Chapter 8

GENERAL ALGEBRAIC MODELING SYSTEM (GAMS)

Michael R. Bussieck & Alex Meeraus
GAMS Development Corporation
Washington D.C., U.S.A.
{MBussieck,AMeeraus}@gams.com

Abstract In this chapter we will categorize the development of mathematical programming tools into three major phases and will define three major categories of models and their use. This classification will be used to describe some of the features found in `GAMS` to support different application environments. Selected language features, and examples of external functions and privacy and security issues are discussed and separate annexes. Finally, we will list the inputs to a problem that has been provided to all the authors by the editor of this volume.

Introduction

Today, algebraic modeling languages are widely accepted as the best way to represent and solve mathematical programming problems. Their main distinguishing features are the use of relational algebra and the ability to provide partial derivatives on multidimensional, very large and sparse structures. In this chapter we will describe some of the origins of `GAMS` and provide background information that shaped early design decisions. Initial Research and Development of `GAMS` was funded by the International Bank for Reconstruction and Development, usually referred to as The World Bank, through the Bank's Research Committee (RPO 671-58, RPO 673-06) and carried out at the Development Research Center in Washington DC. Since 1987, R&D has been funded by *GAMS Development Corporation*.

The system was developed in close cooperation of mathematical economists who were and still are an important group of `GAMS` users. The synergy between economics, computer science and operations research was the most important success factor in the development of the system. Mathematical Programming and economics theory are closely intertwined. The Nobel Prize in Economics awarded to Leonid Kantorovich and Tjalling Koopmans in 1975 for their "contribution to the theory of optimal allocation of resources" was really a prize in mathematical programming. Other Nobel laureates like Kenneth Arrow in 1972, Wassily Leontief in 1973, and Harry Markowitz in 1990 are well known names in math programming. Another early example of this synergy is the use of LP in refining operations, which was started by Alan Manne, an economist, with his book on *Scheduling of Petroleum Refinery Operations* in 1956 [148].

The origins of linear programming algorithms all go back to George Dantzig's early work in the 1940s and 1950s [44, 45]. Computing technology and algorithmic theory had developed at a rapid pace. Thirty years later, we could solve problems of practical size and complexity that allowed us to test of the economic theory on real life problems. The research agenda at the World Bank in the 1970s and 1980s created the perfect environment to bring different disciplines together to apply mathematical programming to research and operational questions in Economic Development.

8.1 Background and Motivation

From the very beginning, the driving force behind the development of the General Algebraic Modeling System (GAMS) [26] has been the users of mathematical programming who believed in optimization as a the powerful and elegant framework for solving real life problems in the sciences and engineering. At the same time, these users were frustrated with the high cost, skill requirements, and overall low reliability of applying optimization tools. Most of our initiatives and support for new development came from the worlds of economics, finance, and chemical engineering. These disciplines find it natural to view and understand the world and its behavior as a mathematical program.

GAMS's impetus for development arose out of the frustrating experiences of a large economic modeling group at the World Bank. In hindsight, one may call it a historical accident that in the 1970s mathematical economists and statisticians were assembled to address problems of development. They used the best techniques available at the time to solve multisectoral economy-wide models and large simulation and optimization models in agriculture, steel, fertilizer, power, water use, and other sectors. Although the group produced impressive research, initial successes were difficult to reproduce outside their well functioning research environment. The existing techniques to construct, manipulate, and solve such models required several manual, time-consuming, and error-prone translations into the different, problem-specific representations required by each solution method. During seminar presentations, modelers had to defend the existing versions of their models, sometimes quite irrationally, because the time and money needed to make proposed changes were prohibitive.

Their models just could not be moved to other environments, because special programming knowledge was needed, and data formats and solution methods were not portable.

The idea of an algebraic approach to represent, manipulate, and solve large-scale mathematical models fused old and new paradigms into a consistent and computationally tractable system. Using matrix generators (see appendix *GAMS versus Fortran Matrix Generators*) for linear programs taught us the importance of naming rows and columns in a consistent manner. The connection to the emerging *relational data* model became evident. Painful experience using traditional programming languages to manage those name spaces naturally lead one to think in terms of sets and tuples, and this led to the relational data model. Combining multidimensional algebraic notation with the relational data model was the obvious answer. Compiler writing techniques were by now widespread, and languages like GAMS could be implemented relatively quickly. However, translating this rigorous mathematical representation into the algorithm specific format required the computation of partial derivatives on very large systems. In the 1970s, TRW developed a system called PROSE [203] that took the ideas of chemical engineers to compute *point derivatives* that were exact derivatives at a given point, and to embed them in a consistent, fortran-style calculus modeling language. The resulting system allowed the user to use automatically generated exact first and second order derivatives. This was a pioneering system and an important demonstration of a concept. However, in our opinion PROSE had a number of shortcomings: it could not handle large systems, problem representation was tied to an array-type data structure that required address calculations, and the system did not provide access to state-of-the art solution methods. From linear programming, we learned that exploitation of sparsity was the key to solve large problems. Thus, the final piece of the puzzle was the use of *sparse data structures*.

With all pieces in place, all we had to do was adopt the techniques to fit into one consistent framework and make it work for large problems.

8.2 Design Goals and Changing Focus

The original and still valid goal is to improve the model builder's productivity, reduce costs, and improve reliability and overall credibility of the modeling process. To achieve this, we established the following key principles to guide the GAMS development:

- The problem representation is independent of the solution method.
- The data representation follows the relational data model.
- The problem and data representations are independent of computing platforms.

- The problem and data representations are independent of user interfaces.

- Optimization methods will fail, and systems have to be designed to be fail-safe.

Another way to express these principles is to think in terms of layers of representations and capabilities that have clearly defined interfaces and functions. The oldest and most basic layer is the *solver layer* or implementation of a specific algorithm. Above the solver is the model layer, expressed in an algebraic modeling language. The *modeling layer* translates the mathematical representation into a computational structure required by a specific solution method and provides various services such as function and derivative evaluations and error recovery. Above the modeling layer is the *application or domain layer*, which is highly context sensitive and has knowledge about the problem to be solved and the kind of user interacting with the system.

