language stream was on declarative program languages and systems for developing applications of constraints. The language stream gave many special purpose declarative languages and also general programming languages such as constraint logic programming. The focus of the algorithm stream was on algorithms and heuristics

for the constraint satisfaction framework. The algorithm stream gave constraint propagation algorithms such as algorithms for arc consistency and also heuristics and constraint propagation within backtracking search. Ultimately, the language stream and the algorithm stream merged to form the core of the new field of constraint programming.

In Chapter 3, Christian Bessiere surveys the extensive literature on constraint propagation. Constraint propagation is a central concept—perhaps *the* central concept—in the theory and practice of constraint programming. Constraint propagation is a form of reasoning in which, from a subset of the constraints and the domains, more restrictive constraints or more restrictive domains are inferred. The inferences are justified by local consistency properties that characterize necessary conditions on values or set of values to belong to a solution. Arc consistency is currently the most important local consistency property in practice and has received the most attention in the literature. The importance of constraint propagation is that it can greatly simplify a constraint problem and so improve the efficiency of a search for a solution.

The main algorithmic techniques for solving constraint satisfaction problems (CSPs) are backtracking search and local search. In Chapter 4, Peter van Beek surveys backtracking search algorithms. A backtracking search algorithm performs a depth-first traversal of a search tree, where the branches out of a node represent alternative choices that may have to be examined in order to find a solution, and the constraints are used to prune subtrees containing no solutions. Backtracking search algorithms come with a guarantee that a solution will be found if one exists, and can be used to show that a CSP does not have a solution or to find a provably optimal solution. Many techniques for improving the efficiency of a backtracking search algorithm have been suggested and evaluated including constraint propagation, nogood recording, backjumping, heuristics for variable and value ordering, and randomization and restart strategies.

In Chapter 5, Holger H. Hoos and Edward Tsang survey local search algorithms for solving constraint satisfaction problems. A local search algorithm performs a walk in a directed graph, where the nodes represent alternative assignments to the variables that may have to be examined and the number of violated constraints is used to guide the search for a solution. Local search algorithms cannot be used to show that a CSP does not have a solution or to find a provably optimal solution. However, such algorithms are often effective at finding a solution if one exists and can be used to find an approximation to an optimal solution. Many techniques and strategies for improving local search algorithms have been proposed and evaluated including randomized iterative improvement, tabu search, penalty-based approaches, and alternative neighborhood and move strategies.

In Chapter 6, Willem-Jan van Hoeve and Irit Katriel survey global constraints. A global constraint is a constraint that can be over arbitrary subsets of the variables. The canonical example of a global constraint is the `all-different` constraint which states that the variables in the constraint must be pairwise different. The power of global constraints is two-fold. First, global constraints ease the task of modeling an application using constraint programming. The `all-different` constraint, for example, is a pattern that reoccurs in many applications, including rostering, timetabling, sequencing, and scheduling applications. Second, special purpose constraint propagation algorithms can be designed which take advantage of the semantics of the constraint and are therefore much more efficient. Van Hoeve and Katriel show that designing constraint propagation algorithms for global constraints draws on a wide variety of disciplines including graph theory, flow theory, matching theory, linear programming, and finite automaton.

A fundamental challenge in constraint programming is to understand the computational complexity of problems involving constraints. In their most general form, constraint satisfaction problems (CSPs) are NP-Hard. To counter this pessimistic result, much work has been done on identifying restrictions on constraint satisfaction problems such that solving an instance can be done efficiently; that is, in polynomial time in the worst-case. Finding tractable classes of constraint problems is of theoretical interest of course, but also of practical interest in the design of constraint programming languages and effective constraint solvers. The restrictions on CSPs that lead to tractability fall into two classes: restricting the topology of the underlying graph of the CSP and restricting the type of the allowed constraints. In Chapter 7, Rina Dechter surveys how the complexity of solving CSPs varies with the topology of the underlying constraint graph. The results depend on properties of the constraint graph, such as the well-known graph parameter tree-width. In Chapter 8, David Cohen and Peter Jeavons survey how the complexity of solving CSPs varies with the type of allowed constraints. Here, the results depend on algebraic properties of the constraint relations.

