Solving Conditional Models by Using a Boundary Crossing Algorithm

V. Rico-Ramírez and A. W. Westerberg[1]

Institute for Complex Engineered Systems
Carnegie Mellon University, Pittsburgh, PA 15213

Abstract: In this paper we investigate the solving of conditional models using the boundary crossing algorithm proposed by Zaher (1995). He proposed this algorithm as an alternative to using MINLP techniques for solving. This algorithm involves the execution of several well differentiated activities including logical analysis and continuous reconfiguration of the equations constituting the problem, calculation of Newton-like steps, and the calculation of subgradient steps. In this paper, we describe the practical implementation of the boundary crossing algorithm as a conditional modeling solution tool. In such an implementation, we have integrated the entities created and/or used to perform each of the activities in an object oriented solving engine: the conditional modeling solver CMSlv. Also, we describe and solve several examples of conditional models in chemical engineering. Finally we discuss the scope and limitations of the algorithm.

Keywords: Conditional model, boundary crossing.

1. Author to whom all correspondence should be addressed

Table of Contents

1 Introduction....1

2 The Boundary Crossing Algorithm....3

2.1 The Formulation....3

2.2 The Solution Algorithm....5

2.2.1 Boundary Crossing....6

2.2.2 Assumption of Continuity in the Boundary Crossing Algorithm....8

3 Implementation in an Equation Based Environment....8

3.1 The Modeling Tools....12

3.2 The Solving Engine....12

3.2.1 Logical Analysis....13

3.2.2 Configuration of the System of Equations....16

3.2.3 Conventional Nonlinear Solver: Newton-like Step....16

3.2.4 Setting Up the Optimization Subproblem....16

3.2.5 Solution of Optimization Subproblems: Gradient Step....17

3.2.6 Termination....17

3.2.7 About CMSlv....17

4 Illustrative Examples....19

5 Results....21

6 Conclusions....22

7 Acknowledgments....22

8 References....23

Appendix A ASCEND Models of the Examples Solved with the Boundary Crossing Algorithm....25

Appendix B Derivation of the Optimization Subproblem of the Boundary Crossing Algorithm....29

List of Figures

Figure 1 MINLP approach to the solution of conditional models.....2

Figure 2 The boundary crossing algorithm in a simple flash equilibrium calculation..... 7

Figure 3 Vectors of gradients are of opposite direction at a boundary..... 9

Figure 4 Flowsheet of the boundary crossing algorithm implementation.....10

Figure 5 The laminar-turbulent transition in the ASCEND modeling language..... 13

Figure 6 Object-oriented architecture of the boundary crossing implementation.....14

Figure 7 Implementation of the logical solver LRSlv.....15

Figure 8 Setting up the optimization subproblem..... 18

List of Tables

Table 1 Results.....21

1 INTRODUCTION

Process engineering design and simulation generally require one to find the solution to a large system of nonlinear equations. Conventional process models consist of a set of variables and a unique set of m equations that related them. On the other hand, conditional models constitute a means to formulate alternative sets of m equations depending on the values of the modeling variables. The system of equations of the model is different for each of the alternatives. Thus, the main difficulty while dealing with conditional models is that their solution involves simultaneously selecting the equations to be solved and solving them.

In conditional models, the equations within each alternative set are confined to a subset of the search space, which results in the dissection of the feasible region into subregions (Zaher, 1995). Frequently, however, a set of equations is found to be common to all of the subregions. Hence, a conditional model consists of a system of equations expressed by two sets: a globally defined invariant set of equations (common to all of the subregions) and a variant set of conditional (or locally defined) equations which, in general, is formulated disjunctively (Zaher, 1995; Grossmann and Turkay, 1996)

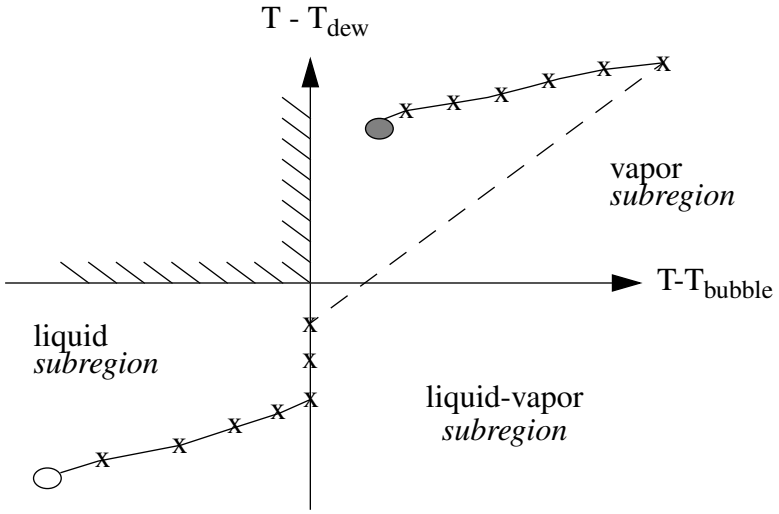
While dealing with conventional models, one generally applies iterative nonlinear techniques in order to solve the equations simultaneously. Newton-like numerical techniques are the methods most commonly applied to find the solution of conventional models. Such methods require both continuity and differentiability of the system of equations over the entire feasible region in order to guarantee convergence. In conditional models, however, because of the loss of continuous functionality between the surfaces of neighboring subregions in the search space, Newton-like iterative techniques cannot be applied if the solution to the system requires a step from one subregion to another.

The literature reports several approaches to the solution of conditional models. One can formulate conditional models as mixed integer programming problems. Grossmann and Turkay (1996) show that, in the case of linear equations, the solution of a conditional model can be found by solving a MILP problem that is often solvable as a relaxed LP. Also, based on the works of Raman and Grossmann (1993, 1994), Turkay and Grossmann (1996) address the solution of disjunctive set of nonlinear programming problems. They show that the use of logic can improve

the efficiency and robustness of MINLP algorithms in the solution of structural optimization problems. Zaher (1995) makes the following observations about the MINLP approach (big-M formulation) to the solution of conditional models:

1. One can expect that, during the solution of a MINLP, the algorithm will jump from one subregion to another which is remote from it, leading to the poor initialization of some of the nonlinear subproblems (See Figure 1).
2. The nonlinear subproblems insist on converging the equations of what might be an incorrect subregion.
3. This method remains as a rigorous general purpose algorithm designed for problems where discrete decision making cannot be avoided.

FIGURE 1 MINLP approach to the solution of conditional models. Note the jump from the liquid to the vapor region, leading to poor initial conditions from which to converge the vapor model equations.



Besides the mixed-integer formulation, some alternative approaches exist which, in principle, can avoid the need of discrete decisions. Rico-Ramírez and Westerberg (1998) propose an extension to the standard complementarity formulation for the representation of conditional models. The

complementarity approach performs well for models containing a small number of conditional equations, but it relies on the nondegeneracy of the resulting complementarity problem in order to avoid numerical singularities. Therefore, it leaves out those cases on which multiple set of alternative equations may be simultaneously satisfied (degenerate case).

This work focuses on the implementation of a third approach to the solution of conditional models, the boundary crossing algorithm (Zaher, 1991).

2 THE BOUNDARY CROSSING ALGORITHM

Zaher (1991) introduced a boundary crossing algorithm as an alternative approach to the solution of conditional models. In the most recent of his works, Zaher (1995) describes the boundary crossing algorithm in the context of an optimization problem. He recognizes the problem of solving a conditional model as one of nondifferentiable optimization, and proposes an augmented-Lagrangian SQP approach for the solution of such a problem. A thorough report of the results obtained by that approach is in preparation. In this work, we focus in the boundary crossing algorithm as a technique for the solution of a fully determined nonlinear system of equations. Here we show that, by isolating in boundaries the nondifferentiable functionality of conditional models, the solution of such problems can be tackled by an algorithm mainly based on the use of conventional nonlinear techniques (such as Newton-based methods).

