Proceedings of the 1999 Winter Simulation Conference
P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, eds.

AN ALGORITHM FOR GOAL-DRIVEN SIMULATION

Michel Page
Jérôme Gensel
Mahfoud Boudis

Université Pierre Mendès France
Projet Sherpa INRIA Rhône Alpes
655 avenue de l'Europe
F-38330 Montbonnot, FRANCE

ABSTRACT

This paper addresses the problem of goal-driven simulation. Goal-driven simulation is a task frequently performed by users of simulation systems. It consists in determining, when possible, an assignment of one or several decision variable(s) in order to obtain a particular value for a specific goal variable. This task is poorly supported in simulation systems because of lack of appropriate algorithms. Some systems assist goal-driven simulation with a functionality called *target value computation*. This functionality allows users to set a value for a goal variable and to get the value of a decision variable by running a simulation "backwards" from this goal. However, target value computation is insufficient in current simulation systems: it does not deal with models involving conditional expressions in equations – a common case in practice – nor with under and over-constrained problems, which frequently occur during goal-driven simulation. We present an algorithm which overcomes these difficulties. We propose to combine graph theoretic methods for monitoring the numerical solving process of the model and interval constraint reasoning for dealing with under-constrained and over-constrained problems. This algorithm, implemented in a simulation environment called AMIA, has been successfully applied to several large models containing thousands of equations.

1 INTRODUCTION

Simulation is the most widely used decision support technique in Economics and Management. One of the main reasons for the popularity of simulation systems comes from the assistance they give to decision makers for performing *what-if*? analysis, i.e., for determining the possible outcomes of decision hypotheses. It is now widely accepted that supporting decision more efficiently requires to assist *how-to*? analysis as well as *what-if*? analysis. *How-to*? analysis starts with a goal and determines one (or several) decision(s) which allows one to reach this goal. *How-to*? analysis is very important in today's management practice because many decisions are made in such a goal-driven way. In the context of simulation, *how-to*? analysis can be performed by *goal-driven simulation.* Goal-driven simulation consists in using a simulation model for finding, when possible, the value of a specific decision variable in order to reach a specific value for a particular output (or goal) variable. Goal-driven simulation is very important for users of simulation tools because it is complementary with traditional data-driven simulation. Goal-driven simulation is used in many fields like financial and marketing planning, economic forecasting and engineering design. However, in many simulation tools, this task is cumbersome: the user must adjust the value of the decision variable by trial and error until she gets a value of the goal variable which satisfies her. Goal-driven simulation can be assisted by a functionality called *target value computation*. Target-value computation automates the determination of a value for a decision variable by running the simulation model "backwards" from the expected goal. However, because of lack of appropriate algorithms, target value computation is seldom available in current simulation systems. When it is available, it is only with limited possibilities. In HEQS (Derman and Sheppard 1985) and some commercial spreadsheet packages for instance, target value computation is possible, but limited to hierarchical systems of equations (i.e. without simultaneous equations) and fails to solve seemingly simple problems involving conditions such as: given the equation: X = if Y = 1 then 2 else 0, how to set Y in order to get X = 2 ?

Target value computation is, in general, a difficult problem. Two difficulties must be overcome. First, as shown in the above example, target value computation should deal with conditions in equations (today, most modeling languages allow the definition of equations involving such conditions). Second, the problem can be

under-constrained (this would be the case for the problem: how to set Y in order to get $X = 0$?). The set of solutions must be in this case characterized and given to the user in a meaningful form.

Two approaches can be considered for tackling target value computation problems. The first one, initially proposed by Serrano and Gossard (Serrano and Gossard 1987) for solving systems of constraints in engineering design, extended in (Porté et al. 1988) and (Ait-Aoudia, Jegou and Michelucci 1993), exploits results of graph theory concerning *matchings*. It helps in characterizing constrained systems, i.e., systems of equations which can be numerically solved. However it does not take conditions into account. The second approach is based on interval reasoning. It stems from the observation that target value computation problems form a subset of interval constraint satisfaction problems. Interval constraint reasoning (Older and Vellino 1993), a set of powerful techniques for solving constraint satisfaction problems is hence a good candidate for solving target value computation problems. These techniques cope with under-constrained problems. However, up until now, they lack efficiency for tackling large models.