It is instructive to put the development of modeling systems into some historic perspective and see how the focus and technical constraints have changed in the last 30 years. We can observe three major phases that shift the emphasis from computational issues to modeling issues and finally the application or the real problems. Each phase defined one of the main system layers discussed above. The dominant constraints in the first phase were the *computational limits* of our algorithms. Problem representation had to abide by algorithmic convenience, centralized expert groups managed large, expensive and long lasting projects and end users were effectively left out. The second phase has the *model* in focus. This volume is about languages and systems supporting this stage. Applications are limited by modeling skill, project groups are much smaller and decentralized, the computational cost are low and the users are involved in the design of the application. Applications are designed to be independent of computing platforms and frequently operate in a client-server environment.

We believe that we are entering a third phase which has the *application* as its focus and the optimization model is just one of many analytic tools that help making better decisions. The users are often completely unaware of any optimization model or use a mental model that is different from the actual model to solved by optimization techniques. User interfaces are build with off-the-shelf components and frequently change to adjust to evolving environments and new computing technologies. As with databases, modeling components have a much longer life than user interfaces. We have observed cases where the model has remained basically unchanged over many years, whereas the computing environments and user interfaces have changed several times. The solvers used to solve the models have changed, the computing platforms have changed, the user interfaces have changed and the overall performance of the model has changed without any change in the model representation.

8.3 A User's View of Modeling Languages

Today's modeling systems have achieved an interface between the solver and the model world. They also attempt to provide solutions for interfacing models and applications. These attempts focus mainly on data exchange capabilities, source code generation, and access to model object libraries. None of these suggested solutions has yet been widely accepted. This area will remain very active for the next decade, and may ultimately decide which optimization technology becomes the most widely used in tomorrow's applications. In this section we describe requirements for a modeling system for different classes of models and users. We segment the world of models into three categories:

- academic research models,
- domain expert models, and
- black box models.

We also illustrate various concepts, currently implemented in the `GAMS` system, that support the modeling process from each user's perspective. Each section starts by describing the typical environment and user supported by some actual examples. Furthermore, we discuss how particular `GAMS` features, that are not necessarily found in other systems, might help these users.

8.3.1 Academic Research Models

Academic research models implement mathematical programming problems often found in academic publications. The model source consists almost exclusively of highly complex algebra that defines the variables and equations. Frequently, these kinds of models, that are benchmarked using a given set of publicly available test instances, support statements made in research papers. An impressive library of test problems from Operations Research can be found in the OR Library [9]. In addition to obvious selection criterion such as personal taste for syntax and the development environment, the most important criteria for selecting a modeling system for these kind of models are the availability of the user's platform of choice and the availability of the model type and appropriate solver.

`GAMS` offers mixed integer linear and nonlinear programming model types, mixed complementarity problem (MCP) type, and mathematical programs with equilibrium constraints (MPEC). MCP and MPEC play an important role in economic modeling. In addition, `GAMS` can represent multi-level stochastic optimization problems and models with cone constraints. Furthermore, application languages like MPSGE for general equilibrium models and LOGMIP for disjunctive programming provide access to optimization techniques to researchers outside our narrow field of mathematical programming. To seamlessly support new model types or solvers, the GAMS language offers sufficient flexibility to represent new ideas within the current syntax (although certain constructs may look somewhat awkward). The idea is to first understand each new concept's model, algorithms, and most important application areas, before adding syntax from that concept to an increasingly cluttered language.

One of GAMS' early design criteria was to be both platform and solver independent. GAMS users immediately benefit from hardware and operating system improvements, and improvements on the solver side. Throughout its evolution, the GAMS system has seen the rise and fall of quite a few platforms and mathematical programming solvers. We anticipate a similar pattern in the future. The solver suite of the actual GAMS system includes about 20 supported solvers and a handful of contributed plug-and-play solvers. Each supported GAMS solver complies to a strictly defined solver interface, making the seamless exchange of solvers possible for a GAMS modeler. Common solver attributes (*e.g.*, maximum resource time) can be set through general GAMS options, and solver-specific options can be set through option files. Linking a solver to GAMS means complying with this strict interface, which can be a challenging task[1]. There are easier ways to connect to a modeling system, especially for research codes. For example, the AMPL (see Chapter 7) input/output library is geared towards convenience for the algorithm provider. We find many research codes hooked up to AMPL (*e.g.*, most NEOS solvers have an AMPL interface). Making these codes available for GAMS models would allow researchers to benchmark their GAMS models against a larger set of solvers, without reformulating their model in different modeling languages. Rather than compromising the strict GAMS solver interface, the GAMS system comes with a translation *solver* called CONVERT [29].

Modeling languages have a rich syntax that is usually based on sets and indexed variables, equations, and parameters. This syntax, the corresponding structure in the model, and the data is very useful to the model developer. However, this structure is not used by the solvers. Most solvers see the world as consisting of a list of variables (X_1 to X_n), a list of equations or constraints (E_1 to E_m), and the relationship between these variables and equations, as represented in some form. It is therefore acceptable for a translator to remove the structure, as long as the model as seen by the solver remains unchanged.

The translation solver CONVERT transforms models into a very simple internal scalar format. This internal format can then be written out in many different formats.

With GAMS as output format, the scalar model consists of

[1]The title of the corresponding documentation says it all: *Linking your Solver to GAMS. The Complete Notes. Don't Panic!!*

```
Set I Products         /P1*P2/
    J Cutting Patterns /C1*C2/;

Parameter c(J)    cost of raw material                 /C1 1, C2 1/
          cc(J)   cost of change-over of knives        /C1 0.1, C2 0.2/
          b(I)    width of product roll-type I         /P1 460, P2 570/
          nord(I) number of orders of product type I   /P1 8, P2 7/
          Bmax    width of raw paper roll              /1900/
          Delta   tolerance for width                  / 200/
          Nmax    max number of products in cut        /   5/
          bigM    max number of repeats of any pattern /  15/;

Variable  y(J)    cutting pattern
          m(J)    number of repeats of pattern j
          n(I,J)  number of products I produced in cut J
          obj     objective variable;

Binary Variable y; Integer Variable m, n;

Equation defobj, max_width(J), min_width(J), max_n_sum(J),
                 min_order(I), cut_exist(J), no_cut(J);

defobj..         sum(j, c[j]*m[j] + cc[j]*y[j]) =e= obj;
max_width(j)..   sum(i, b[i]*n[i,j])            =l= Bmax;
min_width(j)..   sum(i, b[i]*n[i,j]) + Delta    =g= Bmax;
max_n_sum(j)..   sum(i, n[i,j])                 =l= Nmax;
min_order(i)..   sum(j, m[j]*n[i,j])            =g= nord[i];
cut_exist(j)..   y[j]                           =l= m[j];
no_cut(j)..      m[j]                           =l= bigM*y[j];

m.up[j] =  bigM; n.up[i,j] = nmax;

model trimloss /all/;
solve trimloss minimize obj using minlp;
```

Figure 8.3.1. The orginial GAMS model for trim loss optimization

- declarations of the variables, with extra declarations for the subsets of positive, integer, or binary variables;
- declarations of the equations;
- the symbolic form of these equations; and
- assignment statements for non default bounds and initial values.

