Optimization Modeling Languages

Emmanuel Fragnière

HEC, Department of Management, University of Lausanne

BFSH1, 1015 Dorigny-Lausanne, Switzerland

e-mail: `emmanuel.fragniere@hec.unil.ch`

Jacek Gondzio

Department of Mathematics and Statistics, The University of Edinburgh

James Clerk Maxwell Building, King's Buildings, Edinburgh, EH9 3JZ, UK

e-mail: `gondzio@maths.ed.ac.uk`

March 16, 1999, revised in September 24, 1999

Abstract

The access to advanced optimization software needs more and more sophisticated modeling tools. Optimization modeling languages are tools that facilitate the decision making process based on the optimization paradigm. Among the variety of modeling tools, the algebraic modeling languages seem to be the leaders. The optimization modeling tools evolve together with the progress in optimization techniques. Links with solvers are crucial, especially, in nonlinear optimization. New classes of problems such as complementarity, stochastic programming, combinatorial optimization or global optimization problems can be modeled with these tools. In a wider context, interesting developments of modeling tools also emerge from other fields such as chemical engineering or computer science. Indeed, approaches based on object-oriented modeling or constraint programming influence the evolution of optimization modeling languages too. Optimization modeling tools evolve toward fully integrated modeling management systems opening the access to databases, spreadsheets and graphical user interfaces.

Keywords: Algebraic modeling languages, optimization modeling tools, optimization solvers, modeling management systems.

1 Introduction

This handbook shows the rich area of optimization including the algorithms that prove efficient in practice and powerful in solving real-world problems. Numerous problems are solved routinely by optimization tools. Todays state of the art in optimization techniques enables the use of optimization modeling in decision making. The access to advanced optimization software needs equally sophisticated optimization modeling tools. We discuss these issues in this chapter. Many decision problems can benefit from an optimization-type approach. Indeed, the paradigm of optimization seems to be well adapted to needs in different domains such as, economics, engineering, management or physics. In many applications, it is quite natural to formulate the problem as a choice of one solution from a set of possible solutions. Mathematically this can be written as

$$\min \quad z = f(x) \qquad \text{s.t.} \quad x \in X, \tag{1}$$

where f is an objective function defined over a domain X of feasible solutions. Typically, models involve relations defined over many different dimensions that inevitably leads to the increase of the complexity of problem (1). Optimization modeling languages bridge the gap between model formulation and the appropriate solution technique. Optimization modeling languages are computer programs that help to develop and maintain optimization models.

Modeling is a highly iterative process. It includes such steps as problem definition, collection of data, formulating and solving mathematical problem and the analysis of results. The optimization modeling languages support this process. We develop these issues at the beginning of Section 2. To introduce the reader to this subject we first discuss one particular type of modeling tools - the algebraic modeling languages. We illustrate their use with an example from economics that needs nonlinear optimization techniques.

Optimization modeling started from the need of interfacing to the simplex method for linear programming. Modeling tools followed the progress in optimization techniques giving access, for instance, to mixed integer and nonlinear programming solvers. In these cases, more complicated data had to be sent to the solvers. For instance, the nonlinear optimization solver may require the derivatives of the objective and constraints.

Independently, new needs arose on the modeling side, such as the need to include uncertainty in planning models or to handle complementarity constraints in economics. They have become a driving force in the development of optimization modeling tools. Several examples of such needs that led to techniques which are currently being implemented are discussed in Section 4. Summing up, the evolution of optimization modeling languages follows both the progress in optimization techniques and the needs of modelers.

Note that important contributions to optimization tools come from other fields such as chemical engineering or computer science. Specific optimization requirements in these fields led to independent developments in modeling. Typically, chemical engineers build their models of chemical processes from reusable modules, which naturally favors the object-oriented modeling. These issues are discussed in Section 5.

Finally, there are other tools such as databases, model viewers, debuggers, graphical user interfaces that may facilitate modeling process. Today's tendency is to integrate them into the modeling management systems. These issues are discussed in Section 6.

2 Optimization Modeling Languages

In this section, we explain the general process used to model real-life problems and justify the use of modeling languages. (We focus our attention on modeling that by-default involves optimization.) We then present the category of modeling languages that are used most, that is, the algebraic modeling languages.

2.1 The Need of Optimization Modeling Tools

Modeling is a highly iterative process as is illustrated in Figure 1. We distinguish 5 steps in this process. Usually a modeler will go through them repeating certain steps many times.

Recognition and definition of the real problem is often the most difficult step. For such a problem, the modeler can formulate the objective function and the set of constraints so producing an optimization model. Then, the third step, the collection of data is, according to many researchers and consultants, the most demanding task. Typical difficulties at this stage are the lack of data (or its confidentiality) or

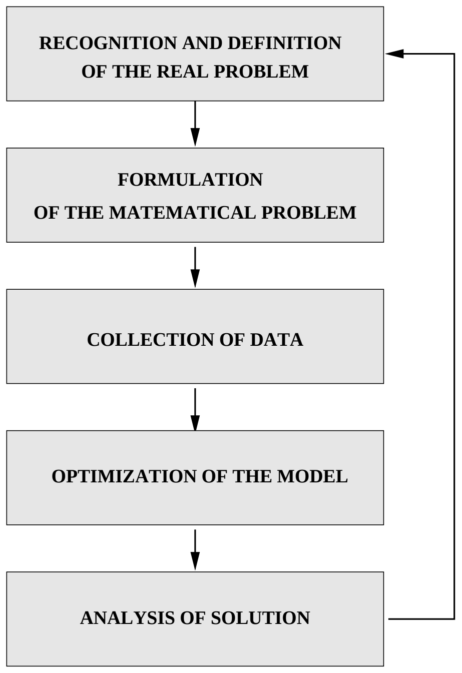

RECOGNITION AND DEFINITION OF THE REAL PROBLEM
FORMULATION OF THE MATEMATICAL PROBLEM
COLLECTION OF DATA
OPTIMIZATION OF THE MODEL
ANALYSIS OF SOLUTION

Figure 1: Modeling Process.

just the opposite, the oversupply of information and the need of selecting the appropriate data from it (e.g. data-mining techniques). The data collected must then be calibrated and validated if the results are to gain credibility. These first three steps of modeling end up with some mathematical model of the real process. The model has now to be translated into the form that is understandable to an appropriate solver. This step needs a computer-based tool - *optimization modeling language*. Indeed, the purpose of the modeling language is to bridge the gap between the "natural" formulation of the problem proposed by the modeler and the input data scheme for the solver. Finally, when the solver finds (hopefully) an optimal solution to the optimization problem, the modeling support system provides the modeler with solutions that can be validated and interpreted. This interpretation may indicate the need to revise the original problem definition. We should point out that in practice the modeling process does not have to follow these 5 steps in the order indicated in Figure 1. Also the learning process about the situation being modeled is often as important as the results obtained from the solver. Indeed, knowledge gathered during this process helps to develop a pertinent expertise.