In this paper, we propose to solve target value computation problems by taking advantage of the two above approaches. First, we extend the graph-theoretic method proposed by Serrano and Gossard in order to deal with conditions. Second, we combine it with interval constraint reasoning only when necessary, i.e., when dealing with under-constrained problems. An algorithm based on these ideas has been implemented in AMIA (Page 1996), a discrete-time simulation workbench. This algorithm has successfully been applied to some large models. One of them, described in (Camos, Dumort and Valette 1986), contains several thousands of equations.

The paper is organized as follows. The target value computation problem is precisely stated in section 2. Our algorithm for target value computing is presented in section 3. Section 4 presents two examples illustrating our algorithm. Section 5 discusses related work. Section 6 summarizes the contribution of this paper and indicates its applications.

2 PROBLEM STATEMENT

From a general standpoint, this work is concerned with discrete-time simulation models (Cellier 1991). Models of this kind are generally represented by difference equations and, in many simulation systems, the equations are piece-wise defined. In this paper, we consider a more simple class of models: sets of algebraic piece-wise defined equations (PWDE); the generalization to difference equations is discussed in section 6. These models form a triple $<X,K,E>$ where X is a set of unknown real-valued variables; K is a set of real-valued variables whose value is known; E is a set of PWDEs. In each PWDE, the left-hand side is an unknown and the right-hand side is an expression which may contain conditions. These PWDEs have the following syntax:

piece-wise-defined-equation ::= *var* = *expr*
expr ::= if *cond-expr* then *expr* else *expr* | *arithm-expr*

where *var* is an unknown variable, arithmetic expressions (*arithm-expr*) are formed using numbers, variables, usual arithmetic operators and functions (+, -, ×, /, log, ...), while conditional expressions (*cond-expr*) are made up of relational and logical operators (=, ≠, >, ..., and, or, ...) combining arithmetic expressions. In the remainder of the paper, we will use the term *equation* to refer to a PWDE which does not contain any condition.

A *target value computation problem* associated with a model $<X,K,E>$ consists in:

assigning a value to a particular unknown variable G called *goal variable*,

removing the value of a particular known variable I called *instrument variable*,

determining *one* value of I as well as an assignment of the variables in X-$\{G\}$ such that all the PWDEs of E are satisfied.

A target value computation problem can be seen as a numerical constraint satisfaction problem (CSP) in which the set of variables is $(X - \{G\}) \cup \{I\}$, the domain of these variables is $\Re$ and the set of constraints is E.

Target value computation raises the problem of solving sets of nonlinear equations. Because interactivity is often a primary concern in simulation, the approach we have chosen to trade completeness for efficiency. It relies on classical numerical algorithms using floating-point arithmetic which may not converge, but which are more efficient than the complete methods developed with interval arithmetic (see section 5).

3 SOLVING TARGET VALUE COMPUTATION PROBLEMS

This section presents our algorithm for target value computation. The reader is invited to consult the example in section 4 to find illustrations of the concepts and properties introduced in this section.

3.1 Definitions and Properties

As stated above, our work is based on numerical algorithms. To be as generic as possible, we consider these algorithms as a black box called *equation solver* which, when given a set of equations, terminates but may not successfully find an instantiation of the unknown variables that satisfies the set of equations.

Using an equation solver as a black box implies to make assumptions about the properties of the sets of equations it is able to solve. In this paper, we make two important assumptions. First, a PWDE should not be sent to the equation solver as long as its condition contains a variable whose value is unknown. This assumption is made because continuity is required by almost all numerical algorithms that solve sets of equations whereas conditions usually break the continuity of the function in which they appear. The consequence of this first assumption is that conditional expressions must be handled outside the numerical machinery. The second assumption we make is also required by most numerical equation solving algorithms: the set of equations should be *constrained*.

Definition 1 (constrained set of equations) Let E be a set of n equations. E is a *constrained set of equations* if and only if :

it contains as many equations as unknowns;

in every subset of k equations ($0 \leq k \leq n$), at least k different unknowns appear;

The second condition of this definition stems from Simon's concept of *self-contained structure* (Simon 1953). It ensures that no part of the model is over-constrained. This condition is not easy to handle, from a computational point of view. For this reason, several researchers have proposed a more tractable formulation based on a representation of the set of equations in the form of a bipartite graph (Serrano and Gossard 1987).

