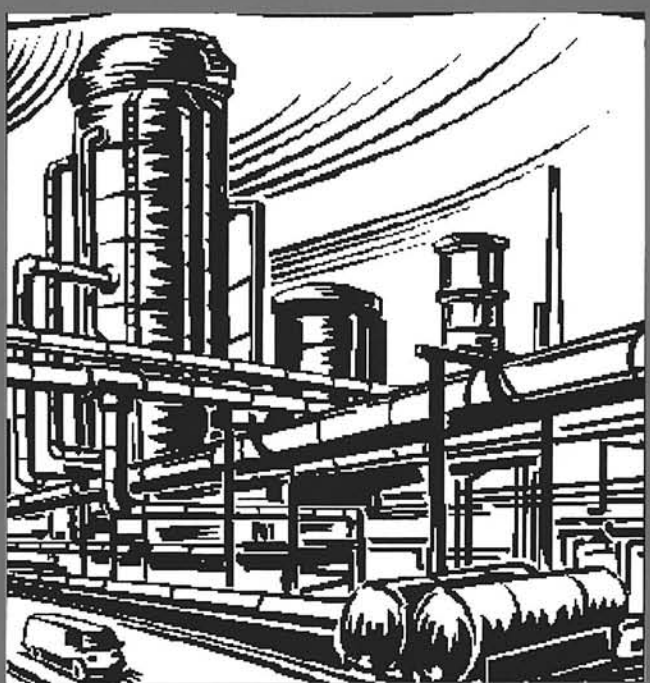
Refinery-planning with Gemms and PlanStar

Linear Programming applied to Mixed-integer and Nonlinear Models

M.R.B. Kreuk



WBBM Report Series 40

Delft University Press

Refinery-planning with Gemms and PlanStar

Linear Programming applied to Mixed-integer and Nonlinear Models

WBBM Report Series 40

WBBM
Delft University of Technology
Faculty of Information Technology and Systems
Department of Mathematics and Computer Science
Room ET 05.040
Mekelweg 4
2628 CD Delft, The Netherlands
Phone +31 15 278 16 35
Fax +31 15 278 72 55

Refinery-planning with Gemms and PlanStar

Linear Programming applied to Mixed-integer and Nonlinear Models

M.R.B. Kreuk

Delft University Press, 1998

The WBBM Report Series is published by:

Delft University Press
Mekelweg 4
2628 CD Delft, The Netherlands
Phone +31 15 278 32 54
Fax +31 15 278 16 61

Editors:

E. de Klerk
H. van Maaren

Delft University of Technology
Faculty of Information Technology and Systems
Department of Mathematics and Computer Science

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK DEN HAAG

Kreuk, M.R.B.

Refinery-planning with Gemms and PlanStar — Linear Programming applied to Mixed-integer and Nonlinear Models / M.R.B. Kreuk - Delft : Delft University Press. – Ill. – (WBBM Report Series 40)
ISBN 90-407-1766-4
NUGI 841
Trefw.: oil refinery, production planning

Acknowledgments

This report concludes my two-year post-MSc. ('TwAIO') education in mathematical design at the Delft University of Technology. It describes all my relevant findings of the past year, which I spent at ORTEC Consultants bv at its Oil&Gas department as a contractor for the Shell Oil Company.

In this respect, I'd first of all like to thank everyone on my examination committee:

drs R.J. Flik,	ORTEC Consultants bv, Gouda
dr. D.C. Ament,	ORTEC Consultants bv, Gouda
ir. P.J. Bost,	Shell International Oil Products bv, The Hague
dr. E. de Klerk,	University of Technology, Delft
dr. H. van Maaren,	University of Technology, Delft
prof. R.M. Cooke	University of Technology, Delft

and in specific my supervisors: Rob Flik, Etienne de Klerk and Hans van Maaren. I'd also like to mention Peter Bost explicitly for his useful remarks during my research, and for his involvement in the development of the enumeration approach for cargo-analysis, described by chapter 4.

Second, I owe my thanks to everyone at ORTEC's Oil&Gas department who all generate a great atmosphere to work and have fun in, and to all the people at SIOP (group OGBH/1) with whom I have worked, and will continue to work, with much pleasure.

Furthermore, I mustn't forget the first year of my education, in which I had a great time with my university colleagues. That means just about everybody at the department of Statistics, Stochastics and Operations Research, but especially my fellow TwAIOs and AIOs.

Milo Kreuk

Gouda, August 1998

Contents

Chapter 1

Introduction

Within the ORTEC Oil&Gas department, a group called 'HCL' has been installed to deal with all problems concerning hydrocarbon logistics, which will be discussed in the chapter 2. This group (myself included) functions mainly as a contractor for the Shell Oil Company, department Shell International Oil Products (SIOP).

Practically all of my work at ORTEC was based on a combination of two software systems, called Gemms and PlanStar. Gemms has been created by Shell, but since May 1996, it is being maintained and developed by ORTEC. It can create an LP-matrix, give it to an LP-solver and, from the solution, deduce the best possible strategy of purchasing and processing crude oil on some oil-refinery, given all its market-requirements, refinery structure, transport possibilities, and so forth. PlanStar is used for Plannning-data Storage and Retrieval, developed by ORTEC on contract for SIOP. Amongst others, it can be used to maintain Gemms-related data in a relational database. Very briefly put, the combination of Gemms and PlanStar works as follows: with PlanStar, all relevant planning-data is maintained, from this data, input for Gemms is created and Gemms is run. Then, Gemms' ASCII-file output can be presented more user-friendly via PlanStar (which calls the Business Objects software system to generate reports). Most Shell-refineries which use Gemms for their refinery-planning, don't fully operate with PlanStar yet, since it has only just been released.

To gain some insight in the functionality of Gemms, a few aspects of its linear modeling are also described in the second chapter. The description is a small draft from a confidential document which was written for both ORTEC and SIOP, which holds a full description of Gemms' data-handling and matrix-generation.

Gemms can only create linear models. However, some aspects in refinery-planning simply are nonlinear by definition. Some of those can be modeled by Gemms with penalty sequential linear programming (PSLP). The user only has to enter the relevant data and switch on Gemms' PSLP-algorithm. In order to gain insight in Gemms' nonlinearity-modeling, chapter 3 describes the implementation of the PSLP-algorithm, which depends on the software used to run it. Throughout this report, this software (either OMNI or GSolve), will be referred to as the 'recursion-guide', although this term only covers a small part of its actual functionality. Although PSLP seems to give satisfactory results in practice, one might expect that the spectacular development in nonlinear solvers should be able to give us more than this recursion, which is, mathematically speaking, quite an ancient solving-method. Therefore, a small refinery-model was built with several common nonlinearities both in Gemms and in the AIMMS modeling language. Gemms solved it with its sequential linear programming and AIMMS with its accompanied nonlinear solver CONOPT. The solving method of CONOPT is based on the commonly used Reduced Gradient

Method, which will be discussed briefly. Now, because of the fact that the model is mostly linear, and probably most importantly, because the objective function remains linear (both of which are also the case in actually used models), CONOPT performs worse than Gemms. The reader should note, however, that the goal of investigating Gemms' nonlinearities was *not* to explicitly mention if nonlinear solvers should be used to solve refinery-planning models, but just to find out exactly what functionality currently is covered by the recursion, and how it has been implemented. Therefore, the comparison between Gemms and AIMMS results is based on only one generic model which is supposed to be representative for actual models. Probably, AIMMS would perform relatively better if the objective function would become nonlinear, since the algorithm in most nonlinear solvers is based on the gradient of this function. Then, the nonlinear solver would adjust the initial solution in such a way that the objective value increases. The PSLP-algorithm in Gemms wouldn't guarantee any such thing, because it doesn't adjust the solution directly. Instead, it adjusts the matrix sequentially.

Although the possible relevance of nonlinear solvers is acknowledged by the author, it falls beyond the scope of this report and is left for future research. Still, with current Gemms models, there seems no need in using any other algorithm than PSLP.

The fourth chapter describes a possible extension of the functionality of PlanStar. PlanStar not only manipulates and controls data used for refinery-planning, but can also steer the optimization. Currently, the only true optimization is based on the refinery as a whole, including all crude oils which could possibly be purchased. After optimization, a package of crude oils is presented which gives the maximum refinery-margin (profit). However, users often wish to evaluate crude oils separately or in a small combination and examine what the results are if this crude is purchased. Furthermore, actual purchases are often based on cargoes, that is, a refinery oil-trader purchases a whole oil-tanker full with oil, and not, for example, just three quarters of a tanker. Currently, Gemms only uses continuous variables to represent oil-purchase. On first sight, especially for a mathematician, it looks like a typical example of simple (mixed) integer programming. However, once the wishes of the planners and traders in real life are investigated, another approach seems more appropriate: the enumeration approach.

Finally, chapter 5 gives my conclusions on the current implementation of the PSLP algorithms and on the possible future implementation of cargo-analysis in PlanStar.

Appendices are adopted to give examples of Gemms' PSLP algorithm and a piece-wise linear approximation of a nonlinear relationship and to describe required data-entries for nonlinear pooling and its implementation in the AIMMS modeling system.

