
AIMMS User's Guide - Math Program Inspector

This file contains only one chapter of the book. For a free download of the complete book in pdf format, please visit www.aimms.com or order your hard-copy at www.lulu.com/aimms.

Copyright © 1993–2011 by Paragon Decision Technology B.V. All rights reserved.

Paragon Decision Technology B.V.	Paragon Decision Technology Inc.	Paragon Decision Technology Pte.
Schipholweg 1	500 108th Avenue NE	Ltd.
2034 LS Haarlem	Ste. # 1085	80 Raffles Place
The Netherlands	Bellevue, WA 98004	UOB Plaza 1, Level 36-01
Tel.: +31 23 5511512	USA	Singapore 048624
Fax: +31 23 5511517	Tel.: +1 425 458 4024	Tel.: +65 9640 4182
	Fax: +1 425 458 4025	

Email: info@aimms.com
WWW: www.aimms.com

AIMMS is a registered trademark of Paragon Decision Technology B.V. IBM ILOG CPLEX and sc CPLEX is a registered trademark of IBM Corporation. GUROBI is a registered trademark of Gurobi Optimization, Inc. KNITRO is a registered trademark of Ziena Optimization, Inc. XPRESS-MP is a registered trademark of FICO Fair Isaac Corporation. MOSEK is a registered trademark of Mosek ApS. WINDOWS and EXCEL are registered trademarks of Microsoft Corporation. $\text{T}_{\text{E}}\text{X}$, $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$, and $\text{A}_{\text{M}}\text{S}_{\text{L}}\text{A}_{\text{T}}\text{E}_{\text{X}}$ are trademarks of the American Mathematical Society. LUCIDA is a registered trademark of Bigelow & Holmes Inc. ACROBAT is a registered trademark of Adobe Systems Inc. Other brands and their products are trademarks of their respective holders.

Information in this document is subject to change without notice and does not represent a commitment on the part of Paragon Decision Technology B.V. The software described in this document is furnished under a license agreement and may only be used and copied in accordance with the terms of the agreement. The documentation may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from Paragon Decision Technology B.V.

Paragon Decision Technology B.V. makes no representation or warranty with respect to the adequacy of this documentation or the programs which it describes for any particular purpose or with respect to its adequacy to produce any particular result. In no event shall Paragon Decision Technology B.V., its employees, its contractors or the authors of this documentation be liable for special, direct, indirect or consequential damages, losses, costs, charges, claims, demands, or claims for lost profits, fees or expenses of any nature or kind.

In addition to the foregoing, users should recognize that all complex software systems and their documentation contain errors and omissions. The authors, Paragon Decision Technology B.V. and its employees, and its contractors shall not be responsible under any circumstances for providing information or corrections to errors and omissions discovered at any time in this book or the software it describes, whether or not they are aware of the errors or omissions. The authors, Paragon Decision Technology B.V. and its employees, and its contractors do not recommend the use of the software described in this book for applications in which errors or omissions could threaten life, injury or significant loss.

This documentation was typeset by Paragon Decision Technology B.V. using $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ and the LUCIDA font family.

Chapter 9

The Math Program Inspector

In this chapter you will find the description of an extensive facility to analyze both the input and output of a generated optimization model. This mathematical program inspector allows you to make custom selections of constraints and variables. For each such selection, you can inspect statistics of the corresponding matrix and the corresponding solution. The main purpose of the math program inspector, however, is to assist you in finding causes of infeasibility, unboundedness and unrealistic solutions.

This chapter

The design and contents of the math program inspector is strongly influenced by the papers and contributions of Bruce A. McCarl on misbehaving mathematical programs. The example in the last section of this chapter is a direct reference to his work.

Acknowledgement

9.1 Introduction and motivation

Even though you have taken the utmost care in constructing your linear optimization model, there are often unforeseen surprises that force you to take a further look at the particular generated model at hand. Why is the model infeasible or unbounded? Why is the objective function value so much different from what you were expecting? Are the individual constraints generated as intended? Why is the number of individual constraints so large? Why are the observed shadow prices so unrealistically high (or low)? These and several other related questions about the matrix and the solution need further investigation.

Unforeseen surprises ...

The answer to many model validation questions is not easily discovered, especially when the underlying optimization model has a large number of individual constraints and variables. The amount of information to be examined is often daunting, and an answer to a question usually requires extensive analysis involving several steps. The functionality of the math program inspector is designed to facilitate such analysis.

... are not easily explained

There are many causes of unforeseen surprises that have been observed in practice. Several are related to the values in the matrix. Matrix input coefficients may be incorrect due to a wrong sign, a typing error, an incorrect unit of measurement, or a calculation flaw. Bounds on variables may have been omitted unintentionally. Other causes are related to structural information. The direction of a constraint may be accidentally misspecified. The subsets of constraints and variables may contain incorrect elements causing either missing blocks of constraints and variables, or unwanted blocks. Even if the blocks are the correct ones, their index domain restrictions may be missing or incorrect. As a result, the model may contain unwanted and/or missing constraints and/or variables.

Some of the causes

The purpose of the mathematical program inspector included in AIMMS is to provide you with

Purpose

- insight into the (structure of the) generated model and its solution (if present), and
- a collection of tools to help you discover errors in your model ,

9.2 Functional overview

In this section you will find a description of the functionality available in the mathematical program inspector. Successively, you will learn

This section

- the basics of the trees and windows available in the mathematical program inspector,
- how you can manipulate the contents of the variable and constraint trees through variable and constraint properties, but also using the **Identifier Selector** tool,
- how to inspect the contents and properties of the matrix and solution corresponding to your mathematical program, and
- which analysis you can perform using the mathematical program inspector when your mathematical program is infeasible.

9.2.1 Tree view basics

The math program inspector window displays the set of all generated variables and all generated constraints, each in a separate tree (see the left portion of Figure 9.1). In these trees, the symbolic identifiers are the first-level nodes and on every subsequent level in the tree, one or more indices are fixed. As a result, the individual variables and constraints in your model appear as leaf nodes in the two tree view windows.