The first part ends with three chapters concerned with modeling real world problems as CSPs. In many real world problems, not all constraints are hard. Some constraint may be "soft" and express preferences that we would like to satisfy but do not insist upon. Other real world problems may be over-constrained. In both cases, an extension of the basic framework of constraint satisfaction to soft constraints is useful. In Chapter 9, Pedro Meseguer, Francesca Rossi, and Thomas Schiex survey the different formalisms of soft constraints proposed in the literature. They describe the relationship between these different formalisms. In addition, they discuss how solving methods have been generalized to deal with soft constraints.

Symmetry occurs in many real world problems: machines in a factory might be identical, nurses might have the same skills, delivery trucks might have the same capacity, etc. Symmetry can also be introduced when we model a problem as a CSP. For example, if we introduce a decision variable for each machine, then we can permute those variables representing identical machines. Such symmetry enlarges the search space and must be dealt with if we are to solve problems of the size met in practice. In Chapter 10, Ian P. Gent, Karen E. Petrie, and Jean-François Puget survey the different forms of symmetry in constraint programming. They describe the three basic techniques used to deal with symmetry: reformulating the problem, adding symmetry breaking constraints, and modifying the search strategy to ignore symmetric states. Symmetry is one example of the sort of issues that need to be considered when modeling a problem as a CSP. In Chapter 11, Barbara M. Smith surveys a range of other issues in modeling a problem as a CSP. This includes deciding on an appropriate viewpoint (e.g. if we are scheduling exams, do the variables represent exams and their values the times, or do the variables represent the times and their values the exams?), adding implied constraints to help prune the search space, and introducing auxiliary variables to make it easier to state the constraints or to improve propagation.

Part II: Extensions, Languages, and Applications

To increase the uptake, ease of use, extensibility, and flexibility of constraint technology, constraints and search have been integrated into several programming languages and programming paradigms. In Chapter 12, Kim Marriott, Peter J. Stuckey, and Mark Wallace

survey constraint logic programming (CLP), the integration of constraint solving into logic programming languages. Constraint solving and logic programming are both declarative paradigms, so their integration is quite natural. Further, the fact that constraints can be seen as relations or predicates, that a set of constraints can be viewed as the conjunction of the individual constraints, and that backtracking search is a basic methodology for solving a set of constraints, makes constraint solving very compatible with logic programming, which is based on predicates, logical conjunctions, and backtracking search. Marriott, Stuckey, and Wallace cover the elegant semantics of CLP, show the power of CLP in modeling constraint satisfaction problems, and describe how to define specific search routines in CLP for solving the modeled problem.

In Chapter 13, Thom Frühwirth, Laurent Michel, and Christian Schulte survey the integration of constraints into procedural and object-oriented languages, concurrent languages, and rule-based languages. Integrating constraint solving into these more traditional programming paradigms faces new challenges as these paradigms generally lack support for declarative programming. These challenges include (i) allowing the specification of new search routines, while maintaining declarativeness, (ii) the design of declarative modeling languages that are user-friendly and based on well-known programming metaphors, and (iii) the integration of constraint solving into *multi*-paradigm languages. Frühwirth, Michel, and Schulte include a discussion of the technical aspects of integrating constraints into each programming paradigm, as well as the advantages and disadvantages of each paradigm.

In Chapter 14, Christian Schulte and Mats Carlsson survey finite domain constraint programming systems. One of the key properties of constraint programming systems is the provision of widely reusable services—such as constraint propagation and backtracking search—for constructing constraint-based applications. Schulte and Carlsson discuss which services are provided by constraint programming systems and also the key principles and techniques in implementing and coordinating these services. For many applications, the constraint propagation, backtracking search, and other services provided by the constraint programming system are sufficient. However, some applications require more, and most constraint programming systems are extensible, allowing the user to define, for example, new constraint propagators or new search strategies. Schulte and Carlsson also provide an overview of several well-known finite domain constraint programming systems.

Operations research (OR) and constraint programming (CP) are complementary frameworks with similar goals. In Chapter 15, John N. Hooker surveys some of the schemes for incorporating OR methods into CP. In constraint programming, constraints are used to reduce the domains of the variables. One method for incorporating an OR method is to apply it to a constraint to reduce the domains. For example, if a subset of the constraints are linear inequalities, the domain of a variable in the subset can possibly be reduced by minimizing and maximizing the variable using linear programming on the subset of linear constraints. This example is an instance of a popular scheme for incorporating OR into CP: create a relaxation of the CP problem in the form of an OR model, such as a linear programming model. Other schemes for creating hybrid OR/CP combinations decompose a problem so that CP and OR are each used on the parts of the problem to which they are best suited. Hooker shows that OR/CP combinations using both relaxation and decomposition can bring substantial computational benefits.