Optimization modeling languages are computer-based mathematical modeling tools. Their main role is to help the modeler to formulate the model as a mathematical programming problem and to

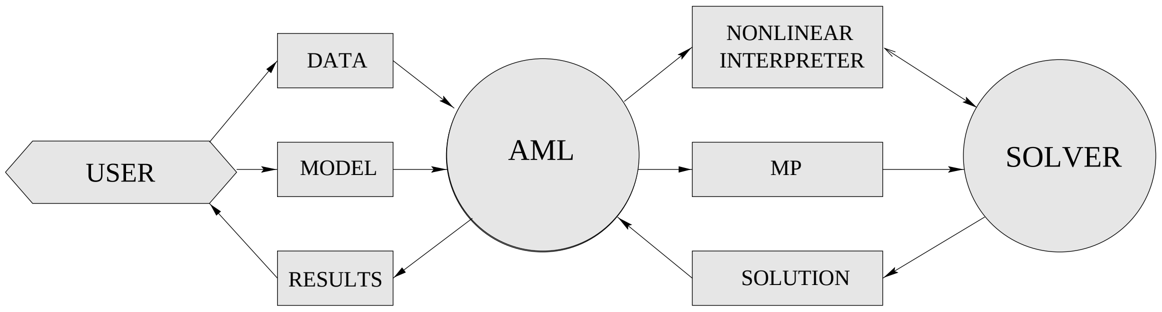


Figure 2: Links between an AML and a solver.

translate it into the form in which optimization code can handle it. Real life problems may involve thousands of constraints and variables and it is impossible to generate by hand each constraint and each variable individually. Rather, the constraints are partitioned into classes, each class having its generic representative. The model builder just provides the mathematical formulation for the constraint or variable representative. This step mimics the mathematical way model builders describe their problem in a clear, concise and efficient way using indices to specify variables and constraints with a common structure.

There is a great variety of modeling languages. In the next section, we address one particular class, algebraic modeling languages.

2.2 Algebraic Modeling Languages

Algebraic Modeling Languages (AMLs for short) enable decision models to be formulated with an algebraic notation. They use a generic model description in a form of a data file. The models developed with AMLs can be easily modified. AMLs are employed within the scheme displayed in Figure 2. The user builds the model and provides the AML with the appropriate data. The AML translates the model into a form that is understandable to a solver and evokes the appropriate solver. In this setting, the solver is seen as a black box. The optimization code may query the AML about an additional information on the problem. For example, nonlinear optimization code may ask for the function values as well as the first and the second derivatives at a given point. Once the solution of the mathematical program is found, it is returned to the AML and the results are reported to the user.

AML enables a modeler to express the problem in an *indexed-based* mathematical form with abstract entities: sets, indices, parameters, variables and constraints. The key notion in the AML is the ability to group conceptually similar entities into a set. Once the entities are grouped in a given set, they can be referenced by indices to the elements of this set. That way the groups of entities (such as variables or constraints) can be represented in a compact way with one algebraic expression. For instance, the mathematical formulation $\sum_{i \in I} X_i$ is translated into the expression `SUM(I, X(I))` in the GAMS modeling language. This leads to a *problem formulation* that is very close to the formulation using algebraic notations. The role of the AML is to expand this compact problem formulation (problem structure and data) into the *problem instantiation*, the one ready to be solved by an appropriate optimization code. This operation is realized within the AML by replicating every entity over the different elements of the set. This is often referred to as a *set-indexing* ability of the AML. The user of an AML can define generic expressions that are indexed over several sets. Set-indexing in such cases involves *compound sets*.

As opposed to programming languages, in which the use of procedural statements dominates over the use of declarative ones, the AMLs use essentially declarative statements. The few procedural statements used in AMLs are `read/write data` and `solve` commands. The need of dealing with more complicated models encouraged AML developers to add more statements typical for programming languages, like e.g., `if-then-else` or commands creating loops. Such commands enable the modeler to write even certain solution algorithms directly in the modeling language. Seminal papers concerning AMLs are [Bisschop and Meeraus, 1982] and [Fourer, 1983]. For a survey of recent conceptual development in AMLs, we refer the reader to [Huerlimann, 1999] and the references therein. The book [Williams, 1993] is a standard reference to modeling.

An important feature of some algebraic modeling languages is the capability of dealing with nonlinear models. To illustrate this point we recall briefly an intertemporal aggregate growth model stemming from work of [Ramsey, 1928]. This model involves three decision variables in each time period $t \in T$: consumption, investment, and capital stock (C_t, I_t, K_t). Consider an economy with a single agent acting as producer, consumer, investor, and saver. Given initial levels of our decision variables (C_0, I_0, K_0), and a Cobb-Douglas production function that is a function of capital and labor $(f(K_t, L_t) = A * K_t^{\beta} L_t^{(1-\beta)})$, and given exogenous labor supplies, L_t, we want to find an optimal level of consumption, investment, and capital stock. Labor L_t has a fixed growth rate, i.e. $L_t = L_0 * (1+g)^t$. To find this optimal level, we maximize a discounted logarithmic utility function under a capital stock constraint and a production constraint. With a fixed growth rate, g, and a utility discount factor, udf, the model is written as:

$$\begin{array}{rll} \max & \sum_t (udf)^t * log(C_t) & \\ \text{s.t.} & K_{t+1} = K_t + I_t, & t \in T \\ & a_t * K_t^{\beta} = C_t + I_t, & t \in T \\ & a_t = A * (L_t)^{1-\beta}, & t \in T \\ & L_t = L_0 * (1+g)^t, & t \in T \\ & C_t, K_t, I_t \geq 0, & t \in T. \end{array} \qquad (2)$$

An extract of the corresponding model written in GAMS algebraic modeling language is displayed in Figure 3.

In this model `C(T)`, `I(T)` and `K(T)` correspond to consumption, investment, and capital stock (C_t, I_t, K_t) at the time t, respectively. `T` is an index (time period) that varies from 1 to 20. The line

```
 CC(T)..   AL(T)*K(T)**B  =E=  C(T) + I(T);
```

corresponds a capacity constraint

$$a_t * K_t^{\beta} = C_t + I_t.$$

We notice that the notation in the AML is close to the mathematical formulation of problem (2). This example well demonstrates the capabilities of todays algebraic modeling languages. These capabilities are the result of many years of development of the optimization modeling languages.