Definition 2 (bipartite graph associated with a set of equations) The *bipartite graph associated with a set of equations* E in the set X of unknowns is the bipartite graph $G(E,X)$, such that $(e \in E, v \in X)$ is an edge in G if and only if variable v appears in equation e.

Constrained sets of equations can be characterized by bipartite graphs using the following theorem (Porté et al. 1988):

Theorem 1 (characterization of constrained sets of equations) Let E be a set of equations in the set X of unknowns. E is a constrained set of equations if and only if its associated bipartite graph $G(E,X)$ admits a perfect matching.

Under the assumption that the equation solver can deal with systems of PWDEs which are constrained and whose conditions contain no variable whose value is unknown, a naïve approach for solving a target value computation problem would be to consider each condition in a system of PWDEs as an hypothesis which can be either true or false. Thus, a system of PWDEs containing m conditions results in 2^m different systems of equations. One can then attempt to sequentially solve these systems (some of which may be non-constrained) until a solution consistent with the hypotheses is discovered. However, this solution is grossly inefficient because inconsistencies are discovered only when the whole system of equations has been built up.

A more efficient approach consists in trying to discover inconsistencies as soon as they appear in the system of PWDEs. Following this idea, one can notice that before any hypothesis is made, some *part* of a model may satisfy the two assumptions stated above, even if the model as a whole does not. Before explaining how to determine the constrained subsets of equations contained in a model, we need to introduce some concepts and properties.

Definition 3 (active PWDE) A PWDE e is said *active* if every variable appearing in its condition (if any) is known; otherwise, it is said *inactive*.

It should be noticed that the state (active or inactive) of a PWDE is not defined once for all: during the solving process, an inactive PWDE may later become active because the unknowns it contains in its condition have been computed. When a PWDE is active, the arithmetic expression in its right-hand side is determined. This is embodied in the following definition.

Definition 4 (active part of an active PWDE) The *active part* of an active PWDE e is defined by the result of the recursive function *active* below, applied on e:

$$active(var = arithm\text{-}expr) = arithm\text{-}expr$$

$$active(var = \text{if } cond\text{-}expr \text{ then } expr_1 \text{ else } expr_2) = \begin{cases} active(var = expr_1) & \text{if } cond\text{-}expr \text{ evaluates to } true \\ active(var = expr_2) & \text{otherwise} \end{cases}$$

Theorem 1 provides an interesting characterization of constrained sets of equations. In order to adapt it to the context of PWDEs, we introduce the following definition.

Definition 5 (**bipartite graph associated with a set of PWDEs**) The bipartite graph associated with a set of PWDEs P in the set X of unknowns is the bipartite graph $G(E,X)$ such that E is the subset of PWDEs in P which are active; $(e \in E, v \in X)$ is an edge in G if and only if variable v appears in the left-hand side or in the active part of e.

In order to characterize the maximum subset of equations in a set of PWDEs that can be sent to the equation solver, we have stated and demonstrated the following theorem (the demonstration is presented in (Boudis 1997)) using alternating paths, *i.e.* paths whose successive edges are alternatively inside and outside a matching.

Theorem 2 (characterization of the maximum constrained subset of equations in a set of PWDEs) Let P be a set of PWDEs containing a set of unknown variables X, $G(E,X)$ be the bipartite graph associated to P and W a maximum matching on G. The set of equations in E matched by W which cannot be reached by an alternating path *w.r.t.* W from a variable in X not matched by W is a maximum constrained subset of equations (MCSE).

3.2 Algorithm for Target Value Computation

The algorithm for target value computation is based on theorem 2 and on an interval constraint solver to deal with non-constrained systems. Its principle can be stated as follows. A basic step iteratively scans the set of PWDEs which are not yet solved in order to determine an MCSE and then to solve it using the equation solver. Each time an MCSE is successfully solved, the unknowns it contains become known. Some previously inactive PWDEs may thus become active *i.e.* become new candidates to be included in an MCSE at the next iterations. When no more active PWDE can be solved while some inactive PWDEs are still to be considered, the conditions of these inactive PWDEs are treated as choice points. A backtracking structure is established. For each inactive PWDE, two branches corresponding respectively to the assumptions that the condition is either true or false, are set. Both branches are explored in a depth-first search manner. When every inactive PWDE has been considered and the value of the instrument variable is still unknown, it means that the system is under-constrained. In this case, the interval constraint solver is invoked to find a value for the instrument value which is consistent with both the other values found so far and the set of equations. The algorithm which performs these operations is embodied in the target_value function described in the APPENDIX.