Real-world problems often take us beyond finite domain variables. For example, to reason about power consumption, we might want a variable to range over the reals and

to reason about communication networks we might want a variable to range over paths in a graph. Constraint programming has therefore been extended to deal with more than just finite (or enumerated) domains of values. In Chapter 16, Frédéric Benhamou and Laurent Granvilliers survey constraints over continuous and interval domains. The extension of backtracking search over finite domains to interval constraints is called branch-and-reduce: branching splits an interval and reduce narrows the intervals using a generalization of local consistency and interval arithmetic. Hybrid techniques combining symbolic reasoning and constraint propagation have also been designed. Benhamou and Granvilliers also discuss some of the applications of interval constraints and the available interval constraint software packages. In Chapter 17, Carmen Gervet surveys constraints over structured domains. Many combinatorial search problems—such as bin packing, set covering, and network design—can be naturally represented in the language of sets, multi-sets, strings, graphs and other structured objects. Constraint propagation has therefore been extended to deal with constraints over variables which range over such datatypes.

Early work in empirical comparisons of algorithms for solving constraint satisfaction problems was hampered by a lack of realistic or hard test problems. The situation improved with the discovery of hard random problems that arise at a phase transition and the investigation of alternative random models of constraint satisfaction, satisfiability, and optimization problems. Experiments could now be performed which compared the algorithms on the hardest problems and systematically explored the entire space of random problems to see where one algorithm bettered another. In Chapter 18, Carla Gomes and Toby Walsh survey these alternative random models. In addition to their interest as an experimental testbed, insight gained from the study of hard problems has also led to the design of better algorithms. As one example, Gomes and Walsh discuss the technique of randomization and restarts for improving the efficiency of backtracking search algorithms.

In Chapter 19, Manolis Koubarakis surveys temporal constraint satisfaction problems for representing and reasoning with temporal information. Temporal reasoning is important in many application areas—including natural language understanding, database systems, medical information systems, planning, and scheduling—and constraint satisfaction techniques play a large role in temporal reasoning. Constraint-based temporal reasoning formalisms for representing qualitative, metric, and combined qualitative-metric temporal information have been proposed in the literature and many efficient constraint satisfaction algorithms are known for these formalisms. Koubarakis also demonstrates the application-driven need for more expressive queries over temporal constraint satisfaction (especially queries combining temporal and non-temporal information) and surveys various proposals that address this need including the scheme of indefinite constraint databases.

In Chapter 20, Boi Faltings surveys distributed constraint satisfaction. In distributed constraint satisfaction, constraint solving happens under the control of different independent agents, where each agent controls a single variable. The canonical example of the usefulness of this formalism is meeting scheduling, where each person has their own constraints and there are privacy concerns that restrict the flow of information, but many applications have been identified. Backtracking search and its improvements have been extended to the distributed case. In synchronous backtracking, messages are passed from agent to agent with only one agent being active at any one time. A message consists of either a partial instantiation or a message that signals the need to backtrack. In asynchronous backtracking, all agents are active at once, and messages are sent to coordinate their the assignments that are made to their individual variables. Asynchronous backtracking has been

the focus of most of the work in distributed constraint satisfaction. Faltings also surveys the literature on open constraint satisfaction, a form of distributed CSP where the domains of the variables and the constraints may be incomplete or not fully known.

The basic framework of constraint programming makes two assumptions that do not hold in many real world problems: that the problem being modeled is static and that the constraints are known with certainty. For example, factory scheduling is inherently dynamic and uncertain since the full set of jobs may not be known in advance, machines may break down, employees may be late or ill, and so on. In Chapter 21, Kenneth N. Brown and Ian Miguel survey the uses and extensions of constraint programming for handling problems subject to change and uncertainty. For dynamically changing problems, two of the alternatives are to record information about the problem structure during the solving process, such as explanation or nogood recording, so that re-solving can be done efficiently; and to search for robust or solutions that anticipate expected changes. For uncertain problems, different types of uncertainty can be identified including: the problem itself is intrinsically imprecise; there is a set of possible realizations of the problem, one of which will be the final version, and there are probability distributions over the full realizations. As well, many CSP formalisms have been proposed for handling uncertainty including fuzzy, mixed, uncertain, probabilistic, stochastic, and recurrent CSPs.