3 Links with Solvers

The growth of modeling languages was motivated by the need to get easy access to modern optimization technology. The word “modern” referred to different solvers in the beginning of sixties (the only mature optimization software at that time was the simplex method for linear programming) and at the end of nineties when the whole variety of optimization algorithms became available. The interface between the modeling language and the solvers evolved together with the development of the new modeling features and the new optimization techniques.

At the very beginning of the practical use of optimization algorithms the needs were limited to formulating and maintaining linear programs. Linear programs have a compact formulation involving only

```
 SET  T         TIME PERIODS       / 1*20/

      TFIRST(T) FIRST PERIOD

      TLAST(T)  LAST PERIOD


 PARAMETERS BETA(T)  DISCOUNT FACTOR

            ALPHA(T) PRODUCTION FUNCTION CONSTANT ;

 ...


 TFIRST(T) = YES$(ORD(T) EQ 1);

 TLAST(T) = YES$(ORD(T) EQ CARD(T));


 BETA(T)  = BET**ORD(T);

 BETA(TLAST) = BETA(TLAST)/(1-BET);

 A     = (C0+I0)/K0**B;

 AL(T) = A*(1+G)**((1-B)*(ORD(T)-1));


 VARIABLES C(T) CONSUMPTION (TRILLION RUPEES PER YEAR)

           I(T) INVESTMENT (TRILLION RUPEES PER YEAR)

           K(T) CAPITAL STOCK (TRILLION RUPEES)

           UTILITY


 EQUATIONS CC(T) CAPACITY CONSTRAINT (TRILLION RUPEES PER YEAR)

           KK(T) CAPITAL BALANCE (TRILLION RUPEES)

           TC(T) TERMINAL CONDITION (PROVIDES FOR POST-TERMINAL GROWTH)

           UTIL  DISCOUNTED LOG OF CONSUMPTION: OBJECTIVE FUNCTION ;


 CC(T)..   AL(T)*K(T)**B  =E=  C(T) + I(T);

 KK(T+1).. K(T+1) =E=  K(T) + I(T);

 TC(TLAST)..G*K(TLAST) =L=  I(TLAST);

 UTIL..    UTILITY =E=  SUM(T, BETA(T)*LOG(C(T)));


 MODEL  RAMSEY   /CC, KK, TC, UTIL/;


 K.LO(T) = K0; C.LO(T) = C0;  I.LO(T) = I0;

 K.FX(TFIRST) = K.LO(TFIRST);

 I.UP(T) = I0*((1+AC)**(ORD(T)-1));


SOLVE RAMSEY MAXIMIZING UTILITY USING NLP;
```

Figure 3: Extract of the Ramsey model written by Alan Manne.

$$\begin{array}{llcrcr}
\max & 120x_1 & + & 80x_2 & & \\
\text{s. to} & 2x_1 & + & x_2 & \leq & 6 \\
 & 7x_1 & + & 8x_2 & \leq & 28 \\
 & x_1 \geq 0 & & x_2 \geq 0. & &
\end{array}$$

```
NAME          Production Problem
ROWS
 N  PROFIT
 L  MATERIAL
 L  WORKTIME
COLUMNS
    PRODUCT1  PROFIT           120.0
    PRODUCT1  MATERIAL           2.0   WORKTIME            7.0
    PRODUCT2  PROFIT            80.0
    PRODUCT2  MATERIAL           1.0   WORKTIME            8.0
RHS
    RHS       MATERIAL           6.0
    RHS       WORKTIME          28.0
ENDATA
```

Figure 4: Production problem: LP formulation and MPS file

the constraint matrix and a few vectors associated with it; hence their generation and storing was fairly obvious. In sixties IBM's linear optimization code, MPSX specified a file format for the matrices and vectors in a linear programming model and this quickly became the standard. It is called the MPS format and is illustrated in Figure 4. It became common to write programs called *matrix generators* to produce the models in the MPS format.

MPS format allowed to standardize the formulation of linear programs. Its design was clearly influenced by the methodology used to solve linear programs at that time. A linear program is represented in MPS format as a sparse matrix accessed column-wise and the vectors providing constraint right-hand-sides and ranges, and variable bounds. MPS has several extensions that followed the progress in optimization. In particular, MPS format can be used today to formulate mixed integer, quadratic or stochastic optimization problems. In Figure 4 we give an example of a simple linear program and the corresponding MPS file. The linear program is a production model in which we maximize the profit from the sales of two products. The products x_1 and x_2 are subject to material and work time constraints.

Matrix generators were data driven tools that could run on mainframes of the 1960's and 1970's: they were in fact specialized languages designed to write the linear program and to generate the solution report. We mention three such languages PDS, MAGEN and OMNI; the last one is used till today, for example, in the modeling of refineries. The data transfer between the matrix generator and the solver involved writing the problem in a form of a text file onto a disc (or tape) and reading it from there. Clearly, such connection was not efficient. In the next step, the matrix generator and the solver were integrated to enable direct passing of column-wise linear program data from the generator to the solver. Until the 1970's such integrated systems were tailored to each specific application. Naturally, the maintenance of such programs and models could not be done by a non-specialist. At the end of 1970's matrix generators gave way to *algebraic modeling languages* [Bisschop and Meeraus, 1982, Fourer, 1983]. The first modeling language, GAMS [Brooke et al., 1992] was developed at the World Bank at the end of 1970's.

Independently of the progress in modeling tools, nonlinear optimization software became available. The need of solving more complicated (nonlinear) optimization problems led Brooke et al. [1985] to including into GAMS the feature to model nonlinear problems and to interfacing GAMS to MINOS - a nonlinear optimization code of Murtagh and Saunders [1983]. Nonlinear solvers need to be able to evaluate problem functions and possibly derivatives at points which only become known when the optimization proceeds. This is quite different from the case with linear problems where all the data of the problem can be transfered to the solver before the solution starts. Nonlinear optimization codes may differ significantly between one another in the type of information they need to solve the problem. Thus the standardization of the interface between algebraic modeling languages and the nonlinear solvers is a complex issue.