Viewing generated variables and constraints

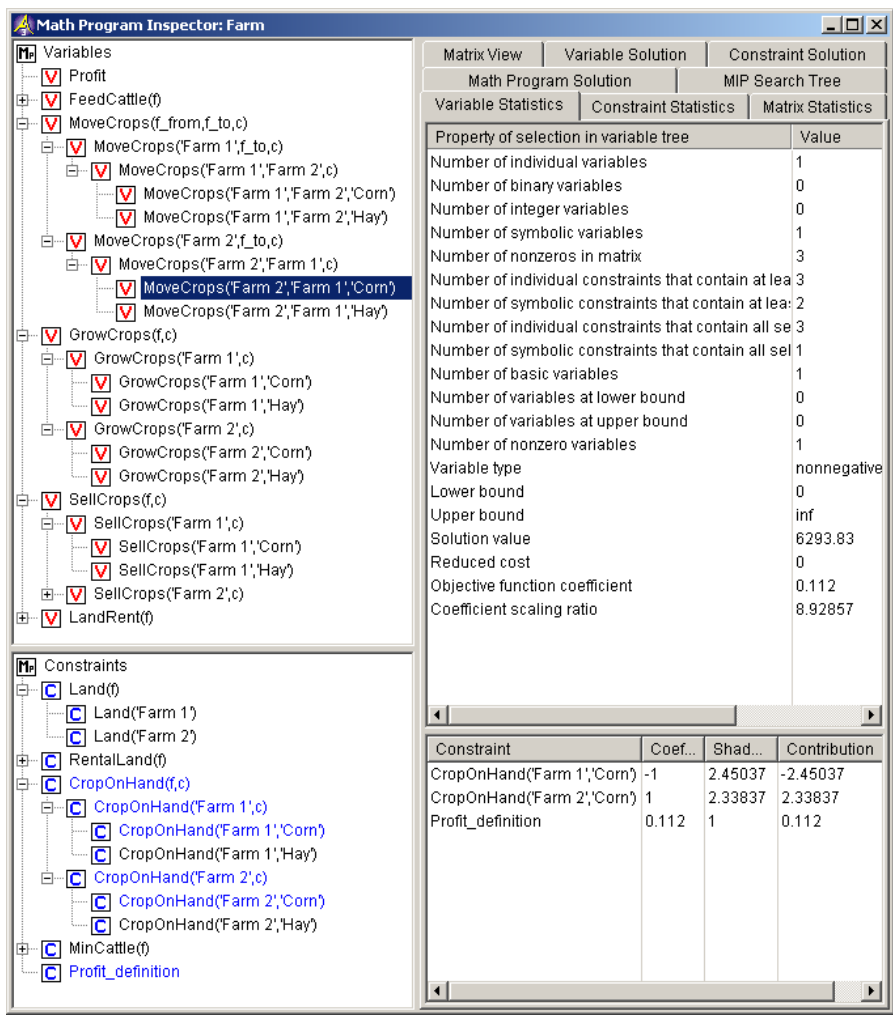


Figure 9.1: The math program inspector window

The math program inspector contains several tabs (see the right portion of Figure 9.1) that retrieve information regarding the selection that has been made in the tree views. Common Windows controls are available to select a subset of variables and constraints (mouse click possibly in combination with the *Shift* or *Control* key). Whenever you select a slice (i.e. an intermediate node in the tree) all variables or constraints in that subtree are selected implicitly. You can use the **Next Leaf Node** and **Previous Leaf Node** buttons in the toolbar for navigational purposes. In Figure 9.1 a single variable has been selected in the variable tree. In addition, the text on most tabs can be copied to the Windows clipboard using the familiar **Ctrl-C** shortcut key.

Tree view selections

Bookmarks allow you to temporarily tag one or more variables or constraints. While navigating through the tree you will always change the current selection, while the bookmarked nodes will not be affected. Whenever you bookmark a node, all its child nodes plus parent nodes are also bookmarked. Using the **Bookmarks** menu you can easily select all bookmarked nodes or bookmark all selected nodes. Bookmarks appear in blue text. Figure 9.1 contains a constraint tree with three bookmarked constraints plus their three parent nodes. You can use the **Next Bookmark** and **Previous Bookmark** buttons in the toolbar for navigational purposes. In case your Figure 9.1 is not displayed in color, the light-gray print indicates the bookmarked selection.

Bookmarks

By default only one index is fixed at every level of the tree views, and the indices are fixed from the left to the right. However, you can override the default index order as well as the subtree depth by using the **Variable Property** or **Constraint Property** dialog on the first-level nodes in the tree. The subtree depth is determined by the number of distinct index groups that you have specified in the dialog.

Domain index display order

The linkage between variables and constraints in your model is determined through the individual matrix coefficients. To find all variables that play a role in a particular constraint selection, you can use the **Associated Variables** command to bookmark the corresponding variables. Similarly, the **Associated Constraints** command can be used to find all constraints that play a role in a particular variable selection. In Figure 9.1, the associated constraint selection for the selected variable has been bookmarked in the constraint tree.

Finding associated variables/constraints

9.2.2 Advanced tree manipulation

Using the right-mouse popup menu you can access the **Variable Properties** and **Constraint Properties**. On the dialog box you can specify

Variable and constraint properties

- the domain index display order (already discussed above), and
- the role of the selected symbolic variable or constraint during infeasibility and unboundedness analysis.

The math program inspector tool has two tabs to retrieve statistics on the current variable and constraint selection. In case the selection consists of a single variable or constraint, all coefficients in the corresponding column or row are also listed. You can easily access the variable and constraint statistics tabs by double-clicking in the variable or constraint tree. Figure 9.1 shows the variable statistics for the selected variable.

Variable and constraint statistics

In addition to **Variable Properties** and **Constraint Properties**, you can use the right-mouse popup menu to

*Popup menu
commands*

- open the attribute form containing the declaration of an identifier,
- open a data page displaying the data of the selected slice,
- make a variable or constraint at the first level of the tree inactive (i.e. to exclude the variable or constraint from the generated matrix during a re-solve), and
- bookmark or remove the bookmark of nodes in the selected slice.

Using the identifier selector you can make sophisticated selections in the variable and/or constraint tree. Several new selector types have been introduced to help you investigate your mathematical program. These new selector types are as follows.

*Interaction with
identifier
selector*

- **element-dependency selector:** The element-dependency selector allows you to select all individual variables or constraints for which one of the indices has been fixed to a certain element.
- **scale selector:** The scale selector allows you to find individual rows or columns in the generated matrix that may be badly scaled. The selection coefficient for a row or column introduced for this purpose has been defined as

$$\frac{\text{largest absolute (nonzero) coefficient}}{\text{smallest absolute (nonzero) coefficient}}$$

The **Properties** dialog associated with the scale selector offers you several possibilities to control the determination of the above selection coefficient.

- **status selector:** Using the status selector you can quickly select all variables or constraints that are either basic, feasible or at bound.
- **value selector:** The value selector allows you to select all variables or constraints for which the value (or marginal value) satisfies some simple numerical condition.
- **type selector:** With the type selector you can easily filter on variable type (e.g. continuous, binary, nonnegative) or constraint type (e.g. less-than-or-equal, equal, greater-than-or-equal). In addition, you can use the type selector to filter on nonlinear constraints.

9.2.3 Inspecting matrix information

Most of the statistics that are displayed on the Variable Statistics tab are self-explanatory. Only two cases need additional explanation. In case a single symbolic (first-level node) has been selected, the *Index domain density* statistic will display the number of actually generated variables or constraints as a percentage of the full domain (i.e. the domain without any domain condition applied). In case a single variable (a leaf node) has been selected, the statistics

*Variable
Statistics tab*

will be extended with some specific information about the variable such as bound values and solution values.

In case a single variable x_j has been selected, the lower part of the information retrieved through the Variable Statistics tab will contain a list with all coefficients a_{ij} of the corresponding rows i , together with the appropriate shadow prices y_i (see Figure 9.1). The last column of this table will contain the dual contributions $a_{ij}y_j$ that in case of a linear model together with the objective function coefficient c_j make up the reduced cost \bar{c}_j according to the following formula.

$$\bar{c}_j = c_j - \sum_i a_{ij}y_i$$

Coefficients of variables that appear in nonlinear terms in your model are denoted between square brackets. These numbers represent the linearized coefficients for the current solution values.

The Constraints Statistics tab and the Variable Statistics tab retrieve similar statistics. Figure 9.2 shows the constraint statistic for the selection consisting of a single constraint. Note that in this particular case the symbolic form of the constraint definition will also be displayed. In case the selected constraint is nonlinear, the individual nonlinear constraint as generated by AIMMS and communicated to the solver is also displayed.

