CONSTRAINT-BASED SCHEDULING:
Applying Constraint Programming to Scheduling Problems
CONSTRAINT-BASED SCHEDULING:
Applying Constraint Programming to Scheduling Problems

PHILIPPE BAPTISTE
CNRS, UMR 6599 Heudiasyc
Université de Technologie de Compiègne
F-60200 Compiègne

CLAUDE LE PAPE
ILOG
9, rue de Verdun
F-94253 Gentilly

WIM NUIJTEN
ILOG
9, rue de Verdun
F-94253 Gentilly

SPRINGER SCIENCE+BUSINESS MEDIA, LLC
Contents

Foreword ix
Preface xi
Acknowledgments xiii

1. INTRODUCTION 1
  1.1 Introduction to Constraint Programming 1
  1.2 Scheduling Theory 9
    1.2.1 Graham's Classification 10
  1.3 A Constraint-Based Scheduling Model 12
    1.3.1 Activities 12
    1.3.2 Temporal Relations 15
    1.3.3 Resource Constraints 15
    1.3.4 Extensions of the Model 16
    1.3.5 Objective Function 18

2. PROPAGATION OF THE ONE-MACHINE RESOURCE CONSTRAINT 19
  2.1 Non-Preemptive Problems 19
    2.1.1 Time-Table Constraint 19
    2.1.2 Disjunctive Constraint Propagation 21
    2.1.3 Edge-Finding 22
    2.1.4 Not-First, Not-Last 26
    2.1.5 More Propagation 29
  2.2 Preemptive Problems 31
    2.2.1 Time-Table Constraint 32
    2.2.2 Disjunctive Constraint Propagation 33
    2.2.3 Network-Flow Based Constraints 33
    2.2.4 Preemptive Edge-Finding 37

3. PROPAGATION OF CUMULATIVE CONSTRAINTS 43
  3.1 Fully Elastic Problems 44
3.2 Preemptive Problems 46
  3.2.1 Time-Table Constraint 46
  3.2.2 Disjunctive Constraint 46
  3.2.3 Partially Elastic Relaxation 46

3.3 Non-Preemptive Problems 58
  3.3.1 Time-Table Constraint 58
  3.3.2 Disjunctive Constraint 59
  3.3.3 Edge-Finding 59
  3.3.4 Extended Edge-Finding 64
  3.3.5 Not-First, Not-Last 65
  3.3.6 Energetic Reasoning 67

4. COMPARISON OF PROPAGATION TECHNIQUES 77
  4.1 Constraint Propagation Rules 77
    4.1.1 Time-Table Constraints 80
    4.1.2 Disjunctive Constraints 81
    4.1.3 Edge-Finding 81
    4.1.4 Not-First, Not-Last 83
    4.1.5 Energetic Reasoning 83
  4.2 Dominance Relations 85
    4.2.1 The Fully Elastic Case 87
    4.2.2 The Cumulative Preemptive Case 87
    4.2.3 The One-Machine Preemptive Case 87
    4.2.4 The Cumulative Non-Preemptive Case 88
    4.2.5 The One-Machine Non-Preemptive Case 91
  4.3 Non-Dominance Relations 92
    4.3.1 General Counterexamples 93
    4.3.2 A One-Machine Preemptive Counterexample 96
    4.3.3 Cumulative Counterexamples 96
  4.4 Summary 99
    4.4.1 The Fully Elastic Case 99
    4.4.2 The Cumulative Preemptive Case 100
    4.4.3 The One-Machine Preemptive Case 100
    4.4.4 The Cumulative Non-Preemptive Case 100
    4.4.5 The One-Machine Non-Preemptive Case 101

5. PROPAGATION OF OBJECTIVE FUNCTIONS 105
  5.1 Total Weighted Number of Late Activities 105
    5.1.1 Complexity Results 106
    5.1.2 A Lower Bound of the Number of Late Activities 108
    5.1.3 Constraint Propagation 117
  5.2 Sum of Transition Times and Sum of Transition Costs 122
    5.2.1 Route Optimization Constraint 124
    5.2.2 Precedence Graph Constraint 126
  5.3 Conclusion 127
Foreword