The target_value function takes three parameters; the first one, I, is the instrument variable. A variable is treated as a record with a field value containing its value (or ? when the value is unknown). The second parameter, PWDEs, is a set of PWDEs which have not already been solved. Each PWDE is treated as a record with four fields: var is the variable in the left-hand side, if is the expression in condition; then is the expression corresponding to the right-hand side of the PWDE when its condition is true; else is the expression when the condition is false. The third parameter, CONSTRAINTS, is the set of current constraints. This set is initially empty and is augmented each time an hypothesis is made about the condition of a PWDE.

The target_value function is roughly divided in two parts. The first one spans from line 9 to line 20. It is basically an iteration which determines a new MCSE, then solves it. Each time an MCSE is solved, the consistency of the solution is checked against CONSTRAINTS. This iteration stops either when an inconsistency is encountered (CONSISTENCY = false), or when an MCSE cannot be solved (SOLVED = false), or when no new MCSE can be found (MCSE = ∅).

The second part of the target_value function which spans from line 21 to the end deals with the termination of the above iteration. If an inconsistency occurs or if an MCSE cannot be solved, the value and the domain of variables which have been backed up (line 10) before MCSEs computations start are restored with their previous value (line 33) and the target_value function returns a value indicating its failure (RESULT = false), so that backtracking can take place. If the iteration terminates with an empty MCSE, two possibilities must be considered. The first one is that every inactive PWDE has been processed. In this case, either the instrument variable has been determined, the problem is then solved, or the instrument variable is still unknown and the interval constraint solver is invoked to determine it (line 26). The second possibility is that some inactive PWDEs have not been processed. In this case, the first inactive PWDE is selected (line 24) and a choice point is established from its condition (lines 27-31).

The auxiliary functions and procedures used by the target_value function are described below:

- function mcse(PWDEs: set of pwde) → set of pwde returns an MCSE in PWDEs. It incrementally maintains the bipartite graph associated with the set of unsolved active PWDEs. A maximum matching is computed using the algorithm described in (Hopcroft and Craft 1973). The equations which are reached by an alternating path w.r.t. to this matching from a non-matched variable are removed to obtain the MCSE, according to theorem 2.
- function equation_solver(PWDEs: set of pwde) → boolean invokes the equation solver on the constrained set of active PWDEs; it returns true if PWDEs can be solved, false otherwise. As a side effect, if PWDEs can be solved, the unknowns it contains are instanciated. The equation solver implements a set of classical numerical methods. An appropriate method is chosen according to the set of equations. For a set of linear equations, Gaussian elimination is used. For a set of nonlinear equations, Levenberg-Marquardt algorithm is used. Like any algorithm for solving sets of nonlinear equations using floating-point arithmetic, this algorithm is not guaranteed to converge towards a solution.
- Function check_consistency(CONSTRAINTS: set of constraint) → boolean invokes the interval constraint solver described in section 3.3. It returns true if CONSTRAINTS is consistent and false otherwise.
- function inactive_pwde(PWDEs: set of pwde) → pwde returns an inactive PWDE in PWDEs or nil if there is no such PWDE.
- function solve_constraints(CONSTRAINTS: set of constraint) → boolean returns true if a

value of the instrument variable satisfying CONSTRAINTS has been found, false otherwise. This function is detailed in section 3.3.

- procedure backup_variables and procedure restore_variables respectively backs up and restores the values and the domains of the unknown variables which have been reduced by constraint propagation.

3.3 The Interval Constraint Solver

The interval constraint solver we have chosen for the algorithm is MICRO (Gensel 1995). It handles linear and nonlinear constraint systems of equations, inequalities and disequalities, involving boolean and numerical variables (integers and float numbers) whose domain is represented by a union of intervals.

Operators involved in constraint expressions correspond to those found in the syntax of PWDEs. Complex constraints are built up from standard comparison operators and arithmetic or boolean (for conditional expressions) operators. Each operator is represented by a primitive constraint. Each primitive constraint is attached a set of rules of consistency which is fired whenever a change occurs in the arguments of the constraint. The range of constraint arguments is computed using interval arithmetic rules given in (Moore 1966). These principles are common to most existing interval constraint solvers, including CLP(BNR) (Older and Vellino 1993) and Inc++ (Hyvönen, De Pasquale and Lehtola 1993). Interval computations are performed with outward ranging (rounding the left endpoint down and the right endpoint up) so that bounds are always correct. Whenever it is possible, the evaluated range of a constraint expression is further narrowed using both the centered and the mean value forms (Alander 1985).