Constraint programming has proven useful—indeed, it is often the method of choice—in important applications from industry, business, manufacturing, and science. In the last five chapters of the handbook, some of these applications of constraint programming are highlighted. Each of the chapters emphasizes *why* constraint programming has been successful in the given application domain. As well, in the best traditions of application-driven research, the chapters describe how focusing on real-world applications has led to basic discoveries and improvements to existing constraint programming techniques. In a fruitful cycle, these discoveries and improvements then led to new and more successful applications.

In Chapter 22, Philippe Baptiste, Philippe Laborie, Claude Le Pape, and Wim Nuijten survey constraint programming approaches to scheduling and planning. Scheduling is the task of assigning resources to a set of activities to minimize a cost function. Scheduling arises in diverse settings including in the allocation of gates to incoming planes at an airport, crews to an assembly line, and processes to a CPU. Planning is a generalization of scheduling where the set of activities to be scheduled is not known in advance. Constraint programming approaches to scheduling and planning have aimed at generality, with the ability to seamlessly handle real-world side constraints. As well, much effort has gone into improved implied constraints such as global constraints, edge-finding constraints and timetabling constraints, which lead to powerful constraint propagation. Baptiste et al. show that one of the reasons for the success of a constraint programming approach is its ability to integrate efficient special purpose algorithms within a flexible and expressive paradigm. Additional advantages of a constraint propagation approach include the ability to form hybrids of backtracking search and local search and the ease with which domain specific scheduling and planning heuristics can be incorporated within the search routines.

In Chapter 23, Philip Kilby and Paul Shaw survey constraint programming approaches to vehicle routing. Vehicle Routing is the task of constructing routes for vehicles to visit customers at minimum cost. A vehicle has a maximum capacity which cannot be exceeded and the customers may specify time windows in which deliveries are permitted. Much work on constraint programming approaches to vehicle routing has focused on alternative

constraint models and additional implied constraints to increase the amount of pruning performed by constraint propagation. Kilby and Shaw show that constraint programming is well-suited for vehicle routing because of its ability to handle real-world (or side) constraints. Vehicle routing problems that arise in practice often have unique constraints that are particular to a business entity. In non-constraint programming approaches, such side constraints often have to be handled in an ad hoc manner. In constraint programming a wide variety of side constraints can be handled simply by adding them to the core model.

In Chapter 24, Ulrich Junker surveys constraint programming approaches to configuration. Configuration is the task of assembling or configuring a customized system from a catalog of components. Configuration arises in diverse settings including in the assembly of home entertainment systems, cars and trucks, and travel packages. Junker shows that constraint programming is well-suited to configuration because of (i) its flexibility in modeling and the declarativeness of the constraint model, (ii) the ability to explain a failure to find a customized system when the configuration task is over-constrained and to subsequently relax the user's constraints, (iii) the ability to perform interactive configuration where the user makes a sequence of choices and after each choice constraint propagation is used to restrict future possible choices, and (iv) the ability to incorporate reasoning about the user's preferences.

In Chapter 25, Helmut Simonis surveys constraint programming approaches to applications that arise in electrical, water, oil, and data (such as the Internet) distribution networks. The applications include design, risk analysis, and operational control of the networks. Simonis discusses the best alternative formulations or constraint models for these problems. The constraint programming work on networks vividly illustrates the advantages of application-driven research. The limited success in this domain of classical constraint programming approaches, such as backtracking search, led to improvements in hybrid approaches which combine both backtracking and local search or combine both constraint programming and operations research methods. A research hurdle that must still be overcome, however, is the complexity and implementation effort that is required to construct a successful hybrid system for an application.

In Chapter 26, Rolf Backofen and David Gilbert survey constraint programming approaches to problems that arise in bioinformatics. Bioinformatics is the study of informatics and computational problems that arise in molecular biology, evolution, and genetics. Perhaps the first and most well-known example problem in bioinformatics is DNA sequence alignment. More recently, constraint programming approaches have made significant progress on the important problem of protein structure prediction. The ultimate goals and implications of bioinformatics are profound: better drug design, identification of genetic risk factors, gene therapy, and genetic modification of food crops and animals.