Column coefficients

Nonlinear coefficients

Constraint Statistics tab

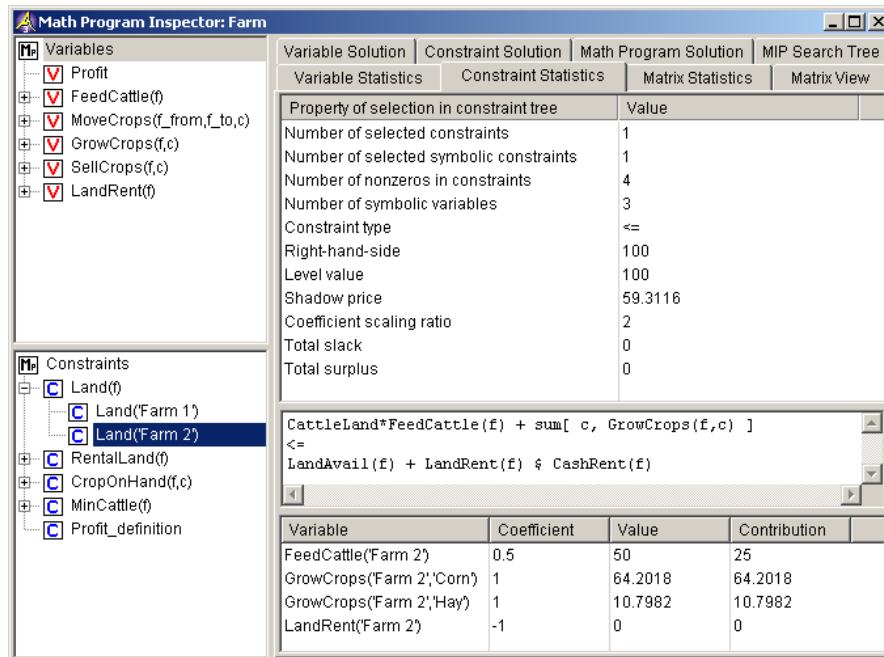


Figure 9.2: The math program inspector window

In case a single row i has been selected, the lower part of the Constraint Statistics tab will contain all coefficients a_{ij} in the corresponding columns j , together with their level values x_j . The last column of this table lists the primal contributions $a_{ij}x_j$ that together in case of a linear model with the right-hand-side make up either the slack or surplus that is associated with the constraint according to the following formula.

$$\text{slack}_i - \text{surplus}_i = \text{rhs}_i - \sum_j a_{ij}x_j$$

Row coefficients

As is the case on the Variable Statistics Tab, all coefficients corresponding to nonlinear terms are denoted between square brackets. For these coefficients, the last column displays all terms that contribute to the linearized coefficient value.

Nonlinear constraints

The Matrix Statistics tabs retrieves information that reflects both the selection in the variable tree and the selection in the constraint tree. Among these statistics are several statistical moments that might help you to locate data outliers (in terms of size) in a particular part of the matrix.

Matrix Statistics tab

The Matrix View tab contains a graphical representation of the generated matrix. This view is available in two modes that are accessible through the right-mouse popup menu. The symbolic block view displays at most one block for every combination of symbolic variables and symbolic constraints. The individual block view allows you to zoom in on the symbolic view and displays a block for every nonzero coefficient in the matrix. It is interesting to note that the order in which the symbolic and individual variables and constraints are displayed in the block view follows the order in which they appear in the trees.

Matrix View tab

The colors of the displayed blocks correspond to the value of the coefficient. The colors will vary between green and red indicating small and large values. Any number with absolute value equal to one will be colored green. Any number for which the absolute value of the logarithm of the absolute value exceeds the logarithm of some threshold value will be colored red. By default, the threshold is set to 1000, meaning that all nonzeros $x \in (-\infty, -1000] \cup [-\frac{1}{1000}, \frac{1}{1000}] \cup [1000, \infty)$ will be colored red. All numbers in between will be colored with a gradient color in the spectrum between green and red.

Block coloring

Any block that contains at least one nonlinear term will show a hatch pattern showing diagonal lines that run from the upper left to the lower right of the block.

Block patterns

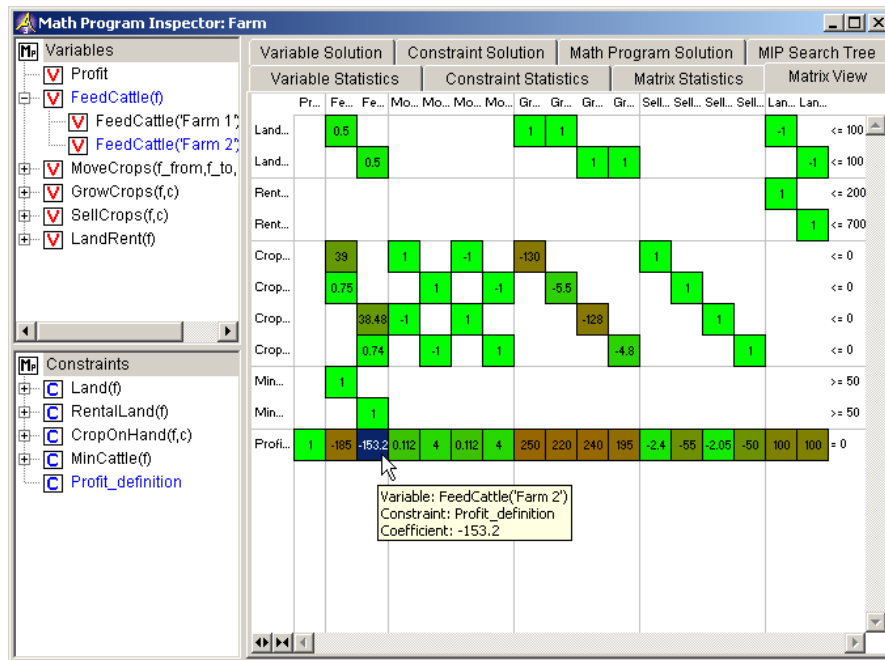


Figure 9.3: The matrix view (individual mode)

The value of the threshold mentioned in the previous paragraph is available as an AIMMS option with name *bad_scaling_threshold* and can be found in the **Project - Math program inspector** category in the **AIMMS Options** dialog box.

While holding the mouse inside a block, a tooltip will appear displaying the corresponding variables and constraints. In the symbolic view the tooltip will also contain the number of nonzeros that appear in the selected block. In the individual view the actual value of the corresponding coefficient is displayed.

Having selected a block in the block view you can use the right-mouse popup menu to synchronize the trees with the selected block. As a result, the current bookmarks will be erased and the corresponding selection in the trees will be bookmarked. Double-clicking on a block in symbolic mode will zoom in and display the selected block in individual mode. Double-clicking on a block in individual mode will center the display around the mouse.

When viewing the matrix in individual mode, *linear* coefficient values can be changed by pressing the F2 key, or single clicking on the block containing the coefficient to be changed.

9.2.4 Inspecting solution information

The tabs discussed so far are available as long as the math program has been generated. As soon as a solution is available, the next three tabs reveal more details about this solution.

Solution tabs

The Variable Solution tab shows the following seven columns

*Variable
Solution tab*

- Variable Name,
- Lower Bound,
- Value (i.e. solution/level value),
- Upper Bound,
- Marginal (i.e. reduced cost),
- Basis (i.e. *Basic*, *Nonbasic* or *Superbasic*), and
- Bound (i.e. *At bound* or *In between bounds*).

