PECOS : a high level constraint programming language

J-F Puget

ILOG SA

2 avenue Gallieni, BP 85

94253 Gentilly Cedex, FRANCE

email : puget@ilog.ilog.fr

Subject : Knowledge representation and Reasoning

Abstract For applying CSP consistency techniques in real world problems, domain specific constraints must be easy to describe. We propose a high level language for station constraints intensionally, as opposed to the classical extentional definition made of the tuples satisfying the constraint. An extension of these techniques to finite set variables is also presented. This in turn enables the implementation of a very powerful cardinality operator, which is shown to be more efficient by a factor linear in the size of the problems than best previously published approaches. This work is implemented in LE-LISP, in the PECOS system.

Key words CSP, Arc-consistency, Constraint Logic Programming.

Introduction

Constraint satisfaction problems (CSP) are a class of combinatorial problems that can be solved efficiently by combining consistency methods, such as arc-consistency [Ma 77], together with a backtracking search. The idea of consistency method is to assign a *domain* (i.e a set of possible values) to each variable, and to prune the search by preventing variable instantiation that are not consistent with the constraints of the problem. Different methods for this kind of pruning have been studied, e.g., [Mo 74], [Wa 75], [Ma 77], [Lau 78], [Mo 86], [Mo 88]. In these work, a CSP is usually defined as a graph, where the nodes are the variables of the problem, and the arcs are the constraints. Moreover, the algorithms proposed are often restricted to unary and binary constraint. Last, the binary constraints are represented by the set of pairs of values that satisfy them : they are represented *extensionally.*

We think that this approach is not practical for real world problems, because constraints are often described at a higher level as functional relationships, or by predicate to be satisfied, but almost never as a set of tuples. Even in general domains such as integer arithmetics, constraints are not easily defined extensionaly. For instance, is it easy to define a constraint such as $x \neq y$ by the pairs of values that satisfy the constraint? The answer is clearly not, since we first have to know the domains of the variables x and y. Let us suppose that these variables take their values in the set $\{1, 2, 3, \}$. The constraint can then be defined (for this special case only), by the following set of pairs :$\{< 1, 2 >, < 13 >, < 2, 1 >, < 2, 3 >, < 3, 1 >, < 3, 2 >\}$. More generally, if the domains of the variables is the range $(1, ..., N)$, $N.(N - 1)$ pairs are needed for the definition of the constraint, which is totally unthinkable if $N = 10000$ for instance. Thus, the definition of a constraints should be done at a higher level when possible. Following similar work by Van Hentenryck and Deville [VHDe 91], we propose a higher description language for defining constraints. In this language, the

constraint $x \neq y$ is defined as follows:
$\forall a, x \neq a \rightarrow y \neq a$
$\forall a, y \neq a \rightarrow x \neq a$

We call this kind of definition, an *intensional* definition of the constraint, as opposed to the extensional description by set of tuples.

However, this abstract language has to be compiled in order to be used efficiently for consistency checking. PECOS compiles these propagation rules into demons posted on variables. The semantics underlying this compiler, is described in [Pu 92].

1 Integers

In what precedes, the representation of variable domains variations can take several possibilities. We now show how this can be applied to other computation domains by providing other domain reduction representations.

It is shown [DA 87] that efficient consistency algorithm can be built for arithmetic constraints over integer valued variables by reasoning about the bounds of variable domains. We can incorporate this into our algorithm by defining two new domain reduction descriptions $<\leq m>$ and $<\geq m>$. The first one means that all the values less than m have been removed from the domain, the second one means all the values greater than m have been removed from the domain.

The set of relations for describing propagation rules $\{=, \in, \leq, \geq \neq\}$.