Chapter 2

Hydrocarbon Logistics

The field of hydrocarbon logistics (HCL) covers all difficulties occurring with the logistics of oils and gases ('hydrocarbons'). Examples are (oil-)refinery-planning, refinery-scheduling and crude-valuation. ORTEC Consultants manages some of these aspects as a contractor for the Shell Oil Company, which comes down to maintenance, support and development of computer systems within the area of HCL.

This chapter first gives some of the logistic problems within the area of oils and gases and then briefly describes three software systems, managed by ORTEC, which deal with various of these. Sections 2.3 through 2.5 describe two of these, Gemms and PlanStar, in more detail because they are at the basics of all research presented in this report.

2.1 Hydrocarbon logistic problems

Many logistic problems are present at an oil-refinery and it's therefore, that many techniques developed in the field of operations research are used. One of the most widely used techniques is linear programming, which has proven to be very useful within the oil business because of it's easy understanding and little solving time. The first is important because decisions must be made based on LP-specific outcomes like marginal values, the latter because the people dealing with the LP's often do studies in which several model-runs (and solutions) are required in a short period of time.

Logistic problems occur at various levels of the refinery, like scheduling, planning and crude valuation.

The oil refinery

An oil refinery consists of several units (or factories) which all have their specific function to convert one set of components into another. For example, a crude distillation unit takes in crude oil and distillates it in order to allow other units to further process the distillates. The units which process (some of) the distillates coming from the crude distiller cannot process the crude oil itself. When several units have processed that what started out as crude oil, components result which can be blended (mixed) to obtain final products like kerosene, LPG and gasoline.

Not all crude oils can be processed by every refinery. Some are just to difficult to handle for refineries with only one simple crude distillation unit. Every refinery differs from another in its complexity (e.g. number of units present). Of course, the

more complex and complete the refinery, the more crude oils it can process. One of the largest oil-refineries in the world is the Shell Oil refinery in Pernis, Holland.

Typical logistic questions at a refinery are:

- which crude oils can be processed by the crude distillation unit?
- Which units are available in the coming period?
  (Some units may have to be shut down temporarily.)
- Which external transport-possibilities must be used (e.g. pipeline, barge)?
- Which crude oils are worth the most to this refinery and should therefore be purchased (assuming that they can indeed be processed)?
- How can the already purchased (and soon incoming) crude oils be processed?
- How should all components be routed through all units?
- How should components be blended to the final products?

2.1.1 Scheduling

Every refinery has its scheduling-department. This department has to make sure that all incoming crudes are processed (or transported) by this refinery and that all market requirements (for example, contractual kerosene supplies) are met. This means leading incoming crudes through the right pipelines at the right time, guiding them through the right units at the right time and blending the resulting components together in the right way to obtain final products which meet market requirements, as well as quality-demands. Scheduling is based on short-term (usually several days).

2.1.2 Planning

A refinery-planning department needs to advise the oil-trader which crude oils to purchase in order to maximize the refinery margin. Planning-studies may also be used to determine the 'health' of a refinery (study possible gains and losses when applying some specific strategy), or the consequences of unit-shutdowns or breakdowns. Planning is either a medium or a long-term study, depending on its purpose (e.g. long-term studies are used for strategic planning). Medium-term means from about two weeks up to a year, and long-term usually means more than a year.

2.1.3 Crude valuation

Crude valuation is used to determine the value of some crude oil, relative to all other available crudes. The third chapter of this report describes the possible implementation of crude-evaluation via cargo analysis, using PlanStar.

The right method for crude valuation is hard to detect and discussions about it are still well alive. Which method to use depends on the reason for the crude valuation. For example, when a new crude oil is found during experimental drillings, a marginal refinery may be the best way to model the situation to estimate the crude value, whereas the value for a specific refinery may be estimated best by cargo analysis.

2.2 Operational hydrocarbon logistic tools

This section describes some of the software programs maintained and developed by ORTEC Consultants, which are operational at various refineries of the Shell Group.

2.2.1 CAS: scheduling

CAS is a Unix-based refinery-scheduling system. The actually purchased crude diet is known and must be processed in the best possible way. That is, products must be obtained according to their demands (e.g. much gasoline must be blended in the first three days due to high demands) and, given the purchased crudes, CAS can optimize when to blend the products as well as how much of each and every component should be used for blending.

CAS offers quite a user-friendly graphical interface and has a more detailed representation of the refinery than the model used for planning.

2.2.2 Gemms/PlanStar: planning

Gemms/PlanStar is a combination of two systems which is used as a refinery-planning tool. Simplified, one could say that Gemms handles matrix generation and that PlanStar handles data-manipulation, data-control and optimization-control. Gemms runs under MS-DOS, whereas PlanStar is a Windows 95 and Windows NT compliant (database) application.

Gemms uses a Linear Programming model to decide, amongst others, what amounts and types of crude oils (usually abbreviated to 'crudes') a refinery should purchase and what amount and type of product it must sell. The model maximizes profit subject to several constraints. Gemms is completely data-driven, which means that the framework of all models produced and optimized by Gemms is the same, but size and complexity depend completely on the entered data.

The constraints in the model reflect refinery- and market-restrictions, such as quality demands (e.g. the octane number of gasoline), unit capacity limits (such as maximum throughput) and other specifications that restrict possible purchases, processes and supplies. The model may be multi-period, that is, crude can be purchased at the beginning of several time-periods and for each of the periods available in the model, a decision on purchases and/or refinery-processing must be made. Section 2.3 describes the Gemms-model in more detail and sections 2.4 and 2.5 give some more comments on the Gemms and PlanStar systems.

Some of the differences between CAS and Gemms are:

- Gemms plans for a long period, CAS schedules for a short period.
- The Gemms refinery-model (description of units) is less accurate than the CAS model, because many details are irrelevant for medium- or long-term planning (such as tanks).

- CAS uses the timing-aspect required for scheduling, whereas Gemms doesn't. For example, suppose that components A and B can blend to final product P and suppose that component A is produced in the first week and component B is produced in the second. This cannot be considered by Gemms, which simply allows components A and B to blend together to product P. In CAS, however, this wouldn't possible if component A cannot be held in a tank, because A and B wouldn't be available for blending at the same time (individual blending would remain possible, of course).
- CAS is mainly simulation, with only the blending process being optimized as an LP model, whereas Gemms runs a complete LP maximization.

2.2.3 CrudeVal: crude valuation

The software package CrudeVal is used for crude valuation and runs under MS-DOS. It is mainly used to estimate the value of some new discovered crude-type, based only on its property-values obtained from some sample (or crude assay). It can also estimate the relative value of a specific crude-type for a specific refinery with respect to other crudes.

2.3 Details of the linear Gemms model

This section describes some parts of refinery-planning models for the Gemms. A complete overview of the Gemms functionality and modeling is confidential and can be found in Kreuk [8].

The following description of Gemms linear modeling should help the reader enough to understand chapters 3 and 4. The Gemms linear model is quite powerful (especially because it can be solved rapidly) and can even be used (sequentially) to efficiently model various nonlinearities, such as specific Mixed Integer models (chapter 4) and nonlinear constraints (chapter 3).

An overview of the model which Gemms generates for its specified LP-solver reads:

maximize Profit
s. t. constraints:
1 non period-specific
2 non refinery-specific
3 unit
4 balance
5 quality
6 limit
7 pooling
8 user defined

This model is completely data-driven. That is, the generation of constraints and the numerous variables and coefficients which they contain, depends completely on the data-dictionary (a set of data-tables presented by the user as an ASCII-file in OMNI-format). Therefore, the user determines size, complexity and exact formulation of the model at all times.

2.3.1 Profit maximization

The total profit of a refinery-complex is defined as the sum of incomes ('netbacks') minus the sum of costs of all refineries in all periods available to the system. This section describes Gemms' objective function in some detail.

Income = netbacks of [{sales} + {product exchange} + others]
Cost = costs of [{product exchange}] + {stocks} + {purchases} + {operating} + {transfers} + {lead} + {blend} + others]

The largest part of netbacks is obtained by selling products (e.g. gasoline) and/or utilities (e.g. electricity) to markets or refineries. Netbacks are given in money units (US$ by default) per unit of sale (m^3, barrels(bbls) or metric tons(mt)).

Much of the total cost is due simply to the purchase of crudes. Stock costs are only present in multi-period models, where they represent the missed interest-revenues per stock-entity for a given period and component or product. They are defined as $(val \cdot d \cdot i/100)/365$, with *val* the component- or product-value for stock cost calculation for the period, *d* the length of the period in days and *i* the interest rate.

Operating costs occur from activating units for processing, transportation costs result from transferring products within a set of refineries and markets and lead costs are defined as the given costs per gram Pb/Lt per m^3 times the obtained lead level of a product. Blend costs are due to costly mixing several components to produce a final grade like gasoline or LPG.

Other parts of the objective function are taxes, costs of additives, penalties for infeasibilities and penalties which are used for a recursive-pooling procedure (see chapter 3).

2.3.2 Variables

Thousands of variables are usually present in Gemms models, though many have the same interpretation, such as:
- amount of component in stock,
- amount of crude to be purchased,
- amount of product which should be transferred between refineries and/or markets,
- amount of product which will be sold to some market,
- amount of component blending to a specific product,
- obtained property-values for a specific product.