As shown in Figure 2, AMLs integrate nonlinear interpreters. These interpreters rely on symbolic and/or automatic differentiation tools to provide the solvers with the required information. Most interpreters included in todays AMLs are based on automatic differentiation. Automatic differentiation [Griewank and Corliss, 1991] exploits the chain rule for computing derivatives. This technique is based on the acyclic expression graph where nodes of the graph correspond to the computation of partial derivatives. The graph can be explored with the *forward method* of automatic differentiation meaning that the nodes are analyzed sequentially. A more sophisticated *backward method* identifies common factors that may be exploited while computing different derivatives. Although this complicates the expression graph, it enables more efficient differentiation especially in the case of larger dimensions. The backward method applies first a forward exploration of the intermediate nodes, and next a backward exploration where derivatives are computed from a triangular set of adjoint equations.

The few last years brought a fast development of new optimization techniques and led to the creation of codes able to solve very large stochastic optimization problems, linear complementarity problems and global optimization problems. To make these algorithms as efficient as possible, one has to pass fairly complex information to the solver and the solver has to query the modeling language while solving the problem. For example, in integer optimization the modeler can provide the solver with the branching priorities, in stochastic optimization an event tree has to be passed to the solver, and the solver may use sampling techniques to examine only selected branches of the event tree. Global optimization algorithms often rely on a good starting point or a hint of how to branch to examine the most interesting local optima. Summing up, to be used in the most efficient way, todays optimization software needs often to access *nonstandard* information. This creates a need for an open interface between the AMLs and the solver. The open interface between AMLs and the solver must be sufficiently flexible to pass the type of information required by recently developed optimization algorithms (e.g. in global optimization) and by the need to model new classes of problems (e.g. complementarity problems). Yet these extensions cannot continue indefinitely because they inevitably complicate the interface. We shall present some of these extensions in Section 4.

Rather than complicating the interface, one can change the modeling process leaving more freedom in building models from smaller elements each communicating on its own with the necessary optimization tool. This gives rise to *modeling management systems* that we shall discuss in Section 6.

4 Optimization Modeling Extensions

In this section, we discuss four extensions of optimization modeling languages that are currently included or are being considered for inclusion in algebraic modeling languages: complementarity problems, combinatorial optimization, stochastic programming and global optimization.

4.1 Complementarity Problems

An important class of problems are those containing complementarity constraints. These constraints imply that in two inequalities that must be satisfied at least one should hold as an equality. This means for instance that either $x_j = 0$ and $g_j(x) \geq 0$, or $x_j \geq 0$ and $g_j(x) = 0$. This particular constraint can be naturally represented by a logical constraint such as $x_j = 0$ or $g_j(x) = 0$ or as a nonlinear constraint such as $x_j g_j(x) = 0$.

Complementarity problems arise typically when one deals with variational inequalities. Such applications appear in computational economics (e.g. computational equilibrium models) as well as in other areas. The possibility of writing complementarity problems directly in modeling languages greatly reduces the work on model formulation; additionally, it simplifies passing appropriate problem to a suitable solver such as MILES [Rutherford, 1995] and PATH [Dirkse and Ferris, 1995]. Syntax extensions to deal with complementarity problems have been added to most algebraic modeling languages.

4.2 Combinatorial Optimization

Many combinatorial optimization problems may be reformulated as integer optimization ones. Since AMLs can invoke mixed integer programming solvers, this offers a possibility of modeling combinatorial optimization problems. However, this way of modeling is certainly not the most efficient one. Combinatorial optimization problems arise, for instance, in scheduling, sequencing, location and assignment. The modeling of these problems needs the logical expressions (`and`, `or`, `not`) as well as procedural operators such as `do-while`, `if-then-else` [Fourer and Gay, 1995]. In this sense, it highlights the need for procedural statements in the modeling environment as noted in Section 2. Some production models, for example, involve fixed costs. It is natural to express the overall production cost through the following procedural construct

$$\texttt{if}\ \ Prod_i \geq MinProd\ \ \texttt{then}\ \ FixedCost_i + VarCost_i \times Prod_i\ \ \texttt{else}\ \ 0.$$

Another example justifies the use of logical operators in a scheduling problem. We assume that a set of different jobs is to be executed on a machine and there are set-up times to be taken into account. The corresponding constraint can be expressed with the following procedural construct

$$\{\forall i, j \in JOBS : i < j\} \qquad Start_j \geq Start_i + SetTime_{ij}\ \ \texttt{or}\ \ Start_i \geq Start_j + SetTime_{ji}.$$

Both these representations are much more natural than the representation with integer variables. For more details, see Chapter XXX of this book.

4.3 Stochastic Programming

In the classical optimization problems, exogenous parameters are deterministic. Typically, in planning applications these parameters can be replaced with random variables. The corresponding branch of mathematical programming is known as *stochastic programming* (see Chapter XXX of this book for a comprehensive introduction into the topic).

Unfortunately, AMLs are not yet well adapted to handle stochastic programming modeling. Although there is a lot of research going on that aims to allow the modeler to write stochastic programming models directly in the AML, it is still not clear which standard will be adopted. The complexity and variety of formulations (see [Gassmann and Ireland, 1995] for a comprehensive taxonomy of stochastic programming problems) are slowing down the integration of specific language items. We should emphasize that the possibility of modeling stochastic programming problems directly in AMLs is not the only issue in this field. Indeed, the size of these problems tend to explode since it grows exponentially with the number of periods and the number of outcomes at each period. Another important point is that these huge problems are structured and they can be solved by specialized optimization techniques only if their structure is exploited. Several powerful codes have been developed such as MSLiP, DECIS and SP/OSL, but they still need to be linked with modeling languages. Recent research in this area shows the need to model multistage stochastic programming problems directly in AMLs. Gassmann and Ireland [1995] note that stochastic programming modeling could largely benefit from specific contributions to AMLs. *Scenarios*, for instance, can be seen like a collection of data. New features must thus be provided, like the implicit declaration of scenarios through the declaration of random parameters. Among other interesting directions, we could mention the recursive definition of stochastic programming problems to express their dynamic character within AMLs [Buchanan et al., 1999]. This brief discussion shows the versatility of approaches proposed. It is thus understandable that AMLs developers hesitate and wait before imposing a language syntax that should engage them in the long run. Consequently, these concepts are still only incorporated into prototype AMLs.