Column	Column Value	Column Margin...	Column...	Column Bound
Profit	14988.9	0	Basic	In between bounds
FeedCattle(Farm 1)	161.38	0	Basic	In between bounds
FeedCattle(Farm 2)	50	0	Basic	In between bounds
MoveCrops(Farm 1, Farm 2, Corn)	0	-0.224	Nonbasic	At bound
MoveCrops(Farm 1, Farm 2, Hay)	0	-8	Nonbasic	At bound
MoveCrops(Farm 2, Farm 1, Corn)	6293.83	0	Basic	In between bounds
MoveCrops(Farm 2, Farm 1, Hay)	14.8312	0	Basic	In between bounds
GrowCrops(Farm 1, Corn)	0	-24.85036	Nonbasic	At bound
GrowCrops(Farm 1, Hay)	19.3098	0	Basic	In between bounds
GrowCrops(Farm 2, Corn)	64.2018	0	Basic	In between bounds
GrowCrops(Farm 2, Hay)	10.7982	0	Basic	In between bounds
SellCrops(Farm 1, Corn)	0	-0.05037	Nonbasic	At bound
SellCrops(Farm 1, Hay)	0	-1.98157	Nonbasic	At bound
SellCrops(Farm 2, Corn)	0	-0.28837	Nonbasic	At bound
SellCrops(Farm 2, Hay)	0	-2.98157	Nonbasic	At bound
LandRent(Farm 1)	0	-6.60134	Nonbasic	At bound
LandRent(Farm 2)	0	-40.68844	Nonbasic	At bound

Figure 9.4: The variable solution

By clicking in the header of a column you can sort the table according to that specific column.

A similar view is available for the constraints in your mathematical program. The Constraint Solution tab contains the following five columns

*Constraint
Solution tab*

- Constraint Name,
- Value (i.e. solution),
- Marginal (i.e. shadow price),
- Basis (i.e. *Basic*, *Nonbasic* or *Superbasic*), and
- Bound (i.e. *Binding* or *Nonbinding*).

By default AIMMS will only store marginal solution values if explicitly specified in the **Property** attribute (through the *ReducedCost* or *ShadowPrice* property). An more convenient way to ensure that all marginal solution information is available to the math program inspector is by setting the option `Store_complete_solver_solution_tree` to *yes*. When the nonlinear presolver has been activated (by setting the `Nonlinear_presolve` option (in the Solvers General category) to *on*), the option `Store_nonlinear_presolve_info` has to be set *yes* to make sure that the math program inspector is able to display information about the reductions that have been achieved by the nonlinear presolver.

*Solution related
AIMMS options*

The Math Program Solution tab retrieves solution information about the mathematical program that has been solved. This information is similar to that in the AIMMS **Progress** window.

*Math Program
Solution tab*

The lower part of the information retrieved by this tab is used to display logging messages resulting from the **Bound Analysis** and **Unreferenced Identifiers** commands in the **Actions** menu.

*Logging
messages*

Whenever your linear model is a mixed-integer model, the solver will most probably use a tree search algorithm to solve your problem. During the tree search the algorithm will encounter one or more solutions if the model is integer feasible. Once the search is completed, the optimal solution has been found.

*Solving MIP
models*

With the MIP Search Tree tab you can retrieve branching information about the search tree. Only CPLEX 9.0 or higher provides this information. In addition the option `Show_branch_and_bound_tree` has to be set to *on* (before the solve) to instruct AIMMS to store search tree information during the solve.

*MIP Search Tree
tab*

The size and shape of the search tree might give you some indication that you could improve the performance of the solver by tuning one or more solver options. Consider the case in which the search tree algorithm spends a considerable amount of time in parts of the tree that do not seem interesting in retrospect. You might consider to use priorities or another branching rule, in an attempt to direct the search algorithm to a different part of the tree in an earlier stage of the algorithm.

*Improving the
search process*

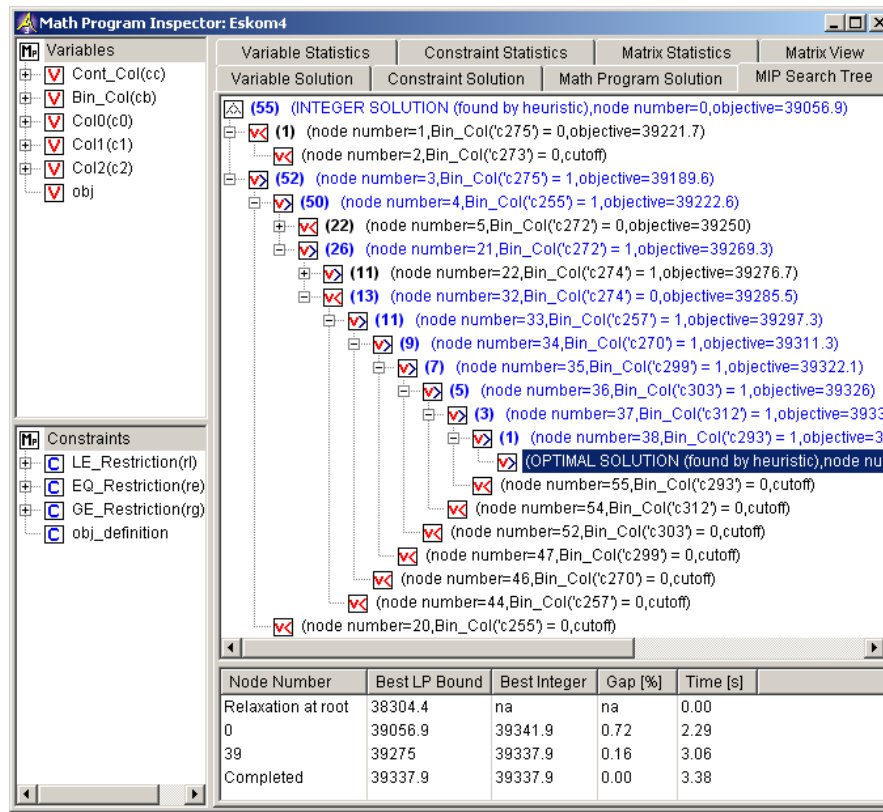


Figure 9.5: The MIP search tree

Because all structural and statistical information is kept in memory, displaying the MIP search tree for large MIPs might not be a good idea. Therefore, you are able to control to the and size of the stored search tree through the option `Maximum_number_of_nodes_in_tree`.

*Controlling
search tree
memory usage*

For every node several solution statistics are available. They are the sequence number, the branch type, the branching variable, the value of the LP relaxation, and the value of the incumbent solution when the node was evaluated. To help you locate the integer solutions in the tree, integer nodes and their parent nodes are displayed in blue.

*Search tree
display*

The lower part of the MIP Search Tree tab retrieves all incumbent solutions that have been found during the search algorithm. From this view you are able to conclude for example how much time the algorithm spend before finding the optimal solution, and how much time it took to proof optimality.

*Incumbent
progress*

9.2.5 Performing analysis to find causes of problems

One of the causes of a faulty model may be that you forgot to include one or more variables or constraints in the specification of your mathematical model. The math program inspector helps you in identifying some typical omissions. By choosing the **Unreferenced Identifiers** command (from the **Actions** menu) AIMMS helps you to identify

*Unreferenced
identifiers*

- constraints that are not included in the constraint set of your math program while they contain a reference to one of the variables in the variable set,
- variables that are not included in the variable set of your math program while a reference to these variables does exist in some of the constraints, and
- defined variables that are not included in the constraint set of your math program.

The results of this action are visible through the Math program solution tab.

In some situations it is possible to determine that a math program is infeasible or that some of the constraints are redundant even before the math program is solved. The bound analysis below supports such investigation.