Most variables depend on the refinery, the product and the period in the model. The product property-values deserve some further explanation, since they are a large part of chapter 3. For each component which can blend to a product, some property-values are known (e.g. Sulfur percentage, cloud point or octane number). For the obtained products, these aren't known beforehand, since they depend on the components which actually blend to them. Restrictions are placed on the product-property values (e.g. a maximum octane number of 98 for premium gasoline), so the generated variables are required to specify such constraints. Furthermore, the generated variable is defined equal to the mixture of the blending components' property-values, which usually is just their weighted average. Sometimes, however, this average doesn't give the correct resulting property-value and specific blend-rules are required. For example, if API-property-values are given instead of densities, they are converted to densities via the formula DENS=141.36/(API + 131.5). Then, the DENS property is used in the linear blending constraints.

2.3.3 Constraints

To demonstrate some of the numerous constraints (usually thousands), this section gives two general balance constraints, which determine a large part of the actual model.

The first constraint makes sure that crudes which are purchased are also either processed by the refinery's units or are put into stock. The second defines balance for a refinery's incoming and outgoing components. Some of Gemms' notation is used to allow the reader to gain more insight in Gemms' matrix generation and to aid future Gemms developers. Comments are stated between parentheses.

Crude-balance

For all crude-types X, periods P and refineries R:

$$
\begin{aligned}
&- A(R)(P)(X)00 - \Sigma_{(I)}A(R)(P)(X)00(I) \\
&\qquad \{\text{normal purchase and purchase by tier (I)}\} \\
&- O(R)1(X)00 \qquad \{\text{opening stock}\} \\
&+ F(R)(P)(X)00 \qquad \{\text{closing stock}\} \\
&+ \Sigma_{(UN)}\Sigma_{(MOD)}P(R)(P)(UN)(MOD) \\
&\qquad \{\text{total amount of crude oil processed by the available units at the specific refinery}\} \\
&+ P(R)(P)(X)00 - N(R)(P)(X)00 \\
&\qquad \{\text{feasibility variables: surplus minus deficit}\} \\
&= 0
\end{aligned}
$$

In short: "the amount of crude processed at (or 'going out of') the refinery minus the amount of crude coming into the refinery equals zero".

In the generated LP-matrix, every letter or word between round brackets is replaced by the appropriate period-number (P), refinery (R), crude (X), tier (I), unit (UN) or

unit-mode of operation (MOD). The notation '(X)00' is predefined Gemms-notation to differentiate crudes from other components.

Summations over *I*, *UN* and *MOD* are only over those tiers, units and processing-modes which can handle crude *X*, and all variables are measured in kilotons.

Opening stocks are given by the Gemms or PlanStar data, but intakes, purchases, closing and inter-period stocks are decision variables. All of these may be bounded (and possibly fixed) by other data-tables.
Finally, so-called feasibility-variables were mentioned in the crude balance-equation. The user can choose whether or not these variables should be generated. If generated, they get a very large penalty (by default 9999,000 US dollars) in the objective function. Therefore, they will only hold a positive value in the optimal solution if no other (and therefore originally feasible) solution can be found without using them. This helps the user to deal with infeasible models.

Component-balance

Many components (here: other than crudes), can be transferred between refineries by various means. They can also be held in stock or blended to products (e.g. component butane can blend to LPG). For each component, all of these possibilities, including transportation costs, are entered in the OMNI or PlanStar data-tables and converted by Gemms to the LP-matrix. The resulting constraints can be much more complex than those for crudes.

A general (though not complete) Gemms description for component-balance constraints reads: for all components *XXX*, refineries *R* and periods *P*:

F(R)(P)(XXX) - O(R)1(XXX)
{closing minus opening stocks}
$- \Sigma_{(UN)}\Sigma_{(MOD)}$ *YIELD*·{P(R)(P)(UN)(MOD)}
{ component yields, that is, how much of component XXX is produced at unit UN of refinery R at period P, with UN processing at mode MOD. Summation over all units and modes of operation available to the refinery to obtain the total amount of produced component XXX}
$+ \Sigma_{(PR)}$Y(R)(P)(PR)(XXX)
{ component-weight blended 'free' to product PR (not by recipe) }
$+ \Sigma_{(PR,I)}$(r/100)·D(R)(P)(PR)(I)
{ recipe blending:
(r/100) is the fixed fraction of weight of component XXX which blends to product PR via so-called recipe blending (see example 2.1)}
$+ \Sigma_{(M)}$I(R)(P)(XXX)(F)(M) - $\Sigma_{(M)}$I(F)(P)(XXX)(R)(M)
{ component transfers out of refinery *R* minus component transfers into the refinery, summed over all transport means *M* (pipe, barge, etc.) }
+ P(R)(P)(XXX) - N(R)(P)(XXX)
{ Feasibility surplus minus deficit.}
= 0.

In short: "the amount of component going out of the refinery minus the amount of component coming into the refinery equals zero".

Example 2.1 Component recipe-blending on volume

Suppose that the following OMNI table is entered (or otherwise represented by the PlanStar database):

```
TABLE A1RECI
        >        V1PR
        CP1       70
        CP2       10
        CP3       20
```

Here, V1PR means 'based on (V)olume (here m^3), recipe-nr 1, blending to product PR'. Furthermore, CP1, CP2 and CP3 are defined components and product PR is assumed to be sold on weight.

Gemms now generates a variable 'D(R)(P)(PR)(I).' (here: 'DA1PR1.') to represent the total weight of blended product PR by recipe 1. This variable will be present in the weight-balance equations of all components CP1, CP2 and CP3, and in the weight-balance of product PR. In the product's balance it will simply have a coefficient equal to -1, representing produced weight. In the components' balances, its coefficient may differ for each and, for component i, is calculated as $\frac{r_i \rho_i}{\sum_i r_i \rho_i}$, with ρ_i defined as the density of component CPi (used to convert m^3 to metric tons), and r_i the required volume-fraction of component i in the recipe-blend (e.g. r_1=0.70). (So $\Sigma_i r_i \rho_i$ equals the resulting blend-density.)

2.4 Running the Gemms system

2.4.1 Building

First of all, Gemms users build the model by entering data at specific tables. For Gemms (standalone), these tables are represented by one or more ASCII-files in OMNI-format. For PlanStar, an Oracle database is available and data can be entered in windows with table-formats. Then, before Gemms is run, PlanStar must be instructed to automatically generate ASCII-files which will hold all relevant OMNI-tables. Finally, when all data is available in OMNI-format, Gemms is run to convert all tables to an LP-matrix in MPS-format, which can be read by all prominent LP-solvers.

2.4.2 Solving

The user can choose which solver to use for the optimization (three solvers are usually available: HSLP, XPRESS and OSL). The selected solver is called with the matrix-file as input and the results are saved to an ASCII-file and, if selected in PlanStar, to the database.

2.4.3 Reporting

Various reports can be generated by Gemms. Of course, the objective value can be reported, as well as the 'optimal crude diet', that is the package of crude purchases which maximizes the refinery-margin (profit). But also other aspects can be shown, such as the unit-capacity usage ('which units run at their maximum capacity?'), obtained product-properties and product-composition ('which components (and how much) should blend to the product?').

If the model is infeasible, Gemms shows which constraints are likely to cause the problems.

Gemms can only generate reports in ASCII-files. Via the software systems PlanStar and Business Objects, more user-friendly (ad hoc) reports can be created.

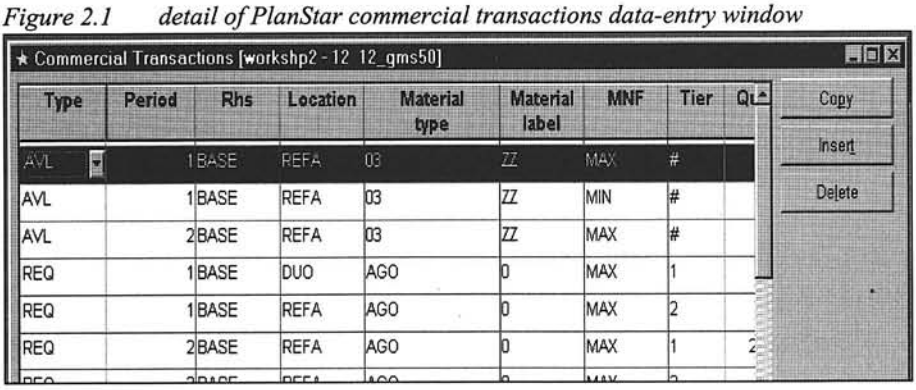
2.5 Running the PlanStar system

This section briefly describes the PlanStar system, and is adopted in this report only to show its connection to the Gemms system, which is required to understand the additional functionality developed in chapter 4.

The PlanStar system is a tool for refinery-planning datamanagement and control. All data is kept in a relational Oracle database. This data can be changed in various data-entry windows within PlanStar. One of these windows allows the user to enter or update commercial transactions (see figure 2.1), such as possible product-exports or crude purchases. These transactions are relevant for chapter 4.

PlanStar can be used to run the Gemms system. If the user chooses to run a Gemms LP model, PlanStar converts all data-tables from its database to OMNI-tables in various ASCII-files which Gemms uses as data-input. Then, if required, Gemms runs its building, solving and reporting steps as described in the previous section.