2.1 THE FORMULATION

In conditional models, the equations within each alternative set are confined to a subset of the feasible region. In the boundary crossing algorithm, a set of boundary expressions of the form $\underline{b}(\underline{x}) \geq 0$ is used to associate each set of conditional equations to each of the subregions. Each boundary expression partitions the feasible region into two neighboring subregions, each of the subregions characterized by either the satisfaction or non-satisfaction of the expression. Thus, each subregion in the feasible region is identified by a unique combination of truth values for all of the boundary expressions. In this way, the partitioning of the feasible region into subregions is combinatorial. If l is the number of boundary expression (i.e. $b_i(\underline{x}) \geq 0$ where $i \in \{1 \ldots l\}$), the number of subregions is 2^l. For most of the conditional models, though, it is physically not meaningful for the solution to lie in some of the subregions. These subregions, therefore, should

be removed from the feasible region and no conditional equations need to be associated to them. One must emphasize that the boundary expressions are not inequality constraints but rather are used to associate values of the modeling variables with the appropriate set of equations. A way of identifying a specific subregion is by using a set s where $s \in \{1 \ldots l\}$. The set s contains the indices of the boundary expressions which must be satisfied in order for the solution to reside in that specific subregion. If R is the set of all of the subsets s belonging to the feasible region with which m conditional equations are to be associated in the form $\underline{r}_s(\underline{x}) = 0$, then the conditional model can be formulated in the following disjunctive representation:

$$\bigvee_{s \in R} \begin{bmatrix} b_i(\underline{x}) \geq 0, \forall i \in s \\ b_i(\underline{x}) < 0, \forall i \in \{1 \ldots l\} - s \\ \underline{r}_s(\underline{x}) = 0 \end{bmatrix} \tag{1}$$

Note that, when the number of boundary expressions is zero, only one subregion is created using $s = \varnothing$ and the previous formulation reduces to a conventional nonlinear system of equations.

Since frequently a large number of equations are common to all of the subregions, it is convenient to include those invariant equations outside the disjunction:

$$\begin{gathered} \underline{g}(\underline{x}) = 0 \\ \bigvee_{s \in R} \begin{bmatrix} b_i(\underline{x}) \geq 0, \forall i \in s \\ b_i(\underline{x}) < 0, \forall i \in \{1 \ldots l\} - s \\ \underline{r}_s(\underline{x}) = 0 \end{bmatrix} \end{gathered} \tag{2}$$

where now $\underline{g}(\underline{x})$ is m_g dimensional and $\underline{r}_s(\underline{x})$ is m_r dimensional such that $m_g + m_r = m$. Note that the formulations given by (1) and (2) represent one disjunction over all of the subregions in the feasible region of a conditional model. Such a global disjunctive statement can be generated from the combination of all of the individual disjunctions in the problem. However, it is not necessary to formulate the conditional model strictly in the form given by (1) or (2) in order to solve it by using the boundary crossing algorithm. A representation in which each individual disjunction is explicitly formulated is also suitable.

2.2 THE SOLUTION ALGORITHM

The goal of a conditional model solver is to find a solution to the disjunctive system of equations (2), a solution which consists in a vector of variables satisfying the invariant set of equations and exactly one set of conditional equations, providing that the truth values of the boundary expressions are consistent with the set of variant equations constituting the solution.

The most popular iterative numerical techniques for the solution of a system of nonlinear equations involve strategies designed to enforce descent of some objective function with each iteration. This objective function is generally chosen as the square norm of the residuals of the model equations, assuming the equations are all appropriately scaled. Hence, for each subregion the objective function $\phi_s(\underline{x})$ is defined by:

$$\underline{f}_s(\underline{x}) = \begin{bmatrix} \underline{g}(\underline{x}) \\ \underline{r}_s(\underline{x}) \end{bmatrix}$$
$$\phi_s(\underline{x}) = \frac{1}{2} \cdot \underline{f}_s^T(\underline{x}) \cdot \underline{f}_s(\underline{x}) \qquad \textbf{(3)}$$

In the boundary crossing algorithm, all of the points where the functionality is nondifferentiable are characterized by boundary hyper-surfaces as given by the boundary expressions $\underline{b}(\underline{x}) \geq 0$. Therefore, the equations within each subregion form a continuous and differentiable objective function. Hence, a Newton-based numerical technique may find the solution when the initial point is internal to the correct subregion. On the other hand, if one places the initial point inside an incorrect subregion, one could expect to encounter a subregion boundary in a finite number of iterations.

According to the above discussion, the boundary crossing algorithm applies a Newton-like step for all the points internal to a subregion, taking advantage of the superlinear convergence feature of such a step. If during the iterative process a boundary is reached, then a Newton-like step cannot be taken because of the nondifferentiability of the objective function at that point. A boundary crossing based on subgradient methods (linear convergence methods) takes place instead. Once off the boundary, the superlinear convergence steps resume using a possibly new set of equations. Figure 2 illustrates the boundary crossing algorithm.

2.2.1 Boundary Crossing

In the boundary crossing algorithm, because of the lack of differentiability at a boundary, an analogue of the steepest descent vector is derived in order to advance the solution progress into the correct subregion.

While solving continuous and differentiable models, the steepest descent direction at some point with values $\underline{a}$ for the variables is in the opposite direction of the gradient of the objective function at that point. Hence, a vector $\underline{d}$ with a direction of steepest descent direction is a vector which solves the following problem:

$$\begin{aligned} min \quad & \nabla_x \phi(\underline{a})^T \cdot \underline{d} \\ s.t. \quad & \underline{d}^T \cdot \underline{d} \leq 1 \end{aligned} \tag{4}$$

When a descent direction exists at $\underline{a}$, the solution vector $\underline{d}$ has the same direction as $-\nabla_x \phi(\underline{a})$ but it is of unit length. In other words, the steepest descent direction for a differentiable function is the direction which yields a minimum in the directional derivative.

For a conditional model, each subregion has a different value for the directional derivative at a boundary. For that case, Zaher (1991) defines the directional derivative as the maximum over all of the subregion directional derivatives. If B is the set of all of the subregions in the neighborhood of a boundary, then the directional derivative for the problem at such boundary is given by:

$$\begin{aligned} max \quad & \nabla_x \phi_i(\underline{a})^T \cdot \underline{d} \\ s.t. \quad & i \in B \end{aligned} \tag{5}$$

Hence, the minimum of this directional derivative constitutes the steepest descent direction at the boundary:

$$\begin{aligned} min \quad & \begin{pmatrix} max & \nabla_x \phi_i(\underline{a})^T \cdot \underline{d} \\ s.t. & i \in B \end{pmatrix} \\ s.t. \quad & \underline{d}^T \cdot \underline{d} \leq 1 \end{aligned} \tag{6}$$

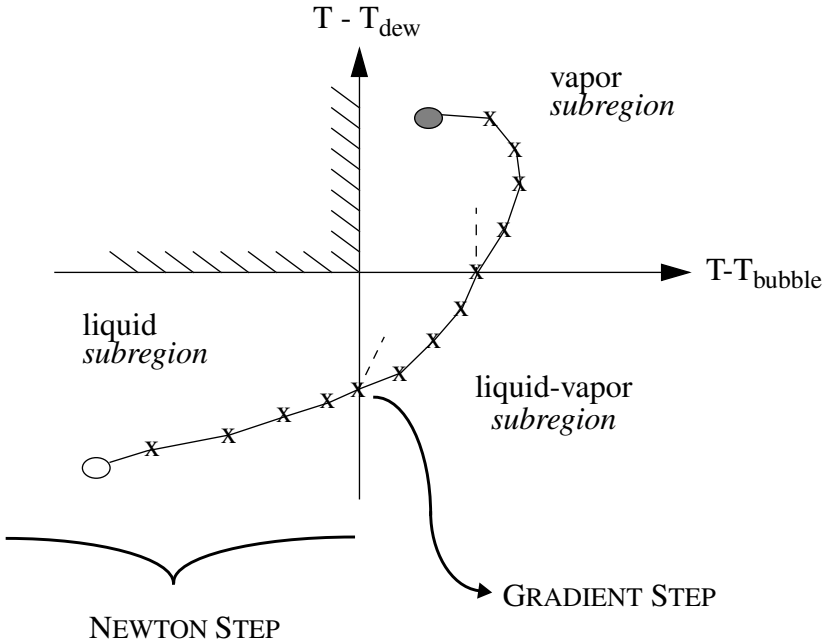
The definition of directional derivative given by (5) aims for a descent direction with respect to all the subregions neighboring the boundary. By doing that, one will prevent a cycling situation in