4.4 Global Optimization

To simplify the optimization problem and to render it tractable, the mathematical programming community used to require the objective function and the constraint functions to be convex. However, many real-life problems are inherently nonconvex. Just to simplify our explanations we focus solely on continuous optimization problems (many combinatorial optimization problems can be viewed as global optimization ones, but they were discussed in the earlier section). Nonconvex optimization problems may have multiple local optima. The objective of global optimization is to find a global one (or, more realistically, a ”good” one satisfying the decision maker). The general global optimization problems are usually very difficult and this is certainly one of the reasons why AMLs do not offer global optimization as an option. However, for many classes of global optimization problem (see Chapter XXX), there exist quite efficient solution approaches that work well in practice. They often require some additional knowledge of the problem. For instance, the modeler should be able to provide his own heuristics and search strategies to explore the search spaces or to provide a good starting point. Such an additional knowledge can hardly be quantified in a general way, which is certainly the second obstacle in including global optimization into modeling tools. On the other hand, the progress in global optimization encouraged the search for a specialized modeling environment dedicated to global optimization such as NUMERICA [Van Hentenryck et al., 1997] and LGO [Pintér, 1997].

5 Non-algebraic Modeling Languages

In all the previous sections, particular attention was made to AMLs. However, we should not forget the progress in optimization modeling languages motivated by developments in other areas. We mention two classes of such modeling languages: object-oriented languages that arose from the field of chemical engineering and constrained programming languages that arose from the field of computer science. We begin this section with the concept of structured modeling. This research trend has influenced developers of modeling languages, in particular *non-algebraic* modeling languages.

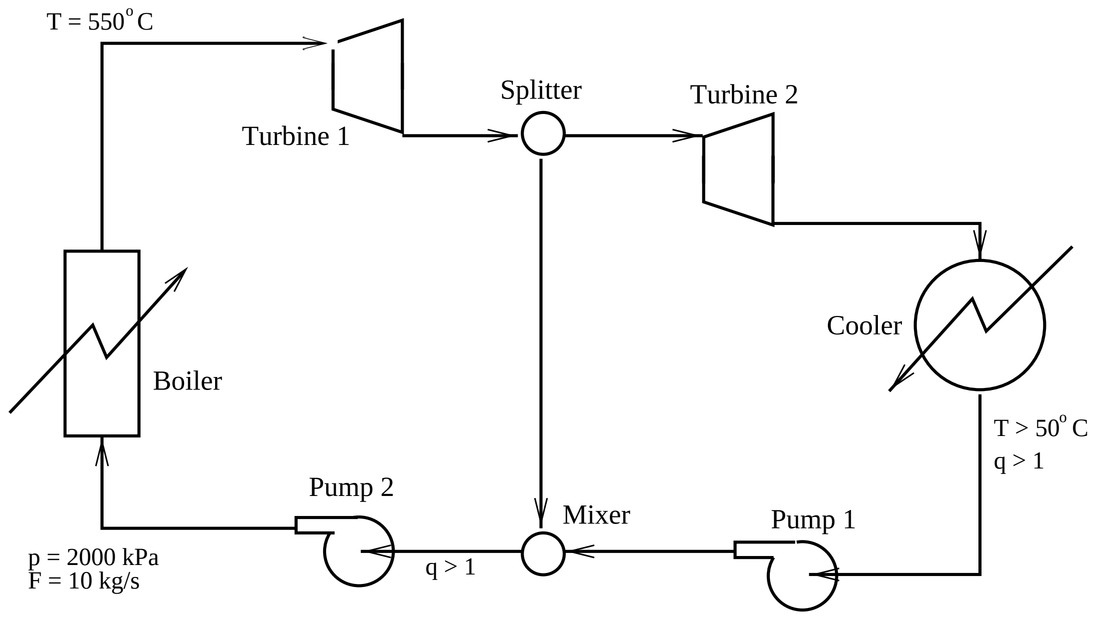


Figure 5: Chemical Model.

5.1 Structured Modeling

The birth of the idea of structured modeling [Geoffrion, 1987] in the 80's was prompted by the limitations of most languages available at this time. Structured modeling proposed a list of criteria intended to lead to good modeling. The main concept underlying structured modeling is that every model can be viewed as a set of smaller models or distinct elements. As stated in structured modeling literature, elements are categorized into five types: *primitive entity, compound entity, attribute, function and test.* Moreover, these elements are organized into a hierarchical structure. A directed acyclic graph is often used to represent the dependencies that exist among them. The development of certain languages have been inspired by ideas found in structured modeling. It is especially the case for those giving the possibility of building models from smaller objects (object-oriented modeling features) as can be seen in the next section.

5.2 Optimization Modeling in Chemical Engineering

The modeling of chemical engineering processes is so particular that optimization modeling tools were developed to handle specifically these kinds of problems. To understand the nature of these problems, we present in Figure 5 a flow diagram containing two turbines, a cooler, two pumps and a boiler that are linked together in a two-turbine power generation cycle.

The first modeling tools developed in this field had some peculiarities inherent to chemical processes. These problems involve certain reusable units. Typically, for simulating the flows that pass in the unit, a set of input variables is specified (energy, temperature, pressure, entropy) that describe the chemical process. Those variables are linked through possibly very complicated physical or chemical laws (equations) and determine the output of the unit. The blocks are linked with each other via specialized syntax. The links use conservation laws to reproduce the flow diagram. From the beginning, chemical engineering modeling languages were linked to simulation software (a Monte-Carlo type simulation). In the 1970's, a new trend emerged called *equation-based* modeling that employed optimization to help decision-making. The problem was viewed as designing a flow process (capacity variables of each unit had to be determined) in conjunction with the optimal use of the system. In economic terms, this means investing in capacity and minimizing the overall operating costs.

The modeling environments developed by chemical engineers were flexible enough to preserve their object-type modeling approach while incorporating optimization. There exist many modeling languages specifically devoted to chemical engineering. They are built of blocks that can use simulation or complex mathematical relations describing the given process (e.g. ordinary and partial differential equations). Two such modeling languages are certainly worth to mention at that point, namely gPROMS [Barton et al., 1991] and ASCEND [Piela et al., 1991]. The model is built of separate objects linked through constraints being usually conservation laws (energy, mass, etc.) and this creates a problem instance that is readable for a solver. Typically, these models lead to nonlinear optimization; moreover, the convexity of constraints can rarely be satisfied hence the models often give rise to global optimization problems.

Chemical engineering optimization modeling languages are not the only ones to build models from *reusable blocks*. In modeling manufacturing processes, simulation tools such as EXTEND use a modeling interface to create discrete event production process from building blocks. Such an interface known as ”drag-n-flop” flowsheeting enables modelers without sophisticated skills to build a complicated model. We could imagine the evolution of optimization modeling languages in a similar direction. Summing up, different research areas come up with different modeling styles, including sometimes the optimization paradigm. Let us discuss another modeling approach called constraint programming.