1.3 Future Research

The field of constraint programming is rapidly progressing. Many new research results are being published and new research areas are being opened in the field of constraint reasoning. We conclude this introduction with some speculation on lines of research that appear interesting and promising to us, and that in the future could be mature enough to constitute entire chapters in future revisions of this handbook.

Quantified constraint problems are a very interesting extension of classical CSPs where some variables may be universally quantified. This can help modeling scenarios where uncertainty does not allow us to decide the values for some variables. Many theoretical results on the complexity of such problems have already been developed. We envision a fast growth of this area and its applications.

When using a constraint solver, often it is not easy to understand what went wrong, or why a certain solution is returned rather than another one. *Explanation* tools could greatly help in making constraint technology easy to use in an interactive system. In general, *user interaction* in constraint systems deserves much attention and effort. Improvements in this respect could greatly widen the usability of constraint-based tools.

It is rare that all constraints are collected at the same time from the user of a constraint system. Usually such constraints, or preferences, are collected some at a time, but the system must be able to perform some amount of reasoning also with partial knowledge. Moreover, based on the partial knowledge it has, it should be able to ask the user only for those constraints or preferences that are useful to make the next inference. The issue of *preference elicitation* is crucial in such situations, and allows users to intelligently interact with a constraint system without being forced to state all their constraints, or preferences, at the beginning of the interaction. This can also be useful in scenarios where the users want to avoid revealing all their preferences, for example for privacy reasons.

Even when the user is willing to state all the information at the beginning of the interaction, sometimes it may be difficult for him to actually state it in terms of constraints. For example, it could be easier to state examples of desirable or unacceptable solutions. In this cases, machine *learning techniques* can be helpful to learn the constraints from the partial and possibly imprecise user statements. As for explanation and preference elicitation, this can greatly help in easing the interaction between users and constraint solvers.

Satisfiability is a mature research area with much interaction with constraint reasoning, since a satisfiability problem is just a constraint problem with Boolean variables. Thus, many theoretical results can be adapted from one field to the other one. We hope to see many such results in the future.

This handbook contains chapters on just some of the main application areas for constraint programming. Other application fields, which look very promising, are design, constraint databases, web services, global computing, and security. We hope to see constraint programming to be the base of many useful tools for such applications.

Acknowledgements

A project like this, which lasted almost two years and involved about sixty people, would not be possible without the support and encouragement of a great many people within the constraint programming community. First, we wish to thank the many authors of the chapters within this handbook. Many of them also helped us by reviewing other chapters. Additionally, we would like to thank Claire Bagley, Roman Bartak, Andrei Bulatov, Martin Henz, Andrea Lodi, Michela Milano, Luis Quesada, Francesco Scarcello, Peter Van Roy, and Roland Yap, who reviewed other chapters. Thanks also to Ugo Montanari, a pioneer of constraint programming, who wrote the foreword for the book.

We also wish to thank Zeger Karssen, originally at Elsevier and now at Atlantic Press, and Bernhard Nebel, one of the editors of the series where this book will appear. They

have been very enthusiastic about this project since the very first time we described it to them in the Summer of 2004. Zeger and his assistants have helped us greatly to put the project together and to smoothly reach a satisfactory agreement on the format and style of the book.

Finally, we also would like to thank Helmut Simonis, who, besides being an author of the handbook, provided the very nice cover picture for this handbook. We think his beautiful rose can represent very well the spirit of this handbook: the petals are the many authors, who worked together in cooperation to produce what we hope is a book as beautiful as this rose.

Bibliography

[1] K. R. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.
[2] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.
[3] F. Fages. *Programmation logique par contraintes*. Ellipses Marketing, 1998.
[4] T. Frühwirth and S. Abdennadher. *Essentials of Constraint Programming*. Springer, 2003.
[5] J. Hooker. *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction*. Wiley-Interscience, 2000.
[6] K. Marriott and P. J. Stuckey. *Programming with Constraints*. The MIT Press, 1998.
[7] E. P. K. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.
[8] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT Press, 1989.