All operations involving sets are unrolled, and all expressions involving parameters are evaluated and replaced by their numerical values. Since there are no sets or indexed parameters in the scalar models, most of the differences between modeling systems have disappeared. Therefore, the GAMS format can be transformed easily into the format of another language. In AMPL, for example,

```
*  MINLP written by GAMS Convert

Variables  b1,b2,i3,i4,i5,i6,i7,i8,x9;

Binary Variables b1,b2;
Integer Variables i3,i4,i5,i6,i7,i8;

Equations  e1,e2,e3,e4,e5,e6,e7,e8,e9,e10,
           e11,e12,e13;

e1..  0.1*b1 + 0.2*b2 + i3 + i4 - x9 =E= 0;
e2..  460*i5 + 570*i7 =L= 1900;
e3..  460*i6 + 570*i8 =L= 1900;
e4..  460*i5 + 570*i7 =G= 1700;
e5..  460*i6 + 570*i8 =G= 1700;
e6..  i5 + i7 =L= 5;
e7..  i6 + i8 =L= 5;
e8..  i3*i5 + i4*i6 =G= 8;
e9..  i3*i7 + i4*i8 =G= 7;
e10.. b1 - i3 =L= 0;
e11.. b2 - i4 =L= 0;
e12.. - 15*b1 + i3 =L= 0;
e13.. - 15*b2 + i4 =L= 0;

* set non default bounds
i3.up = 15; i4.up = 15; i5.up = 5;
i6.up = 5;  i7.up = 5;  i8.up = 5;

Model m / all /;
Solve m using MINLP minimizing x9;
```

```
#  MINLP written by GAMS Convert

var b1 binary;
var b2 binary;
var i3 integer >= 0, <= 15;
var i4 integer >= 0, <= 15;
var i5 integer >= 0, <= 5;
var i6 integer >= 0, <= 5;
var i7 integer >= 0, <= 5;
var i8 integer >= 0, <= 5;


minimize obj: 0.1*b1 + 0.2*b2 + i3 + i4;
subject to

e2:  460*i5 + 570*i7 <= 1900;
e3:  460*i6 + 570*i8 <= 1900;
e4:  460*i5 + 570*i7 >= 1700;
e5:  460*i6 + 570*i8 >= 1700;
e6:  i5 + i7 <= 5;
e7:  i6 + i8 <= 5;
e8:  i3*i5 + i4*i6 >= 8;
e9:  i3*i7 + i4*i8 >= 7;
e10: b1 - i3 <= 0;
e11: b2 - i4 <= 0;
e12: - 15*b1 + i3 <= 0;
e13: - 15*b2 + i4 <= 0;
```

Figure 8.3.2. The Scalar GAMS and AMPL versions of the trim loss model

the keyword *var* is used instead of *Variable*, bounds and initial values are written using a different format, the equation declarations are missing, the equation definitions start with *subject to Ei* instead of just *Ei..*, and a few operators are named differently. In a few cases, a model cannot be translated into a particular language because special functions (*e.g.*, errorf) or variable types (*e.g.*, SemiCont) are not available in that language. GAMS models have an objective variable rather than an objective function. If there is one defining equation for that variable, CONVERT will eliminate the objective variable and will introduce a real objective for formats that support objective functions (*e.g.*, AMPL).

Figure 8.3.1 represents a user-written GAMS model for trim loss minimization, and Figure 8.3.2 represents the scalar models GAMS and AMPL versions.

Currently, CONVERT translates into AMPL [67], BARON [179], LINGO [190], LGO [166], MINOPT [194], and the usual MPS and LP formats. Details about CONVERT can found in the GAMS Solver Guide [72]. There is a translation server to support those who do not have access to a GAMS system. The interface to this server is email-based. An email to gms2xx@gamsworld.org with a GAMS model attached and the requested language name (*e.g.*, AMPL) in the subject line triggers a translation process on this server. Details about the service can be found at the translation server web site at http://www.gamsworld.org/translate.htm.

8.3.2 Domain Expert Models

Domain expert models are often used as tools for problem analysis. To be a useful tool and to be able to support a user over a decade or more, these *models* need to be highly flexible. During its lifetime, a domain expert model grows and changes with every new project the domain expert takes on. The meaning of the term *model* also changes dramatically, from a collection of equations and variables to a compilation of sophisticated modules that prepare data, find a *solution* to a problem, and provide reporting capabilities. Mathematical programs are often included in the data preparation model (*e.g.*, calibration of data), and in the solution module, intertwined with other algorithmically steps. The part that represents mathematical programs (*i.e.*, equations and variables) does not, however, often exceed 10% of the overall code (the major part of GAMS source code is dedicated to data calculations and reporting).

The domain experts who use these models have entirely different needs than do the users of academic research models. A perfect example of such a domain expert user is the consulting firm Hill & Associates in Annapolis, Maryland, U.S.A. This firm specializes in providing sound advise and management services to the coal and electricity markets, based on supply, demand, and transportation data gathered over the past two decades. The models used in their *Electric Generation, Coal and Emission Forecasting System* include several extremely challenging linear and mixed-integer linear problems with millions of variables and hundreds of thousands of constraints. Since the requirements for this system change with nearly every project, the computing environment needs to be highly modular and to grow over time. Since the final product of a consultant is some written report, the models need to communicate with many data systems, including databases of different vintage and presentation systems such as MapInfo. Highly skilled experts, primarily economists and engineers, operate these models, and have deep insight into them and their application. Failures are the norm in such a constantly changing system, and handling failures is part of the routine operation.