One of the roots of constraint programming is artificial intelligence where researchers focused on the use of logics and deduction for the resolution of complex problems. Principles of constraint programming became precise in the late 1970's and early 1980's. These principles are: deduction of additional constraints from existing ones by logical reasoning, and the application of search algorithms which are used to explore the solution space. Although these principles are very general they have been most successfully implemented in connection with scheduling problems. Scheduling problems are usually defined by temporal and capacity constraints which can be formulated explicitly in an easy way. Furthermore, efficient operations research algorithms can be adapted to the constraint programming framework in connection with scheduling problems. Early success in this area stimulated further research aimed at the design and implementation of further algorithms embeddable into constraint programming tools.

Based on these ideas ILOG S.A., a French software company, developed a commercial constraint programming solver and constraint-based scheduling and vehicle routing systems. A fruitful interaction between basic research and software development resulted in further progress in constraint-based scheduling.

The authors of this book are part of this success story. As software developer and project manager they have been involved in both the theoretical foundation and implementation of ILOG's constraint programming systems. Their excellent articles on constraint-based scheduling which resulted from this work are well known. The book covers the most widely used constraint-based scheduling techniques. It is a valuable source for students, practitioners, and researchers who are interested in the still growing research area of constraint programming.

Professor Peter Brucker
University of Osnabrück
Preface

Constraint Programming is a problem-solving paradigm which establishes a neat distinction between, on the one hand, a precise definition of the constraints that define the problem to be solved and, on the other hand, the algorithms and heuristics enabling the selection and cancellation of decisions to solve the problem. The main principles of Constraint Programming are:

- the separation between deductive "constraint propagation" methods, generating additional constraints from existing constraints, and search algorithms, used to systematically explore the solution space [149];

- the "locality principle", which states that each constraint must propagate as locally as possible, independently of the existence or the non-existence of other constraints [150];

- the distinction between the logical representation of constraints and the control of their use, in accordance with the equation stated by Kowalski for logic programming: Algorithm = Logic + Control [92].

These principles have been widely applied in the area of scheduling, enabling the implementation of flexible and extensible scheduling systems. Indeed, with Constraint Programming all the specific constraints of a given problem can be represented and actually used as a guide toward a solution.

As the number of applications grew, the need emerged to reconcile the flexibility offered by Constraint Programming with the efficiency of specialized Operations Research algorithms. The first step consisted in adapting well-known Operations Research algorithms to the Constraint Programming framework. As a second step, the success of the resulting
TOOLS opened a new area of research aimed at the design and implementation of efficient algorithms embeddable in Constraint Programming tools.

The aim of this book is to provide a non-exhaustive overview of the most widely used Constraint-Based Scheduling techniques. Following the principles of Constraint Programming, the book consists of three distinct parts:

- The first chapter introduces the basic principles of Constraint Programming and provides a model of the constraints that are the most often encountered in scheduling problems.

- Chapters 2, 3, 4, and 5 are focused on the propagation of resource constraints which usually are responsible for the "hardness" of the scheduling problem. In accordance with the locality principle, Chapters 2 and 3 focus on the propagation of one resource constraint considered independently of any other constraint. Chapter 2 deals with a resource of capacity one (which can execute one activity at a time). Chapter 3 considers the more general case in which a resource can execute several activities in parallel. Chapter 4 provides a theoretical comparison of most of the constraint propagation techniques presented in Chapters 2 and 3. Chapter 5 considers a more recent line of research, allowing to efficiently handle optimization objectives within resource constraints.

- Chapters 6, 7, and 8 are dedicated to the resolution of several academic scheduling problems. These examples illustrate the use and the practical efficiency of the constraint propagation methods of the previous chapters. They also show that besides constraint propagation, the exploration of the search space must be carefully designed, taking into account specific properties of the considered problem (e.g., dominance relations, symmetries, possible use of decomposition rules).

In conclusion, Chapter 9 mentions various extensions of the model and presents promising research directions.

