Contraintes, objets et connaissances

Pierre Roy, François Pachet

Laforia-IBP, Université Paris 6, Boîte 169

4, place Jussieu,

75252 Paris Cedex, France.

Tel: (33) 1 44277004

Fax: (33) 1 44277000

E-mail: {roy|pachet|jfp}@laforia.ibp.fr

Résumé : La satisfaction de contraintes à domaines finis est un paradigme puissant permettant de résoudre de nombreux problèmes combinatoires complexes. Nous nous intéressons a l'intégration de ces mécanismes au sein de la programmation par objets. Cette integration est envisagée en prenant la PPO comme couche de représentation de base, les classes étant vues comme des définitions naturelles de domaines structurés, et les méthodes associées aux classes fournissant alors un langage pour exprimer des contraints complexes. Cet article décrit comment exploiter au mieux la structure des domaines objets pour définir des problèmes combinatoires complexes. Nous illustrons notre propos sur le système BackTalk développe au Laforia, et sur un problème d'harmonisation musicale.

1. Introduction

Constraint satisfaction programming is a powerful paradigm for solving complex combinatorial problems, which has gained attention recently. The notion of constraint was initially seen as an algorithmic problem, e.g. by [Mackworth 77] and [Laurière 78] who see constraint graphs as networks of relations for finite domains. Complex combinatorial problems have been studied extensively in operation research, graph theory and artificial intelligence for over two decades, leading to the elaboration of a rich theoretical framework. The main notion that came up from these works is arc-consistency [Macworth 77]. Most existing algorithms are based on the exploitation of arc-consistency : forward-checking [Haralick & Elliot 80], full look-ahead, and various extensions (e.g. backjumping, [Prosser 93]). These mechanisms have been later incorporated into logic programming languages ([Colmerauer 1990], CHIP [Van Hentenryck 89], CLP (R) [Jaffar & Lassez 87]). More recently these mechanisms have been integrated with object-oriented languages [Puget 94], [Caseau 94] or [Avesani et al. 90].

However, most difficult problems are still out of reach, even using state of the art CSP algorithms or languages. The main reason is well known in AI : general-purpose algorithms are, by definition, limited, because they do not have the knowledge specific to the problem instance. The idea of exploiting knowledge about problem instances has been explored already by J.-L. Laurière in the Alice system [Laurière 78]. Although Alice was able to adapt its reasoning to particular problem instances, it used only general-purpose heuristics and knowledge, and the user has no possibility of expressing domain specific knowledge to help the engine.

Following Laurière, we are convinced that knowledge is needed to improve the efficiency of enumeration algorithms. However, departing from his approach, we do not believe in general-purpose heuristics that are applicable to all domains. We claim that domain-specific knowledge is the key to improving the efficiency of CSPs, but that this knowledge must be carefully carved-up to fit the constraints imposed by the CSP technology. This paper describes a framework for expressing and solving combinatorial problems, in which domain specific knowledge can be expressed to increase the efficiency of the resolution.

2. The BackTalk framework

In order to study the relevance of domain specific knowledge, we designed a framework called BackTalk (standing for Backtracking in Smalltalk) [Roy & Pachet 97] that achieves simultaneously two goals : 1) provide state of the art enumeration algorithms that are usable off-the-shelf to solve combinatorial problems on arbitrary domains, and 2) provide « safe » entry points to express specific knowledge, that will be exploited by the system to speed up the resolution.

2.1 Main loop

The main loop of the system consists is a general generate-and-test procedure augmented with constraint propagation. At each step, the system picks up a variable, and instantiates it with a value of its domain. This assignment is then propagated to the other variables, in order to reduce their domains, as much as possible.

One of the key results of CSP is to show that the maximum amount of reduction is given by the property of *arc-consistency*, if one considers constraints individually [Macworth 77]. The full look ahead algorithm, for instance, consists in achieving arc-consistency of the whole CSP after each instantiation. The literature provides us with many algorithms to enforce arc-consistency at each step: AC-3 [Macworth 77], AC-5 [Deville & Van Hentenryck 91], etc. These algorithms basically loop over the set of constraints, calling a function, called `revise`, on each constraint, until it reaches a fixed point. This function achieves the arc-consistency of the constraint.