The GAMS system fits nicely into such a diverse environment because it interfaces well with other systems. Data exchange with other modules is accomplished through the GAMS Data eXchange (GDX) facilities, a collection of ready-made programs that communicate with standard applications and an application programming interface (API) for custom-made data exchange. Furthermore, the GAMS execution system seamlessly integrates with external procedures, including a model's equations (see *External Functions in GAMS* in the appendix). This capability makes it very easy to integrate a GAMS model into a large application (a collection of such application systems was presented on the back cover of OR/MS Today issues from 1999 to 2001).

The GDX facilities complement the GAMS system's ASCII-based input and output. In addition to improving input/output time for bulk data movements, the data coming through the GDX interface is guaranteed to be syntactically correct. This inconspicuous feature, combined with a well-defined data contract (a mapping between the name-space in the GAMS model and the name-space in the external data source), allows the user to segment responsibility for interfacing the model with other systems. This segmentation is often reflected in the optimization project structures found at larger companies, where model analysts and developers ensure the correctness of the model, and the IT department guarantees the accurate transfer of data. Smooth operation is guaranteed if both groups adhere the data contract. If an error occurs, it can easily be traced to its source, to ensure that repair responsibility is properly assigned.

In modeling systems where the data exchange is part of the language, the data contract must reside inside the model. GAMS and GDX allow this contract to reside anywhere. This is especially helpful with spreadsheets and other cases where it is advantageous to place the data contract inside the data source.[2] In other applications where model and database are components of a larger system, it may be necessary to place the data contract between the model and the database inside the driving application.

The GDX interface can also be used to capture data in a database type file, called the GDX file. This file contains parameters, sets, and even scalar-like level and marginal values of variables and equations. All input data required for a model run can be represented in this platform-independent file (even across big and little endian computers). This ensures capture of the instance of a model failure. The problem can then be analyzed off-line together with the model source, and does not require otherwise vital connections with the rest of the application system.

Capturing the model source can frequently be as complex a process as obtaining the data itself, since the source can reside in different files located in different directories. Therefore, GAMS provides a source dumping mechanism (dumpopt) to collect all model sources in one file.

Independent of the original data source, the GDX file represents a highly structured database. Supplemented by a collection of GDX tools, the GDX facilities provide an environment that makes it easy to analyze both the data that goes into a model and the results that come out of each model run. For example, the tool *gdxdiff* compares two GDX files and reports differences similar to the UNIX *diff* text utility. To compare numeric data settings, an absolute tolerance allows the user to filter the differences. The utilities *gdxmerge* and *gdxsplit* combine structurally identical GDX files and add a scenario index to

[2]The GDX utility *gdxxrw*, which exchanges data with MS-Excel, supports this with its *param=index* option. For details see http://www.gams.com/dd/docs/gams/gdxutils.pdf.

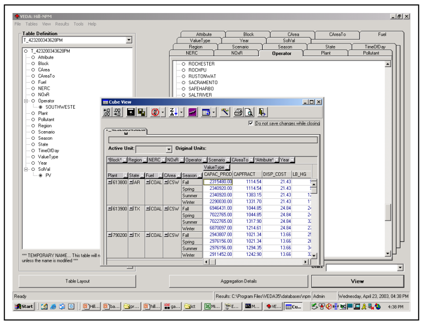
VEDA: Hill-NPM
File Tables View Results Tools Help
Attribute Block CArea CAreaTo Fuel
ValueType Year SolVal
Region Scenario Season State TimeOfDay
NERC NOxR Operator Plant Pollutant
Cube View
Table Layout
Aggregation Details
View

Table Definition
T_423200343628PM

- T_423200343628PM
  - Attribute
  - Block
  - CArea
  - CAreaTo
  - Fuel
  - NERC
  - NOxR
  - Operator
    - SOUTHWESTE
  - Plant
  - Pollutant
  - Region
  - Scenario
  - Season
  - State
  - TimeOfDay
  - ValueType
  - Year
  - SolVal
    - PV

- ROCHESTER
- ROCHPU
- RUSTONWAT
- SACRAMENTO
- SAFEHARBO
- SALTRIVER

☑ Do not save changes while closing

Active Unit: Original Units:

Block Region NERC NOxR Operator Scenario CAreaTo *Attribute* Year

Plant	State	Fuel	CArea	Season	CAPAC_PROD	CAPFRACT	DISP_COST
613800	AR	COAL	CSW	Fall	2315480.00	1114.54	21.43
				Spring	2340920.00	1114.54	21.43
				Summer	2340920.00	1383.15	21.43
				Winter	2290030.00	1331.70	21.43
613900	TX	COAL	CSW	Fall	6946431.00	1044.85	24.84
				Spring	7022765.00	1044.85	24.84
				Summer	7022765.00	1317.90	24.84
				Winter	6870097.00	1214.61	24.84
790200	TX	COAL	CSW	Fall	2943807.00	1021.34	13.66
				Spring	2976156.00	1021.34	13.66
				Summer	2976156.00	1294.35	13.66
				Winter	2911452.00	1242.90	13.66

*** TEMPORARY NAME... This table will n unless the name is modified ***

Ready | Results: C:\Program Files\VEDA35\databases\npm | Admin | Wednesday, April 23, 2003, 04:38 PM

4:38 PM

Figure 8.3.3. Scenario Management Tool: VEDA

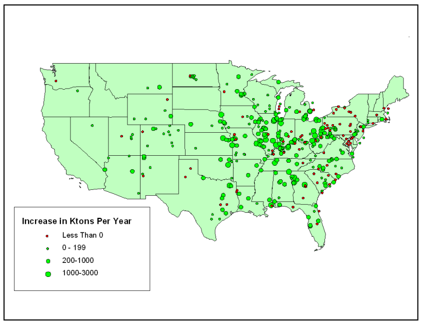

Figure 8.3.3. Geographical Information with MapInfo

every symbol for managing scenario analysis. *Gdxsplit* reverses the effect of *gdxmerge*. GDX files also represent a convenient way of storing model results in a compact format for deferred reporting of results. Using the GDX API data links to several non-standard reporting systems (including MapInfo, a geographical information system), MATLAB, and VEDA, a scenario management tool and data cube, have been successfully built. The screen shots in Figs. 8.3.3 and 8.3.3 show VEDA and MapInfo interfacing with GAMS models from Hill and Associates. The screen shot in Fig. 8.3.3 is a MATLAB visualization of a treatment plan calculated by a GAMS model for radiosurgery with the Gamma Knife [57].

