Compact Modeling in Constraint Programming with Hybrid Tables

Christophe Lecoutre, Mouny Samy Modeliar, Nicolas Paris, Nicolas Szczepanski

CRIL, University of Artois & CNRS, France
{lecoutre,modeliar,paris,szczepanski}@cril.fr

Mots-clés : *Modeling, Table Constraints*

Hybrid tables (called 'smart' in [6]) are a useful modeling tool for Constraint Programming (CP). Such tables allow us to handle disjunctive cases (constraints) in a compact and structured way. An hybrid table constraint is defined from a table authorizing entries to contain simple arithmetic restrictions (which can be seen as intern constraints). In this paper, we show the practical interest of using hybrid tables on a very simple problem.

Illustration with the 1D Rubik's Cube. The 1D Rubik's Cube is a vector composed of 6 numbers, (1 2 3 4 5 6), which can be rotated in 3 different ways in groups of four :

```
(1 2 3 4) 5 6  --(1)->  (4 3 2 1) 5 6
1 (2 3 4 5) 6  --(2)->  1 (5 4 3 2) 6
1 2 (3 4 5 6)  --(3)->  1 2 (6 5 4 3)
```

Given a scrambled vector, the objective is to return the shortest sequence of rotations so as to restore the original ordered vector. Of course, this problem can be generalized with n values. Here, we have $n = 6$, and the possible rotations are 1, 2, 3, as well as 0 for indicating that no rotation is performed.

When building a CP model, one can introduce a two-dimensional array x of variables indicating what is the status of the vector at each time unit. At $x[0]$, we set the initial scrambled vector, and at $x[-1]$ we set the target ordered vector (-1, as in Python, for designating the last element of the array). We also need a one-dimensional array y of variables for indicating which rotation is applied at each time unit. The declaration of these variables in PyCSP^3 [5] is :

```
# x[t][i] is the value of the ith element of the vector at time t
x = VarArray(size=[nSteps + 1, n], dom=range(1, n + 1))

# y[t] is the rotation chosen at time t (0 for none)
y = VarArray(size=nSteps, dom=range(nRotations))
```

It is possible to avoid some useless sequences of operations : i) if at time t, no operation (0) is performed, then at time $t+1$, we can force 0 too ; ii) applying two times in sequence the same operation lets the vector unchanged, which is just a waste of time. Actually, with an hybrid table, we can combine these restrictions, which gives (when focusing on $y[0]$ and $y[1]$ only) in format XCSP^3 [1, 2] :

```
<extension type="hybrid-1">
  <list> y[0] y[1] </list>
  <supports> (0,0)(1,≠1)(2,≠2)(3,≠3) </supports>
</extension>
```

To ensure that we pass from a state to another one that is valid (i.e., can be reached), we can use an hybrid table involving some binary restrictions. For example, if $y[0]$ is set to 0, then we want $x[0][0] = x[1][0]$, $x[0][1] = x[1][1]$, ... By introducing (in XCSP[3]) an expression of the form 'ci' in the jth element of a tuple, we indicate that we want the variable in the column of index i being equal to the jth variable. We can then combine all possible transition cases with a single table. This table is given here in the context of the transition between time 0 and time 1 (note how rotations are managed by the choice of indexes after the symbol 'c') :

```
<extension type="hybrid-2">
  <list> y[0] x[0] x[1] </list>
  <supports>
    (0,*,*,*,*,*,*,c1,c2,c3,c4,c5,c6)
    (1,*,*,*,*,*,*,c4,c3,c2,c1,c5,c6)
    (2,*,*,*,*,*,*,c1,c5,c4,c3,c2,c6)
    (3,*,*,*,*,*,*,c1,c2,c6,c5,c4,c3)
  </supports>
</extension>
```

Hence, a PyCSP[3] model for the 1D Rubik's Cube can be mainly composed of hybrid table constraints : one group of hybrid tables with unary restrictions of the form '$\neq$ i' (hybridization level 1) and one group of hybrid tables with binary restrictions of the form '=ci', abbreviated as 'ci' (hybridization level 2). Solving with our constraint solver ACE [4] the most difficult instance (12, 2, 7, 3, 4, 11, 1, 10, 8, 9, 6, 5), mentioned by H. Kjellerstrand on his page www.hakank.org/common_cp_models, gives the following result. The hybrid model instance involves 57 constraints (29 hybrid table constraints, 24 unary constraints, and 4 side constraints) and can be solved in 20 seconds. The instance built by means of classical intensional constraints involves 2 239 constraints (most of them being reified constraints due to complex expressions) and cannot be solved within 1 hour.

It is important to note that modeling with hybrid tables can be applied to various contexts, and notably when one has to simulate a planning process. For example, for the classical board of the English peg solitaire [3], we have written an efficient model composed of only 31 hybrid table constraints (because a sequence of 31 operations has to be executed).

Useful links :
— PyCSP[3] website : pycsp.org
— ACE Github : https://github.com/xcsp3team/ace
— XCSP[3] website : xcsp.org

Références

[1] F. Boussemart, C. Lecoutre, G. Audemard, and C. Piette. XCSP3 : an integrated format for benchmarking combinatorial constrained problems. *CoRR*, abs/1611.03398, 2016.

[2] F. Boussemart, C. Lecoutre, G. Audemard, and C. Piette. XCSP3-core : A format for representing constraint satisfaction/optimization problems. *CoRR*, abs/2009.00514, 2020.

[3] C. Jefferson, A. Miguel, I. Miguel, and A. Tarim. Modelling and solving english peg solitaire. *Computers & Operations Research*, 33(10) :2935–2959, 2006.

[4] C. Lecoutre. ACE, a generic constraint solver. *To Appear*, 2022.

[5] C. Lecoutre and N. Szczepanski. PyCSP3 : modeling combinatorial constrained problems in Python. *CoRR*, abs/2009.00326, 2020.

[6] J.-B. Mairy, Y. Deville, and C. Lecoutre. The smart table constraint. In *Proceedings of CPAIOR'15*, pages 271–287, 2015.