Improving Lotos Simulation Using Constraint Propagation

Malek Mouhoub and Samira Sadaoui
University of Regina
Dept of Computer Science
Wascana Parkway, Regina, SK, Canada, S4S 0A2
{mouhoubm,sadaouis}@cs.uregina.ca

Abstract

Lotos is the ISO formal specification language for describing and verifying concurrent and distributed systems. The simulation or execution of complex Lotos specifications is, however, not always efficient due to the space explosion problem of their corresponding transition systems. To overcome this difficulty in practice, we propose in this paper the integration of constraint propagation techniques into the Lotos simulation. Indeed, constraint propagation techniques are very powerful for solving hard discrete combinatorial problems. Experimental tests, we have conducted on the simulation of several specified combinatorial problems, demonstrate the efficiency of integrating constraint propagation into Lotos simulation.

1 Introduction

A Constraint Satisfaction Problem (CSP) involves a list of variables defined on finite domains of values and a list of relations restricting the values that the variables can simultaneously take [8, 11, 12, 13]. A solution to a CSP is a set of assigned values to variables that satisfy all the constraints. CSPs are very powerful for solving discrete combinatorial problems including frequency assignment, configuration and conceptual design, molecular biology, chemical hypothetical reasoning and scene analysis. Since a CSP is known to be an NP-Hard problem in general[1], a backtrack search algorithm of exponential time cost is needed to find a complete solution. In order to overcome this difficulty in practice, constraint propagation techniques have been proposed [14, 11, 8, 12, 13]. Our goal in this paper is to use constraint propagation techniques in order to improve the simulation phase running time of Lotos [4] specifications. This will enable us to simulate complex specified combinatorial problems in very efficient running time.

[1]There are special cases where CSPs are solved in polynomial time, for instance, the case where a CSP network is a tree [13].

Lotos is the ISO formal specification language [4] for describing and verifying concurrent and distributed systems. Lotos is composed of two complementary parts: *(i)* a process algebra that expresses temporal constraints of system actions or events, and *(ii)* algebraic data types that represent data structures and value expressions. Lotos provides the ability to describe complex data structures by composition and extension, and the equations can concisely specify very complex constraints. We use here the CADP toolbox [7] to compile, simulate and verify with model-checking the Lotos CSP specifications. The compilation generates a labeled transition system (LTS) which encodes all the possible execution sequences of a specification. To efficiently handle value expressions, variables, sorts and operations, CADP applies concrete implementation of data types rather than rewriting or symbolic evaluation. Indeed, CADP translates the data part of Lotos specifications into libraries of C types and functions.

In practice, for complex systems, the simulation are not always efficient due to the space explosion problem of the LTS [19]. Many methods [9, 19, 5, 10] have been proposed to cope with this difficulty. In this paper, we address this problem in a new way: first by expressing, in Lotos, CSP constraints and then by using efficient CSP techniques to improve the simulation time cost. The challenge is how to integrate CSP techniques into Lotos specifications in order to provide a more flexible and efficient way for describing and solving large complex systems such as combinatorial problems. In this paper, we define a generic Lotos-based CSP framework including:

- A generic template that can be customized to represent and solve a given CSP. This template reuses "as-is" a library of data types where the provided operations and equations are expressive enough to clearly and concisely specify complex constraints of CSPs. Besides, we use the simulators of CADP to find solutions for Lotos CSP specifications.

- The constraint propagation algorithms [8, 12, 13, 11] used to significantly improve the performance of problem solving. These algorithms are integrated into Lotos specifications through the external implementation of data types.

Our approach makes a clear separation between CSP specifications (customized template and its reusable library) and CSP algorithms (that we have implemented here in C) in order to freely apply any combination of CSP algorithms for any new problem and to include any new techniques in the future.

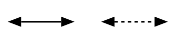
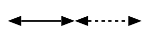
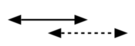
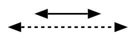
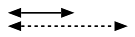
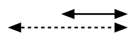
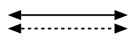
The paper is organized as follows. Section 2 introduces CSPs and Temporal CSPs. Through examples, Section 3 presents the integration of constraint propagation into Lotos simulation. Section 4 describes the experimentation we have conducted in order to evaluate the efficiency of our method. Concluding remarks and some perspectives are finally covered in Section 5.

