An On-line Course on Constraint Programming

Christine Solnon
LIRIS CNRS FRE 2672, Nautibus, Université Lyon I
43 Bd du 11 novembre, F-69 622 Villeurbanne cedex
christine.solnon@liris.cnrs.fr

Abstract

This paper describes an on-line course on constraint programming. This course is dedicated to students of the "e-miage" formation, which is a french remote formation to "Information Systems for Companies Management". This course is available (in french) at
http://www710.univ-lyon1.fr/∼csolnon/Site-PPC/e-miage-ppc-som.htm

1 Introduction

The MIAGE (Méthodes Informatiques Appliquées à la Gestion des Entreprises) is a popular french formation on "Information Systems for Companies Management" which is delivered in twenty french universities. This formation lasts for three years and delivers a "Maîtrise" diploma, nearly corresponding to a Master's degree.

The e-miage [4] is an on-line version of the MIAGE formation: students of the e-miage can earn their degree via the Internet. This remote formation aims at reaching new trainees who cannot follow traditional courses, either because they cannot physically attend them (foreign or handicapped students), or because they are professionals that are continuing education during their release time. Another goal is to improve the actual learning conditions of basic education students by providing them with complete courses and training exercises available via Internet.

E-learning allows students to earn their degrees with maximum efficiency and flexibility, without commuting nor schedule conflicts: students study via Internet whenever and wherever they choose. However, remote formation also raises drawbacks, and students may feel isolated, or get stucked on difficult points. To overcome these drawbacks, student progresses are followed by tutors: a tutor is a professor of the university from which the student is registered; he answers

student's questions via e-mail and evaluates his knowledge at the end of each course.

The e-miage formation is composed of fifty course units, each course unit roughly corresponding to fourty learning hours and to three ECTS (European Credit Transfer System). One of these course units is entitled "Artificial Intelligence" and is composed of 36 working sessions of one hour: 2 introductory sessions about artificial intelligence, 8 sessions about logic, 5 about Prolog, 7 about constraint programming, 4 about ontologies, 2 about problem solving, 3 about machine learning, and 5 about expert systems. There is no predefinite order for studying these lessons. However, some sessions may be required for studying other sessions. In particular, logic must be studied before Prolog, and Prolog before constraint programming.

This paper describes the sessions dedicated to constraint programming. We first discuss instructional objectives, and the reasons that guided the choice of the programming language used to illustrate this course. We then describe the content of each session. We conclude on a first feedback on this course, and on some related sources of information.

2 Instructional objectives and choices

This course does not aim at training experts of constraint programming, but is an introduction to this field: the goal is to train students to use a constraint programming language to solve problems. By means of competences, one can summarize our objectives by the 4 following points:

- knowing what is a constraint satisfaction problem (CSP),
- being able to model a problem as a CSP,
- knowing how a constraint solver works, and
- being able to use a constraint programming language to solve a CSP.

These objectives must be achieved within 7 training sessions of one hour so that we have limited this course to the very basic concepts: we mainly deal with basic constraint satisfaction problems, and only briefly introduce their different extensions, such as Max-CSP or Valued-CSP; also, we have limited the study of constraint solvers to complete approaches, based on the "branch and propagate" schema, and to constraints on finite domains.

To illustrate constraint programming, we have chosen a logic programming language. Indeed, it would have been interesting to illustrate constraint program-

ming with different languages, based on different paradigms. However, considering the small number of sessions, we have limited the study to one language. We have more particularly chosen Gnu-Prolog [8], a language developped by Daniel Diaz. Indeed, Gnu-Prolog is used to illustrate the Prolog course within the same course unit; it integrates a constraint solver over finite domains and provides a large number of built-in predicates for defining and solving constraints; finally, it is free and easy to install on most computers and operating systems.

3 Content of the sessions

The course on constraint programming is divided into 7 learning sessions of one hour.

Session 1 introduces constraint satisfaction problems.