which a loop of connected subregions are visited continually. The problem given by (6) is recognized as a mini-max problem. Because of the convexity of this problem, one can derive the dual problem to obtain:

$$\begin{aligned} min \quad & \underline{d}^T \cdot \underline{d} \\ s.t. \quad & \underline{d} = -N \cdot \underline{\alpha} \\ & \underline{e}^T \cdot \underline{\alpha} = 1 \\ & \alpha_i \geq 0, \forall i \in B \end{aligned} \tag{7}$$

where $\underline{e}$ is a vector with all its elements equal to one and N is a matrix whose columns are the vectors of gradients of the subregions in the neighborhood of the current boundary. That is, column i of N is the vector $\nabla_x \phi_i(\underline{a})$, $\forall i \in B$. Also, both $\underline{e}$ and $\underline{\alpha}$ are vectors with dimension equal to the number of subregions neighboring the current boundary. The derivation of equation (7) from (6) has been described by Zaher (1991,1995) and it is given in Appendix B. If all the gradients of the subregions cannot be confined to one side of the boundary hyper-surface, the result of (7) is a minimum of zero and we terminate the algorithm. We terminate because no

FIGURE 2 The Boundary Crossing Algorithm in a simple flash equilibrium calculation.



descent direction with respect to all the neighboring subregions has been found. Otherwise, If the

result of (7) is nonzero, the linear combination of the elements of the vector $\underline{\alpha}$ is used to generate the steepest descent direction, along which one should move away from the boundary.

2.2.2 Assumption of Continuity in the Boundary Crossing Algorithm

In the boundary crossing algorithm, one has to assume the continuity of the conditional model. In conditional models, continuity is ensured as follows: if the solution to the equations of an arbitrary subregion lies on a boundary hyperplane, then that solution must satisfy the equations of all of the subregions neighboring that boundary.

The reason for this assumption of continuity is best described by analyzing Figure 3. Figure 3 presents two cases in which continuity is not preserved in the conditional model and discovers the potential difficulties of such cases. In case (a) there is no solution to the problem, and in case (b) the conditional model has two solutions. Note that, in both cases in Figure 3, the vectors of gradients of the subregions neighboring the boundary are of opposite direction at the point *(x=10,y=7.5)*. If we solve problem (7) for such a point at the boundary, the solution of (7) would be zero in both cases, and the boundary crossing algorithm would terminate. Hence, for the case (b), the boundary crossing algorithm would terminate without finding any of the two existing solutions to the conditional model. A possible cure for this complication would be to search in each neighboring subregion for a solution interior to any of them. Such an approach has been considered throughout this research, but it is not yet a part of the implementation that we describe in this paper.

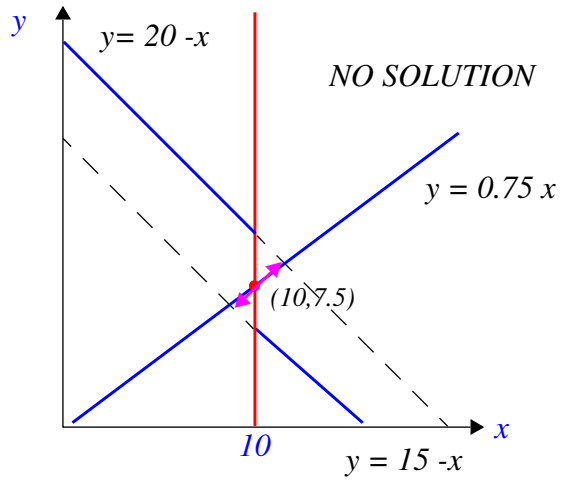
3 IMPLEMENTATION IN AN EQUATION BASED ENVIRONMENT

In this section, we present some of the details of the implementation of the boundary crossing algorithm in an equation-based environment.

In Figure 4, we show a flowsheet of the main activities that a solver applying the boundary crossing algorithm must perform. Given the initial guess for the model variables, the algorithm starts by evaluating the logical boundary expressions in order to determine whether that initial guess resides on a boundary or it is internal to any of the subregions:

FIGURE 3 Vectors of gradients are of opposite direction at a boundary.

(a)



(b)

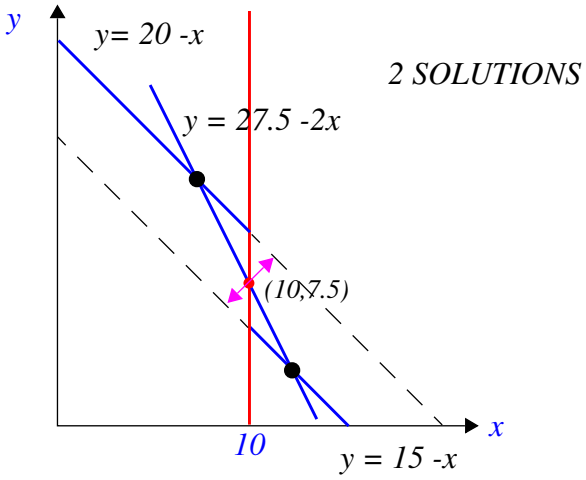
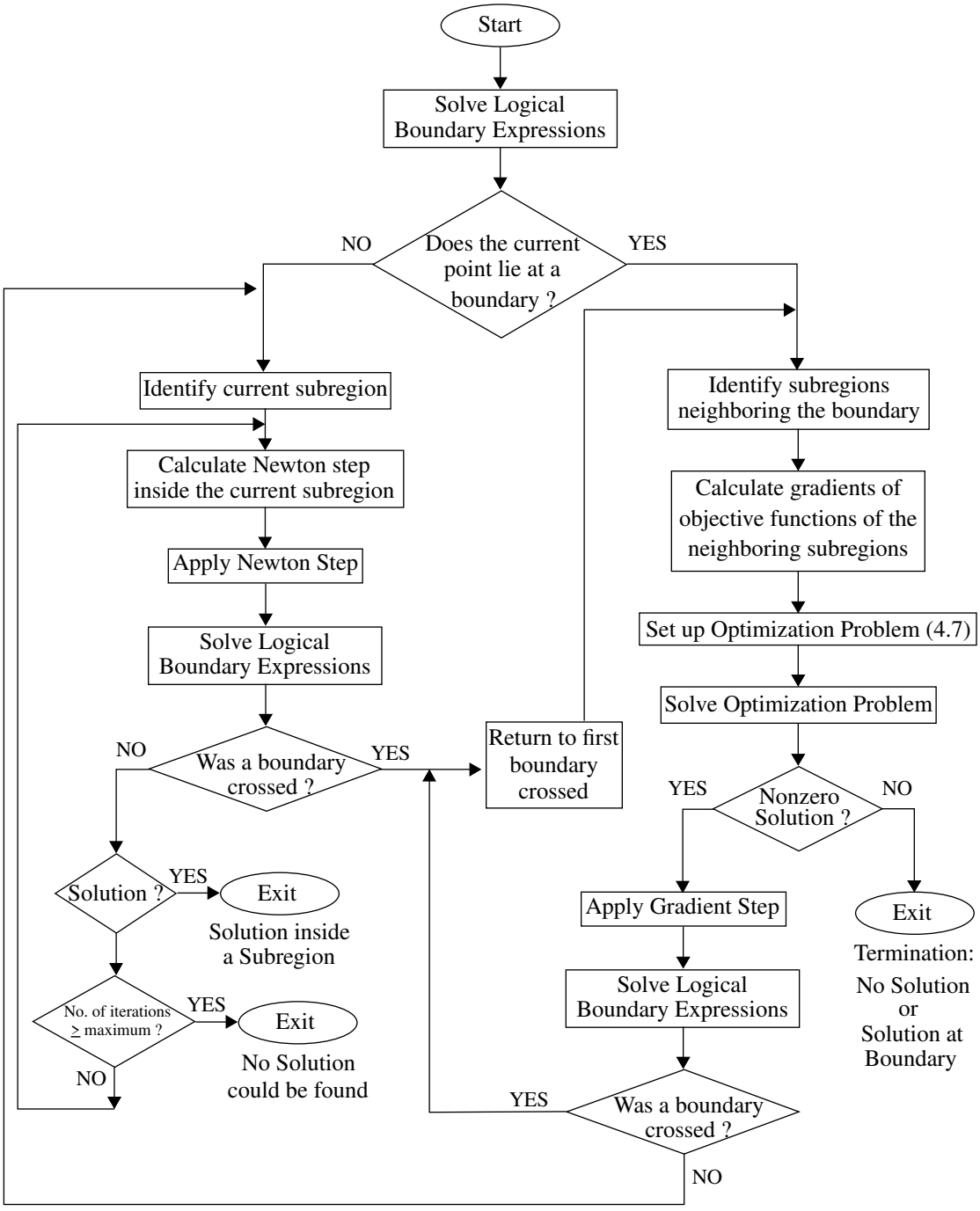