2 Constraint Propagation for CSPs

The basic way to solve a CSP is the systematic search (called also search by enumeration) which explores the search space, such as standard Backtracking (BT). BT incrementally attempts to extend a partial solution towards a complete one, by repeatedly choosing a value for another variable [8, 12]. The late detection of inconsistency is the disadvantage of BT. Constraint propagation approach uses BT with local consistency algorithms. This allows the early detection of inconsistencies. Local consistency is used before and during BT phase to prune earlier later failure. In binary CSPs (CSPs where constraints are unary or binary relations), various local consistency techniques have been proposed [2, 11, 12] among which the most important and widely used is arc consistency. Arc consistency converts a CSP into an equivalent and simpler one by removing all inconsistencies involving all subsets of 2 variables.

AC-3 is one of the simplest arc consistency algorithms and is known to be practically efficient [12]. The time complexity of AC-3 is majored by $O(ed^3)$, where d denotes the size of the largest domain, and e the number of constraints. AC3.1 [20, 3] improves AC3 by using additional space to remember the resumption point of any value with respect to a constraint. The worst case time complexity of AC-3.1 can be achieved in $O(ed^2)$. AC3-FC (forward-checking), and AC3-LA (look-ahead) are two backtrack search algorithms using constraint propagation via arc consistency [8]. Forward-checking is the easiest way to prevent future conflicts. It performs a restricted form of arc consistency between the current variable (the variable that is being assigned a value) and the future ones (the variables that will be assigned a value). Look-ahead does more than forward-checking by further detecting the conflicts between future variables and therefore allows more branches of the search tree that will lead to failure to be pruned earlier than with forward checking.

A particular case of CSPs called Temporal CSPs or TCSPs[2][16, 15] is used to represent combinatorial problems involving numeric and symbolic temporal information. This include many application areas such as scheduling, planning, natural language processing, molecular biology and temporal databases. Using the model TemPro [16] a given problem under qualitative and quantitative temporal constraints is converted into a TCSP using a discrete representation of time and Allen interval Algebra[1]. More precisely, in TemPro temporal objects are called events. The domain of an event is the finite set of numeric intervals with constant duration. This domain is represented by the fourfold [*begintime, endtime, duration, step*] where *begintime* and *endtime* are respectively the earliest start time and the latest end time of the event, *duration* is how long the event lasts, and *step* is the distance (number of time units) between the starting time of two adjacent intervals within the domain. The qualitative relation between two events is represented by the disjunction of *Allen* primitives [1] (see Figure 1 for the definition of the thirteen *Allen* primitives).

Relation	Symbol	Inverse	Meaning
X before Y	b	bi	
X meets Y	m	mi	
X overlaps Y	o	oi	
X during Y	d	di	
X starts Y	s	si	
X finishes Y	f	fi	
X equals Y	eq	eq	

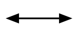
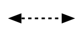
X:  Y: 

Figure 1. Allen Primitives.

The following example illustrates the transformation of a problem including numeric and symbolic temporal constraints into a TCSP using the model TemPro.

Example 1 : the scheduling problem

The production of five items A, B, C, D and E requires three mono processor machines M_1, M_2 and M_3. Each item can be produced using two different ways depending on the order in which the machines are used. The process time of each

[2]Note that this name and the corresponding acronym was used in [6]. The TCSP, as defined by Dechter et al, is a quantitative temporal network used to represent only numeric temporal information. Nodes represent time points while arcs are labeled by a set of disjoint intervals denoting a disjunction of bounded differences between each pair of time points.

machine is variable and depends on the task to be processed. The following lists the different ways to produce each of the five items (the process time for each machine is mentioned in brackets) :

item A:	$M_2(3), M_1(3), M_3(6)$ *or* $M_2(3), M_3(6), M_1(3)$
item B:	$M_2(2), M_1(5), M_2(2), M_3(7)$ *or* $M_2(2), M_3(7), M_2(2), M_1(5)$
item C:	$M_1(7), M_3(5), M_2(3)$ *or* $M_3(5), M_1(7), M_2(3)$
item D:	$M_2(4), M_3(6), M_1(7), M_2(4)$ *or* $M_2(4), M_3(6), M_2(4), M_1(7)$
item E:	$M_2(6), M_3(2)$ *or* $M_3(2), M_2(6)$