The constraint stating that $x <= y$ can thus be defined by the following, and will be compiled very efficiently :
$\forall a, x \geq a \rightarrow y \geq a$
$\forall a, y \leq a \rightarrow x \leq a$

The addition constraint $x = y + z$ is expressed as follows:

$\forall a, b,\ (y \geq a\ \wedge z \geq b)\ \rightarrow\ x \geq +(a, b)$
$\forall a, b,\ (y \leq a\ \wedge z \leq b)\ \rightarrow\ x \leq +(a, b)$
$\forall a, b,\ (x \geq a\ \wedge y \leq b)\ \rightarrow\ z \geq -(a, b)$
$\forall a, b,\ (x \geq a\ \wedge z \leq b)\ \rightarrow\ y \geq -(a, b)$
$\forall a, b,\ (x \leq a\ \wedge y \geq b)\ \rightarrow\ z \leq -(a, b)$
$\forall a, b,\ (x \leq a\ \wedge z \geq b)\ \rightarrow\ y \leq -(a, b)$

Now this constraint can be used for additions with more than two variables. For instance, PECOS treats the constraint $z = w + x + y$ as $z = w + u \wedge u = x + y$ The introduction of a new variable u is automatically done by PECOS, thus enabling the use of the primitive ternary addition constraint defined above.

2 Sets

Another computation domain has been very useful for real world applications, namely finite set, but consistency techniques have never been applied to it before. In this domain, a variable represents an unknown subset of $\mathcal{D}$. The domain of such a variable can in principle be described as a finite set of subsets of $\mathcal{D}$. This can lead to problems for practical applications since the size of such a domain may be extremely large. We use the partial ordering defined by inclusion for representing domains by the greatest lower bound (the intersection), and the least upper bound (the union) of its elements, in a way similar to the representation of integer domains by intervals. The bounds of a domain $dom(s)$ are noted $inf(dom(s)$, and $sup(dom(s))$.

For instance, a possible domain assignment for a set variable s is $dom =< \{1,2\}, \{0,1,2,3\} >$, with $inf(dom(s)) = \{1,2\}$, and $sup(dom(s)) = \{0,1,2,3\} >$. In this case, the possible values for s are $\{1,2\}, \{0,1,2\}, \{1,2,3\}, \{0,1,2,3\}$.

The set of primitive relations is extended with $\supset$ and $\subset$. The first one is used when the lower bound of the domain is modified, the second one when the upper bound is modified. The system can then derive specific filterings using properties of functions used in the definition of a constraint. For instance the inclusion constraint $\sigma_1 \subset \sigma_2$ is described by:

$\forall D, \sigma_1 \supset D \rightarrow \sigma_2 \supset D$
$\forall D, \sigma_2 \subset D \rightarrow \sigma_2 \subset D$

The combination of set and integer variables can be used to define a cardinality constraint between an integer valued variable i, and a set valued variable σ. We note this constraint $i = \sharp(\sigma)$ We will see that this constraint can be used in a very powerful and flexible way.

Set variables are useful for symmetrical problems. For instance, if the problem is to find location for N identical warehouse, it is better to use a single set variable with cardinality equals to N, than to use N variables. Indeed, in the latter case, the problem is symmetric: you can apply any permutation on the values of the variables since the warehouses are identical. In other words, for any solution, there exists $factorial(N)$ different instantiations of the variables corresponding to the same "real" solution.

3 Meta level extension

In what precedes, we have never made nor used the assumption that a variable cannot be used as a value. Thus we can use variables as possible values for other variables, which is a meta level extension, departing significantly form previously published CSP techniques.

A special case of interest is to be able to use a set variable to represent sets of other variables. Our system is flexible enough to implement this as a constraint: if $\{x_0, x_1, ..., x_n\}$ is a finite subset of $\mathcal{V}$, and d an element of $\mathcal{D}$, the $setofequal$ constraint state that a set valued variable σ is equal to the set of x_i instantiated to d. It is noted $setofequal(\sigma, d, \{x_i\})$.