The constraint solver plays two roles in the algorithm. First, the function check_consistency invokes the constraint solver with the conditions corresponding to the hypotheses made so far and with the set of equations associated with these conditions. The solver transforms these data into an interval CSP. Then, a constraint propagation phase checks this set of constraints against consistency. If it is not consistent, there is no solution and the solving process backtracks to the previous choice point. Otherwise, the propagation ensures that if a solution exists, it is present in the domains of the constrained variables. During the propagation, the domains of the PWDEs variables are reduced by eliminating values which cannot appear in a solution. These reductions are effective all along a branch of the search tree until backtrack is performed.

Second, when called, the solve_constraints function attempts to find one solution for a system which is always made of n active equations in n+1 unknowns plus a set of hypotheses. These data are transformed in an interval CSP. Domain splitting is performed on the domain of the instrument variable. At each splitting step, new bounds are propagated in the interval CSP. If one of the two split sub-domains is inconsistent, it is left aside, otherwise the instrument variable is instantiated to the mid-value of this sub-domain and the equation solver is called. When the equation solver fails, dynamic splitting is further applied until either a solution is found in the reduced domains or no further splitting is possible.

4 EXAMPLES

Let us illustrate the target value computation algorithm using the model (M) below:

$$
\begin{array}{lll}
 & A = B \times E2 & (1)\\
 & B = \text{if } C > 4 \text{ then } A - 1 \text{ else } C + 2 & (2)\\
(M) & C = \text{if } E1 \geq 1 \text{ then } D - 1 \text{ else } 0 & (3)\\
 & D = \text{if } \log(E1) \leq 3 \text{ then } 2 \times C - 2 \text{ else } 0 & (4)\\
 & E1 \leftarrow 1 \quad E2 \leftarrow 2 &
\end{array}
$$

The set of unknowns is X = {A,B,C,D}, the set of known variables is K = {E1,E2} and the set of PWDEs is E = {(1),(2),(3),(4)}. Let us consider for the first example the following target value computation problem: how to set the value of E2 in order to get A = 10 ? The set of unknowns becomes X = {B,C,D,E2}; the set of known variables becomes K = {E1,A}. Figure 1 depicts the bipartite graph associated with the model (M) resulting from this problem. The PWDE (2) is not in the graph because it is not active.

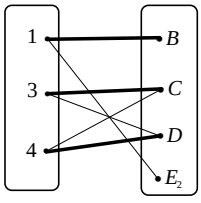

Figure 1: The Bipartite Graph Associated with the Model (M) Resulting from the Target Value Computation Problem "How to Set the Value of E2 in Order to Get A = 10 ?". Thick lines correspond to the maximum matching W (see below).

Following the algorithm presented above, an MCSE is first searched for. Among the different maximum matchings existing for this graph, let us suppose that W = {((1), B), ((3), C), ((4), D)} is obtained (it can be shown that when several different perfect matchings exist, they induce the same MCSE). E2 is not matched by W and {(E2,(1)), ((1),B)} is an alternating path starting at E2. Hence, equation (1) cannot be solved. No other equation is reached

by an alternating path starting at a variable not matched by W. Therefore, according to theorem 2, {(3),(4)} is an MCSE. The solving process then enters the loop in lines 13-20. {C = D - 1, D = 2 × C - 2} is sent to the equation solver, yielding C = 3 and D = 4. The solving process goes on with the bipartite graph depicted on Figure 2.

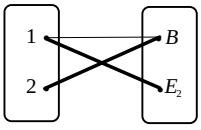

Figure 2: The Bipartite Matching Associated to the Model (M) at the Second Iteration.

PWDE (2) is now active, because C > 4 evaluates to false. Equations (3) and (4) which are solved have been removed. The maximum matching is {((1), E2), ((2), B)}. Since every variable is matched, {(1),(2)} is an MCSE. The solving process enters for the second time the loop in lines 13-20; {10 = B × E2, B = 3 + 2} is sent to the equation solver, yielding B = 5 and E2 = 2. All the PWDEs have been processed, so PWDEs is now empty, as well as MCSE; then, the solving process goes out of loop 10-20. No more inactive PWDE remains to be processed, the solving process is thus over with E2 = 2 as a solution.