The goal here is to find a possible schedule of the different machines to produce the five items and respecting all the constraints of the problem. In the following, we will describe how is the above problem transformed into a TCSP using our model TemPro. Figure 2 illustrates the graph representation of the TCSP corresponding the constraints needed to produce items A and B. We assume that items A and B should be produced within 25 and 30 units of time respectively. A temporal event corresponds here to the contribution of a given machine to produce a certain item. For example, AM_1 corresponds to the use of machine M_1 to produce the item A, ... etc. In the particular case of item B, machine M_2 is used twice. Thus there are two corresponding events : BM_{21} and BM_{22}. 16 events are needed in total to produce the five items. Most of the qualitative information can easily be represented by the disjunction of Allen primitives. For example, the constraint (disjunction of two sequences) needed to produce item A is represented by the following three relations :

$$AM_2 \;\; B \vee M \;\; AM_1$$
$$AM_2 \;\; B \vee M \;\; AM_3$$
$$AM_1 \;\; B \vee M \vee Bi \vee Mi \;\; AM_3$$

However the translation to Allen relations of the disjunction of the two sequences required to produce item B needs a 3-ary relation involving BM_1, BM_{22} and BM_3. This relation states that BM_{22} should occur between BM_1 and BM_3. Since our temporal network handles only binary relations, the way we use to represent this kind of 3-ary relation is as follows : we create an additional event (Evt_1) and represent the constraints for producing item B as shown in figure 2. The duration X of Evt_1 is greater (or equal) than the sum of the durations of BM_1, BM_{22} and BM_3.

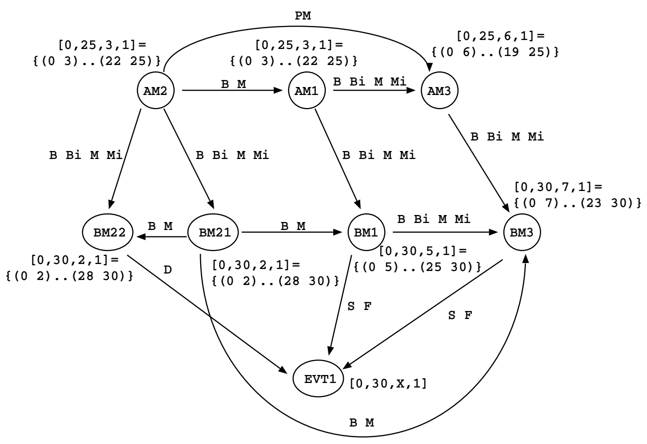


Figure 2. TCSP corresponding to a subset of the problem presented in example 1.

3 Integrating Constraint Propagation into Lotos Simulation

In [17] we have proposed a first framework to describe and solve CSPs using Lotos. The data part of Lotos is used to specify the different constraints of a CSP and the temporal constraints of a TCSP, their corresponding variables and domains. In addition, the process part of Lotos corresponds to the resolution process in CSPs, such as constraint propagation and backtracking algorithms. In Lotos, a CSP (or TCSP) is solved as fellows:

- Checking the Consistency. This consists of checking whether a solution to a CSP problem exists. A problem is consistent if its corresponding Lotos specification is deadlock free i.e., a progress is always possible.
- Finding Possible Solutions. In Lotos, the simulation of a specified CSP can generate one or all the possible solutions.
- Checking if a Path is a Solution. In Lotos, with the model checking we can verify whether a solution, specified as a sequence of actions, is valid. More precisely, this consists of checking whether the LTS of a solution is a sequence of the LTS of the specified CSP.
- Completing a Partial Solution. First, a partial sequence (or partial assignment of variables) is created, then the model checker completes it in an incremental way if it is possible. If the model checking leads to a deadlock, the partial sequence cannot belong to a solution.