8.3.3 Black Box Models

Developing the models used as black boxes in large applications is an extremely challenging task. The black box model user is often unaware that an optimization engine is involved when generating the application problem's so-

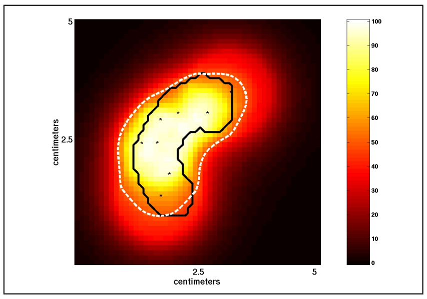

Figure 8.3.3. Visualization with MATLAB

lution. There are two certainties in optimization software that complicate the design of black box models: a) optimization will eventually fail, and b) optimization takes longer than the user is willing to wait. These nasty certainties contrast sharply with the overall system's requirements to a) always deliver a solution, and b) to do so in some measure of real time.

To prevent foreseeable model failures, black box models have an exhaustive and continuously growing list of data checks. Data checks are required to protect the model from delivering useless results, such as those infeasible or unbounded. The black box user does not care if a model is infeasible, but does require detailed, application-specific information on how to overcome the fact that the black box cannot generate a satisfactory solution to his or her problem. Data checks concentrate on particular problems in the data, and can report these problems in terms of data input. Most are simple record-by-record checks, such as ensuring that some numbers are in a particular range (*e.g.*, positive). When the data source is a modern database these simple integrity checks can be built into the database system, but over the model's lifetime such components can be replaced and external data checks can be short-circuited. A fail-safe black box model therefore needs all data checks. Some of these data checks are difficult, and cannot be easily implemented outside the model because they require a mathematical program's solution.

A scheduling project at the United States Military Academy (USMA), West Point, N.Y., U.S.A., is a good example of an effort primarily focused on detecting possible infeasibilities and providing application specific information to overcome them. Unlike other academic institutions, USMA schedules are built around the requirements of each of approximately 4,000 cadets. Each cadet's time is regimented, and USMA must insure that each cadet's schedule allows him and her to succeed in all academic, military, and physical programs at the academy. In some cases the activities of the cadet, the offered courses, and the strict rules of USMA collide, and no feasible schedule can be found for a cadet. Goal programming that penalizes any violation of a constraint while optimizing room assignments and other objectives does not help in this case. The human scheduler must acknowledge each violation of a constraint. Furthermore, assigning proper penalty weights depends on both the constraint and the individual cadet. The complexity of these interacting requirements was making data handling impossible. The solution was to solve a small mixed integer model for each cadet. Cadets with no feasible schedule resulted in an infeasible model. For these cadets, the model generated a set of infeasible schedules and so provided the human scheduler with several suggested ways to resolve each conflict on a cadet-by-cadet basis. This sophisticated data check ensured that the overall model that assigns schedules based on system-wide objectives was feasible, added value to the overall scheduling process, and helped the human scheduler become more productive at resolving scheduling conflicts.

A black box optimization model is often a central but small part of a large application system. The most prominent requirement is reliability in the model and the underlying modeling system. Data checks are one way to make the model reliable. Another way is to anticipate the solution module's failure and have a back-up solution ready. The GAMS execution system's flow control statements allow the user to build simple heuristics for most problems. The user gets a back-up solution, and the model can automatically capture the problematic instance (*e.g.*, in a GDX file) for off-line analysis by the model developer. The overall model cannot collapse just because a solver crashed with some hard exception error. The modeling system must be able to recover from such failures, and also needs watchdog capabilities to keep solver resources (*e.g.*, time and memory) in check. The GAMS system can run the solver completely detached from the overall model processing. This protects the model from unintentional interference from the solver or other external processes.

The world of platform independent development with Java and advances in internet-ready application shed new light on the importance of platform independence for modeling systems. For example, an optimization application's development might start on Windows PCs, undergo testing on Solaris workstations, deploy an optimization module on an HP workstation, and move to a 64-bit Linux PC after five years.

8.4 Summary and Conclusion

In this chapter we provided some background information to support and explain some of the early design decisions, which we believe are still valid today and guide us into the future. As a language and modeling system designers it is important to keep the focus on the existing and future user community. Mundane concepts as backward compatibility are an essential element in protecting the investments of our users. They demand that models built 10 years ago can be operated today without any change and at the same time are able to take advantage of the latest computer architecture and technology as well allow the use of any state-of-the-art solvers. The same holds for an unknown future modeling environments. The users of our systems are entitled to expect that their models will operate in 10 years from now with out any change. The incorporation of new technologies and theoretical advances will have to be done in a way to protect existing investments.

Finally we would like to recognize the contribution of the editor of this volume, Dr. Josef Kallrath, to bring together a group of highly individualistic and sometimes very competitive developers at the 2003 meeting in Bad Honnef and manage to make them contribute the chapters in a timely fashion. Congratulations.

Appendix

A Selected Language Features

The modeling systems described in this volume are closely related to each other and share many important design principles and in many cases look very similar. The following is a somewhat arbitrary list of features that are not shared by other systems:

Model Types. A model is a collection of symbolic equations which can be used to solve a specific model type with a statement like *solve mymodel using NLP minimizing myobjective*. GAMS requires the user to specify the general model type. This allows the system to verify that a suitable solver is available and that the symbolic algebra fits into the general problem class. Since a nonlinear mixed-integer problem is magnitudes more difficult to solve than a linear program, GAMS requires the user to be aware of the kind of model he or she wants to build and solve. The second feature is that GAMS does not have an objective function and uses instead a simple variable to express the maximand. This allows any variable or indirectly any constraint to serve as an objective at the point of model instantiation.

Attached Comments. Comment and explanatory texts are permanently attached to all symbols and data elements. Descriptive information will never be lost and can automatically be retrieved when producing reports or display intermediate results. This encourages good modeling practice and provides automatic documentation.

Domain Checking. Index positions of all data types are *domain checked*. This is very similar to the concept of referential integrity in data base systems and will guarantee that we will never attempt to reference an index position with an incorrect elopement, set or subset that does not belong there. For example, a variable declared to have the index context of the sets (I,J) can only be indexed by the *domain sets* I and J, their subsets or elements belonging to them. All common errors of inadvertently transposing index references like (J,I) will be flagged as errors.