In a first part, we introduce terminology and definitions: we define what is a constraint, and give an overview of the different kinds of constraints; we formally define what is a constraint satisfaction problem (CSP); we introduce the notion of variable assignment (complete or partial, consistent or inconsistent) and define what is a solution of a CSP; we introduce the concept of over and under constrained problems, and briefly discuss the main extensions to the CSP framework.

In a second part, we illustrate how to model a problem as a CSP through two examples. The first example is the well known n-queens problem: this problem is very simple to describe and allows us to introduce the fact that there may exist different CSP modelings for a same problem. The second example is the stable marriage problem [5], which has more practical applications.

Session 2 is a training session, where the student has to model 5 problems within the CSP framework:

- The first problem involves computing the set of coins that must be returned back by a slot machine, given a price and a quantity of inserted coins. This problem is modeled with integer variables and linear integer constraints. We then ask to add an optimization criterion in order to minimize the number of returned coins.
- The second problem is the classical map coloring problem.
- The third problem is a logical puzzle that has been proposed by Lewis Caroll in [3]: 6 friends have to decide what condiment to take (i.e., either

salt, or mustard, or both salt and mustard, or nothing) while satisfying 5 given logical rules. We ask for two different modelings for this problem: one which associates one 4-valued variable with each friend, and another one which associates one boolean decision variable for each condiment/friend pair.

- The fourth problem is the well-known "SEND + MORE = MONEY" cryptarithmetic puzzle, for which we ask for the two classical modelings: one with a single constraint that expresses the global sum constraint, and another one with carry variables and 5 sum constraints.

- The fifth problem is the well-known "zebra" puzzle, which involves associating a nationality, a colour, an animal, a favorite drinking, and a favorite cigarette tobacco to five consecutive houses, given a set of clues.

For each of these problems, students may ask for some help by clicking on a link that gives indications to help him identifying the variables, their domains and constraints holding between them.

Session 3 introduces basic principles of constraint solvers. We first describe the "generate-and-test" algorithm, which exhaustively enumerates all complete assignments until a solution is found, and we introduce the concept of search space of a CSP. We then describe the "simple-backtrack" algorithm, and we show that the integration of constraint checks within enumeration actually reduces the number of generated combinations. We then introduce the basic principle of constraint propagation and filtering algorithms, and the associated local consistencies, and we show how to integrate these filtering technics within the simple-backtrack algorithm. Finally, we discuss ordering heuristics, and we show that they may speed-up the solution process.

Each algorithm is first informally described. It is then given in imperative pseudo-code, and its run-time behavior is illustrated on the 4-queens problem.

Session 4 is a training session, where the student has to implement in Prolog the different algorithms introduced in session 3. The goal is to let the students have a better understanding and a first practice of the basic principles of enumeration and propagation. Another goal is to let them go deeper into the practice of the Prolog language, which has been studied previously during the five Prolog sessions, and to show that Prolog is very well suited to implement these algorithms.

To simplify constraints representation and consistency checking, we limit ourselves to binary constraints. The different algorithms that are implemented during the session are used to solve the n-queens problem, the map coloring problem, and the stable marriage problem (Prolog predicates describing these three problems are given to the student). Finally, the efficiency of the different algorithms is experimentally compared on the n-queens problem.

As this training session is not directly supervised by a teacher, we have to guide students progression. Hence, for each algorithm, we progressively introduce the different predicates to implement, and for each predicate to implement, we give its template, a description by means of the relationship between its arguments, and some examples of calls and answers.

Session 5 is dedicated to learning and using a constraint programming language, i.e., Gnu-Prolog.

In a first part, we introduce the built-in Gnu-Prolog predicates for defining finite domain variables, and constraints over finite domain variables, and for solving these constraints. For the main built-in predicates, we give a full description, and we illustrate them on different examples. We widely refer to the Gnu-Prolog on-line users' manual [8] for more information on other predicates.