*A priori bound
analysis*

For each linear constraint with a left-hand side of the form

$$\sum_j a_{ij}x_j$$

*Implied
constraint
bounds*

the minimum level value \underline{b}_i and maximum level value \overline{b}_i can be computed by using the bounds on the variables as follows.

$$\begin{aligned}\underline{b}_i &= \sum_{j|a_{ij}>0} a_{ij}\underline{x}_j + \sum_{j|a_{ij}<0} a_{ij}\overline{x}_j \\ \overline{b}_i &= \sum_{j|a_{ij}>0} a_{ij}\overline{x}_j + \sum_{j|a_{ij}<0} a_{ij}\underline{x}_j\end{aligned}$$

By choosing the **Bound Analysis** command (from the **Actions** menu) the above implied bounds are used not only to detect infeasibilities and redundancies, but also to tighten actual right-hand-sides of the constraints. The results of this analysis can be inspected through the Math Program Solution tab. This same command is also used to perform the variable bound analysis described below.

*Performing
bound analysis*

Once one or more constraints can be tightened, it is worthwhile to check whether the variable bounds can be improved. An efficient approach to compute implied variable bounds has been proposed by Gondzio, and is presented without derivation in the next two paragraphs.

Implied variable bounds ...

For i in the set of constraints of the form $\sum_j a_{ij}x_j \leq b_i$, the variable bounds can be tightened as follows.

... for \leq constraints

$$x_k \leq \underline{x}_k + \min_{i|a_{ik}>0} \frac{b_i - \underline{b}_i}{a_{ik}}$$

$$x_k \geq \overline{x}_k + \max_{i|a_{ik}<0} \frac{b_i - \underline{b}_i}{a_{ik}}$$

For i in the set of constraints of the form $\sum_j a_{ij}x_j \geq b_i$, the variable bounds can be tightened as follows.

... and \geq constraints

$$x_k \leq \underline{x}_k + \min_{i|a_{ik}<0} \frac{b_i - \overline{b}_i}{a_{ik}}$$

$$x_k \geq \overline{x}_k + \max_{i|a_{ik}>0} \frac{b_i - \overline{b}_i}{a_{ik}}$$

In case infeasibility cannot be determined a priori (e.g. using the bound analysis described above), the solver will conclude infeasibility during the solution process and return a phase 1 solution. Inspecting the phase 1 solution might indicate some causes of the infeasibility.

Phase 1 analysis

The collection of currently infeasible constraints are determined by evaluating all constraints in the model using the solution that has been returned by the solver. The currently infeasible constraints will be bookmarked in the constraint tree after choosing the **Infeasible Constraints** command from the **Actions** menu.

Currently infeasible constraints

To find that part of the model that is responsible for the infeasibility, the use of slack variables is proposed. By default, the math program inspector will add slacks to all variable and constraint bounds with the exception of

Substructure causing infeasibility

- variables that have a definition,
- zero variable bounds, and
- bounds on binary variables.

The last two exceptions in the above list usually refer to bounds that cannot be relaxed with a meaningful interpretation. However these two exceptions can be overruled at the symbolic level through the Analysis Configuration tab of the **Properties** dialog. These properties can be specified for each node at the first level in the tree. Of course, by not allowing slack variables on all variable

Adapting the use of slack variables

and constraint bounds in the model, it is still possible that the infeasibility will not be resolved.

Note that to add slacks to variable bounds, the original simple bounds are removed and (ranged) constraints are added to the problem definition.

*Slack on
variable bounds*

$$\underline{x}_j \leq x_j + s_j^+ - s_j^- \leq \overline{x}_j$$

After adding slack variables as described above, this adapted version of the model is referred to as the elastic model.

Elastic model

When looking for the substructure that causes infeasibility, the sum of all slack variables is minimized. All variables and constraints that have positive slack in the optimal solution of this elastic model, form the substructure causing the infeasibility. This substructure will be bookmarked in the variable and constraint tree.

*Minimizing
feasibility
violations*

Another possibility to investigate infeasibility is to focus on a so-called *irreducible inconsistent system* (IIS). An IIS is a subset of all constraints and variable bounds that contains an infeasibility. As soon as at least one of the constraints or variable bounds in the IIS is removed, that particular infeasibility is resolved.

*Irreducible
Inconsistent
System (IIS)*

Several algorithms exist to find an *irreducible inconsistent system* (IIS) in an infeasible math program. The algorithm that is used by the AIMMS math program inspector is discussed in Chinneck ([Ch91]). Note that since this algorithm only applies to linear mode, the menu action to find an IIS is not available for non-linear models. While executing the algorithm, the math program inspector

Finding an IIS

1. solves an elastic model,
2. initializes the IIS to all variables and constraints, and then
3. applies a combination of *sensitivity* and *deletion* filters.

Deletion filtering loops over all constraints and checks for every constraint whether removing this constraint also solves the infeasibility. If so, the constraint contributes to the infeasibility and is part of the IIS. Otherwise, the constraint is not part of the IIS. The deletion filtering algorithm is quite expensive, because it requires a model to be solved for every constraint in the model.

*Deletion
filtering*

The sensitivity filter provides a way to quickly eliminate several constraints and variables from the IIS by a simple scan of the solution of the elastic model. Any nonbasic constraint or variable with zero shadow price or reduced cost can be eliminated since they do not contribute to the objective, i.e. the infeas-

*Sensitivity
filtering*

bility. However, the leftover set of variables and constraint is not guaranteed to be an IIS and deletion filtering is still required.

The filter implemented in the math program inspector combines the deletion and sensitivity filter in the following way. During the application of a deletion filter, a sensitivity filter is applied in the case the model with one constraint removed is infeasible. By using the sensitivity filter, the number of iterations in the deletion filter is reduced.

*Combined
filtering*

When the underlying math program is not infeasible but unbounded instead, the math program inspector follows a straightforward procedure. First, all infinite variable bounds are replaced by a big constant M . Then the resulting model is solved, and all variables that are equal to this big M are bookmarked as being the *substructure causing unboundedness*. In addition, all variables that have an extremely large value (compared to the expected order of magnitude) are also bookmarked. Any constraint that contains at least two of the bookmarked variables will also be bookmarked.

*Substructure
causing
unboundedness*

When trying to determine the cause of an infeasibility or unboundedness, you can tune the underlying algorithms through the following options.

Options

- In case infeasibility is encountered in the presolve phase of the algorithm, you are advised to turn off the presolver. When the presolver is disabled, solution information for the phase 1 model is passed to the math program inspector.
- During determination of the substructure causing unboundedness or infeasibility and during determination of an IIS, the original problem is pertubated. After the substructure or IIS has been found, AIMMS will restore the original problem. By default, however, the solution that is displayed is the solution of the (last) pertubated problem. Using the option `Restore_original_solution_after_analysis` you can force a resolve after the analysis has been carried out.

9.3 A worked example

The example in this section is adapted from McCarl ([Mc98](#)), and is meant to demonstrate the tools that were discussed in the previous sections. The example model is used to illustrate the detection of infeasibility and unboundedness. In addition, the example is used to find the cause of an unrealistic solution.

This section

9.3.1 Model formulation

The model considers a collection of farms. For each of these farms several decisions have to be made. These decisions are

*A Farm
planning model*

- the amount of cattle to keep,
- the amount of land to grow a particular crop,
- the amount of additional land to rent, and
- the amount of inter-farm crop transport.

The objective of the farm model is to maximize a social welfare function, which is modeled as the total profit over all farms.

The following notation is used to describe the symbolic farm planning model.