The combination of this constraint with the cardinality constraint above can be used to express constraints on the number of variables that can take a given value. We define the $count$ constraint in the following way. If $\{x_0, x_1, ..., x_n\}$ is a finite subset of $\mathcal{V}$, and $i$vis an integer variable :
$count(i, d, \{x_0, x_1, ..., x_n\}) \equiv \exists\sigma,\ setofequal(\sigma, d, \{x_0, x_1, ..., x_n\}) \wedge i = \sharp(\sigma)$

This constraint has been proved to be very useful in resource allocation problems, such as warehouse location, network configuration with capacity constraints, and so on. We present below a problem which is very combinatorial, and which can be solved elegantly with such a constraint. This problem is stated in [VH 89], [VHDe 91] for instance. An integer N is given. The problem is to find a finite sequence $(x_0, x_1, ..., x_N)$, of integers such that 0 occurs exactly x_0 times in the sequence, 1 occurs

x_1, ..., N occurs exactly x_N times. For instance, for $N = 3$, there are two solutions : $(1, 2, 1, 0)$ and $(2, 0, 2, 0)$. The problem can be straightforwardly represented by N cardinality constraints :

$count(x_0, 0, (x_0, x_1, ..., x_N))$
$count(x_1, 1, (x_0, x_1, ..., x_N))$
...
$count(x_N, N, (x_0, x_1, ..., x_N))$

This problem can be solved faster if a redundant constraint is added [VHDe 91] :

$$\sum_{i=1}^{N} x_i = N + 1$$

4 Related work

In [VHDe 91], a general cardinality operator is described. The main difference between our work and this one is that we use set variables, which simplify a lot the semantics of such an operator. In fact, set variables are a very simple and elegant way of implementing cardinality operators. Moreover, as the next section shows this, our approach is more efficient than Van Hentenryck's one by a factor more than linear with the size of the problems!

Another very interesting piece of work is found [Cas 91], where a system using intensional constraints is described.. This system also automatically derives efficient local filterings from an abstract definition for constraints. One of the main differences between our work and Caseau's work is that his method is restricted to binary constraint.

5 Implementation in PECOS

During the past few years, a growing number of languages integrates such consistency methods in their kernel, especially in the logic programming community. One example is the programming language CHIP [DI 87], [VH 89]. The work presented here has been done while implementing such a language, called PECOS [ILOG 91] on top of the Le-Lisp language [CH 84], [CH 86]. One of the differences between CHIP and PECOS is the integration to a Lisp (thus functional) environment. The description language (and the consistency algorithm) is implemented in PECOS, although with a more lisp-like syntax than the one used in this paper. The descriptions of constraints are automatically compiled into functions : for stating a constraint, one has call the function generated from the constraint description with the variables to be constrained as arguments. A non deterministic programming kernel is also provided which enables PROLOG like control structures (logical and, logical or, and so on). Standard backtracking search using consistency techniques are then easy to implement.

PECOS has been used in various industrial projects, such as the planning of locomotive use at SNCF (the French railway company), network design at CNET (France Telecom's research organization), time table computing in various universities, cutting stock problems, and so on. A scheduling system able to deal with 50000 tasks, with over 250000 constraints with running times less than 10 minutes have been designed with PECOS. Another problem solved within few minutes contains

12000 integer variables with initial domains equals to the range $(0,..15000)$ and over 20000 linear constraints.

PECOS has also been used on theoretical problems. For instance we have been able to write a problem that solves a Ramsey problem. Namely, consider the complete graph with n nodes (each node is connected to every other nodes). The problem is to color the edges of this graph with 3 colors, such that for any 3 nodes $n1, n2, n3$, the three arcs $(n1, n2)(n2, n3)(n3, n1)$ don't have all three the same color (but two of them can have the same color). For $N = 16$, this problem has a lot of solutions. For $N = 17$ there are no solutions. It is not very difficult to write a program for finding solutions for $N \leq 16$. The challenge is to write a program which solves the problem for $N \leq 16$, and proves that there is no solution for $N = 17$. The difficulty lies in the very high number of symetries in this problem, which leads to a combinatorial explosion with classical CSP techniques. The use of set variables enables PECOS to solve this problem for $N \leq 16$, and to stop with no solutions for $N = 17$ because it suppresses the symetries. This had not been achieved before with any other constraint programming approach.

We give below benchmark showing the power of PECOS cardinality. The first table show the cpu time needed to solve the magic series problem for different size on a workstation SUN 3/60. The columns gives respectively, the size of the problem, the time needed for solving the problem with an old version of CHIP (no cardinality operator), with Van Hentenryck's cardinality operator (noted CARD), and with PECOS *count* constraint. The next two columns gives ratios. Times for CHIP and CARD are taken from [VHDe 91].