However, the framework above is not general enough to specify any CSP. On the second hand, the problem specification and solving are really time consuming. For instance, it takes almost 80 seconds to find a solution for the 4-Queen problem. Consequently we have developed a reusable datatype library which is expressive enough to represent any CSP. This library consists of 17 data types (500 lines in total) describing, for instance all the Allen primitives and SOPO. In order to facilitate specifying CSPs, we extend the CSP library with a generic template that can be customized to solve any new problem. As shown in the particular template of Table 1, the variables, domains and (temporal) constraints of CSPs are mapped to sorts, operations and equations of Lotos by defining the:

- Number of Variables. The sort *FixVar* is used to model the variables of CSPs. Variables are enumerable. Each variable can be given a meaningful name along with an ID of natural number. We can also directly use natural numbers to represent the variables when they are many. This later requires to give the number of variables through the operation *varNum*.
- Domain Size of Each Variable. The domain is given through the operation *dSize* which defines the size of the variable domain and the range of value ID. The elements in a domain are in a total ordering. For instance, if a domain size is 4, the value ID of the domain will be 0, 1, 2, and 3. We have also considered the following fact: variables may have different types of domain. For instance, one variable may have a domain of string while another one has a domain of natural number. If variables have the same type of domain, they may have different set of values. For instance, one variable may have the domain of natural number from 5 to 10 while another one is ranged from 15 to 25. It is more convenient to use a value ID rather than a meaningful value.
- Relations and Satisfaction. Constraints between variables are given through the two operations *rel* and *sat* where *rel* defines which two variables have a constraint between them, and *sat* defines how constraints are satisfied given the value ID and variable ID. These considerations ease the definition of constraints because constraint type may be limited but relationships are various. We have defined a data type called Allen to describe any disjunction of Allen primitives. It should be noted that Lotos equations can define more complex temporal satisfaction than *Allen* primitives. For example, we can use equations to specify the relation "event_A should finish within two hours after event_B finishes". This cannot be handled by disjunction of the thirteen basic primitives.

Example 2: N-Queen

Given any integer N, the problem is to place N queens on N distinct squares in an N×N chessboard satisfying the constraints that no two queens should threaten each other, that is no two queens can be placed on the same row, same column, and same diagonal. The specification of N-Queen (for instance for N = 25 see Table 1) is highly reusable because we can change the number of queens by changing only the equations of *varNum* and *dSize*. The simulation of 25-Queen takes 80 seconds. In the next section, we propose to integrate several CSP algorithms into CSP specifications to increase the performance of simulation.

Table 1. 25-Queen Specification

type Template is CSPLibrary
sorts FixVar
...
eqns
varNum = tens(2, 5); (*There are 25 queens*)
dSize(x) = tens(2, 5); (*Each queen takes one of 25 positions*)
rel(x, x) = false; (*A queen cannot attack itself*)
rel(x, y) = true; (*Any two queens may attack each other*)
sat(x, m, y, n) = (abs(id(x), id(y)) ne abs(m, n)) and (m ne n);
(*operation abs, absolute value*)
...

4 Experimentation

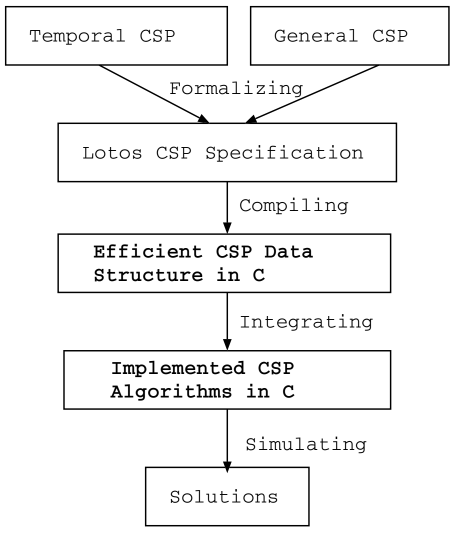


Figure 3. Lotos-based CSP Framework

The CADP tool gives us the opportunity to use external implementation of data types where both sorts and operations can use external operations. For our work, first we automatically translate Lotos CSP description (customized template and its library) into an efficient data structure in C. Then we incorporate to this description the implemented CSP algorithms through external implementation. This