Figure 2.1 detail of PlanStar commercial transactions data-entry window



Chapter 3

Optimization with nonlinear constraints

This chapter describes how nonlinearities are handled by the linear programming-based planning-tool Gemms, and compares it with some results based on nonlinear programming.

3.1 Introduction

Based on data-tables, Gemms builds an LP-model with a linear objective function, which must be maximized subject to linear constraints. The model is written to an ASCII-file in 'MPS'-format, a standard format which can be read by all prominent LP-solvers.

Although Gemms only builds linear models, users can model some nonlinearities, which Gemms solves by a sequence of linear models, using an algorithm called Penalty Sequential Linear Programming (PSLP). Sections 3.2 and 3.3 give an introduction to nonlinear optimization, and briefly describe some general characteristics of algorithms present in nonlinear solvers, some specific techniques often applied by such solvers, and the PSLP algorithm.

Further sections go into all detail of the implemented nonlinearities in Gemms. They describe all nonlinearities which can be modeled within this system, the implementation of the PSLP algorithms which handle them, and a small test on the performance of Gemms' PSLP algorithm versus the performance of a Gradient-technique used by a nonlinear solver within the modeling system AIMMS. Finally, conclusions are presented on resulting performance and on possible advantages or disadvantages of using a nonlinear solver.

3.2 Solving nonlinearities with nonlinear solvers

This report doesn't have the intention to discuss possible techniques of solving nonlinear optimization problems in great detail. Many of them have already been extensively discussed (for example, see Abadie [1] or Abadie and Carpentier [2]). The dissertation of Schweigman [12] is specifically devoted to solving a nonlinear objective function with nonlinear constraints.

First, some general characteristics of the commonly implemented algorithms within nonlinear solvers are given. Then, some specific (gradient) methods of solving nonlinear optimization problems are briefly described (see also Greene [6] and Abadie [1]).

3.2.1 General characteristics of algorithms

Most nonlinear optimization algorithms are implemented to deal with an optimization problem which has a nonlinear objective function and linear constraints. Some algorithms can also handle nonlinear constraints.

The easiest solvable nonlinear maximization would be a quadratic objective function $F(\theta) = a + b'\theta - \frac{1}{2}\theta' C\theta$, with C a positive definite matrix and no constraints. The first-order condition for a maximum is $\frac{\partial F(\theta)}{\partial \theta} = b - C\theta = 0$, a linear set of equations with unique solution $\theta = C^{-1}b$. Note that this solution has a closed-form, so it can be computed directly for any a, b and C.

In a more general situation, the resulting equations $\frac{\partial F(\theta)}{\partial \theta} = 0$ aren't linear and cannot be solved explicitly for θ, leading to a desire for techniques which systematically search for a solution. Most of such techniques are based on an iterative procedure: starting with some initial solution θ_0, if a solution after iteration t (=0,1,2,3,...) isn't locally optimal, it is adjusted using a direction vector Δ_t and step-size λ_t to $\theta_{t+1} = \theta_t + \lambda_t\Delta_t$.

3.2.2 Gradient methods in general

The most widely accepted algorithms to deal with nonlinear optimization are gradient methods, in which $\Delta_t = H_t g_t$, where H_t is a positive definite matrix and g_t is the gradient of $F(\theta_t)$: $g_t = g(\theta_t) = \frac{\partial F(\theta_t)}{\partial \theta}$. The motivation of using such a method is as follows: let $F_{t+1} = F(\theta_{t+1}) = F(\theta_t + \lambda_t\Delta_t)$ and consider its first-order Taylor series approximation around $\lambda=0$: $F_{t+1} \approx F_t + \lambda_t g(\theta_t)'\Delta_t$. Therefore, $F_{t+1}-F_t \approx \lambda_t g_t'\Delta_t = \lambda_t g_t' H_t g_t$ for gradient methods. Now, if the gradient at iteration t isn't 0 and λ_t is small enough, $F_{t+1}-F_t$ must be positive (recall that H_t is assumed to be positive definite), which means that a new gradient-type iteration step will lead to an increase in the objective function. Note that for given vectors Δ_t and θ_t, a secondary optimization is required to find the optimal step-size λ. Usually, however, this size is approximated by a very simple algorithm. The procedure is stopped if the gradient is zero, or if F_{t+1} is close enough to F_t. Then, the resulting θ is considered the locally optimal solution.

Two generally used gradient methods are Newton's method (though usually with some adjustments) and the Generalized Reduced Gradient method. Quite often, parts are solved with the simplest technique available: the steepest ascent method.

3.2.3 Steepest ascent method

The easiest algorithm to use for a nonlinear maximization problem is the steepest ascent method (or descent for minimization), which uses H = I as part of the gradient method, so that $\Delta=g$. In theory, the greatest advantages of the method are that its

search direction is the one with the greatest increase of F, and that the optimal λ, following from $\frac{\partial F(\theta+\lambda\Delta)}{\partial\lambda}=0$, can be determined (at least near the maximum) by $\lambda=\frac{-g'g}{g'Gg}$, with $G=\frac{\partial^2 F(\theta)}{\partial\theta\partial\theta'}$, the Hessian of F. Therefore, the steepest ascent iteration is $\theta_{t+1}=\theta_t-(\frac{g_t'g_t}{g_t'G_tg_t})g_t$.

This method usually performs quite bad because computing the Hessian can be a burdensome task. Furthermore, if G_t isn't negative definite (which is likely if θ_t is far from the maximum), the iteration may diverge.

The steepest ascent (or descent) method is often just part of a nonlinear solving technique.

3.2.4 Newton's method

The original Newton's method is quite simple but can be very effective, especially for quadratic objective functions, in which it reaches the optimum in only one iteration from any starting point. If the problem is approximately quadratic, Newton's method usually still works well, and in fact, if a maximum must be found of a globally concave function, it has proven to be one of the most efficient algorithms available. However, if the problem isn't approximately quadratic or if the initial solution lies far from the optimum (and the objective isn't quadratic), it may perform poorly and even fail to converge at all. For such situations, various improvements are found in Greene [6]. Here, only the basics are discussed.

As described earlier, the set of equations to be solved is $\frac{\partial F(\theta)}{\partial\theta}=0$. The basis for Newton's method is the first-order Taylor series expansion of this set. Around an arbitrary θ_0, this expansion yields $\frac{\partial F(\theta)}{\partial\theta}\approx g_0+G_0(\theta-\theta_0)=0$, with G defined as $G=\frac{\partial^2 F(\theta)}{\partial\theta\partial\theta'}$ and subscript 0 meaning evaluation at θ_0. The iterative equation for θ then reads $\theta_{t+1}=\theta_t-G_t^{-1}g_t$, so, for Newton's method, the elements of the general gradient method are defined as: $H=-G^{-1}$, $\Delta=-G^{-1}g$, and $\lambda=1$.

3.2.5 Generalized Reduced Gradient method

The Generalized Reduced Gradient (GRG) method for nonlinear programming was first introduced by Abadie and Carpentier [2]. For a detailed description of the method, as well as an application, the reader is referred to Abadie [1, pp.191-210]. The key steps, however, are given below.

The GRG method is implemented in the nonlinear solver CONOPT (applied by AIMMS) which is used in the research described at the end of this chapter. Because of the usage of this solver, the method is described here with the general implementation within CONOPT.

The key steps of the algorithm are

1. Initialize the problem and find a feasible solution.
2. Compute the Jacobian J of the constraints.
3. Select a set of n basic variables x_b, such that B, the submatrix of basic columns from J, is nonsingular. Factorize B. The remaining variables x_n are referred to as nonbasic.
4. Solve $B^T p = \frac{\partial f(x)}{\partial f(x_j)}$ for the multipliers p.
5. Compute the reduced gradient r, with $r = df/dx - J^T p$.
6. If r projected on the bounds is small, then stop and the current solution may be considered locally optimal.
7. Otherwise, select the set of superbasic variables x_s as a subset of the nonbasic variables, which can be profitably changed, and find a search direction d_s for the superbasic variables, based on r_s and possibly on some second order information.
8. Perform a line search in the direction d. For each step, x_s is changed and x_b subsequently adjusted to satisfy $g(x_b, x_s)=b$ in a pseudo-Newton process using the factorized B from step 3.
9. Go to 2.

The first step, finding a feasible solution, is solved by Newton's method (with some modifications), but can be just as difficult as finding an optimum. If the objective and constraints are almost linear in step 7, the steepest ascent (or descent) method is used to find a search direction. See [4, pp. 475-477] for more details.

3.2.6 Sequential Linear Programming

Sequential linear programming (SLP) methods replace a nonlinear objective function by a succession of linear approximations. For linearly constrained optimization problems, these approximations allow repeated application of linear algorithms. These algorithms are particularly suitable for linearly constrained optimization problems, but can sometimes, by the use of suitable linear approximations, be extended for problems with nonlinear constraints. Other, currently popular, sequential algorithms are quadratic, which approximate the objective function (or constraints) by a succession of quadratic functions. In practice, these have proven to be more efficient than their linear counterparts, but they can't be applied by the Gemms refinery-planning tool, whereas SLP can. Because of this fact, and the fact that the principle of both methods is the same, only the linear methods are discussed here. A survey of sequential quadratic approximation methods is found in Powell [10].