N	CHIP	CARD	PECOS	$\frac{CHIP}{PECOS}$	$\frac{CARD}{PECOS}$
4	0.20	0.28	0.13	1.5	2.2
5	0.46	0.50	0.22	2.1	2.3
6	1.06	0.88	0.34	3.1	2.8
7	2.30	1.48	0.48	4.8	3.1
8	4.90	2.36	0.66	7.4	3.6
9	10.40	3.62	0.90	11.5	4.0
10	21.92	5.46	1.18	18.5	4.6
11	46.16	7.94	1.54	30.0	5.2
12	101.84	11.30	1.90	53.6	5.9
13	202.84	15.78	2.36	85.9	6.7
14	427.72	21.54	2.86	149.5	7.5

Theses figures shows that PECOS improves the efficiency of CARD more than linearly with the size of the problem, which is due to the use of set variables.

6 Summary

The major contribution of this paper are the following. A high level description language is introduced for defining new domain specific constraints, together with a compiler for this language. Once compiled, constraints are used by an optimal consistency algorithm. An extension of consistency techniques to finite sets is also presented. This extension enables the implementation of a very powerful cardinality operator which is more efficient by a linear factor than the best previously proposed solution.

Set variables offer a pretty good encoding for symmetrical problems, but it requires that the symmetry of the problem is recognized by the user of the system. It would be intersting to automatically discover symetries. Another possibility for future work is the extension to other computation domains, for instance geometrical reasoning.

Acnowledgements I am indebted to Patrick Albert for his support and hints while this work was under progress as well as Hackim Erhili, Jerome Chailloux, Claude Lepape and all ILOG staff. I also thank Yves Caseau and Jean Jacques Chabrier for the interesting discussions we have had. I especially thanks Celine Rouveirol for her support and remarks on this paper.

Bibliography

[CAS 91] Caseau, “Using constraints in an object oriented deductive database” DOOD 91.

[CH 84] Jérôme Chailloux, Matthieu Devin et Jean-Marie Hullot ”Le-Lisp : a Portable an Efficient Lisp System,” 1984 ACM Symposium on Lisp and Functional Programming, Austin, Texas.

[CH 86] Jérôme Chailloux, Matthieu Devin, Francis Dupont, Jean-Marie Hullot, Bernard Serpette and Jean Vuillemin ”Le-Lisp version 15.2: the Reference manual, INRIA Publication, May 1986.

[Dav 87] Davis, E., Constraint Propagation with interval labels, *Art. Int. 24(3).*

[DeVH 91] Deville, Y., and Van Hentenryck, P, “An Efficient Arc Consistency Algorithm for a Class of CSP Problems”, *in proceedings of IJCAI 91, pp 325-330.*

[Din 88] Dincbas, Van Hentenryck, Simonis, Aggoun, Graf, Berthier “The constraint logic programming language CHIP” in *proceedings of the International Conference on Fifth Generation Computer Systems, ICOT, pp 693-702, 1988.*

[ILOG 91] *Manuel PECOS 1.0, mars 1991.*

[Gus 88] Gusgen, Hertzberg “Some fundamental properties of local propagation methods” Artificial intelligence, 1988 pp 237-247.

[Lau 78] Lauriere J.L., A language and a Program for Stating and Solving Combinatorial Problems, *Art. Int. 10(1).*

[Mack 77] Mackworth A.K., “Consistency in networks of relations”, *Art. Int 8, pp 99-118, 1977*

[MoHe 86] Moher, Henderson “Arc and path consistency revisited” *Art. Int. 28, pp 128-233, 1986.*

[MoMa 88] Mohr, Masini, “Good Old Discrete Relaxation”, *in proceedings of ECAI 88.*

[Pu 92a] Intensional and Cardinality Constraints *technical report*, ILOG, January 92.

[VH 89] Van hentenryck, P., *Constraints Satisfaction in Logic Programming*, MIT press, 1989.

[VHDe 91] Van hentenryck, P., Deville Y., “The cardinal operator ; A new logical connective for constraint logic programming” *ICLP 91, pp 745- 759.*