FIGURE 4 Flowsheet of the boundary crossing algorithm implementation.



- If the point resides on a boundary, a subgradient step has to be calculated. In order to do that, the equations and variables of all of the subregions neighboring the boundary have to be identified and differentiated. The problem given by (7) is then generated and solved to obtain the steepest descent direction. If the result of (7) is a minimum of zero, the algorithm terminates concluding that no descent direction could be found for the current point on the boundary. We want to emphasize that the algorithm terminates when the result of the optimization problem is zero because we want to guarantee termination. In such a case, the point on the boundary might not constitute a solution to the conditional model, as we describe in section 2.2.2. If the solution of (7) is nonzero, the subgradient step is taken. After this step, it is necessary to verify is one or several boundaries have been crossed. If so, the subgradient step is reduced until the problem lies exactly on the first boundary crossed, and a subgradient step must be calculated for the new boundary. If no boundary has been crossed, the algorithm proceeds to identify the variables and equations of the current subregion and a nonlinear conventional technique is the used to move towards the solution as explained below.
- If the point does not reside on a boundary, it is necessary to identify the initial subregion. That means that we need to identify the equations and variables constituting the initial problem. The algorithm assumes that the starting point corresponds to a subregion for which the system of equations are defined, i.e. the initial subregion is feasible. An iterative technique is then used in order to generate a Newton-like step towards the solution. After each step, one needs to verify if some of the boundaries have been crossed. If no boundary has been crossed, the numerical technique continues the iterative process in the current subregion until the convergence criteria is satisfied (termination of the algorithm) or until a boundary is encountered. When a boundary has been crossed, the length of the Newton-like step has to be reduced until the system resides exactly on the boundary, and then a boundary crossing has to be performed by using a subgradient step as explained above.

An efficient implementation of this algorithm in an equation-based environment is hard for the following reasons:

1. The implementation requires the incorporation of modeling tools which enables the user of an equation-based environment to represent conditional models. Mainly, these modeling tools have to be capable of representing alternative sets of equations and the logical conditions

given by the boundary expressions.

2. The boundary crossing algorithm involves the performance of several different tasks, including logical analysis, continuous reconfiguration of the equations and variables of the system (switching among alternative subregions), performance of a Newton-like step (which requires the incorporation of a conventional nonlinear technique) and performance of a subgradient step (which requires the solution of an optimization problem). Thus, in the implementation of this technique one has to provide the numerical and algorithmic tools needed to execute each of the previous tasks and to integrate them in a procedural solving engine.

3.1 THE MODELING TOOLS

In this work, we use of the modeling tools described in by Rico-Ramírez et. al. (1998) which allow for the representation of conditional models in an equation based environment Those modeling capabilities meet the representation needs of the boundary crossing algorithm:

1. Boundary expressions can be represented by combining the CONDITION statement and logical relations.
2. Alternative set of model equations can be represented by using the WHEN statement.

These tools have already been incorporated into the ASCEND IV modeling environment (Westerberg et al., 1994; Allan, 1997), hence, such a modeling language was used in this work for representation purposes. Figure 5 show an example of a conditional model (laminar-turbulent flow transition) represented using the ASCEND modeling language. The WHEN statement is used to define the alternative sets of variant equations. The selection between the two alternatives is done in terms of the boolean variable laminar whose truth value depends on the boundary expression defined within the CONDITION statement.

3.2 THE SOLVING ENGINE

We implement our conditional modeling solver in a server-client, object oriented architecture. Figure 6 describes this implementation. This conditional modeling solver (CMSlv) is in charge of the management of external entities which, at the same time, are in charge of executing each of the specific tasks that the algorithm requires. Particularly, calls to external entities are done for the

execution of:

1. The analysis of logical boundary expressions.
2. The calculation of a quadratic converge step inside a subregion (Newton-like iterative technique).
3. The calculation of a subgradient step by solving an optimization subproblem.

In addition, the conditional modeling solver is also in charge of the re/configuration of the appropriate set of equations and variables at each step of the algorithm, convergence tests, and the generation of the optimization subproblem when the system resides on a boundary. In the remainder of this section, we will describe in more detail each of these tasks as well as the external entities created and/or used to provide support to the conditional solver CMSlv.

FIGURE 5 The laminar-turbulent transition in the ASCEND modeling language

```
laminar     IS_A boolean;
Re,f,k      IS_A factor;

CONDITION
      condition: Re <= 2100;
END;

laminar <==> SATISFIED(condition);

invariant: Re = k/sqrt(f);
eq_laminar: Re = 64/f;
eq_turbulent: Re = (0.206307/f)^4;

WHEN (laminar)
      CASE TRUE:
            USE eq_laminar;
      CASE FALSE:
            USE eq_turbulent;
END WHEN;
```