The Frank-Wolfe Sequential Linear Approximation Algorithm

At the basis of SLP lies the algorithm developed by Frank and Wolfe in 1956, originally designed to solve quadratic problems, but easily adapted to the case of a general concave objective function. The summary below is taken from Hillier and Lieberman [7]. An example of the algorithm is found in the same reference, pp. 600-602.

Initialization: Find a feasible initial trial solution $x^{(0)}$ and set k=1.
Iteration:

1. For j=1,2,...,n, evaluate the partial derivatives of the objective function,
$$c_j := \frac{\partial f(x)}{\partial f(x_j)} \text{ at } x = x^{(k-1)}$$
2. Find an optimal solution $x^{(k)*}$ for the following linear programming problem:
$$\text{Maximize} \quad g(x) = \sum_{j=1}^{n} c_j x_j$$
subject to
$$Ax \le b \quad \text{and} \quad x \ge 0$$
3. For the variable t ($0 \le t \le 1$), set $h(t) = f(x)$, for $x = x^{(k-1)} + t(x^{(k)*} - x^{(k-1)})$, such that $h(t)$ gives the value of $f(x)$ on the line segment between $x^{(k-1)}$ and $x^{(k)*}$. Use some search procedure to maximize $h(t)$ over $0 \le t \le 1$, and set $x^{(k)}$ equal to the corresponding x. Check the convergence criterion.

Convergence criterion
If $x^{(k-1)}$ and $x^{(k)}$ are sufficiently close, stop and use $x^{(k)}$ (or some extrapolation of $x^{(0)}$, $x^{(1)}$, ..., $x^{(k-1)}$, $x^{(k)}$) as the estimate of an optimal solution. Otherwise, reset k=k+1 and go to step 1 of the next iteration.

The maximization in step 2 follows from the first-order Taylor expansion, which is used to approximate $f(x)$. That is, $f(x)$ is approximated at some point x' as

$$f(x) \approx f(x') + \sum_{j=1}^{n} \frac{\partial f(x')}{\partial x_j}(x_j - x_j') = f(x') + \nabla f(x')(x - x')$$

with the partial derivatives evaluated at x=x'. Because $f(x')$ and $\nabla f(x')x'$ have fixed values, they can be dropped to give an equivalent linear optimization problem with $g(x)$ as the objective function.

3.3 Solving nonlinearities with Gemms: PSLP

Gemms cannot handle a nonlinear objective function, but several constraints can be modeled nonlinearly. As for standard nonlinear optimization, solving a model with a linear objective function and some nonlinear constraints can result in a local optimum (see Example A.2 of appendix A).

If nonlinear relationships must be modeled and solved, there are really only three ways to deal with them: disregard the nonlinearity and assume a linear relationship,

apply the nonlinear relationship to calculate coefficients of a linear constraint, or approximate the nonlinearity by a sequence of linear models.

Of course, the first mentioned method usually doesn't give a very good representation of reality, but it may suffice if the actual relationships aren't too relevant for the decision process. In Gemms, unit-yields are assumed to increase or decrease linearly with the unit's input. In reality, however, twice as much input doesn't necessarily double all yields, because of reduced reaction-times per molecule (the doubled input may be processed by the unit in the same time as the original input).

The second way, calculation of coefficients with a nonlinear formula, can be used by Gemms in blending. This is required when a property-value of a resulting product isn't a linear combination of the property-values of the blending components. If the formulae give a sufficiently close approximation of resulting product-properties, they are should be used instead of recursive techniques, because their little calculation time.

The third method, a (recursive) sequence of linear models, lies at the basis of this chapter. The algorithm in Gemms, Penalty Sequential Linear Programming, which applies this sequence, is a modification of the just described standard SLP.

It can generally be described by the following steps:

0) derive the Taylor expansion of the nonlinear constraint replace all terms of 2^{nd} order and higher by adding a surplus-variable and subtracting a deficit-variable (these variable receive a penalty in the objective function)
1) initialize coefficients of the linear approximation resulting from 0) and find an initial solution
2) check convergence criteria and go to phase 3 if no convergence has been achieved, stop recursions otherwise
3) adjust coefficients and bounds in the LP-matrix, find an 'optimal' solution and enter phase 2

In practice, PSLP has shown to be quite useful for refinery-planning purposes, because, used correctly, calculation-times are relatively small and results are satisfactory. The term 'used correctly' is important here, and this is what initiated the research for this chapter. Currently, mainly due to a lack of documentation, Gemms-users and developers are insufficiently aware of both the implications of using recursion to solve a nonlinear problem and the required data-input.

An example of the PSLP algorithm, as implemented in Gemms, is given in Appendix A. This may aid the user to fully understand the following section, which describes both the theory and the implementation of Gemms' nonlinearity-modeling in full detail.

3.4 Nonlinearities available for Gemms

Gemms can model two kinds of nonlinearities: pooling and property-blending.

Pooling is very common at an oil-refinery: a number of components may be kept in a tank as a mixture, and later, this mixture can go to various destinations. A nonlinear equality arises because of both the beforehand unknown composition of the tank and the quantities going to any of the destination.

Blended products must satisfy various quality conditions. That is, for some product-property, the obtained value must be higher or lower than a specified value (e.g. a minimum density or a maximum sulfur content). The obtained product property-value is a combination of the property-values of all components which are used to blend the product. For example, the obtained volume of a product is a linear combination of all densities and weights of components used to produce it. Then, a minimum density can be put in a linear restriction as 'total weight of blended product $\leq$ {min. dens}$\cdot${total volume of blended product}', with both the weight and volume of blended product defined by two more linear equations. For some properties, calculating resulting values isn't that straightforward and even nonlinear inequalities may result from specifications. Some of them are modeled with nonlinear index-values, which are assumed to blend linearly. Two, however, can be modeled as nonlinear: cold filter plugging point and p-value.

3.4.1 Pooling

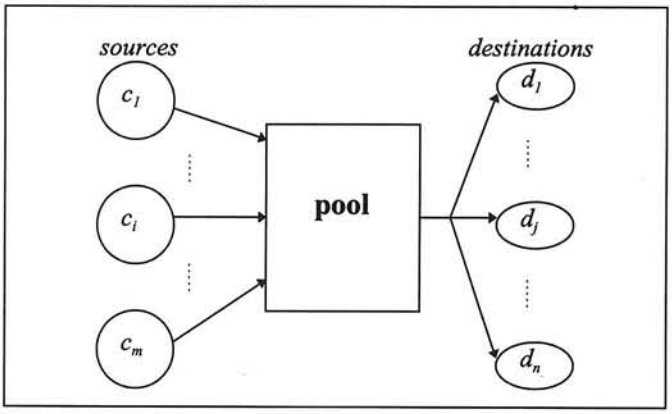
So-called 'pooling' is the most important nonlinearity which can be modeled by Gemms. In Gemms, it is solved by an algorithm of recursive linear programming if the required decision parameter 'REPO' is switched on. Parts of the following description are taken from Brussaard [5] and Pierce [9].

A 'pool' consists of a mixture of several components and can be routed to several destinations (e.g. blended to a product or used as unit feed). Therefore, a pool is often used to model some tank which can hold several components (e.g. butane and propane) for a while and whose contents are later used for final product (e.g. LPG) blending. Two things are important for pooling:

- the contents of the pool, as well as the quantity of the pool going to any of its possible destinations, are unknown beforehand,
- any component which is a possible source of a pool can't go to any of this pool's destinations by any other means then via this pool.

Because of these two aspects, the nonlinearity arises: the ratios of source components in the pool must be the same as the ratios of these source components in the pool's destinations. Suppose we have a pool which can contain up to $m \in \mathbf{N}$ components (or sources) and can be routed to $n \in \mathbf{N}$ destinations. This situation is depicted in figure 3.1.

Figure 3.1 *the pooling problem*



The equality of ratios leads to the following nonlinear constraint:

quantity of component i going to destination j =
{fraction of component i in the pool}·{quantity of pool routed to destination j}.

Gemms handles this constraint with penalty sequential linear programming (PSLP). The details of this algorithm depend on the user-selected 'recursion-guide', which is either OMNI or GSolve.
A summary on required data-entries to model pooling in Gemms, is given by Appendix B.

3.4.2 Linearization of pooling-constraint

Define:

π := set of available pools,
δ_p := set of possible destinations (d) of pool p,
σ_p := set of possible source components (c) for pool p,
$Q_{c,d}$:= quantity of component c going to destination d,
$S_{c,p}$:= quantity of component c in pool p,
$s_{c,p}$:= fraction of component c in pool p,
$D_{p,d}$:= quantity of pool p going to destination d,
X_p := total quantity of pool p.