The next example illustrates a case where constraint processing is necessary both to analyze the consistency of a set of constraints and to solve it because the problem is under-constrained. The problem considered here is: how to set E1 in order to get A = 10 ? The set of unknowns is thus X = {B,C,D,E1} and K = {E2,A}.

The solving process of the algorithm is presented in figure 3. The bipartite graph is not depicted at each step, because of space limitation.

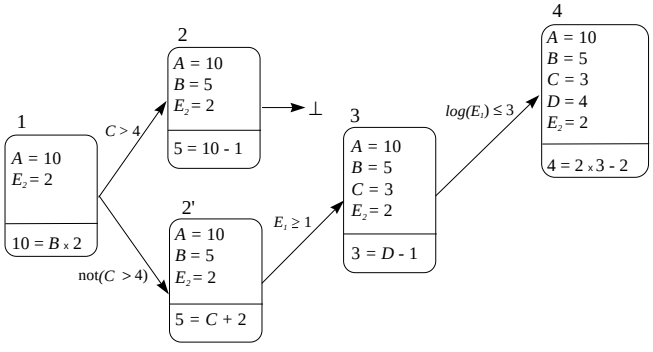

Figure 3: Solving Process for the Problem "How to Set E1 in Order to Get A = 10 ?"

In the initial state 1 (this number corresponds to the level of recursion in function target_value) in figure 3, the bipartite graph contains only PWDE (1) and variable B. The maximum matching is {((1),B)} and the MCSE is {(1)}. {10 = B × 2} is sent to the equation solver, yielding B = 5. No PWDE becomes active, so the resulting MCSE is empty. The solving process thus goes out of the loop 14-21. An inactive PWDE is searched for. Let us suppose that PWDE (2) is first considered. Its condition, C > 4, is asserted to be true. It is thus added to the set of constraints as well as the equation 5 = 10 - 1, which corresponds to this condition. We are now in state 2 in figure 3. The set of constraints {C > 4, 5 = 10 - 1} is sent to the interval constraint solver which detects an inconsistency. Hence, the algorithm backtracks to state 2' which corresponds to the condition not(C > 4). The bipartite graph now contains two vertices: the PWDE (2), which evaluates to {5 = C + 2}, and the variable C. The maximum matching is {((2), C)} and the MCSE is {(2)}. {5 = C + 2} is sent to the equation solver, yielding C = 3. The set of constraints {not(C > 4), 5 = C + 2} is consistent. The MCSE is now empty; a new hypothesis must be made. PWDE (3) is chosen to become active, leading to state 3. D is found to be equal to 4, and the same reasoning leads to state 4. At this point, the constraint 4 = 2 × 3 - 2 is consistent and there is no more inactive PWDE to be processed. Hence, the function solve_constraints is called on the set of constraints {E1 ≥ 1, log(E1) ≤ 3}. It determines that interval [1, 20.08...] is consistent for E1 and returns the mid-value in this interval. The algorithm then stops.

5 DISCUSSION

In this section, we compare our approach with other works. The algorithm for goal-driven simulation presented in section 3 is based on the characterization of the maximum constrained subset of equations in a set of PWDEs. The idea of representing a system of equations by a bipartite graph is due to Serrano and Gossard (Serrano and Gossard 1987). They have shown that a system of equations can be decomposed in sub-systems by determining a matching in the associated bipartite graph. These results have been formalized and extended in (Porté et al. 1988) and (Ait-Aoudia, Jegou and Michelucci 1993). However, these works do not deal with conditional expressions in equations (PWDEs), which are very frequent in simulation systems. In presence of conditional expressions, it is important to determine, not only if a system of equations can be solved, but also whether it contains sub-systems which can be solved. The resolution of these sub-systems may render active other PWDEs and the resolution of the whole system can thus proceed further on. For this reason, theorem 2 of the paper is an important contribution.