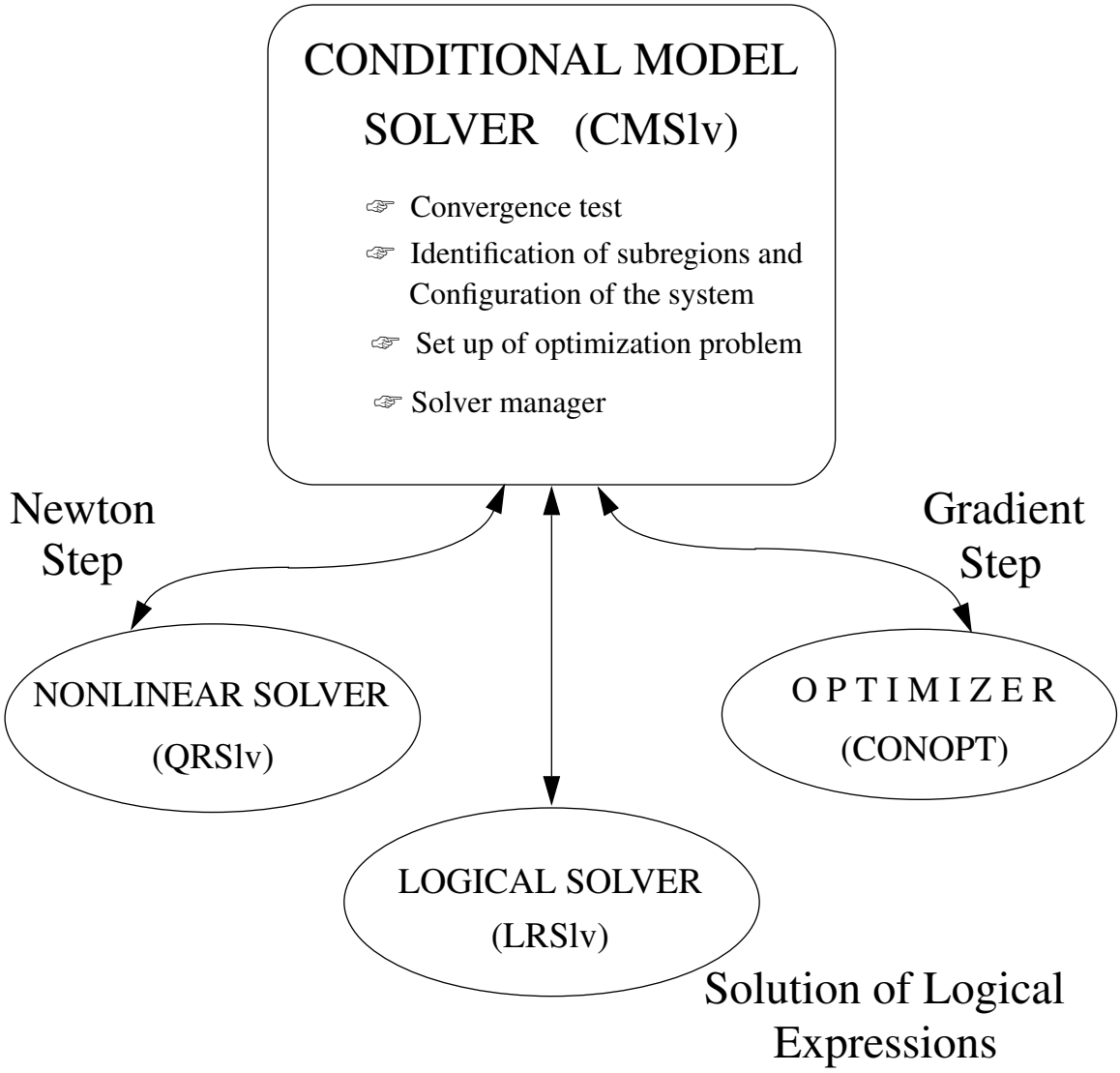
3.2.1 Logical Analysis

The domain of validity of the alternative sets of equations in a conditional model is given in terms

of the truth value of a set of logical boundary expressions. Hence, logical expressions have to be continuously evaluated during the iterative solution scheme. In order to perform this task, we created the logical solver LRSlv. The ASCEND modeling language allows a logical boundary expression to be represented in terms of a combination of:

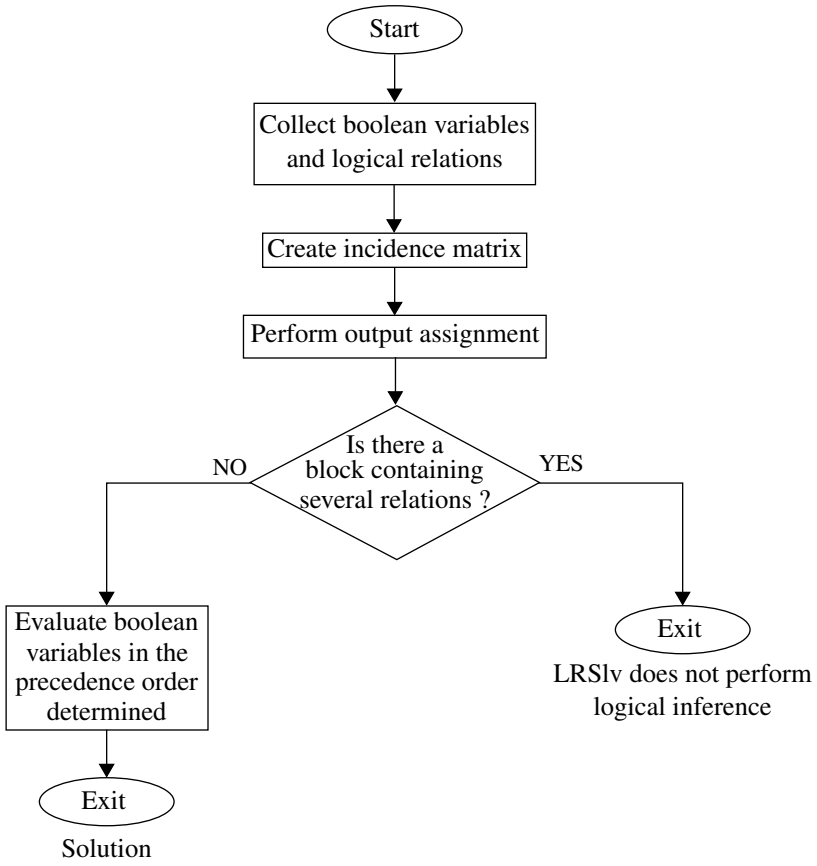
- Boolean variables.
- Boolean operators.
- The truth value of conditions expressed in term of real variables.

FIGURE 6 Object-oriented architecture of the boundary crossing implementation.



Hence, the logical solver LRSlv has been implemented in such a way that it is able to handle this type of flexibility. Figure 7 shows the flowsheet of the activities performed by LRSlv. In this implementation, we assume that we can explicitly evaluate all the boolean variables in some precedence order. In other words, LRSlv does not perform logical inference. Thus, while finding the value of a boolean variable, we require that the truth values of all the boolean variables and conditions on which each boolean variable depends to have been previously given or previously evaluated. Also, we assume that we can evaluate at any time the truth value of a condition expressed in terms of real variables. In general, both of our assumptions will be met while the boundary crossing algorithm is being used to solve conditional models. LRSlv has been attached as an external entity providing support to CMSlv as explained above.

FIGURE 7 Implementation of the logical solver LRSlv.



3.2.2 Configuration of the System of Equations

The conditional solver is capable of determining whether the current point resides on a boundary or is internal to any of the subregions. If the point is internal to any of the subregions, CMSlv must identify the equations and variables constituting the problem in that specific subregion. To perform this task, CMSlv uses the notion of active equation and active variable. A consistent mechanism to select the structure of the system inside a specific subregion is described in Rico-Ramírez et. al. (1997). That mechanism is used in this work.

We emphasize that a change in the configuration of the problem during an iterative solution process may also cause a change in the partitioning of the variables and in the partitioning of the equations. Zaher (1995) developed algorithms for simplifying the partitioning of variables and equations in a conditional model, but those algorithms are not part of this implementation yet.

3.2.3 Conventional Nonlinear Solver: Newton-like Step

When the values of the variables of the model correspond to a point internal to any of the subregions, the boundary crossing algorithm requires the calculation of a Newton-like step. CMSlv calls an external solver to perform this task. We do not impose any restrictions for the selection of this external solver. CMSlv can be able to interact with any solver which is able to calculate a step based on derivative and function evaluations and on the structural information of the system of equations. In the current implementation of CMSlv, the Newton-like step is calculated by the ASCEND nonlinear solver QRSlv (Westerberg, 1989). In order to enforce the descent of the objective function local to the subregion, QRSlv uses the modification to the Levenberg-Marquardt algorithm given by Westerberg and Director (1979).

3.2.4 Setting Up the Optimization Subproblem

When CMSlv determines that the current point of the system resides on a boundary, a subgradient step has to be calculated by solving an optimization subproblem. Setting up this optimization subproblem requires the calculation of the gradients of the objective functions of all of the subregions neighboring a boundary. Also, it results in a system of equations which is quite different from the system of equations of any of the subregions of the original problem. That implies that the data structures required to provide the information about this optimization subproblem have to be dynamically created and destroyed. In addition, in order to minimize the

number of function and gradient evaluations employed for setting up each optimization subproblem, it is necessary to identify the invariant set of equations (if there is one) and to avoid multiple evaluations for those equations. Figure 8 presents the flowsheet of the activities performed by CMSlv while setting up an optimization subproblem at a boundary.