For these variables, the following relations hold (see table 3.1 for details): for all $p \in \pi$,

$$\sum_{d \in \delta_p} D_{p,d} = X_p$$

$$X_p = \sum_{c \in \sigma_p} S_{c,p}$$

$$\forall c \in \sigma_p: \; S_{c,p} = \sum_{d \in \delta_p} Q_{c,d}$$

$$\forall d \in \delta_p: D_{p,d} = \sum_{c \in \sigma_p} Q_{c,d}$$

For each available pool, we have a nonlinear constraint for all its possible components and destinations as mentioned in the 'recursive pooling' introduction. In the just defined terms, this reads:

$$\forall p \in \pi | \forall c \in \sigma_p,\ d \in \delta_p:\ Q_{c,d} = s_{c,p} D_{p,d},$$

a quadratic expression for all sources and destinations of all pools, with $s_{c,p} = \frac{S_{c,p}}{X_p}$. To simplify the following linearization, we rewrite this expression as $f(x,y) = xy$. The Sequential Linear Programming algorithm approximates this function in a point (x_0, y_0) by the first-order Taylor-approximation

$$\begin{aligned} f(x,y) &\approx f(x_0,y_0) + \Delta x \frac{\partial f(x,y)}{\partial x}\bigg|_0 + \Delta y \frac{\partial f(x,y)}{\partial y}\bigg|_0 \\ &= x_0 y_0 + (x - x_0) y_0 + (y - y_0) x_0 = x_0 y + y_0 \Delta x, \end{aligned}$$

or

$$\forall p \in \pi | \forall c \in \sigma_p,\ d \in \delta_p:\ Q_{c,d} \approx (s_{c,p})_0 D_{p,d} + (D_{p,d})_0 \Delta s_{c,p}.$$

This linear equation will be active in every recursion step (after the first non-pooled optimization). Now, to ensure feasibility in each recursion step, penalty variables $H_{c,d}$ (deficit) and $M_{c,d}$ (surplus) are introduced to get

$$\forall p \in \pi | \forall c \in \sigma_p,\ d \in \delta_p:\ Q_{c,d} = (s_{c,p})_0 D_{p,d} + (D_{p,d})_0 \Delta s_{c,p} + M_{c,d} - H_{c,d} \qquad (3.1),$$

with $\Delta s_{c,p}$ a new variable in the LP-formulation. This formulation doesn't completely cover the nonlinear problem yet, because the new variable $\Delta s_{c,p}$ depends on both $S_{c,p}$ and X_p in a nonlinear fashion. Therefore, the relationship $s_{c,p} = (s_{c,p})_0 + \Delta s_{c,p}$ isn't guaranteed, which makes the algorithm worthless. This relationship will be enforced by another equation, again with approximation by the first-order Taylor expansion (here, in the point $(S_{c,p}, X_p)_0$): $s_{c,p} = \frac{S_{c,p}}{X_p} \approx \frac{(S_{c,p})_0}{(X_p)_0} + \Delta S_{c,p} \frac{1}{(X_p)_0} - \Delta X_p \frac{(S_{c,p})_0}{(X_p)_0^2}$, which can be rewritten to $(X_p)_0^2 \Delta s_{c,p} - (X_p)_0 S_{c,p} + (S_{c,p})_0 X_p = 0$, with all 2nd and higher order terms neglected. However, instead of using this equation in the LP-model, a much easier restriction can be used: $\forall p \in \pi | \forall c \in \sigma_p:\ \sum_d (M_{c,d} - H_{c,d}) = 0$ (3.2)

Proof:
With (3.1) follows

$$\forall c:\ \sum_d Q_{c,d} = (s_{c,p})_0 \sum_d D_{p,d} + \Delta s_{c,p} \sum_d (D_{p,d})_0 + \sum_d (M_{c,d} - H_{c,d})$$

and with the previously defined relationships between all variables, we find

$$\forall c:\ S_{c,p} = (s_{c,p})_0 X_p + (X_p)_0 \Delta s_{c,p} + \sum_d (M_{c,d} - H_{c,d}) \quad \text{or}$$

$$\forall c:\ -(X_p)_0 S_{c,p} + (S_{c,p})_0 X_p + (X_p)_0^2 \Delta s_{c,p} = \sum_d (M_{c,d} - H_{c,d})$$

which, with (3.2), exactly states the required expression. □

Avoiding numerical problems

Inserting equations (3.1) and (3.2) into our LP-matrix will result in numerical problems for small values of $(X_p)_0$, given the fact that $(s_{c,p})_0 = \left.\frac{S_{c,p}}{X_p}\right|_0$. Therefore, restriction (3.1) is scaled by multiplying left- and right-hand-sides by $(X_p)_0$. We keep the same notation $H_{c,d}$ and $M_{c,d}$ for the feasibility-variables because we can just as well add them only to the constraint *after* multiplication by $(X_p)_0$, which has no relevant consequences for our model.

One more change is made to the above formulation, before the model is actually presented to the LP-solver. That is, equation (3.2) also receives further feasibility-variables to lead to:

$$\forall p \in \pi \; \forall c \in \sigma_p: \; (WM_c - WU_c) + \sum_{d \in \delta_p} (M_{c,d} - H_{c,d}) = 0$$

The penalty-values of WM_c and WU_c equal the value for parameter PEN2, entered at the control-table.

Conclusion: added matrix-rows and variables

The linearized recursion constraints which will become active after the first (non-pooled) optimization of the model are:

$\forall p \in \pi \; \forall c \in \sigma_p, d \in \delta_p$:

$$-(X_p)_0 Q_{c,d} + (S_{c,p})_0 D_{p,d} + (X_p)_0 (D_{p,d})_0 \Delta s_{c,p} + M_{c,d} - H_{c,d} = 0 \qquad (3.3)$$

$\forall p \in \pi \; \forall c \in \sigma_p$:

$$(WM_c - WU_c) + \sum_{d \in \delta_p} (M_{c,d} - H_{c,d}) = 0 \qquad (3.4)$$

All values, except for $\Delta s_{c,p}$, are greater than or equal to 0. The delta-variable receives initial upper- and lower-bounds equal to +1 and -1, respectively.

The resulting objective function from recursive pooling reads: Obj(with pools) = Obj(without pools) - $\sum_{p \in \pi} \sum_{c \in \sigma_p} \{ \text{PEN2} \cdot (WM_c + WU_c) + \text{PEN1} \cdot \sum_{d \in \delta_p} (M_{c,d} + H_{c,d}) \}$,

with PEN1 and PEN2 the penalty values given by the pooling-control table. Default values are: PEN1 = 1 ; PEN2 = 10·PEN1.

Taylor-approximation as recursion

Gemms iteratively uses the above given first-order Taylor expansion to deal with the nonlinear equation. That is, in any iteration, the Gemms model is run with the coefficients of the linear approximation obtained from the previous iteration (and with some initialization for the first run). With superscripts used to indicate the iteration-number, the recursive equation of the n-th iteration reads

$$-X_p^{(n-1)} Q_{c,d}^{(n)} + S_{c,p}^{(n-1)} D_{p,d}^{(n)} + X_p^{(n-1)} D_{p,d}^{(n-1)} \Delta s_{c,p}^{(n)} + M_{c,d}^{(n)} - H_{c,d}^{(n)} = 0 \qquad (3.5)$$

Note that variables with superscripts n-1 are known before running iteration n. Eventually, the series $Q_{c,d}^{(n)}$ and $X_p^{(n)}$ should converge to some $Q_{c,d}$ and X_p, and $\Delta s_{c,p}^{(n)}$ should go to 0. For a big part, actual convergence depends on the initial solution.

3.4.3 Nonlinear property-blending

Apart from pooling, nonlinearities can also arise when calculating resulting properties for products which are produced from blending various materials.

For example, suppose we blend 1 kiloton of material A with 2 kilotons of material B to produce final grade P. Then, for most material-properties (e.g. sulfur-content percentage) , the resulting property-value of P will simply be the weighted average [1·{property-value of A} + 2·{property-value of B}]/3, possibly corrected with material-densities when the property blends linearly on volume. A specification may be placed on such a property, e.g. 'Sulfur contents of P must be less than 3 percent', which leads to a linear constraint on the quantities of materials A and B blending to P.

Now, some properties don't blend linearly (neither on weight nor on volume). In such a case (e.g. cloud point), Gemms uses indices which are presumed to blend linearly (usually on volume). For example, the data contains cloud point values c_A and c_B for materials A and B, respectively. These materials blend to product P and the cloud point of P must be less than some value K. Then, Gemms recalculates actual values c_A, c_B and K to indices I_A, I_B and I_K and generates a blending-constraint like the one mentioned earlier as if actual material-properties equal I_A and I_B and as if the specification had value I_K.
For two properties, P-value and CFPP, neither the actual property-value nor the index-value suffices to linearly model their specifications correctly. Therefore, recursive calculations can be used.

3.4.3.1 P-value

In fuel oil blends, the combination of cracked and straight-run components can cause stability problems. To control the stability, the user must specify a minimum acceptable so-called *P-value*, which leads to a nonlinear blend-restriction. Gemms can solve this with its PSLP algorithm.

Required Gemms material-properties for this nonlinear blending are *FRMX* (flocculation ratio), *ASPH* (asphaltenes), *p0* (peptizing power) and *DENS* (density). With W_i the total weight of material i blending to the product and with *VOL* the product's total blended volume (in m^3), resulting product-properties are given by:

$$FRMX = \Sigma_i (FRMX_i \cdot ASPH_i \cdot W_i),$$
$$ASPH = \Sigma_i (ASPH_i \cdot W_i),$$
$$p0 = \Sigma_i (p0_i \cdot W_i / DENS_i),$$
$$VOL = \Sigma_i (W_i / DENS_i),$$

with properties with subscript *i* being constants equal to component *i*'s property-value. All variables at the left-hand-sides depend on the refinery, the period and the product.