5.3 Constraint Programming

Consider the following problem. A message "send more money" is to be coded as the addition:

```
  code(   S E N D )

+ code(   M O R E )

------------------

= code( M O N E Y ).
```

The question is to choose distinct digits associated with all 8 letters used in the message in such a way that the arithmetical operation gives correct result. This problem can be modeled as an integer programming problem. There exist different formulations of it, but each of them involves about a hundred integer variables, which renders the associated integer program quite difficult. Strictly speaking, we should say that these integer programming formulations are difficult for a classical integer optimization approaches, e.g. for branch and bound solvers.

The logical analysis of this problem makes it possible to guess a feasible solution. For example, we note that neither S nor M can be zero. Next, we find that the only possible code of M is equal to 1. (We could never produce $M = 2$ or larger on a fifth position from adding digits of thousand $S + M$). Next, we realize that as M is 1, S can only be 9. We also realize that O has to be coded with 0. By analyzing the addition of digits corresponding to hundreds we find that since O is coded with zero and E and N have to be different, they must satisfy $N = E + 1$. By continuing such a logical analysis of constraints, we eventually come to a feasible solution of the coding problem: $E = 5, N = 6, D = 7, R = 8$ and $Y = 2$.

This example shows the obvious limits of classical integer programming approach as well as highlights the need to incorporate logical information into the optimization algorithm. The main difficulty in modeling this problem consists in a constraint that requires all variables (corresponding to distinct letters) be different. This problem can be modeled in a compact form if a logical predicate `All_different` is used. Unfortunately, the classical modeling languages do not offer such predicates.

Two main techniques used in constraint logic programming are: *consistency techniques* and *local propagation*. The idea behind *consistency techniques* is to reduce the assignable domains of variables. *Local propagation* means solving an equation in one variable and propagating the value of that variable to all other equations, which may then become solvable. In constraint logic programming indices can be variables and logical predicates can be used. Moreover, every constraint is treated as a separate object. The objects communicate with each other trying to restrict their feasible domains and coming eventually up with a feasible solution (or at least a limited number of possible solutions). Clearly, each object can solve auxiliary integer optimization problems; for this an access to a fast integer optimization solver is crucial. The most natural implementation of constraint logic programming takes full advantage of the object-oriented design. Constraint logic programming is a new successful area of optimization. The applications of this technique have increased very quickly during the last few years. The technique was applied to solve a number of problems that could not be solved with the classical integer optimization approach.

Constraint logic programming [see Jaffar and Maher, 1994] is implemented in ILOG software. It has been applied, for instance, to solve a complicated logistics problem of scheduling airplanes to parking slots. This optimization problem relies on sophisticated information system and combines modeling, difficult on-line planning and advanced optimization technology. The problem has been solved and fully automated decision support system combining graphical user interface and real-time access to a database has been implemented. ILOG developed this system primarily for Orly, one of Paris international airports. Today, the same solution is used by a number of airports through the world, which is certainly a spectacular success of the constraint logic programming. This approach shows a lot of promise in solving different kinds of scheduling problems.

6 Other Modeling Aid Tools

There is a crucial need for integrated systems that include optimization modeling languages. As shown in Figure 1, the analysis of results is an important element of modeling process.

Most modeling languages include or make use of different kinds of tools such as the following ones: debuggers to check the model formulation syntax, high-level data checking procedures, model analyzers, text editors for model or report writing, database systems and Graphical User Interface (GUI).

In this section, we focus on two such tools: model analyzers and database systems with a particular emphasis on Geographical Information Systems (GIS). We conclude this section by commenting on the recent wide dissemination of spreadsheet modeling which, in a sense, is a modeling language that emerged from a database environment.

6.1 Model Analyzers

Programming language compilers include powerful user-friendly debuggers. The needs in modeling languages are similar. However, syntax errors in the model are not the only difficulty the user may encounter. Another modeling concern is checking the feasibility of the model and detecting the source of possible infeasibility. There exist tools such as ANALYZE [Greenberg, 1983] and MPROBE especially designed to analyze mathematical programs written with optimization modeling languages. ANALYZE was designed to analyze linear programs and to be used with matrix generators. MPROBE is a more recent tool inspired by ANALYZE that is devoted to nonlinear modeling. For instance, it offers useful features such as function shape analysis, constraint effectiveness checking, objective effect analysis, variables and constraints sorting.

The ability to view a mathematical programming model has also attracted a lot of attention. To be able to represent different dependencies among elements of linear programs one can use different types of graphs, e.g., an activity-constraint graph or a value-dependency graph, or an index-dependency graph. The visualization techniques can help the modeler to build the model and to correct mistakes. Graph representation may sometimes be a more efficient way to assist the modeler than using uniquely algebraic expressions [see Jones, 1996]. Certain integrated modeling environments (e.g., MIMI) offer such visual supports in their modeling environment.