3.2.5 Solution of Optimization Subproblems: Subgradient Step

After CMSlv has created the optimization subproblem at a boundary, this subproblem is solved with a call to another external entity. This time, the external solver must be an optimizer. Once again, there are no limitations in the implementation of CMSlv as to which optimizer should be used. The current implementation interacts with the subroutine CONOPT (Drud, 1985), which uses a reduced gradient approach to the solution of the optimization problem. CMSlv provides the values of the derivatives of the linear constraints (elements of the vectors of gradients of the objective functions), the residual of these constraints, and the derivatives and values of the objective function. CONOPT (like QRSlv in the case of the calculation of the Newton-like step) is considered a black box which provides the subgradient step based on that information.

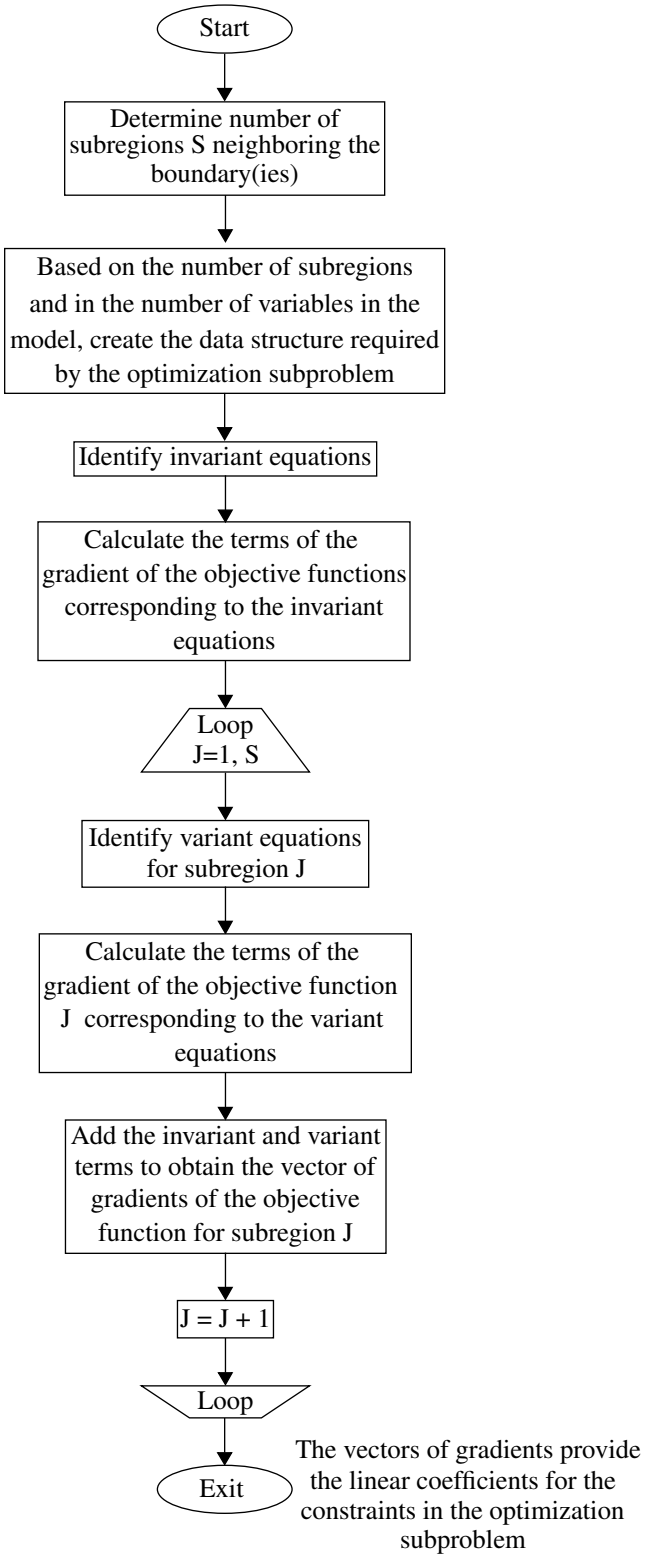
3.2.6 Termination

Termination of the algorithm can occur on a point internal to any subregion or on a point residing on a boundary. It occurs on a boundary when the result of the optimization subproblem is zero, or internal to a subregion when the norm of the residuals of the equations in that subregion is less than a specified tolerance. As we already mentioned, if the algorithm terminates at a point lying on a boundary, such a point on a boundary might not constitute a solution to the conditional model as described in section 2.2.2. CMSlv is in charge of the convergence test after both the subgradient step and the Newton-like step.

3.2.7 About CMSlv

Besides tasks like the reconfiguration of the system, termination tests, setting up of the optimization problem, etc., CMSlv is in charge of managing the logic of the algorithm and interacting with the external solvers. It provides the information required by the external solvers (LRSLv, QRSlv and CONOPT) and processes the results obtained from the calls to them. The conditional modeling solver (CMSlv) and the logical analyzer (LRSlv) have been attached to the ASCEND environment.

FIGURE 8 Setting up the optimization subproblem



4 ILLUSTRATIVE EXAMPLES

We have modeled and solved several examples of conditional models found in the literature by using the ASCEND modeling language and our implementation of the boundary crossing algorithm. Rico-Ramírez and Westerberg (1998) give a detailed description of each of these examples. in Appendix A, we show a representative section of the ASCEND model for each of the examples.

In all of the examples, the initial values of the variables correspond to an incorrect subregion. We do that in order to test the ability of the algorithm for crossing boundaries until finding the correct subregion and iterating until obtaining the solution to the problem. In particular, the initial values for Example 4 are those defined in Bullard and Biegler (1992), and the initial values for Example 6 are those defined in Rico-Ramírez and Westerberg (1998). Examples 1 through 3 were introduced by Zaher(1995). In that work, Zaher solves example 1 and 3 as optimization problems in order to test the performance of his approach to sequential quadratic programming. In this work we solve those problems as simulation problems by eliminating the objective function and defining fixed values for the degrees of freedom.

EXAMPLE 1 Fluid Transition (Zaher, 1995).

This example describes the flow of a compressible gas in an adiabatic frictional circular pipe of constant diameter. Nonsmooth functionality occurs due to the possible transition between sonic and non-sonic flow at the outlet of the pipe. This example corresponds to the simplest case of a conditional model, which involves only one boundary expression (and, therefore, only $2^1=2$ subregions) and contains only one conditional equation in each alternative set of variant equations.

EXAMPLE 2 Phase Equilibria (Zaher, 1995).

An isothermal flash is applied to a ternary system involving benzene, ethanol and water. According to the phase diagram of this mixture and, depending on the values of pressure and temperature, three phases can be expected to exist simultaneously, an aqueous liquid phase, an organic liquid phase, and a vapor phase. The existence or nonexistence of each phase can be

represented as a conditional statement. Since there are three possible phases, we require three disjunctions to represent the behavior, which means that our search space has $2^3=8$ subregions.

EXAMPLE 3 Heat Exchanger (Zaher, 1995).

It represents a case in which a conditional model contains differential equations that have to be integrated. The approach suggested is to discretize the differential equations and treat the problem as a conditional model with only algebraic equations. The difficulty with this model is that, in addition to solving for the temperature profile, the dimension of the finite elements are to be solved for as well.

There are 2 boundary expressions in the model which would involve 4 subregions in the search space. However, one of the subregions is infeasible and can be eliminated from the search space.

EXAMPLE 4 Pipeline Network (Bullard and Biegler, 1992)

Here we consider the pipe network solved previously by Bullard and Biegler (1992). The specifications of the problem as well as the initial and converged values of the flowrates can be found in Bullard and Biegler (1992) and in Rico-Ramírez and Westerberg (1998). Since the problem contains 38 pipes, and, therefore, 38 boundaries are defined, the search space is constituted by $2^{38}=2.748779\text{x}10^{11}$ potential subregions. Hence, this example represents a case in which the combinatorial nature of the problem could have a severe effect in the performance of the algorithm.