No Dummy Index. Index references do not use *dummy indices*. In most cases, the use of *dummy indices* makes the algebra more difficult to read and leads to a loss of context. For example, a matrix multiplication is simply written as *c(i,j) = sum(k, a(i,k)*b(j,k))*. A good naming convention then leads to a very compact and direct representation. A consequence of this choice is need for 'aliases' to allow driving more than one index with the same set.

Relaxed Punctuation. A certain amount of context sensitivity and *relaxed* punctuation is allowed. Non-programmers become very impatient if every string has to be quoted, or a missing comma at the end of a line in some list causes a compilation error. Another example is a missing statement separator ';' not causing any ambiguity will not cause an error.

Common Syntax for Declarative and Procedural Components. The syntax for data transformations and symbolic equation definitions are alike and treated internally in the same way. Symbolic equation, model and data definitions are declarative, all other statements are procedural. As problems become more complex, the procedural aspects of the system become more important. Models of different types feed result into other models, all embedded in some complex control structures.

Simple Macros. Extensive compile time source preprocessing and recursive calling of GAMS itself allows easy tailoring of complex applications.

Computational Performance. Sparsity and parallelism are exploited to skip over "dead" indices and in most cases the order of computational burden is proportional to the number of no default (zero) entries in its operands. For example, the computational burden of *sum((i,j), a(i,j)*x(i,j))* will be proportional to the non-zero entries in 'a' and not the product of the cardinality of the sets i and j.

Save and Restart. The state of the system can be saved completely and allows to be restarted compilation and execution at any time, possibly on a different computing platform and even encrypted (see appendix *Secure Work Files*). A simple example is to do the report generation at different times from the solution of the model. All information belonging to different scenarios can easily be stored away and later reused to produce specific reports.

B GAMS External Functions

Although GAMS provides a powerful language for manipulating data and defining highly structured collections of variables and equations, there are times when one would like to define some parts of a model using a more traditional programming language such as Fortran or C.

GAMS External Functions connect code written in Fortran, C, Java, Delphi, or some other programming language to equations and variables in a GAMS model. We will refer to these GAMS equations as *external equations*, and the compiled version of the programming routines as the *external module* defining the *external functions*. The form of the external module depends on the operating system used. The external module under Windows is a Dynamic Link Library (.dll), and the external module under Unix is a shared object (.so). In principle, any language or system can be used to build the .dll or shared object defining the external module, as long as the interface conventions are not changed.

The basic mechanism is to declare all the equations and variables using the normal GAMS syntax. The interpretation of the external equations is done in a special way. Instead of the usual semantic content, the external equations specify the mapping between 1) the equation and variable names used in GAMS and the function, and 2) variable indices used in the external module which can be written in C, Fortran or most other programming languages.

The external equation interface is not intended as a way to bypass some of the very useful model checking done by GAMS when external equations are used with an NLP solver. External equations are still assumed to be continuous, and to have accurate and smooth first derivatives. The continuity assumption implies that the external functions must have very low noise levels, considerably below the feasibility tolerance used by the solver. The assumption about accurate derivatives implies that derivatives must be computed more accurately that can be done with standard finite differences. If these assumptions are not satisfied, there is no guarantee that the NLP solver can find a solution that has the mathematical properties of a local optimum (*i.e.*, that satisfies the Karush-Kuhn-Tucker conditions within the standard tolerances used by the solver). More information about GAMS External Functions can be found at http://www.gams.com/docs/extfunc.htm.

C Secure Work Files

Issues of privacy, security, data integrity and ownership arise when models are distributed to users other than the original developers, or are embedded in applications to be deployed by other developers. We may have to hide, protect, or purge some parts of the model before it can be released. The information to be protected can be of numeric or symbolic nature, and the protection requirements may be driven by:

Privacy. A Social Accounting Matrix supplied by a Statistical Office is required in a general equilibrium model to be used by the Ministry of Finance. The data from the statistical office needs to be protected for obvious privacy reasons and the model experiments are used to evaluate policy options that are highly confidential. Most of the model structure is public, most of the data however is private and model results need to be transformed in such a way as to prohibit the discovery of the original data.

Security. Components of a model contain proprietary information that describes mathematically a chemical reaction. The associated algebra and some of the data are considered of strategic importance and need to be hidden completely. However, the final model will be used at different locations around the world.

Integrity. Data integrity safeguards are needed to assure the proper functioning of a model. Certain data and symbolic information needs to be protected from accidental changes that would compromise the model's operation.

To address these secure work file issues, access control at a symbol level and secure restart files have been added to the `GAMS` system.

Access Control. The access to `GAMS` symbols, including sets, variables, parameters, and equations, can be changed once with the compile time commands $purge, $hide, $protect and $expose. $Purge will remove any information associated with this symbol. $Hide will make the symbol and all its information invisible. $Protect prevents changes to information. $Expose will revert the symbol to its original state.

Secure Restart Files. The `GAMS` licensing mechanism can be used to save a secure model in a secure work file (work files are used to save and restart the state of a `GAMS` program). A secure work file behaves like any other work file, but is locked to a specific users license file. A privacy license, the license file of the target users, is required to create a secure work file. The content of a secure work file is disguised and protected against unauthorized access via the `GAMS` license mechanism.

A special license is required to set the access controls and to create a corresponding secure work file. Reporting features have been added to allow audits and traces during generation and use of secure work files. More information about secure work files can be found at http://www.gams.com/docs/privacy.pdf

D GAMS versus Fortran Matrix Generators

In the late 70's and early 80's, the MPS file was the universally accepted format to describe a linear program. Those MPS files were either described using specialized Matrix Generator languages, like the very successful and durable `MAGEN/OMNI` family of products from Haverly Systems [92], or general purpose programming languages, like Fortran and PL/I were used to write matrix generator programs. Considerable programming experience and a large stock of programs had been accumulated and novel approaches like algebraic modeling languages were viewed as irrelevant academic exercises. Computing power was still the limiting factor and the idea of replacing the MPS file with an algebraic model representation with the need to create a new instance of this model for each optimization step was considered totally impractical.