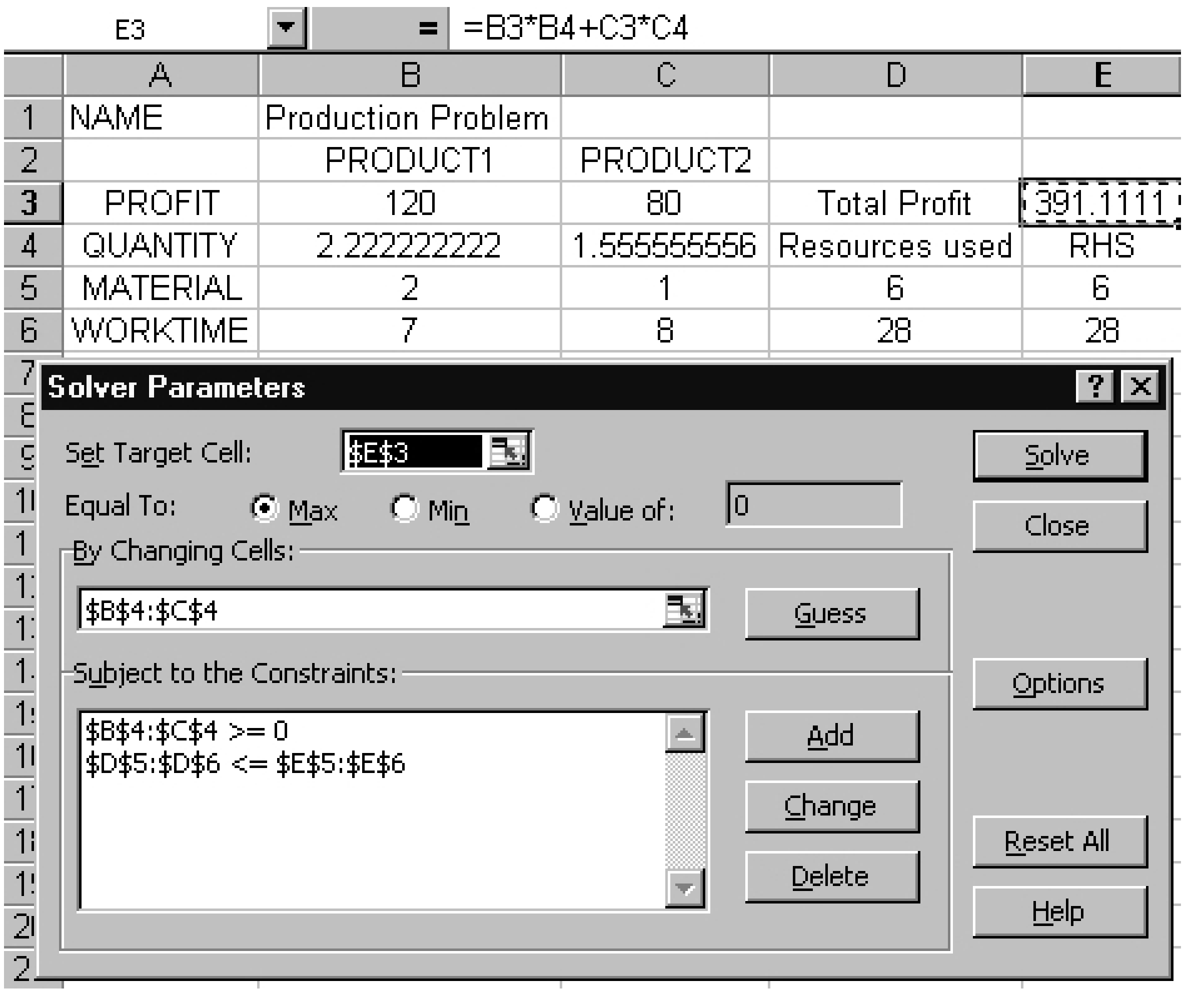
6.2 Links with Databases and GIS Extensions

A current trend is to combine optimization modeling languages and databases systems. An AML-style model file is an ASCII file containing definitions of sets and parameters in the form of scalars, vectors and tables. Each line is either a comment, a set definition, a parameter definition or a continuation line. An import-export filter can translate the data into a database format. The interfacing between a user, without advanced modeling skills, and the language can thus be facilitated by the development of links between database management systems and the model and data files. These capabilities are particularly well suited to handle data. Because data appears to be more volatile than the problem structure, most modeling languages designers insist on data and model structure being separated [Huerlimann, 1999]; hence data management should benefit from appropriate tools. Data storage of the optimization modeling language should respect the structure query language standards thus becoming an element of a wider database environment. This enables an easy exchange of information between modeling, language database and other (external) databases. AMPL PLUS is an example of modeling language which enables data queries to be made with MS-Windows databases or spreadsheets, thus allowing users to produce customized modeling products.

The technology of geographical information systems (GIS), has shown a very rapid development in recent years. A GIS is a database in which the information is associated with a spatial location indicator. Most GIS environments provide standard tools for creation and visualization of geographic data. The data management is handled using the relational database paradigm with its standard structured query language and its extension to incorporate spatial operators. These tools can be particularly useful for handling network modeling (e.g. routing, shortest path) or environmental analysis. The possibility to display the solutions resulting from the use of the model together with maps that could be further analyzed with GIS tools, give to decision makers very appealing insights. Moreover, results obtained from optimization may become an input to some other modeling tools (e.g. simulation) also linked with the GIS environment. An example of merging integer programming, network optimization and GIS is given in [Camm et al., 1997].

6.3 Spreadsheet Modeling

Compared with what was presented in previous sections, spreadsheet modeling could be considered as a very simple modeling approach. Figure 6 shows an extract of an optimization problem written in EXCEL. Spreadsheets provide a set of simple functions such as the scalar product and `copy and paste` operations, which makes modeling possible. Using copying and pasting to replicate equations and variables does not ensure that the model formulation will remain compact, and this limits the size of problems that can be modeled. It is the responsibility of the modeler to organize the cells of the spreadsheet in the most convenient way. Models can, for example, be made of data spread across the spreadsheet. This may lead to creating so-called "spaghetti models" with a danger that modeling errors can be difficult to detect. Indeed spreadsheet modeling has the main disadvantage that the mathematical structure of the model cannot be visualized.



E3 = =B3*B4+C3*C4

	A	B	C	D	E
1	NAME	Production Problem			
2		PRODUCT1	PRODUCT2		
3	PROFIT	120	80	Total Profit	391.1111
4	QUANTITY	2.222222222	1.555555556	Resources used	RHS
5	MATERIAL	2	1	6	6
6	WORKTIME	7	8	28	28

Solver Parameters

Set Target Cell: $E$3

Equal To: [● Max] Min Value of: 0

By Changing Cells: $B$4:$C$4

Subject to the Constraints:

$B$4:$C$4 >= 0
$D$5:$D$6 <= $E$5:$E$6

Figure 6: Excel Modeling

The solver interface of the spreadsheet is quite simple to use. *The objective cell* is the variable that will be optimized and *changing cells* correspond to the decision variables. Concerning the definition of constraints, here, the logic is to define a column of cells where the use of resources is computed for a given solution (value entered in the changing cells). These values are then put into relation with the right hand side vector and the user needs to choose for each of them the type of the constraint (the sense of inequality). Cells may compute nonlinear function values too; this opens the possibility to model nonlinear problems. As the simple example in Figure 6 shows, spreadsheet modeling is not the most convenient modeling tool. However, it is sufficient to develop a small prototype. It is certainly a controversial statement but we wonder if due to millions of licenses sold across the world, spreadsheet modeling of optimization problems is not the most widely used modeling tool today. EXCEL contains an optimization solver [Fylstra et al., 1998] which provides most PC owners with an access to optimization techniques.

On this point, we should mention that many management science courses are based on spreadsheet modeling enabling the teaching of optimization with a user-friendly environment. As stated in [Erkut, 1998], optimization courses have often been devoted to large scale modeling. In many companies there is a need to solve small optimization problems that can be handled directly by managers and probably do not require the presence of specialized consultants. This can give meaningful business insights. Students seems to take readily to this way of learning optimization modeling. Also they can incorporate this knowledge into simple programs in visual basic and develop quite impressive decision support systems.