EXAMPLE 5 Simple L-V Flash Calculation (King, 1980)

This situation corresponds to a simple equilibrium calculation as given by King (1980). Basically, the problem consists of finding a solution to the well known Rachford-Rice equation. Depending on the specifications of the problem, the system may be present as subcooled liquid, superheated vapor, or a mixture liquid-vapor in equilibrium.

EXAMPLE 6 Linear Mass Balance (Grossmann and Turkay, 1996)

This example represents a problem in which each of the six processing units interconnected in a flowsheet has three operating regions, each region with different mass balance coefficient in terms of the main product flowrate. The mass balance coefficients and the bounds for each of the

flowrates have been reported in Grossmann and Turkay (1996) and Rico-Ramírez and Westerberg (1998). Because of the three operating regions, the disjunctive linear mass balance for each of the units is represented for a disjunction containing 3 disjunctive terms. The search space is constituted by 729 subregions. Also, the set of invariant equations as well as all of the sets of conditional equations contain only linear equations. Therefore, the equations in each subregion are all linear.

5 RESULTS

The number of iterations that we used to obtained the solution of each of these examples is shown in Table 1 . The initial values are the same as those values reported by Rico-Ramírez and Westerberg (1998) while employing the complementarity approach. As mentioned before, the definition of those initial values for the variables ensures that the algorithm starts in an incorrect subregion, which enforces the application of the algorithm for the crossing of boundaries.

Table 1: Results

Example	Reference	Number of Equations	Number of Disjunctions	Number of boundary crossings	Number of Iterations
Flow Transition (sonic-subsonic)	Zaher (1995	5	1	1	10
Phase Equilibria	Zaher (1995	12	3	5	17
Heat exchanger	Zaher (1995)	48	2	1	8
Pipeline network	Bullard and Biegler (1992)	98	38	44	106
Simple L-V flash	King (1980)	15	1	1	12
Linear mass balance	Grossmann and Turkay (1996)	13	6	2	8

As expected, the combinatorial complexity of the example of the pipeline network (2.748779×10^{11} possible subregions) affects the effectiveness of the boundary crossing algorithm. The solution path crossed 44 boundaries. Thus with 106 iterations, the average number of Newton

steps per region entered is only of the order of two. For the rest of the examples, the algorithm performance is very encouraging. For the linear mass balance example, the number of possible subregions is 729, but the fact that the equations are linear facilitates the convergence of the algorithm. A clear advantage of the boundary crossing implementation is the problem size. Note the number of equations inside each subregion is always the minimum, and it is not affected by the combinatorial nature of the problem.

6 CONCLUSIONS

Following a brief description of the boundary crossing algorithm, we have described the details of its practical implementation. The conditional modeling solver CMSlv is based on an server-client, objected oriented architecture. CMSlv executes a number of activities including:

- Configuration of the systems of variables and equations consistent with the restrictions imposed by the boundary expression.
- Creation of optimization subproblems at boundaries.
- Managing the logic of the algorithm
- Performing termination tests.
- Managing the interaction with external solvers; LRSlv for logical analysis, QRSlv for iterating within a subregion and CONOP for the solution of the optimization subproblems at boundaries.

The CMSlv and LRSlv solver have been attached to the ASCEND environment, whose modeling language provides the tools required for the representation of conditional models in an equation oriented manner. Finally, the modeling and solution of a number of examples show the scope of application of the algorithm and reveal its very encouraging performance.

7 ACKNOWLEDGMENTS

This work has been supported by the Institute for Complex Engineered systems at Carnegie Mellon University (NSF Grant No. EEC-8943164), by the Computer Aided Process Design (CAPD) Consortium, and by the “Consejo Nacional de Ciencia y Tecnología (CONACYT)”.

8 REFERENCES

Allan, B. A.; A More Reusable Modeling System; PhD thesis, Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania,1997.

Bullard, L. G. and Biegler, L. T.; Iterative Linear Programming Strategies for Constrained Simulation. *Comput. Chem. Eng.*, 15(4):239–254, 1991.

Bullard, L. G. and Biegler, L. T.; Iterated Linear Programming Strategies for Nonsmooth Simulation: Continuous and Mixed-Integer Approach. *Comput. Chem. Eng.*, 16(10), 1992

Drud, A.; CONOPT: A GRG Code for Large Sparse Dynamic Nonlinear Optimization Problems. *Mathematical Programming*, 31, 153-191, 1985.

Grossmann, I. E. and Turkay, M.; Solution of Algebraic Systems of Disjunctive Equations. *Comput. Chem. Eng.*, 20:S339–44, 1996. Suppl. Part A.

King, C. J.; Separation Processes, Chemical Engineering Series, 2nd. Edition, McGrawHill, 77-79, 1980.

Raman, R. and Grossmann, I. E.; Symbolic Integration of Logic in Mixed-Integer Linear Programming Techniques for Process Synthesis. *Comput. Chem. Eng.*, 17(9):909–927, 1993.

Raman, R. and Grossmann, I. E.; Modeling and Computational Techniques for Logic Based Integer Programming. *Comput. Chem. Eng.*, 18(7):563–578, 1994.

Rico-Ramirez, V., Allan, B. A., and Westerberg, A. W.; Conditional Modeling in an Equation-based Environment, ICES Technical Report 06-242-98, Carnegie Mellon University, Pittsburgh, PA, 15213, 1998.

Rico-Ramirez, V. and Westerberg, A.W.; Complementarity Formulation for the Representation of Algebraic Systems Containing Conditional Equations, ICES Technical Report 06-243-98, Carnegie Mellon University, Pittsburgh, PA, 15213, 1998.

Turkay, M. and Grossmann, I. E.; Logic-Based MINLP Algorithms for the Optimal Synthesis of Process Networks. Comput. Chem. Eng., 20(8):959–978, 1996.

Westerberg, A.W., Abbott, K. A., and Allan, B. A.; Plans for ASCEND IV: Our Next Generation Equational-Based Modeling Environment. Boston, Massachusetts, November 1994. AspenWorld'94.

Westerberg, A. W. and Director, S. W.; A Modified Least Square Algorithm for Solving Sparse n x n Sets of Nonlinear Equations. Technical report, Carnegie Mellon University, Engineering Design Research Center, EDRC-06-5-79, January 1979.

Westerberg, K.M.; Development of Software for Solving Systems of Nonlinear Equations. Technical Report. Carnegie Mellon University, Engineering Design Research Center, EDRC 05-36-89, 1989.

Zaher, J. J.; Conditional Modeling. PhD thesis, Department of Chemical Engineering Carnegie Mellon University, Pittsburgh, Pennsylvania, May 1995.

Zaher, J. J.; Conditional Modeling in an Equation-Based Modeling Environment. The Annual AIChE National Meeting, 1991. paper 138c.

Zaher, J. J.; Conditional Programming. The Annual AIChE National Meeting, March 1993.