Table 2. Integration of CSP Algorithms

```
type Algorithms is Template
opns
   BT (*!implementedby ALG_BT*):→ Nat (*backtrack*)
   AC3 (*!implementedby ALG_AC3*) : → Nat (*use AC-3*)
   AC31 (*!implementedby ALG_AC31*) : → Nat (*use AC-3.1*)
   FC (*!implementedby ALG_FC*) :→ Nat (*forward checking*)
   LA (*!implementedby ALG_LA*) :→ Nat (*lookahead*)
   ARC (*!implementedby ALG_ARC*) : → Nat (*check AC first*)
eqns ofsort Nat
   BT = 1; AC3 = 2; AC31 = 4; FC = 8;LA = tens(1, 6);
   ARC = tens(3, 2);
endtype
```

later is totally independent of any simulation and model-checking tools. In the following we present the evaluation of our framework for solving the scheduling problem presented in example 1. A part of the specification of this problem is given in Table 3. The simulation of the specification of this example is illustrated in the left chart of Figure 4. Arc consistency is very helpful to solve CSPs. From the benchmarks, we can see that AC3.1 is a little faster than AC3. AC3.1 requires more memory space than AC3, but this does not affect the running time in practice. The improvement of AC3.1 is not beneficial in this particular example. The experiments also show that some Lotos data types operations are really fast as all the SOPOs operations use only pure Lotos data types without external implementation.

Table 3. Specification of a Scheduling Problem

```
Type Template is ConstraintProblem, SOPO, Allen
AM1, AM2, AM3, ..., DM3, EVT17 :→FixVar (*17 events*)

dSopo(AM1) = sp(0, tens(2, 6), 3, 1);
dSopo(AM2) = sp(0, tens(2, 6), 3, 1);
dSopo(AM3)= sp(0, tens(2, 6), 6, 1);
dSopo(BM1)= sp(0, tens(2, 6), 5, 1); ...
(*There are 56 constraints, C_ij and C_ji are considered
   as one constraint*)
tRel(AM2, AM3) = a_b + a_m;
tRel(AM2, AM1) = a_b + a_m;
tRel(AM1, AM3) = a_b + a_m + a_bi + a_mi; ...
```

We have also tested our framework on the N-Queen problem as shown in the right chart of Figure 4 to examine its performance. The result of using only abstract Lotos specifications and standard simulation is not shown in the figure because its performance is too slow. In order to thoroughly evaluate the performance, statistically significant means and variances, of different algorithms, we also need to test our framework via the Random Uniform CSP Generators [18]. The testing result using *ARC + AC3* algorithm is shown in Table 4. *Time (N*N)* refers to the time needed to find the solution of CSPs having N variables, each defined on a domain of size *N*. The time unit in the table is 0.1 seconds and its precision is 0.1 seconds too, where *0* denotes that the time is less than 0.1 second and symbol "..." denotes that the program runs too long time. As we can see, the hardest problems are those with 50% incompatible pairs (that we call middle-constrained problems). Indeed, under-constrained and over-constrained problems (corresponding respectively to 20% and 80% incompatible pairs) are easy to solve. In the case of under-constrained problems, there are many solutions satisfying all the constraints and in the case of over-constrained problems the arc consistency at the preprocessing stage reduces considerably the size of the search space before the back-track search.

Table 4. Benchmarks on random CSPs

% of const	% of incomp pairs	Time 20*20	Time 30*30	Time 50*50	Time 100*100
100	20	0	0.2	0.7	12.1
100	**50**	0.6	1.8	32.5	...
100	80	0	0	0.2	8.3
75	20	0	0	0.6	7.8
75	**50**	0.5	12.3	144.3	...
75	80	0	0	0.3	11.4
50	20	0	0	0.8	45.12
50	**50**	2.2	7.8	27.3	125
50	80	0	0.3	0.3	30.9
25	20	0	0	0	2.1
25	**50**	0	0	17.12	28.1
25	80	0	0.2	0	0

5 Conclusion and Perspectives

The integration of constraint propagation algorithms into Lotos allows this latter to handle large combinatorial problems in a very efficient way, in practice. Indeed, we have significantly improved the simulation process through CSP constraint propagation techniques as demonstrated by the benchmarks we presented in this paper. In the future, we are interested in building a GUI user friendly tool that can assist in the specification of complex systems in an incremental manner. This requires handling the addition and retraction of constraints in a dynamic way using dynamic CSP techniques and the Lotos behavior part to describe dynamic actions.