In practice, this general `revise` function can be drastically improved by taking the nature of the constraint into account, either to specialize the general `revise` function into an *equivalent* and more efficient function, or by implementing a weaker form of revision that reaches only an approximation of full arc-consistency, but with a simpler complexity (e.g. the all-different constraint, [Régin 94]). This procedure is usually called *filtering* procedure.

2.2 Heuristics

The framework allows to specify heuristics for the important steps of the main loop : choice of the next variable, choice of the next value, and choice of the next constraint to filter. These heuristics are represented as methods, called by the solver, so that each problem instance can specify its own set of heuristics.

2.3 Library of constraint classes

Recognizing the importance of exploiting the specificity of constraints, we designed our framework with this idea in mind : constraints are represented as classes, each of which redefining the filtering procedure, as a reasonable approximation of the arc-consistency procedure, thereby achieving a good compromise between efficiency and arc-consistency.

In order to further increase the efficiency of filtering methods, the filtering process is event-based : instead of calling a single filtering method for all cases of variable modification, the framework calls up to 3 specialized filtering methods, corresponding to 3 main cases of variable modification : instantiation, removal of one value, modification of the whole domain. Each class of constraint may accordingly redefine up to 3 different filtering methods, adapted to each of these cases. The user may also specify which events should be raised for each type of constraint.

3. Example : crosswords

We will illustrate how domain-specific knowledge can be expressed in BackTalk on a crossword problem. The problem consists in finding a crossword, given an initial grid with black and white squares and a list of words. The problem is a typical complex combinatorial problem : a reasonable list of words contains about 150,000 words, a standard grid contains about 30 words of size 2 to 12, leading to a search space of about 10^{100} combinations. CSP is therefore particularly well adapted to solve it. We consider a formulation of this problem in which variables are the words to find, and constraints are intersections between two words.

However, the CSP formalism is not enough to cope with the complexity of this problem. We will show here how specific knowledge can be expressed in the BackTalk framework to speed up the resolution. This knowledge is three-fold : topologic knowledge, lexical knowledge, and knowledge on letter distribution.

3.1 Heuristics

A good heuristic for the choice of the next variable to instantiate is to chose the variable with the smallest domain (so-called « first fail » principle). In the case of crosswords, we can use the intuitive knowledge on crosswords that it is better to proceed region by region, rather than exploring several areas at the same time. This corresponds to the « intensification principle » used in Tabu search for instance [Glover 89]. The min size heuristic is a short-sighted strategy that has to be somehow compensated. In this respect, intensification may be seen as a min-size heuristic augmented with a rudimentary anticipation capability.

This knowledge is faithfully represented by a special heuristic requiring that the next variable will be the variable with the smallest domain, within the set of variables connected to the current one.

3.2 Filtering method for intersection constraint

The crossword problem, in its basic form, contains only intersection constraints. Recall that by default, the filtering on binary constraints consists in computing approximately a Cartesian product of the two domains, and retaining only the consistent couples.

Knowledge on intersection can be used to improve the filtering of these constraints. Indeed, checking that two sets of words are « consistent » can be computed much more faster, by noticing that the set of possible intersections is small : there are only 26 letters in the alphabet. The refined procedure is therefore :

```
Compute possibleLetters (X,p) = the set of possible letters at position
p for variable X.
Remove from domain(Y) all words which do not contain one of
possibleLetters (X,p) at position p.
Compute possibleLetters (Y,p) = the set of possible letters at position
p for variable Y.
If possibleLetters (Y,p) ⊄ possibleLetters (X,p) then remove from
domain(X) all words which do not contain one of possibleLetters (Y,p) at
position p.
```

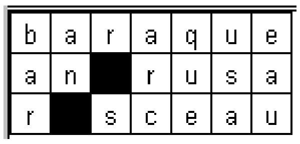
It is easy to show that this procedure achieves indeed arc-consistency for the intersection constraint. The complexity is linear, to be compared to the quadratic complexity of the default filtering method!

