Chapter 9
Continuous Search in Constraint Programming

Alejandro Arbelaez, Youssef Hamadi, and Michèle Sebag

9.1 Introduction

In Constraint Programming, properly crafting a constraint model which captures all the constraints of a particular problem is often not enough to ensure acceptable runtime performance. Additional tricks, e.g., adding redundant and channeling constraints, or using some global constraint (depending on your constraint solver) which can efficiently do part of the job, are required to achieve efficiency. Such tricks are far from being obvious, unfortunately; they do not change the solution space, and users with a classical mathematical background might find it hard to see why adding redundancy helps.

For this reason, users are often left with the tedious task of tuning the search parameters of their constraint solver, and this again is both time consuming and not necessarily straightforward. Parameter tuning indeed appears to be conceptually simple (i/ try different parameter settings on representative problem instances, ii/ pick up the setting yielding the best average performance). Still, most users easily consider instances which are not representative of their problem, and get misled.

The goal of the presented work is to allow any user to eventually get his or her constraint solver to achieve top performance on his or her problems. The proposed

Alejandro Arbelaez
Microsoft-INRIA joint lab, Orsay, France
e-mail: alejandro.arbelaez@inria.fr

Youssef Hamadi
Microsoft Research, Cambridge, CB3 0FB, UK
LIX, École Polytechnique, F-91128 Palaiseau, France
e-mail: youssefh@microsoft.com

Michèle Sebag
Project-Team TAO, INRIA Saclay, Ile-de-France
LRI (UMR CNRS 8623), Orsay, France
Microsoft-INRIA joint lab, Orsay, France
e-mail: michele.sebag@inria.fr

Y. Hamadi et al. (eds.), Autonomous Search, DOI 10.1007/978-3-642-21434-9_9, © Springer-Verlag Berlin Heidelberg 2011
approach is based on the concept of Continuous Search (CS), on gradually building a heuristics model tailored to the user’s problems, and on mapping a problem instance onto some appropriate parameter setting. An important contribution to the state of the art (see [42] for a recent survey and Section 9.6 for more) is the relaxing of the requirement that a large set of representative problem instances be available beforehand to support off-line training. The heuristics model is initially empty (set to the initial default parameter setting of the constraint solver) and it is enriched along the course of lifelong learning approach, exploiting the problem instances submitted by the user to the constraint solver.

Formally, CS interleaves two functioning modes. In production or exploitation mode, the instance submitted by the user is processed by the constraint solver; the current heuristics model is used to parameterize the constraint solver based on the instance at hand. In learning or exploration mode, CS reuses the last submitted instance, running other heuristics than the one used in production mode in order to find which heuristics would have been most efficient for this instance. CS thus gains some expertise with regard to this particular instance, which is used to refine the general heuristics model through Machine Learning (Section 9.2.3). During the exploration mode, new information is thus generated and exploited in order to refine the heuristics model in a transparent manner, without requiring the user’s input and by only using the idle computer’s CPU cycles.

Our claim is that the CS methodology is realistic (most computational systems are always on, especially production ones) and compliant with real-world settings, where the solver is critically embedded within large and complex applications. The CS computational cost must be balanced with the huge computational cost of off-line training [19, 25, 24, 35, 46, 47]. Finally, lifelong learning appears a good way to construct an efficient and agnostic heuristics model, and able to adapt to new modeling styles or new classes of problem.

This chapter is organized as follows. Background material is presented in Section 9.2. Section 9.3 introduces the Continuous Search paradigm. Section 9.4 details the proposed algorithm. Section 9.5 reports on its experimental validation. Section 9.6 discusses related work, and the chapter concludes with some perspectives on further studies.

9.2 Background and Notations

This section briefly introduces definitions used in the rest of the chapter.

9.2.1 Constraint Satisfaction Problems

Definition 1 A Constraint Satisfaction Problem (CSP) is a triple \(\langle X, D, C \rangle\) where, \(X = \{X_1, X_2, \ldots, X_n\}\) represents a set of \(n\) variables.