The first `GAMS` implementation was done on Control Data Corporation's (CDC) super computers and interfaced with the APEX [33] linear programming system which was the first high performance system utilizing super sparse matrix technology allowing all in-core solutions. The `GAMS` development team was determined to disprove the allegation of inherently poor performance of a mathematical approach. The implantation took full advantage of the CDC computing architecture and implemented novel dynamic data structures and just in time generation of machine code was employed. The system was blazingly fast, and attracted the hoped for attention in the traditional LP shops. The second implementation of `GAMS` focused on computing platform and solver independence with the main goal to operate on IBM's mainframes in conjunction with the MPSX [105] or MPSIII [124] linear programming systems, a computing environment that supported 90 percent of all commercial linear programming. A new product from IBM, the Optimization Subroutine Library (OSL) was used as the new performance benchmark. The performance target was to generate a `GAMS` models instance and load it into the OSL workspace in less time than it took to run the Fortran generator and read an MPS file into OSL. One of the benchmark models was a water resource model for Pakistan which was implemented using a Fortran generator and GAMS, one of the GAMS version called INDUS89 [223] can be found in the GAMS model library. The LP matrix has 2,726 row, 6,570 columns and 39,489 non-zero entries. Timings on an high end IBM 3090 costing some 3 million dollars were: OSL optimization time excluding input was 137 seconds, the Fortran MPS generation and OSL import time were 37 and 19 seconds an added 56 seconds, the GAMS generation and OSL import were 41 and 1 second added only 42 seconds. For comparison, the timings on an 1Ghz PC costing only 1,500 dollars with GAMS and OSL 3 are 4 seconds for the optimization phase and 0.5 seconds for generating the model instance.

Over past 15 years, the hardware cost has declined by a factor of 2000 and the speed has increased by a factor of 40. The computational cost of generating and solving one instance of this model has been reduced by a factor of 800,000 to essentially zero.

E Sample GAMS Problem

A portfolio optimization model is used to illustrate in more detail some of the GAMS language features. The same model has been used in other chapters to illustrate the representation and use of other modeling languages. Most of the presentation below will be self-evident and we will just add a few comments for clarification:

Dollar Control Statements. Lines starting with $ are compiler directives. Text between $ontext and $offtext are comments. $eolcom sets the end-of-line comment symbols. The $if statements is patterned after the Windows scripting language and in line 106 we set the local environment string scenario to the value of *s1* if it is still undefined at this point.

Dynamic Sets. Only static sets can be used to define the allowable index domain. Actual references can be done by any single element or set that is contained in the index domain. For example, the equation *TR(rR)* in lines 133 is defined over the domain set *rR*, in the definition at line 145, however, we use the dynamic set *r*. This is a convenient way to resize any model dimension at will.

Suffix Notation. Variables and equations have attributes like primal and dual values, lower and upper bounds, scale factors or priorities. Those attributes are spefied with a suffix notation. For example, in the equation definition in line 215 we reference the variables *f* as *f.lo* which is the lower bound of this variable at the time of model instantiation.

Maps and Filters. Set tuples can be used to limit the domain over which indexed operations are carried out. In line 214 we define an equation over a three-tuple *rpi* and reference the variable *pT2* with the tuple *rpi* insead of the three sets *r, p* and *i*. The use of n-tuples in domain definitions is a convenient way to manage domain restrictions which results in an easy to read, compact notation.

```
\index{GAMS!Applications!product portfolio optimization}
  1 $Title  Product Portfolio Optimization
  2 $ontext
  3
  4 This problem computes minimal cost solutions satisfying the
  5 demand of pre-given product portfolios. It determines the number
  6 and size of reactors and gives a schedule of how may batches of
  7 each product run on each reactor. There are two scenarios (s1 20
  8 products. s2 40 products), add --scenario s2 as a \texttt{GAMS}
  9 parameter to specify the second scenario.
 10
 11 The global optimal reactor volumes are:
 12 data set s1
```

```
vr.fx('r1') = 132.5; vr.fx('r2') = 250;

data set s2
vr.fx('r1') = 20; vr.fx('r2') = 100; vr.fx('r3') = 250;

Two formulations are presented, a compact MINLP formulations
and a linearized MIP formulation using special ordered sets.

Problem sizes for data set s1

                         MINLP       MIP
variables                 102        548
equations                  28        418
Non-zeros                 190       1574
discrete variables         40        186


Kallrath, J. Exact Computation of Global Minima of a Nonconvex
Portfolio Optimization Problem. In Frontiers in Global
Optimization. Eds Floudas C A andd Pardalos P M.
Kluwer Academic Publishers, Dortrecht, 2003.

$offtext
$eolcom //

Sets
       s      scenario / s1,s2 /
       rR     reactors / R1*R3 /
       pP     products / L1*L37 /
       r(rR)  reactors considered in scenorio
       p(pP)  products considered in scenorio

Table RData(rR,s,*) Reactor data

        s1.VMIN  s1.VMAX  s2.VMIN  s2.VMAX
R1        102.14     250     20         50
R2        176.07     250     52.5      250
R3                         151.25      250


Table PData(pP,s,*) Product data

     s1.Dem s1.PTime s2.Dem s2.Ptime
L1     2600      6     2600      6
L2     2300      6     2300      6
L3     1700      6      450      6
L4      530      6     1200      6
L5      530      6      560      6
L6      280      6      530      6
L7      250      6      530      6
L8      230      6      140      6
L9      160      6      110      6
L10      90      6      110      6
L11      70      6       10      6
L12     390      6      110      6
L13     250      6       90      6
L14     160      6       90      6
```

```
L15     100      6       90      6
L16      70      6       70      6
L17      50      6       50      6
L18      50      6       30      6
L19      50      6       10      6
L20                      10      6
L21                      10      6
L22                     190      6
L23                     180      6
L24                      70      6
L25                      70      6
L26                      40      6
L27                      40      6
L28                      40      6
L29                      30      6
L30                      20      6
L31                      20      6
L32                      20      6
L33                      10      6
L34                      10      6
L35                      10      6
L36                      10      6
L37                      10      6


Parameters
      VMIN(rR)     volume flow of products in m^3 per week
      VMAX(rR)     volume flow of products in m^3 per week
      DEMAND(pP)   volume flow of products in m^3 per week
      PRODTIME(pP) production time in hours per batch
Scalars
      WHRS   hours in a week                               / 168  /
      CSTI   in kEuro depreciation per m^3 reactor and week / 0.97 /
      CSTF   in kEuro per week and reactor                 / 2.45 /
      ESF    economies of scale factor                     / 0.5  /;

$if not set scenario $set scenario s1
VMIN(rR)     = RDATA(rR,'%scenario%','VMIN');
VMAX(rR)     = RDATA(rR,'%scenario%','VMAX');
DEMAND(pP)   = PDATA(pP,'%scenario%','Dem');
PRODTIME(pP) = PDATA(pP,'%scenario%','PTime');

* Determine scenario sets
r(rR) = VMAX(rR)   > 0;
p(pP) = DEMAND(pP) > 0;

* definition of compact MINLP model

Variables  cTotal       total  costs
           cInvest      invest cost
           cFixed       fix    costs
           f(rR,pP)     utilization rate
           vR(rR)       reactor volume in m^3
           pS(pP)       surplus production
           bvr(rR)      indicating whether reactor r is active
           nB(rR,pP)    number of batches of product p in reactor r