PHILIPPE BAPTISTE, CLAUDE LE PAPE AND WIM NUIJTEN
Acknowledgments

The authors are indebted to the many people who have helped in previous work and in preparing this book. We are especially grateful to Peter Brucker who has kindly accepted to write down a foreword and to Jacques Carlier, Yves Caseau, and Jean-François Puget for their support and numerous discussions on constraint programming and on several scheduling problems studied in this book. Special thanks to Jérôme Rogerie, to all the members of the ILOG Scheduler team, and to the coauthors of papers that have served as raw material for this book: Emile Aarts, Philippe Couronné, Paul van Erp, Filippo Focacci, Vincent Gosselin, Jeet Gupta, Kees van Hee, Antoine Jouglet, Philippe Laborie, François Laburthe, Emmanuel Néron, Laurent Péridy, Eric Pinson, Benoît Rottembourg, Stephen Smith, Francis Sourd, and Didier Vergamini.
Chapter 1

INTRODUCTION

Constraint-Based Scheduling can be defined as the discipline that studies how to solve scheduling problems by using Constraint Programming (CP). In this introduction we first pay attention to the basics of CP, after which we introduce the scheduling problems that are considered in this book.

1.1. Introduction to Constraint Programming

Constraint Programming is a paradigm aimed at solving combinatorial optimization problems. Often these combinatorial optimization problems are solved by defining them as one or several instances of the Constraint Satisfaction Problem (CSP). Informally speaking, an instance of the CSP is described by a set of variables, a set of possible values for each variable, and a set of constraints between the variables. The set of possible values of a variable is called the variable's domain. A constraint between variables expresses which combinations of values for the variables are allowed. Constraints can be stated either implicitly (also called intentionally), e.g., an arithmetic formula, or explicitly (also called extensionally), where each constraint is expressed as a set of tuples of values that satisfy the constraint. An example of an implicitly stated constraint on the integer variables $x$ and $y$ is $x < y$. An example of an explicitly stated constraint on the integer variables $x$ and $y$ with domains $\{1, 2, 3\}$ and $\{1, 2, 3, 4\}$ is the tuple set $\{(1, 1), (2, 3), (3, 4)\}$. The question to be answered for an instance of the CSP is whether there exists an assignment of values to variables, such that all constraints are satisfied. Such an assignment is called a solution of the CSP.

One of the key ideas of CP is that constraints can be used "actively" to reduce the computational effort needed to solve combinatorial prob-
CONSTRAINT-BASED SCHEDULING

Constraints are thus not only used to test the validity of a solution, as in conventional programming languages, but also in an active mode to remove values from the domains, deduce new constraints, and detect inconsistencies. This process of actively using constraints to come to certain deductions is called constraint propagation. The specific deductions that result in the removal of values from the domains are called domain reductions. The set of values in the domain of a variable that are not invalidated by constraint propagation is called the current domain of that variable. As an example of the benefit of the active use of constraints, let us look at the combination of the three constraints $y > x$, $x > 8$, and $y \leq 9$ on the integer variables $x$ and $y$. Looking at the first two constraints, we can deduce that the value of $y$ is greater than 9. This is administered in the current domain of $y$. Then by using the constraint $y \leq 9$, obviously a contradiction can be detected. Without constraint propagation, the “$y \leq 9$” test could not be performed before the instantiation of $y$ and thus no contradiction would be detected. The current domains of the variables play a central role in constraint propagation as a basic means of communication between constraints. In this small example the fact the current domain of $y$ only includes values greater than 9 makes it trivial for the constraint $y \leq 9$ to detect a contradiction.