Notation

Indices:

f, \hat{f}	<i>farms</i>
c	<i>crops</i>

Parameters:

C_{fc}^g	<i>unit cost of growing crop c on farm f</i>
C_f^c	<i>unit cost of keeping cattle on farm f</i>
C_c^m	<i>unit transport cost for moving crop c</i>
C_f^r	<i>rental price for one unit of land on farm f</i>
P_{fc}^s	<i>unit profit of selling crop c grown on farm f</i>
P_f^c	<i>unit profit of selling cattle from farm f</i>
L_f	<i>amount of land available on farm f</i>
Q	<i>amount of land needed to keep one unit of cattle</i>
Y_{fc}	<i>crop yield per unit land for crop c on farm f</i>
D_{fc}	<i>consumption of crop c by one unit of cattle on farm f</i>
M_f^c	<i>minimum required amount of cattle on farm f</i>
M_f^r	<i>maximum amount of land to be rented on farm f</i>

Variables:

p	<i>total profit</i>
c_f	<i>amount of cattle on farm f</i>
$m_{f\hat{f}c}$	<i>amount of crop c moved from farm f to farm \hat{f}</i>
g_{fc}	<i>amount of land used to grow crop c on farm f</i>
s_{fc}	<i>amount of crop c sold by farm f</i>
r_f	<i>amount of extra land rented by farm f</i>

The *land requirement* constraint makes sure that the total amount of land needed to keep cattle and to grow crops does not exceed the amount of available land (including rented land).

Land requirement

$$Qc_f + \sum_c g_{fc} \leq L_f + r_f, \quad \forall f$$

The total amount of rented land on a farm cannot exceed its maximum.

Upper bound on rental

$$r_f \leq M_f^r, \quad \forall f$$

The *crop-on-hand* constraint is a crop balance. The total amount of crop exported, crop sold, and crop that has been used to feed the cattle cannot exceed the total amount of crop produced and crop imported.

Crop-on-hand

$$\sum_{\hat{f}} m_{f\hat{f}c} + s_{fc} + D_{fc}c_f \leq Y_{fc}g_{fc} + \sum_{\hat{f}} m_{\hat{f}fc}, \quad \forall (f, c)$$

The cattle requirement constraint ensures that every farm keeps at least a pre-specified amount of cattle.

Cattle requirement

$$c_f \geq M_f^c, \quad \forall f$$

The total profit is defined as the net profit from selling crops, minus crop transport cost, minus rental fees, plus the net profit of selling cattle.

Profit definition

$$p = \sum_f \left(\sum_c \left(P_{fc}^s s_{fc} - C^g g_{fc} - \sum_{\hat{f}} C^m m_{f\hat{f}c} \right) - C_f^r r_f + (P_f^c - C_f^c) c_f \right)$$

Once the above farm model is solved, the math program inspector will display the variable and constraint tree plus the matrix block view as illustrated in Figure 9.3. The solution of the particular farm model instance has already been presented in Figure 9.4.

The generated problem

9.3.2 Investigating infeasibility

In this section the math program inspector will be used to investigate an artificial infeasibility that is introduced into the example model instance. This infeasibility is introduced by increasing the land requirement for cattle from 0.5 to 10.

Introducing an infeasibility

By selecting the **Infeasible Constraints** command from the **Actions** menu, all violated constraints as well as all variables that do not satisfy their bound conditions, are bookmarked. Note, that the solution values used to identify the infeasible constraints and variables are the values returned by the solver after infeasibility has been concluded. The exact results of this command may depend on the particular solver and the particular choice of solution method (e.g. primal simplex or dual simplex).

Locating infeasible constraints

Math Program Inspector: Farm	
Variable Solution	Constraint Solution
Math Program Solution	
MIP Search Tree	
Variable Statistics	Constraint Statistics
Matrix Statistics	Matrix View
Property of matrix defined by selection in...	Value
Number of individual variables	17
Number of individual constraints	11
Number of nonzeros	49
Density (%)	26.2032
Smallest nonzero coefficient	-185
Largest nonzero coefficient	250
Smallest nonzero coefficient (absolute)	0.112
Largest nonzero coefficient (absolute)	250
Coefficient scaling ratio	2232.14
Sum of all coefficients	507.244
Average coefficient (over nonzeros only)	10.3519
Population deviation (over nonzeros only)	80.9509
Skewness (over nonzeros only)	47.2989
Kurtosis (over nonzeros only)	285.561

Figure 9.6: An identified substructure causing infeasibility

By selecting the **Substructure Causing Infeasibility** command from the **Actions** menu a single constraint is bookmarked. In this example, one artificial violation variable could not be reduced to zero by the solver used, which resulted in a single infeasibility. Figure 9.6 indicates that this infeasibility can be resolved by changing the right-hand-side of the 'MinCattle' constraint for 'Farm 1'. A closer investigation shows that when the minimum requirement on cattle on 'Farm 1' is decreased from 50 to 30, the infeasibility is resolved. This makes sense, because one way to resolve the increased land requirement for cattle is to lower the requirements for cattle.

Substructure causing infeasibility

By selecting the **Irreducible Inconsistent System** command from the **Actions** menu, an IIS is identified that consists of the three constraints 'RentalLand', 'Land' and 'MinCattle', all for 'Farm 1' (see Figure 9.7).

Locating an IIS

The above IIS provides us with three possible model changes that together should resolve the infeasibility. These changes are

Resolving the infeasibility

1. increase the availability of land for 'Farm 1',
2. change the land requirement for cattle on 'Farm 1', and/or

The screenshot shows the 'Math Program Inspector: Farm' window. On the left, there are tree views for 'Variables' and 'Constraints'. The 'Variables' list includes Profit, FeedCattle(f), MoveCrops(f_from,f_to,c), GrowCrops(f,c), SellCrops(f,c), and LandRent(f). The 'Constraints' list includes Land(f), Land('Farm 1'), Land('Farm 2'), RentalLand(f), RentalLand('Farm 1'), RentalLand('Farm 2'), CropOnHand(f,c), MinCattle(f), MinCattle('Farm 1'), MinCattle('Farm 2'), and Profit_definition. The main window displays a table with columns: Variable, Va., Mar..., Basis..., and Bound Status. The table lists 20 variables with their respective values, maximums, basis types, and bound statuses.

Variable	Va...	Mar...	Basis...	Bound Status
Profit	0	0	Basic	In between bounds
FeedCattle('Farm 1')	0	773.85	Nonbasic	At bound
FeedCattle('Farm 2')	0	500.222	Nonbasic	At bound
MoveCrops('Farm 1','Farm 2','Corn')	0	0.224	Nonbasic	At bound
MoveCrops('Farm 1','Farm 2','Hay')	0	8	Nonbasic	At bound
MoveCrops('Farm 2','Farm 1','Corn')	12800	0	Basic	In between bounds
MoveCrops('Farm 2','Farm 1','Hay')	0	0	Basic	At bound
GrowCrops('Farm 1','Corn')	0	20.5	Nonbasic	At bound
GrowCrops('Farm 1','Hay')	100	0	Basic	In between bounds
GrowCrops('Farm 2','Corn')	100	0	Basic	In between bounds
GrowCrops('Farm 2','Hay')	0	3.064	Nonbasic	At bound
SellCrops('Farm 1','Corn')	12800	0	Basic	In between bounds
SellCrops('Farm 1','Hay')	550	0	Basic	In between bounds
SellCrops('Farm 2','Corn')	0	0.238	Nonbasic	At bound
SellCrops('Farm 2','Hay')	0	1	Nonbasic	At bound
LandRent('Farm 1')	0	17.5	Nonbasic	At bound
LandRent('Farm 2')	0	47.136	Nonbasic	At bound

Figure 9.7: An identified IIS

- decrease the minimum requirement on cattle on 'Farm 1'.

It is up to the producer of the model instance, to judge which changes are appropriate.