```

```
Positive variables f,vR,pS; Integer variable nB; Binary Variables bvr;


Equations DEFcT          total costs
          DEFcF          fix costs
          DEFcI          invest cost
          TR(rR)         production time of reactor r
          SPP(pP)        compute surplus production p
          RVUB(rR)       maximal volume of reactor r
          RVLB(rR)       minimal volume reactor r;


* define the total cost
DEFcT..  cTotal  =e= cFixed + cInvest;
DEFcF..  cFixed  =e= sum (r, CSTF*bvr(r));
DEFcI..  cInvest =e= sum (r, CSTI**ESF*vR(r)**ESF);

* limit the total production time of reactor r
TR(r)..    sum(p, PRODTIME(p)*nB(r,p) ) =l= WHRS*bvr(r);

* compute the surplus production
SPP(p)..   pS(p) =e= SUM(r, nB(r,p)*f(r,p)*vR(r))/DEMAND(p) - 1 ;

* lower and upper bounds on reactor volume
RVLB(r)..  vR(r) =g= VMIN(r)*bvr(r);
RVUB(r)..  vR(r) =l= VMAX(r)*bvr(r);

Model     portfolioMINLP / DEFcT, DEFcF, DEFcI, TR, SPP, RVLB, RVUB / ;

f.lo(r,p) = 0.4; f.up(r,p)  = 1;  // bounds on the utilization rates
pS.lo(p)  =   0; pS.up(p)   = 1;  // bounds on the surplus production

* bounds on the number of batches
nB.lo(r,p) = 0;
nB.up(r,p) = min(WHRS/PRODTIME(p),floor(2*DEMAND(p)/(VMIN(r)*f.lo(r,p))));
nB.up(r,p)$(2*DEMAND(p) < f.lo(r,p)*VMIN(r)) = 0;

vR.l(rR) = 99;
vR.lo(r) = VMIN(r);

solve portfolioMINLP using minlp minimizing cTotal;

* additional variables and equations to define the MIP formulation
* first we need to linearize the product terms:

Sets i            dyadic represenation set       / 0*10 /
     j            discretization points for SOS2 / 0*10 /
     rpi(rR,pP,i) i required for representing np

Parameters
    vRj(rR,j)    x part of SOS2 construct
    ESFvRj(rR,j) y part of SOS2 construct

Positive Variable
      pT(rR,pP)    number of batches x reactor volume in m^3
      pT2(rR,pP,i) same for in dyadic representation
      ESFvR(rR)    economies of scale for vR
```

```

Binary Variables
      nBx(rR,pP,i) dyadic represenation of nB

SOS2 Variables
      lambda(rR,j) approximation of economies of scale function

Equations
      CNP(rR,pP)  compute the nonlinear products nB(rp)*f(rp)*vR(r)
      SPPx(pP)    compute surplus production p
      CNPl0(rR,pP)   linearized version of CNP
      CNPl1(rR,pP)   linearized version of CNP
      CNPl2(rR,pP,i) linearized version of CNP
      CNPl3(rR,pP,i) linearized version of CNP
      CNPl4(rR,pP,i) linearized version of CNP
      DEFSOSx(rR)    SOS2 x construct
      DEFSOSy(rR)    SOS2 y construct
      DEFSOSone(rR)  SOS2 sum construct
      DEFcIlp        linearized version of DEFcI;

* new surplus production equation
SPPx(p)..   pS(p) =e= SUM(r, pT(r,p))/DEMAND(p) - 1 ;

* computes batches x volume
CNP(r,p).. pT(r,p) =e= nB(r,p)*f(r,p)* vR(r);

* Linearized version of CNP
CNPl0(r,p)..         pT(r,p)  =e= sum(rpi(r,p,i),2**(ord(i)-1)*pT2(rpi));
CNPl1(r,p)..         nB(r,p)  =e= sum(rpi(r,p,i),2**(ord(i)-1)*nBx(rpi));
CNPl2(rpi(r,p,i))..  pT2(rpi) =l= VMAX(r)*nBx(rpi);
CNPl3(rpi(r,p,i))..  pT2(rpi) =l= vR(r);
CNPl4(rpi(r,p,i))..  pT2(rpi) =g= f.lo(r,p)*(vR(r)-VMAX(r)*(1-nbx(rpi)));


* SOS2 approximation of economies of scale function
DEFSOSx(r)..  vR(r) =e= sum(j, vRj(r,j)*lambda(r,j));

DEFSOSy(r)..  ESFvR(r) =e= sum(j, ESFvRj(r,j)*lambda(r,j));

DEFSOSone(r).. sum(j, lambda(r,j)) =e= 1;

DEFcIlp..  cInvest =e= sum (r, CSTI**ESF*ESFvR(r));


rpi(r,p,i) = ord(i) <= ceil(log(max(1,nB.up(r,p)))/log(2)) + 1$nB.up(r,p);

vRj(r,j) = (VMAX(r)-VMIN(r))*(ord(j)-1)/(card(j)-1) + VMIN(r);
ESFvRj(r,j) = vRj(r,j)**ESF;

Model PortfolioMIP  /TR, SPPx, RVLB, RVUB, DEFcF, DEFcIlp, DEFcT,
CNPl0, CNPl1, CNPl2, CNPl3, CNPl4, DEFSOSx, DEFSOSy, DEFSOSone/;

portfolioMIP.iterlim = 1e8; portfolioMIP.optcr = .05;
solve portfolioMIP using mip minimizing cTotal;
```