Target value computation problems form a subset of constraint satisfaction problems. Constraint satisfaction techniques are hence candidates for solving them. For instance, constraint logic programming languages like CLP(ℜ) (Jaffar et al. 1992) implement constraint solvers which can be applied to target value computation. Furthermore, these techniques can also handle conditional expressions in equations. However, these systems do not

deal with nonlinear systems (or non-polynomial ones in systems like RISC-CLP(Real) (Hong 1993)). Furthermore, when facing to under-constrained problems, they provide a solution in the form of a symbolic expression, which is not very useful for the user in the context of goal-driven simulation.

Interval constraint satisfaction techniques are also relevant for solving target value computation problems. These techniques are based on branch and prune algorithms. For instance, Newton constraint programming language (Van Hentenryck, Michel and Benhamou 1998) uses interval Newton method, the equivalent for intervals of the well-known method for solving sets of nonlinear equations. This interval method is very interesting in the context of goal-driven simulation. First, its failure proves the absence of a solution. This is interesting in particular for over-constrained problems. Second, it finds all the solutions of a set of linear and/or nonlinear equations. Third, the solution of under-constrained problems is given in the form of an interval solution reaching the goal, which is very useful for the user. However, these methods also have an important drawback: they suffer from inefficiency when dealing with large problems. Benchmarks in (Chiu and Lee 1994) show that the interval Gauss-Seidel method is several orders of magnitude less efficient than its floating-point equivalent. At a lesser degree, the same problem arises with the interval Newton method whose convergence is shown to be slow on large systems of equations. For this reason, we also use interval constraint solving in our work, but only when necessary, i.e., for dealing with under-constrained problems, but not for equation solving.

6 CONCLUSIONS

In this paper, we have presented an algorithm for performing goal-driven simulation. This algorithm monitors the numerical equation solving process using graph theoretic methods in order to determine when a set of piece-wise defined equations can be sent to the equation solver. It deals with non-constrained problems using an interval constraint solver.

The algorithm has been extended to difference equations and integrated in AMIA (Page 1996), an environment for discrete-time simulation based on an algebraic modeling language. Using this system, the target value computation algorithm has been successfully applied to several models. One of these models (called MEDEE (Camos, Dumort and Valette 1986)), developed for energy demand forecasting in European countries, involves several thousands of equations. Despite the size of the model, most goal-driven simulations are performed in less than a minute on a PC, providing energy policy-makers with a powerful decision support tool.

APPENDIX

```
1  function target_value(I: variable; PWDEs: set of pwde;
                CONSTRAINTS: set of constraint) → boolean
2  variables
3    SOLVED: boolean
   // indicates whether the equation solver succeeded or not
4    MCSE: set of pwde
   // maximum constrained set of equations
5    RESULT: boolean
   // result of the target_value function
6    CONSISTENCY: boolean
7    P: pwde
8  begin
9     if not check_consistency(CONSTRAINTS) then
                    return(false);
10    backup_variables;
11    MCSE ←mcse (PWDEs);
12    SOLVED ← true; CONSISTENCY ← true;
13    while MCSE ≠ ∅ and SOLVED and
CONSISTENCY do
14      SOLVED ← equation_solver(MCSE);
15      if SOLVED then
16        CONSISTENCY ←
                    check_consistency(CONSTRAINTS);
17        PWDEs ← PWDEs - MCSE;
18        MCSE ← mcse(PWDEs)
19      end if
20    end while;
21    if not CONSISTENCY or not SOLVED then
22      RESULT ← false
23    else  // MCSE = ∅
24      P ← inactive_pwde(PWDEs);
25      if P = nil then
     // no more inactive PWDE
26        if I.value ≠ ? then RESULT ← true
                    else RESULT ←
          solve_constraints(PWDEs ∪ CONSTRAINTS)
27        else
28        RESULT ←
            target_value(I, {P.var = P.then} ∪
                    (PWDEs - {P}), {P.if} ∪
                {P.var = P.then} ∪ CONSTRAINTS)
29                    or
30          target_value(I, {P.var = P.else} ∪
                (PWDEs - {P}),
      {not(P.if)} ∪ {P.var = P.else} ∪ CONSTRAINTS)
31      end if
32    end if;
33    if not(RESULT) then restore_variables;
34  return(RESULT)
35  end;
```

REFERENCES

Ait-Aoudia S., Jegou R. and Michelucci D. Reduction of constraints systems. In: *Proc. Compugraphic*, pp. 83-92, Alvor, Portugal, 1993.