9.3.3 Investigating unboundedness

The example model is turned into an unbounded model by dropping the constraints on maximum rented land, and at the same time, by multiplying the price of cattle on 'Farm 1' by a factor 100 (representing a unit error). As a result, it will become infinitely profitable for 'Farm 1' to rent extra land to keep cattle.

By selecting the **Substructure Causing Unboundedness** command from the **Actions** menu four individual variables are bookmarked, and all of them are related to 'Farm 1'. Together with all constraints that contain two or more bookmarked variables these bookmarked variables form the problem structure that is subject to closer investigation. From the optimal solution of the auxiliary model it becomes clear that the 'FeedCattle' variable, the two 'GrowCrops' variables and the 'LandRent' variables tend to get very large, as illustrated in Figure 9.8.

Resolving the unboundedness requires you to determine whether any of the variables in the problem structure should be given a finite bounds. In this case, specifying an upper bound on the 'RentalLand' variable for 'Farm 1' seems a natural choice. This choice turns out to be sufficient. In addition, when

Introducing unboundedness

Substructure causing unboundedness

Resolving the unboundedness

Variable Statistics		Constraint Statistics		Matrix Statistics	
Constraint Solution		Math Program Solution		MIP Search Tree	
Matrix View		Variable Solution			
Variable	Value	Marginal	Basis	Bound	
Profit	0	0	Basic	In between bounds	
FeedCattle(Farm 1)	1000000	76216.4	Nonbasic	In between bounds	
FeedCattle(Farm 2)	50	0	Basic	In between bounds	
MoveCrops(Farm 1;Farm 2;Corn)	0	-0.224	Nonbasic	At bound	
MoveCrops(Farm 1;Farm 2;Hay)	0	-2.74861	Nonbasic	At bound	
MoveCrops(Farm 2;Farm 1;Corn)	6689.33	0	Basic	In between bounds	
MoveCrops(Farm 2;Farm 1;Hay)	0	-5.25139	Nonbasic	At bound	
GrowCrops(Farm 1;Corn)	299949	0	Basic	In between bounds	
GrowCrops(Farm 1;Hay)	136364	0	Basic	In between bounds	
GrowCrops(Farm 2;Corn)	67.2917	0	Basic	In between bounds	
GrowCrops(Farm 2;Hay)	7.70833	0	Basic	In between bounds	
SellCrops(Farm 1;Corn)	0	-0.29231	Nonbasic	At bound	
SellCrops(Farm 1;Hay)	0	-3.18182	Nonbasic	At bound	
SellCrops(Farm 2;Corn)	0	-0.53031	Nonbasic	At bound	
SellCrops(Farm 2;Hay)	0	-8.43321	Nonbasic	At bound	
LandRent(Farm 1)	936212	0	Basic	In between bounds	
LandRent(Farm 2)	0	-9.72062	Nonbasic	At bound	

Figure 9.8: An identified substructure causing unboundedness

inspecting the bookmarked variables and constraints on the Matrix View tab, the red color of the objective function coefficient for the ‘FeedCattle’ variable for ‘Farm 1’ indicates a badly scaled value.

9.3.4 Analyzing an unrealistic solution

The example model is artificially turned into a model with an unrealistic solution by increasing the crop yield for corn on ‘Farm 2’ from 128 to 7168 (a mistake), and setting the minimum cattle requirement to zero. As a result, it will be unrealistically profitable to grow corn on ‘Farm 2’.

Introducing an unrealistic solution

Once the changes from the previous paragraph have been applied, the solution of the model is shown in Figure 9.9. From the Variable Solution tab it can indeed be seen that the profit is unrealistically large, because a large amount of corn is grown on ‘Farm 2’, moved to ‘Farm 1’ and sold on ‘Farm 1’. Other striking numbers are the large reduced cost values associated with the ‘FeedCattle’ variable on ‘Farm 2’ and the ‘GrowCrops’ variable for hay on ‘Farm 2’.

Inspecting the unrealistic solution

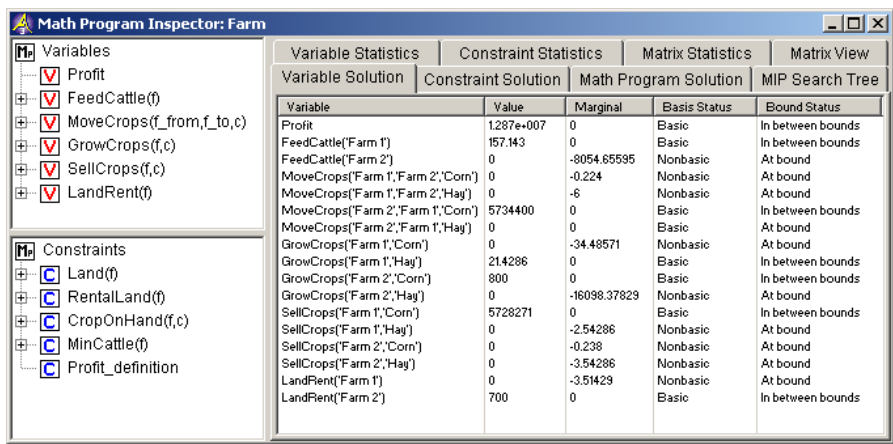


Figure 9.9: An unrealistic solution

When investigating an unrealistic solution, an easy first step is to look on the Matrix View tab to see whether there exist matrix coefficients with unrealistic values. For this purpose, first open the Matrix View tab in symbolic view. Blocks that are colored red indicate the existence of badly scaled values. By double clicking on such a block, you will zoom in to inspect the matrix coefficients at the individual level. In our example, the symbolic block associated with the 'GrowCrops' variable and the 'CropOnHand' constraint is the red block with the largest value. When you zoom in on this block, the data error can be quickly identified (see Figure 9.10).

Badly scaled matrix coefficients

A second possible approach to look into the cause of an unrealistic solution is to focus on the individual terms of both the primal and dual constraints. In a primal constraint each term is the multiplication of a matrix coefficient by the value of the corresponding variable. In the Math Program Inspector such a term is referred to as the *primal contribution*. Similarly, in a dual constraint each term is the multiplication of a matrix coefficient by the value of the corresponding shadow price (i.e. the dual variable). In the Math Program Inspector such a term is referred to as the *dual contribution*.

Primal and dual contributions ...

Whenever primal and/or dual contributions are large, they may indicate that either the corresponding coefficient or the corresponding variable value is unrealistic. You can discover such values by following an iterative process that switches between the Variable Solution tab and the Constraint Solution tab by using either the **Variable Statistics** or the **Constraint Statistics** command from the right-mouse popup menu.