7 Conclusions

The ultimate aim of optimization-based modeling is to help decision makers to find the *best* solution to their problem. It is often said that decision modeling systems have to be designed in a way that allows modelers to formulate their models in the most natural way. However, depending on the background of the modeler, "natural" can mean very different things. For instance, chemical engineers use often graphical representation to model continuous and batch processes. Economists are used to expressing their models with algebraic notations. This observation on different modeling styles in different professional groups extends to students; new generation of modelers usually follow the modeling style of their teachers. The modern way of teaching operations research by using specific modeling languages will certainly give rise to another "natural" way of writing models.

This paper gives an overview of tools that facilitate the decision making process based on the optimization paradigm. It seems to us that the evolution of optimization modeling languages is driven by two factors: the need to formulate particular mathematical problems and the availability of new solution techniques. We have paid special attention in this chapter to the historical development and the current state of algebraic modeling languages, because these tools seem to be widely accepted by the optimization research community.

New developments in optimization modeling languages do not come uniquely from the research of the optimization community. There are important contributions from different research fields (e.g. chemical engineering and computer science).

Finally, we note an increasing influence of the market needs on the developments in optimization modeling languages. Optimization-based modeling has to respond the new requirements imposed by the fast development of information technology, especially in the field of databases.

Acknowledgments

We are grateful to our colleagues: Christian Condevaux-Lanloy, Tony Huerlimann, Ken McKinnon and Bill Morton for many helpful remarks.

The research of the first author was supported by the Fonds National de la Recherche Scientifique Suisse, grant #1214-049696.96/1. The research of the second author was supported by the Engineering and Physical Sciences Research Council of UK, EPSRC grant GR/M68169.

References

P.I. Barton, E.M.B. Smith, and C.C. Pantelides. Combined discrete/continous process modelling using gPROMS. Centre for Process Systems Engineering, Imperial College of Science, Technology and Medicine, London SW7 2AZ, United Kingdom, Prepared for presentation at the 1991 AIChE Annual Meeting, 1991.

J. Bisschop and A. Meeraus. On the development of the general algebraic modeling system in a strategic planning environment. *Mathematical Programming Study*, 20:1–29, 1982.

A. Brooke, A. Drud, and A. Meeraus. High level modeling systems and nonlinear programming. In P. T. Boggs, R. H. Byrd, and R. B. Schnabel, editors, *Numerical Optimization 1984*. SIAM, Philadelphia, 1985.

A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: A User's Guide.* The Scientific Press, Redwood City, California, 1992.

C.S. Buchanan, K.I.M. McKinnon, and G.K. Skondras. The recursive definition of stochastic linear programming problems within an algebraic modeling language. To appear in Annals of Operations Research, 1999.

J.D. Camm, T.E. Chorman, F.A. Dill, J.R. Evans, D.J. Sweeney, and G.W. Wegryn. Blending OR/MS, judgment, and GIS: restructuring P&G's supply chain. *Interfaces*, 27(1):128–142, 1997.

S.P. Dirkse and M.C. Ferris. The PATH solver: A non-monotone stabilization scheme for mixed complementarity problems. *Optimization Methods and Software*, 5:123–156, 1995.

E. Erkut. How to "EXCEL" in teaching management science. OR/MS Today, October, 1998.

R. Fourer. Modeling languages versus matrix generators for linear programming. *ACM Transaction on Mathematical Software*, 9:143–183, 1983.

R. Fourer and D. Gay. Expressing special structures in an algebraic modeling language for mathematical programming. *ORSA Journal on Computing*, 7:166–190, 1995.

D. Fylstra, L. Lasdon, A. Warren, and J. Watson. Design and use of the microsoft excel solvers. To appear in Interfaces, 1998.

H.I. Gassmann and A.M. Ireland. Scenario formulation in an algebraic modelling language. *Annals of Operations Research*, 59:45–75, 1995.

A.M. Geoffrion. An introduction to structured modeling. *Management Science*, 33:547–588, 1987.

H.J. Greenberg. A functional description of ANALYZE: a computer-assisted analysis system for linear programming models. *ACM Transactions on Mathematical Software*, 9:18–56, 1983.

A. Griewank and G.F. Corliss. *Automatic Differentiation of Algorithms: Theory, Implementation, and Application.* SIAM, Philadelphia, PA, 1991.

T. Huerlimann. *Modeling and Optimization.* Kluwer Academic Publishers (to appear), 1999.

J. Jaffar and M.J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20: 503–581, 1994.

C.V. Jones. *Visualization and Optimization.* Kluwer Academic Publishers, Boston, 1996.

B.A. Murtagh and M.A. Saunders. MINOS 5.0 User's guide. Report SOL 83-20, Department of Operations Research, Stanford University, 1983.

P. Piela, T.G. Epperly, and K.M. Westerberg. Ascend: an object-oriented computer environment for modeling and analysis: The modeling language. *Computers and Chemical Engineering*, 15:53–72, 1991.

J.D. Pintér. *LGO - A Model Development System for Continuous Global Optimization, User's Guide.* Halifax, NS, Canada, 1997.

F.P. Ramsey. A mathematical theory of saving. *Economics Journal*, December 1928.

T.F. Rutherford. Extensions of GAMS for complementarity problems arising in applied economics. *Journal of Economic Dynamics and Control*, 19(8):1299–1324, 1995.

P. Van Hentenryck, L. Michel, and Y. Deville. *Numerica A Modeling Language for Global Optimization.* The MIT Press, Cambridge, 1997.

H. P. Williams. *Model Building in Mathematical Programming.* John Wiley & Sons, Chichester, 1993.

A Web Sites of Optimization Modeling Languages

AIMMS: `http://www.aimms.com/`

AMPL: `http://www.ampl.com/ampl/`

ASCEND: `http://www.cs.cmu.edu/afs/cs.cmu.edu/project/ascend/home/`

GAMS: `http://www.gams.com/`

gPROMS: `http://www.psenterprise.com/gPROMS/index.html`

ILOG: `http://www.ilog.com/`

LGO: `http://www.tuns.ca/~pinter/`

LINGO: `http://www.lindo.com/lingo.html`

LPL: `ftp://www-iiuf.unifr.ch/pub/lpl`

MAGIC: `http://www.magic-MP.com/`

MPL: `http://www.maximal-usa.com/mplmain.html`

MPROBE: `http://www.sce.carleton.ca/faculty/chinneck/mprobe.html`

OMNI: `http://www.haverly.com/omni.htm`

XPRESS-MP: `http://www.dash.co.uk/`