Alander J. On Interval Arithmetic Range Approximation Methods of Polynomials and Rational Functions. *Computer and Graphics*, 9(4):365-372, 1985.

Boudis M. *Simulation et Systèmes à Base de Connaissances*. Ph.D. Diss., Univ. Mendès France, Grenoble, France, 1997 (in French).

Camos M., Dumort A. and Valette P. MEDEE 3: modèle de demande en énergie pour l'Europe. *Technique & Documentation-Lavoisier*, Paris, France, 1986 (in French).

Cellier F. *Continuous System Modeling*. Springer Verlag, New York, 1991.

Chiu C.K. and Lee J.H.M. Interval Linear Constraint Solving Using the Preconditioned Interval Gauss-Seidel Method. In: *Proc. ILPS'94*, R. Yap ed. , pp. 17-31, Ithaca, New York, 1994.

Derman E. and Sheppard E. HEQS – a hierarchical equation solver. *AT&T Technical Journal* 64 (9), pp. 2060-2095, 1985.

Gensel J.. *Contraintes et représentation de connaisances par objets. Application au modèle Tropes*. Ph.D. Diss., Univ. Joseph Fourier, Grenoble, France, 1995 (in French).

Hong H.. RISC-CLP(Real): Logic Programming with Non-linear Constraints over the Reals. In: *Constraint Logic Programming: Selected Research*, F. Benhamou and A. Colmerauer eds., pp. 175-196, MIT Press, 1993.

Hopcroft J. and Karp R. An o(n5/2) Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM Journ. of Computing*, 2(4):225-231, 1973.

Hyvönen E., De Pasquale S. and Lehtola A. Interval Constraint programming in C++. In *Proc. of the NATO ASI Conf. on Constraint Programming*, B. Mayoh ed., Parnu, Estonia, 1993.

Jaffar J., Michaylov S., Stuckey P. and Yap R. The CLP($\Re$) Language and System. *ACM Trans. on Prog. Languages and Systems*, 14(3):339-395, 1992.

Moore R. (ed.) *Interval Analysis*. Prentice Hall, New Jersey, 1966.

Older W. and Vellino A. Constraint Arithmetic on Real Intervals. In: *Constraint Logic Programming: Selected Research*, F. Benhamou and A. Colmerauer eds., pp. 175-196, MIT Press, 1993.

Page M. AMIA 3.0: manuel utilisateur. *Tech. Rep. n°176, Univ. Mendès France*, Grenoble, France, 1996 (in French).

Porté N., Boucheron S., Sallantin J., and Arlabosse F.: An Algorithmic View at Causal Ordering. *Tech. Rep. n°45, Centre de Recherche en Informatique*, Montpellier, France, 1988.

Serrano D. and Gossard D. Constraint Management in Conceptual Design. In: *Proc. Knowledge Based Expert Systems in Engineering: Planning and Design*. D. Sriram and R. Adey eds., pp. 211-224, Comput. Mech. Publications, Southampton, UK, 1987.

Simon H. Models of Man. John Wiley & Sons, New York, 1953.

Van Hentenryck P., Michel L., and Benhamou F. Newton: Constraint Programming over Nonlinear Constraints. To appear in: *Science of Computer Programming*, Elsevier, 1998. Available at: http://www.sciences.univnantes.fr/info/perso/permanents/benhamou/papers.html.

AUTHOR BIOGRAPHIES

MICHEL PAGE is an Assistant Professor at the University Pierre Mendès France of Grenoble. He holds a Ph.D. in Computer Science from the University Pierre Mendès France of Grenoble. He works in the Sherpa Project at the INRIA (National French Institute of Research in Computer Science and Automation) Rhône-Alpes. His research interests include numerical simulation, semi-qualitative reasoning, object-based knowledge representation.

JEROME GENSEL is an Assistant Professor at the University Pierre Mendès France of Grenoble. He holds a Ph.D. in Computer Science from the University Joseph Fourier of Grenoble. He works in the Sherpa Project at the INRIA (National French Institute of Research in Computer Science and Automation) Rhône-Alpes. His research interests include constraint satisfaction problems, numerical simulation, object-based knowledge representation.

MAHFOUD BOUDIS holds a Ph.D. in Computer Science from the University Joseph Fourier of Grenoble. His research interests include simulation, graph theory and constraint satisfaction problems.