As the general CSP is NP-complete [77], constraint propagation is usually incomplete. This means that some but not all the consequences of the set of constraints are deduced. In particular, constraint propagation cannot detect all inconsistencies. Consequently, one needs to perform some kind of search to determine if the CSP instance at hand has a solution or not. Most commonly, search is performed by means of a tree search algorithm. The two main components of a tree search algorithm are (i) the way to go “forward”, i.e., the definition of which decisions are taken at which point in the search, and (ii) the way to go “backward”, i.e., the definition of the backtracking strategy which states how the algorithm shall behave when a contradiction is detected. The description of which decisions to take at which point in the search is often referred to as the search heuristic. In general, the decisions that are taken correspond to adding additional constraints. As such during search the constraint propagation reasons on the combination of the original constraints and the constraints coming from the decisions taken. When a contradiction is detected it thus means it is proven that there is no feasible assignment of values to variables given the original data of the CSP and the heuristic decisions that have been made. The most commonly used backtracking strategy is depth first chronological backtracking, i.e., the last decision is undone and an alternative constraint is imposed. More complex backtracking strategies can be found...
for example in [97, 32, 135]. In Section 6.3.4 we present an application where depth first chronological backtracking is compared to alternatives such as limited discrepancy search [83].

**Figure 1.1.** The behavior of a Constraint Programming system.

The overall behavior of a CP system is depicted in Figure 1.1. This figure underlines the fact that problem definition, constraint propagation, and the search heuristic and backtracking strategy are clearly separated. This separation is based upon the following fundamental principles of CP that became precise in the late 1970's and early 1980's. In [149] the separation between deductive methods generating additional constraints from existing constraints, and search algorithms used to systematically explore the solution space is advocated. For CP the deductive method is constraint propagation. The distinction between the logical representation of constraints and the control of their use is in accordance with the equation stated by Kowalski for logic programming: Algorithm = Logic + Control [92]. Another important principle used in CP is the so-called "locality principle", which states that each constraint must propagate as locally as possible, independently of the existence or the non-existence of other constraints [150].

When defining the problem in terms of variables and constraints, in practice the user of a CP tool is offered an array of pre-defined constraints (e.g., constraints on integers, constraints on sets, scheduling constraints) with corresponding propagation algorithms. The locality
principle is of crucial importance for the practical use of the pre-defined constraints as it implies that the constraint propagation algorithms can be reused in all applications where similar constraints apply. On top of this many CP tools offer ways to define new constraints with corresponding propagation algorithms that then seamlessly can be used in cooperation with the pre-defined constraints. CP tools also offer support to specify the search heuristic and the backtracking strategy. Again pre-defined heuristics are offered (e.g., instantiate all variables by choosing the variable with the smallest current domain and assigning it the minimal value in that domain, order all activities on all resources) as well as ways to define one’s own heuristics. The same goes for backtracking strategies: some pre-defined strategies are offered (e.g., depth first chronological backtracking, limited discrepancy search) as well as ways to define one’s own backtracking strategy.

All these properties together contributed to the success of commercial and public domain CP tools such as ILOG SOLVER [136, 137], CHIP [154, 1, 22], PROLOG III, IV [53], ECLIPSE [158], CLAIRE [49] and CHOCO [96]. For a comparison between several CP languages we refer to [71]. For an overview of CP, its principles, and its applications, we refer to [98, 85, 86, 93, 152, 69, 46, 117, 114, 25, 155].

To give an example of several of the aspects of solving a CSP mentioned above, we turn to one of the most popular examples in CP literature, namely the n-queens problem. The n-queens problem involves placing n queens on a chess board in such a way that none of them can capture any other using the conventional moves allowed to a queen. In other words, the problem is to select n squares on a \( n \times n \) chess board so that any pair of selected squares is never aligned vertically, horizontally, nor diagonally. The problem is of limited practical importance, but it does allow to discuss subjects like modeling, constraint propagation, search, and backtracking in more detail.

Let’s first look at the modeling of the problem. The n-queens problem can be modeled by introducing \( n \) integer variables \( x_i \), each representing the position of the queen in the \( i \)-th row, that is, the column number where the \( i \)-th queen is located. As such the domain for each variable is the set of integers 1 to \( n \). The constraints of the problem can be stated in the following way. For every pair \((i, j)\), where \( i \) is different from \( j \), \( x_i \neq x_j \) guarantees that the columns are distinct and \( x_i + i \neq x_j + j \) and \( x_i - i \neq x_j - j \) together guarantee that the diagonals are distinct.
Introduction

Figure 1.2. Solving the 6-queens problem.