In a second part of this session, we illustrate the built-in constraint predicates of Gnu-Prolog through the two examples introduced in session 1, i.e., the n-queen problem and the stable marriage problem.

Sessions 6 and 7 are training sessions, where students have to use Gnu-Prolog for solving the different problems introduced during session 2. For each exercise, we first recall the CSP modeling that has been designed during session 2. Then, we give some indications and we guide the students' progression for solving the CSP with Gnu-Prolog. In particular, we describe the main predicates that should be written, and for each of them we give its template and some examples of calls and answers.

4 Conclusion

First results. A first group of thirteen students has used this course this year. Some feedback on it has been provided through the questions they asked to the tutor... and more over through the questions they did not asked (!): they actually asked very few questions, on minor points only, and globally felt satisfied. For this first experiment, solutions to exercises were available on-line, and the tutor has not checked the solutions found by the students. As a consequence, we had

nearly no feedback about the difficulties they may have uncountered. For the next year, we have decided to deliberately hide the solutions to the students, and to ask them to send their own solutions to their tutor before sending them an "official" solution. This should allow us to evaluate more precisely the difficulties they encountered.

The whole "artificial intelligence" course unit has been ratified by a test. The questions about the constraint programming part mainly dealed with modeling a problem into a CSP. Answers were rather satisfactory, and the average score for the part concerning the constraint programming course has been slightly greater than the average score for the whole artificial intelligence course.

Related work and references. Anybody looking for information on constraint programming via Internet very quickly finds the "Online guide on constraint programming" written by Roman Bartàk [2], which is a very complete tutorial, and which has been a valuable source of inspiration for this course.

However, our goal —and therefore the resulting course— is rather different: we do not aim at training experts and at being complete on the subject, but we aim at training students to model constraint satisfaction problems, and to use a constraint programming language to solve them, within a limited amount of time (7 hours). Hence, our course only focuses on basic aspects and is not as complete as Roman Bartàk's guide on some points. In particular, we do not develop heuristic algorithms (such as local search) and approaches for solving over-constrained problems. As a counterpart, our course includes many exercices, and (try to) guide students to build their own solutions to these exercices. Also, our course includes some other points that are not developped in Roman Bartàk's guide: it introduces a constraint programming language, and illustrates how to use it to solve constraint satisfaction problems.

To write this course, we also took inspiration from many other tutorials and books on constraint programming, e.g., [1, 6, 7, 9, 10]. Students that have been appealed by this introductory course are refered to these books to actually become experts!

Finally, note that if this course has been designed for e-miage students, it is available (in french) at
`http://www710.univ-lyon1.fr/~csolnon/Site-PPC/e-miage-ppc-som.htm`

References

[1] K.R. Apt: Principles of Constraint Programming, Cambridge University Press, August 2003, 407 pages. ISBN: 0521825830.

[2] Roman Bartàk: On-line guide to constraint programming http://kti.ms.mff.cuni.cz/∼bartak/constraints/

[3] Lewis Carrol: Symbolic Logic, 1896 http://durendal.org/lcsl/

[4] e-miage: http://www.u-picardie.fr/∼cochard/IEM/

[5] Ian P. Gent and Patrick Prosser: an empirical study of the stable marriage problem with ties and incomplete lists, in the proceedings of ECAI 2002, IOS Press, pp 141–145

[6] François Fages: Programmation Logique par Contraintes, Collection "Cours de l'Ecole Polytechnique", Ellipses, Paris, 1996.

[7] T. Frühwirth and S. Abdennadher: Essentials of Constraint Programming, Springer Verlag, March 2003.

[8] GNU-Prolog: http://gnu-prolog.inria.fr/

[9] K. Marriott and P.J. Stuckey: Programming with Constraints: An Introduction, The MIT Press, 1998

[10] E. Tsang: Foundations of Constraint Satisfaction, Academic Press, 1993