A product's p-value is given by $Pval = \frac{p0 \cdot ASPH}{VOL \cdot FRMX}$. A specification on this value is always a minimum. Therefore, with a specified minimum p-value, say *Pspec*, the following must hold:

$$Pval \geq Pspec, \text{ or}$$
$$Pspec \cdot FRMX - p0 \cdot ASPH/VOL \leq 0, \qquad (3.6)$$

a nonlinear equation in variables *FRMX*, *p0*, *VOL* and *ASPH*. Note that a variety of formulations is possible for (3.6), but, as will be shown by the following section, this formulation leads to nice results for the linear approximation.

3.4.3.2 Cold filter plugging point

The Cold Filter Plugging Point (CFPP) is another property which doesn't blend linearly and which can be approximated using Gemms' PSLP algorithm.

For all components which can blend to a product with a specified CFPP-value, required properties are: *CFPP* or *CFPPI* (index), *MIDP* (mid boiling point) and *DENS* (density). The index is assumed to blend linearly on a volume-basis.

For any component *i*, either its CFPP-property or its CFPP-index value may be given. From the CFPP-property, the index-value is defined as $CFPPI_i = 10^{F_i}$, with $F_i = 5 - \frac{(850 + MIDP_i)(67 - CFPP_i)}{24{,}000}$.

For a blended product *p*, the resulting CFPP-value follows from the same formula:

$$CFPPI_p = 10^{Fp}, \text{ with } F_p = 5 - \frac{(850 + MIDP_p)(67 - CFPP_p)}{24{,}000}, \text{ or}$$
$$CFPP_p = 67 - 24000 \cdot \{5 - {}^{10}\log(CFPPI_p)\}/(850 + MIDP_p) \qquad (3.7)$$

with $CFPPI_p$ and $MIDP_p$ defined as

$$CFPPI_p = \frac{1}{VOL_p}\sum_i \frac{W_i CFPPI_i}{DENS_i} \text{ and } MIDP_p = \frac{1}{VOL_p}\sum_i \frac{W_i MIDP_i}{DENS_i},$$

with W_i the weight of component *i* blending to the product, $DENS_i$ the density of component *i* and with $VOL_p = \Sigma_i (W_i/DENS_i)$ the total blended volume (here in m^3) of product *p*. Since all W_i's and VOL_p are variables, all equations ($CFPP_p$, $CFPPI_p$ and $MIDP_p$) are nonlinear. In conclusion, the relevant set of constraints to calculate a product's CFPP property reads:

1. $VOL_p = \Sigma_i (W_i/DENS_i)$,
2. $CFPPI_p = \frac{1}{VOL_p} \sum_i \frac{W_i CFPPI_i}{DENS_i}$
3. $MIDP_p = \frac{1}{VOL_p} \sum_i \frac{W_i MIDP_i}{DENS_i}$,
4. $CFPP_p = 67 - 24000 \cdot \{5 - {}^{10}\log (CFPPI_p)\}/(850 + MIDP_p)$,

a set with three nonlinear constraints. This can be rewritten to an easier solvable set of equations (with the product's subscript left out of the notation), with only one nonlinear equation left:

1. $VOL_p = \Sigma_i (W_i/DENS_i)$,
2. $CFPPI = \sum_i \frac{W_i CFPPI_i}{DENS_i}$
3. $MIDP = \sum_i \frac{W_i MIDP_i}{DENS_i}$
4. ${}^{10}\log (CFPPI/VOL) = 5 - (850 + MIDP/VOL) \cdot (67 - CFPP)/24000$

Now, a specification on a product's CFPP-value, say *Cspec*, is always a maximum value, so $CFPP \leq Cspec$. With (3.7) and with the previous set of equations, we can replace the nonlinear equation by one with only three variables left, instead of four:

$$ {}^{10}\log (CFPPI/VOL) - 5 + \frac{(850 + MIDP/VOL)(67 - Cspec)}{24{,}000} \leq 0 \qquad (3.8). $$

All variables are refinery-, period- and product-dependent.

3.4.4 Linear approximation of p-value

Like pooling, nonlinear fuel oil stability is solved with a PSLP algorithm (if activated by decision-parameter FOST). The algorithm uses the first-order Taylor approximation.

To simplify things, we rewrite the p-value constraint with simple names as

$$ f(x,y,v,w) = cx - vw/y \leq 0 \qquad (3.9) $$

with c a constant, and x, v, w and y variables as in (3.6).

The first order Taylor-expansion now tells us

$$ \begin{aligned} f(x,y,v,w) &\approx f(x_0,y_0,v_0,w_0) + c\Delta x - w_0\Delta v/y_0 - v_0\Delta w/y_0 + v_0 w_0 \Delta y/y_0^2 \\ &= cx_0 - v_0 w_0/y_0 + c(x - x_0) - w_0(v - v_0)/y_0 - v_0(w - w_0)/y_0 + v_0 w_0 (y - y_0)/y_0^2 \end{aligned} $$

With (3.9), elimination of terms and multiplication by y_0^2 to allow evaluation at $y_0 = 0$, follows the linearized constraint which will be presented to the LP-solvers:

$(cy_0^2)x - (y_0 w_0)v - (y_0 v_0)w + (v_0 w_0)y \leq 0$, or in Gemms-recursion terms,

for each combination of refinery, period and product, in the *n*-th iteration step:

$$\{Pspec \cdot (VOL^{(n-1)})^2\} \cdot FRMX^{(n)} - \{VOL^{(n-1)} \cdot p0^{(n-1)}\} \cdot ASPH^{(n)}$$
$$-\{VOL^{(n-1)} \cdot ASPH^{(n-1)}\} \cdot p0^{(n)} + \{p0^{(n-1)} \cdot ASPH^{(n-1)}\} \cdot VOL^{(n)} - D^{(n)} \le 0, \qquad (3.10)$$

with $D^{(n)}$ a deficit-variable to ensure feasibility and with all other variables defined as earlier. Note that all values between parentheses are known at the beginning of the *n*-th iteration and that no surplus-variable is necessary because of the 'less than' type of the inequality. Finally, of course, all variables are refinery, period and product-dependent and their definitions are part of the matrix.

The deficit-variable highly improves the flexibility and the performance of the recursion. The idea of using a deficit-variable was already present in Gemms versions 7.1 and lower, but not yet correctly implemented. This could lead to bad convergence speed or to undesired model-infeasibility. Simple tests showed that implementation of $D^{(n)}$ is easy, gives much better results and should be present in future Gemms releases. Note that creation of this variable is handled by OMNI, no matter whether OMNI or GSolve is used to guide the model through the PSLP algorithm.

3.4.5 Linear approximation of CFPP-value

The only required linearization is that of equation (3.8), which, with ${}^{10}\log a = \frac{\ln a}{\ln(10)}$, can be rewritten to

$$f(x,y,z) = [\ln(x/y) - (\alpha\text{-}1)] \cdot y + \beta z \le 0, \qquad (3.11)$$

with $x=CFPPI$, $y=VOL$, $z=MIDP$, $\alpha = \frac{850 \cdot \ln(10) \cdot (67 - Cspec)}{24{,}000} - 5\ln(10) + 1$ and $\beta = \frac{(67 - Cspec) \cdot \ln(10)}{24{,}000}$.

A first-order Taylor approximation of equation (3.11) gives

$$\begin{aligned} f(x,y,z) &\approx f(x_0,y_0,z_0) + (x\text{-}x_0)\cdot(y_0/x_0) + (y\text{-}y_0)\cdot\{[\ln(x_0/y_0)\text{-}c_1] + (y_0/x_0)\cdot(-x_0/y_0^2)\cdot y_0\} + (z\text{-}z_0)\cdot c_2 \\ &= \ldots \\ &= (y_0/x_0)\cdot x + [\ln(x_0/y_0) - \alpha]\cdot y + \beta\cdot z \\ &\le 0 \end{aligned}$$

In recursive Gemms-notation, for iteration *n*, the set of CFPP-constraints which will be part of the LP-matrix reads (for each combination of refinery, period and product):

1. $CFPPI^{(n)} = \sum_i \frac{W_i CFPPI_i}{DENS_i}$
2. $MIDP^{(n)} = \sum_i \frac{W_i MIDP_i}{DENS_i}$
3. $\{\frac{VOL^{(n-1)}}{CFPPI^{(n-1)}}\} \cdot CFPPI^{(n)} + \{\ln(\frac{CFPPI^{(n-1)}}{VOL^{(n-1)}}) - \alpha\} \cdot VOL^{(n)} + \beta \cdot MIDP^{(n)} - D^{(n)} \le 0, \qquad (3.12)$

with $D^{(n)}$ a deficit-variable (with a penalty-value in the objective function) entered to ensure feasibility of the model and to improve the performance of the recursion algorithm.

3.5 General implementation PSLP

This section describes the implementation of the PSLP algorithm in general and holds for both OMNI and GSolve. Furthermore, it holds for all three possible recursions: pooling, fuel oil stability and cold filter plugging point. Several details of the implementation depend both on which recursion(s) is (are) activated and on which recursion-guide is selected (either OMNI or GSolve). These details will be discussed later.

First, we'll give the parameters which are used in the recursion algorithm. Then, the algorithm is described by three different phases: finding an initial solution, checking convergence, and possibly revising the LP-matrix to solve the model again and go back to the convergence check. Generation of matrix-rows which are relevant for the recursion are described under 'phase 0', which is no part of the actual PSLP algorithm.