... can be unrealistic

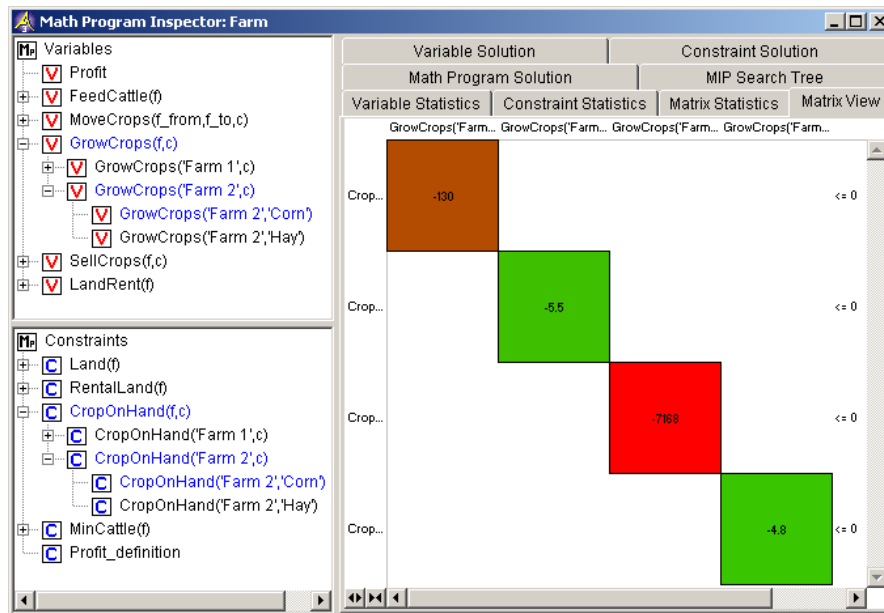


Figure 9.10: The Matrix View tab for an unrealistic solution

The following iterative procedure can be followed to resolve an unrealistic solution.

- Sort the variable values retrieved through the Variable Solution tab.
- Select any unrealistic value or reduced cost, and use the right-mouse popup menu to switch to the Variable Statistics tab.
- Find a constraint with an unrealistic dual contribution.
- If no unrealistic dual contribution is present, select one of the constraints that is likely to reveal some information about the construction of the current variable (i.e. most probably a binding constraint).
- Use the right-mouse popup menu to open the Constraint Statistics tab for the selected constraint.
- Again, focus on unrealistic primal contributions and if these are not present, continue the investigation with one of the variables that plays an important role in determining the level value of the constraint.
- Repeat this iterative process until an unrealistic matrix coefficient has been found.

You may then correct the error and re-solve the model.

In the example, the 'Profit' definition constraint indicates that the profit is extremely high, mainly due to the amount of corn that is sold on 'Farm 1'. Only two constraints are using this variable, of which one is the 'Profit' definition itself. When inspecting the other constraint, the 'CropOnHand' balance, it shows that the corn that is sold on 'Farm 1' is transported from 'Farm 2' to 'Farm 1'.

Procedure to resolve unrealistic solutions

Inspecting primal contributions

This provides us with a reason to look into the 'CropOnHand' balance for corn on 'Farm 2'. When inspecting the primal contributions for this constraint the data error becomes immediately clear (see Figure 9.11).

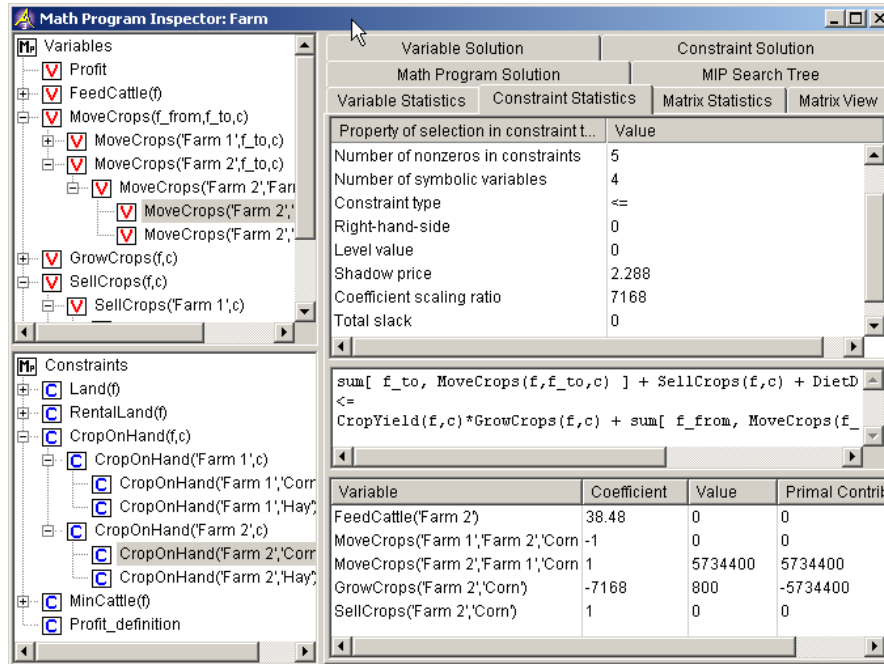


Figure 9.11: Inspecting primal contributions

The same mistake can be found by starting from an unrealistic reduced cost. Based on the large reduced cost for the 'FeedCattle' variable on 'Farm 2', the dual contributions indicate that the unrealistic value is mainly caused by an unrealistic value of the shadow price associated with the 'Land' constraint on 'Farm 2'. While investigating this constraint you will notice that the shadow price is rather high, because the 'GrowCrops' variable for corn on 'Farm 2' is larger than expected. The dual contribution table for this variable shows a very large coefficient for the 'CropOnHand' constraint for corn on 'Farm 2', indicating the data error (see Figure 9.12).

Inspecting dual contributions

The above two paragraphs illustrate the use of just primal contributions or just dual contributions. In practice you may very well want to switch focus during the investigation of the cause of an unrealistic solution. In general, the Math Program Inspector has been designed to give you the utmost flexibility throughout the analysis of both the input and output of a mathematical program.

Combining primal and dual investigation

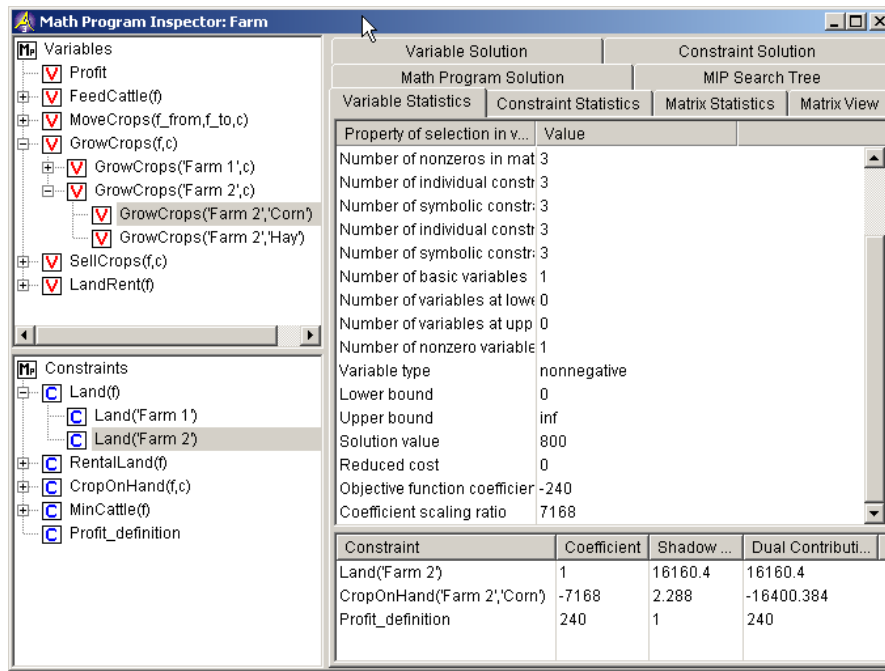


Figure 9.12: Inspecting dual contributions

Bibliography

- [Ch91] J.W. Chinneck, *Locating minimal infeasible constraint sets in linear programs*, ORSA Journal on Computing, vol. 3, (1991), no. 1, 157-168.
- [Mc98] B.A. McCarl, *A note on fixing misbehaving mathematical programs: Post-optimality procedures and GAMS-related software*, Journal of Agricultural & Applied Economics, vol. 30, (1998), no. 2, 403-414.