Another characteristic of this problem is that domains are so large, and constraints are so weak, that it is not worthwhile filtering systematically every constraint after each instantiation. Therefore, only events corresponding to an instantiations will be raised.

3.3 Exploiting knowledge on letter distribution to combine intersection constraints

A last type of knowledge is that letters are not distributed uniformly in words. A typical example of regularity is the fact that the letter « q » is almost always followed by letter « u ». There are numerous examples of this kind of rule : « no word starts by the same consonant twice », and so forth.

These rules basically give information on letters which belong to words not directly intersected with the current word. In the case of a variable `v` instantiated with a word containing a « q », the high probability for the crossing word to have a « u » following the « q » may be used to reduce the domain of the variable `v'` that is just parallel to variable `v` (see figure 1).



b	a	r	a	q	u	e
a	n	■	r	u	s	a
r	■	s	c	e	a	u

Figure 1. The letter « q » implicitly creates a relation between `v` and `v'`. This relation corresponds exactly to the intersection between `v'` and `w`.

cpu : 6192	fail : 598	choices : 603
cpu : 2004	fail : 181	choices : 186

This idea, expressed in terms of constraints and variables, amounts to consider dynamically created constraint between two parallel words only when certain conditions are satisfied (here, letter « q » appears). The filtering of this dynamically created constraint will reduce the domain of a variable not directly related with the current variable. Here, this filtering is naturally represented by raising the « domain » event for the constraint between `v'` and `w`.

3.4 Results

We conducted a series of experiments on various crosswords with and without these three kinds of knowledge, which show clearly the advantage of our approach. Note that the triggering rules are worth representing only if the filtering methods are efficient enough (more details are given in the full paper).

4. Conclusion

We have described a framework for expressing and solving combinatorial problems, that is designed to accommodate various kinds of domain-specific knowledge. We illustrate the framework and three types of knowledge on a crossword solver.

5. References

Avesani, P. Perini, A. Ricci, F. (1990) COOL: An Object System with Constraints. Proc. of TOOLS'2, Paris (France), June 1990.

Caseau, Y. (1994) Constraint Satisfaction with an Object-Oriented Knowledge Representation Language. *Journal of Applied Artificial Intelligence*, 4, pp. 157-184.

Colmerauer, A. (1990) An introduction to Prolog-III. *CACM*, 33 (7): 69.

Deville, Y. Van Hentenryck, P. (1991) An efficient arc-consistency algorithm for a class of CSP problems. Proc. of IJCAI '91, Chambéry (France), pp. 325-330.

Glover, F. (1989) Tabu search - Part I. *ORSA Journal on Computing*, 1 (3), pp. 190-206.

Haralick, R.M. Elliot, G.L. (1980) Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence*, vol. 14, pp. 263-313.

Jaffard, J. Lassez, J.-L. (1987). Constraint logic programming. Proc. of 14th POPL'87, Principles Of Programming Languages, Munich (Germany).

Laurière, J.L. (1978). A language and a program for stating and solving combinatorial problems. *Artificial Intelligence*, Vol. 10, pp. 29-127.

Mackworth, A. (1977). Consistency on networks of relations. *Artificial Intelligence*, (8) pp. 99-118, 1877.

Roy, P. Pachet, F. (1997) Reifying Constraint Satisfaction in Smalltalk. *Journal of Object-Oriented Programming*, to appear.

Prosser, P. (1993) Domain filtering can degrade intelligent backtracking search. Proc. of IJCAI'93, Chambéry (France), pp. 262-267.

Puget, J.-F. (1994) A C++ implantation of CLP. Ilog Solver collected papers. ILOG technical report.

Régin, J-Ch. (1994) A filtering algorithm for constraint of difference in CSPs. Proc. of 12th AAAI' 94, pp. 362-267, Seattle (Washington).

Van Hentenryck, P. (1989) Constraint satisfaction in Logic programming. Logic Programming Series, MIT Press, Cambridge, MA, USA.