APPENDIX A

ASCEND Models of the Examples Solved with the Boundary Crossing Algorithm

Example 1 Fluid Transition (Zaher,1995).

```
CONDITION
      bound: (Pd - Pf) * 1.0 {atm^-1} < Mf - 1.0;
END CONDITION;

sonic_flow <==> SATISFIED(bound,1e-08);

non_sonic: Pf = Pd;
sonic: Mf = 1.0;

WHEN (sonic_flow)
      CASE TRUE:
            USE sonic;
      CASE FALSE:
            USE non_sonic;
END WHEN;
```

Example 2 Phase Equilibria (Zaher,1995).

```
components :== ['B','E','W'];
phases :== ['A','O','V'];
CONDITION
      FOR i IN phases CREATE
            bound[i]: SUM[y[i][j] | j IN components ]+ phi[i] >= 1.0;
      END FOR;
END CONDITION;

FOR i IN phases CREATE
      sum[i]: SUM[y[i][j] | j IN components ] = 1.0;
      frac[i]: phi[i] = 0.0;
      exist[i] <==> SATISFIED(bound[i],1e-08);

      WHEN (exist[i])
            CASE TRUE:
                  USE sum[i];
            CASE FALSE:
                  USE frac[i];
      END WHEN;
END FOR;
```

Example 3 Heat Exchanger (Zaher,1995)

```
components :== ['B','P5','H'];

CONDITION
      bound1: SUM[x[0][i] | i IN components] + phi[0] >= 1.0;
      bound2: SUM[x[2][i] | i IN components] + phi[1] >= 1.0;
END CONDITION;

liq_exist[1] <==> SATISFIED(bound1,1e-08);
liq_exist[2] <==> SATISFIED(bound2,1e-08);

sum0: SUM[x[0][i] | i IN components] = 1.0;
frac0: phi[0] = 0.0;

sum2: SUM[x[2][i] | i IN components] = 1.0;
frac1: phi[1] = 0.0;

p1: eta[1] = 0.5;
p2: eta[2] = 0.5;

sum1: SUM[x[1][i] | i IN components] = 1.0;
p12: eta[1] = 0.0;

WHEN (liq_exist[1])
      CASE TRUE:
            USE sum0;
            USE sum1;
      CASE FALSE:
            USE frac0;
            USE p12;
END WHEN;

WHEN (liq_exist[2])
      CASE TRUE:
            USE sum2;
            USE p1;
      CASE FALSE:
            USE frac1;
            USE p2;
END WHEN;
```

Example 4 Pipeline Network (Bullard and Biegler, 1992).

Model of a pipe with no valve.

```
positive_flow     IS_A  boolean_var;
```

```
      CONDITION
            bound: Q >= 0.0 {gpm};
      END CONDITION;

      positive_flow <==> SATISFIED(bound,1e-08 {gpm});

      positive_head: H = k * sqr(Q);
      negative_head: H = -k * sqr(Q);

      WHEN (positive_flow)
            CASE TRUE:
                  USE positive_head;
            CASE FALSE:
                  USE negative_head;
      END WHEN;
```

Example 5 Simple L-V Flash Calculation (King, 1980).

```
(* Rachford-Rice *)

      SUM[ ( z[i] * ( K[i] - 1 ) ) / ( ((K[i]-1)* V_F) + 1 )
                                                | i IN components ] = R - V_F;

(* Conditional Equations *)
      CONDITION
            bound1: R <= 0.0;
            bound2: R <= 1.0;
      END CONDITION;

(* Logical Relations *)
      bol1 <==> SATISFIED(bound1,1e-08);
      bol2 <==> SATISFIED(bound2,1e-08);

(* Variant Equations*)
      liq_eq: V_F = 0.0;
      two_eq: V_F = R;
      vap_eq: V_F = 1.0;

(* Disjunctions *)
      WHEN (bol1,bol2)
            CASE TRUE,TRUE:
                  USE liq_eq;
            CASE FALSE,TRUE:
                  USE two_eq;
            CASE FALSE,FALSE:
                  USE vap_eq;
            OTHERWISE:
      END WHEN;
```

Example 6 Linear Mass Balance (Grossmann and Turkay, 1996).

```
      CONDITION
            bound1: Fmain <= B[1];
            bound2: Fmain >= B[2];
      END CONDITION;

      bol1 <==> SATISFIED(bound1,1e-08{lb_mole/hour});
      bol2 <==> SATISFIED(bound2,1e-08{lb_mole/hour});

(* Variant Equations *)

      eq1a: Fsub1 = a[1] * Fmain;
      eq1b: Fsub2 = b[1] * Fmain;

      eq2a: Fsub1 = a[2] * Fmain;
      eq2b: Fsub2 = b[2] * Fmain;

      eq3a: Fsub1 = a[3] * Fmain;
      eq3b: Fsub2 = b[3] * Fmain;

(* Disjunction  *)

      WHEN (bol1,bol2)
            CASE TRUE,FALSE:
                  USE eq1a;
                  USE eq1b;
            CASE FALSE,FALSE:
                  USE eq2a;
                  USE eq2b;
            CASE FALSE,TRUE:
                  USE eq3a;
                  USE eq3b;
      END WHEN;
```

APPENDIX B

Derivation of the Optimization Subproblem of the Boundary Crossing Algorithm

Zaher (1991,1995) describes the derivation of Equation (7) as follows. Let us start from Equation (6):

$$min \quad \begin{pmatrix} max & \nabla_x \phi_i(\underline{a})^T \cdot \underline{d} \\ s.t. & i \in B \end{pmatrix}$$
$$s.t. \quad \underline{d}^T \cdot \underline{d} \leq 1$$

where B is the set of all of the subregions in the neighborhood of a boundary, $\nabla_x \phi_i(\underline{a})$ is the gradient vector of the objective function for the neighboring subregion i, and $\underline{d}$ is the descent direction which constitutes the solution to the problem (6). Let us now define the matrix N whose columns are the vectors of gradients of the subregions in the neighborhood of the current boundary. That is, column i of N is the vector $\nabla_x \phi_i(\underline{a})$, $\forall i \in B$. Hence, the inner maximization problem of (6) can be reformulated as finding the value of a scalar β such that:

$$\beta \underline{e} \geq N^T \underline{d} \qquad \textbf{(8)}$$

where $\underline{e}$ is a vector with all its elements equal to one. Then, Equation (6) can be rewritten as:

$$\begin{aligned} min \quad & \beta \\ s.t. \quad & \beta \underline{e} \geq N^T \underline{d} \\ & \underline{d}^T \cdot \underline{d} \leq 1 \end{aligned} \qquad \textbf{(9)}$$

The real-valued Lagrange function, $L(\beta, \underline{d}, \underline{\alpha}, \pi)$, of problem (9) is given by:

$$L(\beta, \underline{d}, \underline{\alpha}, \pi) = \beta - \underline{\alpha}^T (\beta \underline{e} - N^T d) - \pi(1 - d^T d) \qquad \textbf{(10)}$$

where $\underline{\alpha}$ is the vector of multipliers corresponding to first of the inequality constraints in (9) and π is the multiplier (scalar) corresponding to the second inequality of (9). Hence, the dual of the

problem (9) is given by:

$$\begin{aligned} max \quad & \beta - \underline{\alpha}^T(\beta \underline{e} - N^T d) - \pi(1 - d^T d) \\ s.t. \quad & \underline{e}^T \cdot \underline{\alpha} = 1 \\ & 2\pi \underline{d} = -N\underline{\alpha} \\ & \pi \geq 0,\ \alpha_i \geq 0,\ \forall i \in B \end{aligned} \tag{11}$$

The equality constraints of (11) were obtained from the first order conditions of the Lagrangian function with respect to the variables β and $\underline{d}$. After further simplification, (11) reduces to:

$$\begin{aligned} max \quad & -\pi(\underline{d}^T \cdot \underline{d} + 1) \\ & 2\pi \underline{d} = -N \cdot \underline{\alpha} \\ & \underline{e}^T \cdot \underline{\alpha} = 1 \\ & \pi \geq 0,\ \alpha_i \geq 0,\ \forall i \in B \end{aligned} \tag{12}$$

which is equivalent to:

$$\begin{aligned} min \quad & \pi(\underline{d}^T \cdot \underline{d} + 1) \\ & 2\pi \underline{d} = -N \cdot \underline{\alpha} \\ & \underline{e}^T \cdot \underline{\alpha} = 1 \\ & \pi \geq 0,\ \alpha_i \geq 0,\ \forall i \in B \end{aligned} \tag{13}$$

Finally, it should be recognized that, as long as the problem (13) is bounded and the solution is non trivial ($\pi > 0$), the multiplier π will have no effect in the solution vector $\underline{d}$ of the optimization problem (13). Therefore, (13) reduces to equation (7):

$$\begin{aligned} min \quad & \underline{d}^T \cdot \underline{d} \\ s.t. \quad & \underline{d} = -N \cdot \underline{\alpha} \\ & \underline{e}^T \cdot \underline{\alpha} = 1 \\ & \alpha_i \geq 0,\ \forall i \in B \end{aligned}$$