References

[1] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

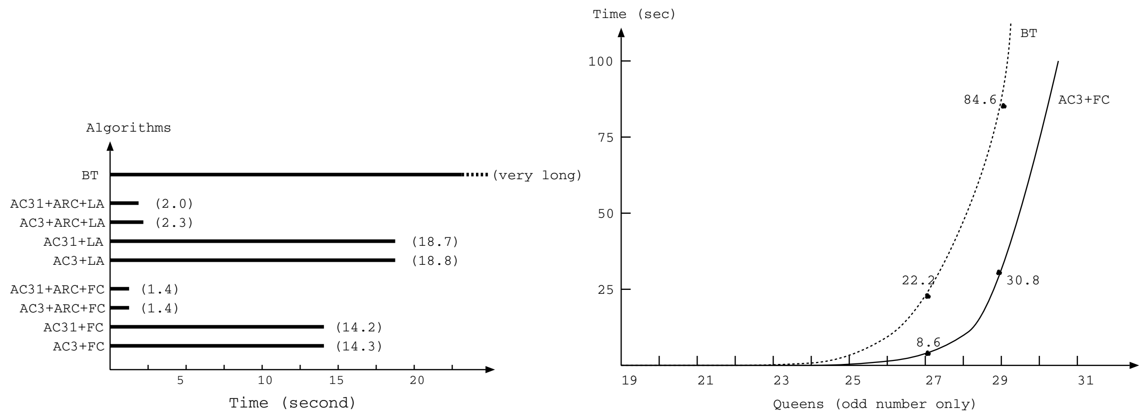


Figure 4. Benchmarks of the Scheduling and the N-Queen Problems

[2] R. Barták. On-line guide to constraint programming. http:// kti.ms.mff.cuni.cz/ b̃artak/ constraints/, 1998.

[3] C. Bessière and J. C. Régin. Refining the basic constraint propagation algorithm. In *Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 309–315, Seattle, WA, 2001.

[4] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language Lotos. *Computer Networks and ISDN Systems*, 14(1):25–59, 1987.

[5] M. Bozga, J. C. Fernandez, and L. Ghirvu. Using static analysis to improve automatic test generation. In *Tools and Algorithms for Construction and Analysis of Systems*, pages 235–250, 2000.

[6] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[7] J. C. Fernandez, H. Garavel, A. Kerbrat, and L. Mounier. CADP: a protocol validation and verification toolbox. *Lecture Notes in Computer Science*, 1102, 1996.

[8] R. Haralick and G. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[9] G. Holzmann. Algorithms for automated protocol validation. In *AT&T technical Journal*, volume 60, pages 32–44, January 1990.

[10] J. P. Krimm and L. Mounier. Compositional state space generation from Lotos programs. In E. Brinksma, editor, *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems (TACAS'97)*, Enschede, The Netherlands, April 1997. Springer Verlag. Extended version with proofs available as Research Report VERIMAG RR97-01.

[11] V. Kumar. Algorithms for Constraint Satisfaction Problems: A survey. *AI Magazine*, 1992.

[12] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[13] A. K. Mackworth and E. Freuder. The complexity of some polynomial network-consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25:65–74, 1985.

[14] U. Montanari. Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.

[15] M. Mouhoub. Reasoning with numeric and symbolic time information. In *Artificial Intelligence Review 21*, pages 25–56. 2004 Kluwer Academic Publishers, 2004.

[16] M. Mouhoub, F. Charpillet, and J. P. Haton. Experimental analysis of number and symbolic constraint satisfaction techniques for temporal reasoning. In B. Steffen, editor, *Constraints: An International Journal, 2:151-164*, pages 2:151–164, Kluwer Academic Publishers, 1998.

[17] M. Mouhoub, S. Sadaoui, and A. Sukpan. Formal description techniques for CSPs and TCSPs. In *Proceedings of International Conference on Software Engineering & Knowledge Engineering (SEKE04)*, pages 406–410, June 2004.

[18] Random Uniform CSP Generators. http:// www.lirmm.fr/ bessiere/ generator.html, 2004.

[19] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. In *7th. Symposium of Logics in Computer Science*, pages 394–406, Santa-Cruz, California, 1992. IEEE Computer Scienty Press.

[20] Y. Zhang and R. H. C. Yap. Making AC-3 an optimal algorithm. In *Proceedings of Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 316–321, Seatle, WA, 2001.