3.5.1 Recursion parameters

The user is able to steer the PSLP algorithm by setting several parameters. Currently, only the first four are actually used by GSolve, whereas all are used by OMNI. The following parameters, with their default values given, are available:

PEN1 = 1
PEN2 = 10·PEN1
MAXITER = 20
DVARBND = 1
TOLKTON = 0.01
TOLZERO = $5 \cdot 10^{-5}$
RCMULT = 1.3
RCDIV = 1.6

PEN1 and PEN2 are penalty-values which will be the initial coefficients of some surplus- and deficit-variables in the objective function, except for the deficit-variable of the linearized CFPP-row. This variable receives an initial cost-factor of PEN, equal to this entry in the general model-control table. MAXITER determines the maximum number of recursion steps for OMNI and GSolve.
In case OMNI is used to guide the recursion, DVARBND, RCMULT and RCDIV determine the adjustment of variable-bounds if variables hit their bounds in recursive solution-steps and TOLZERO and TOLKTON are used in the convergence checks.
From these last-mentioned parameters, GSolve only uses DVARBND to initialize bounds on the delta-variables of pooling-constraints. Convergence checks are based on an internal constant value.

3.5.2 Phases of the algorithm

Phase 0: Matrix-row generation

Check relevance recursive pooling

If pools are present in the Gemms model, several pool-specific constraints are generated, but only if the pool has more than one destination and more than one source. Of course, if a pool contains only one component, there's no need for recursion because the complete composition of the pool is known beforehand. Should a pool have only one destination, then the fraction of components going from a pool to a destination would always be equal for all destinations, so again, recursion wouldn't make any sense.

Gemms detects these compositions and gives a warning if only one source or destination is found (and the component(s) will be treated as if they can go to the destination(s) directly, instead of only via the pool).

Pooling-rows

Table 3.1 gives all generated pooling-rows. In order to assist future Gemms-developers, this table also mentions all relevant OMNI-codes. Note that the first four rows mentioned by this table would generate a set of $m_p = 2 + n_{\sigma_p} + n_{\delta_p}$ constraints for each pool p, with n_{σ_p} and n_{δ_p} the numbers of possible sources and destinations of the pool. If the sixth constraint isn't active, these constraints don't restrict the other parts of the model in any way, which can easily be demonstrated with substitution:

$$\forall p \in \pi: \qquad X_p \overset{(1)}{=} \sum_{d \in \delta_{d,p}} D_{p,d} \overset{(4)}{=} \Sigma_d \Sigma_c Q_{c,d} = \Sigma_c \Sigma_d Q_{c,d} \overset{(3)}{=} \Sigma_c S_{c,p} \overset{(2)}{=} X_p,$$

with the only further restriction on these variables that they are nonnegative.

The other thing that follows from the above substitution is that, for each pool, one of the m_p constraints is redundant. Therefore, Gemms suppresses the pool-destination row $D_{p,d} = \sum_{c \in \sigma_p} Q_{c,d}$ for the last destination from the matrix.

The optimization is only truly influenced if the linear approximation is activated (so decision-parameter REPO is switched on). Then, the balance rows are no longer meaningless, because they restrict X_p and therefore $D_{p,d}$ and so forth. To obtain an initial solution for recursive pooling, the linear approximation won't be active in the first optimization.

Table 3.1 *pooling-specific rows*

Row-type	OMNI-code[1)]	Description	Algebraic
Pool balance	`B(R)(P)(PL)`	The total pool-quantity has a destination.	$\forall p \in \pi$: $\sum_{d \in \delta_{4,p}} D_{p,d} = X_p$
Pool definition	`S(R)(P)(PL)(.)`	The total pool-quantity is determined by the sources actually in the pool.	$\forall p \in \pi$: $X_p = \sum_{c \in \sigma_{c,p}} S_{c,p}$
Source balance	`V(R)(P)(PL)(CP).`	The total quantity of a source in the pool equals the quantity of this source routed to the pool's destinations	$\forall p \in \pi \mid \forall c \in \sigma_p$: $S_{c,p} = \sum_{d \in \delta_{4,p}} Q_{c,d}$
Destination balance	`V(R)(P)(PL).(DE)`	The total quantity of pool going to a destination equals the quantity of all possible pool-sources routed to this destination.	$\forall p \in \pi \mid \forall d \in \delta_p$: $D_{p,d} = \sum_{c \in \sigma_{c,p}} Q_{c,d}$
Delta balance	`W(R)(P)(PL)(CP).`	Ensuring $\Delta s = s - s_0$	See (3.4)
Recursive pooling	`V(R)(P)(PL)(CPD)`	First-order approximation in current solution-point.	See (3.5)

[1)] (R), (P), (PL), (CP), (DE) and (CPD) indicate references to the refinery, the period, the pool, the source-component, and to the pool-destination and a reference-number, respectively.

FOST and CFPP rows

Table 3.2 shows all CFPP- and FOST-(or P-value) specific rows. CFPP- and FOST-recursions are activated if their corresponding decision-parameters are switched on.

Table 3.2 *CFPP- and FOST-specific rows*

Row-type	OMNI-code[1)]	Description	Algebraic
ASPH balance	`Q(R)(P)(PR).FA`	Definition *ASPH*, which blends linear on weight.	$ASPH = \Sigma_i (ASPH_i \cdot W_i)$
p0 balance	`Q(R)(P)(PR).FP`	Definition *p0*, which blends linear on volume.	$p0 = \Sigma_i (p0_i \cdot W_i / DENS_i)$
FRMX balance	`Q(R)(P)(PR).FX`	Definition floc ratio.	$FRMX = \Sigma_i (FRMX_i \cdot ASPH_i \cdot W_i)$
Recursive FOST	`Q(R)(P)(PR).FS`	Linearization p-value.	See (3.10)
CFPPI balance	`Q(R)(P)(PR).CC`	Definition index cold filter plugging point.	$CFPPI = \sum_i \frac{W_i CFPPI_i}{DENS_i}$
MIDP balance	`Q(R)(P)(PR).CV`	Definition mid-boiling point.	$MIDP = \sum_i \frac{W_i MIDP_i}{DENS_i}$
Recursive CFPP	`Q(R)(P)(PR).CM`	Linearization CFFP-value.	See (3.12)

[1)] (R), (P) and (PR) indicate references to the refinery, the period and to the product, respectively. Constraints are generated for all relevant refineries, periods and products.

Phase 1: Finding an initial solution

pooling

For the first model run, the linear approximation for pooling will be present in the matrix, but will receive a 'N' indicator in MPS-format, which tells the solver not to use this row in its optimization. Still, some initial values for all coefficients are required to generate this row. All $(\cdot)_0$ are simply initialized equal to 1. Then, all rows are generated according to the formulae of table 3.1.

FOST and CFPP

Fuel oil stability and CFPP rows are active from the first optimization onwards. Table 3.3 gives all initial values of the variables used for coefficient-calculation. In current Gemms, the model will always hold a linearized P-value (or CFPP) constraint if a specification is given, even if recursion isn't switched on. This linearization is likely to be much different from the required nonlinear restriction. Therefore, the model could be uselessly restricted (if the linearization is binding). This should be either solved for future Gemms releases, or the user should be explicitly noted on this fact. (Then, the user can either deactivate the specification or switch on the recursion.)

In table 3.3, the first mentioned value is tried first for initialization. If this value isn't available, the second is tried, and so forth. If the evaluation of an initial coefficient (between parentheses) is less than 10^{-5}, it is taken equal to 10^{-5}. A new constant called *MID* is introduced to calculate an initial CFPP-index value.

Table 3.3 *Initialization FOST- and CFPP-coefficients*

property	variable	initial value
FOST	*VOL*	1) 1/{reference density} [table PRODS] 2) 1
FOST	*p0*	1) Reference p0 [table PRODS] 2) *Cspec·Fspec*, with *Fspec* a maximum FRMX specification on this product 3) 30
FOST	*ASPH*	1) Reference ASPH [table PRODS] 2) 1
CFPP	*MID*	1) (LOW+HIGH)/2, with entries from table CFPPMIDP 2) (220+350)/2
CFPP	*CFPPI*	1) 10^F, with F=5 - (850+*MID*)(67-*Cspec*)/24000
CFPP	*VOL*	1) 1

With all coefficients initialized as mentioned by the tables, the model is run and the resulting solution, if locally optimal, is used as an initial solution for phases 2 and 3. If the model is infeasible, the recursion is ended. Note that infeasibility cannot be caused by any of the linear approximations, because of the introduced infeasibility-variables (which are just variables with a large costs to the LP-solver).

Phase 2: Checking convergence criteria

To start the recursion, first all required parameters which can direct the sequential linear programming algorithm are read from the data (either from table SLPVAR in OMNI or from the pooling-control of PlanStar's control-parameters section). If for some parameter no entry is found, the default value is used.

When a solution is obtained from phase 1 (or later from phase 3), depending on the user's selection, either OMNI or GSolve takes over from the solver and checks specific convergence criteria. From OMNI or GSolve, any of four conclusions result:

- don't start with recursion because the original model is infeasible (currently, this is only implemented for OMNI, but should become a part of GSolve as well, because recursion can never lead to a feasible solution if the original model isn't feasible),
- end the recursion because the maximum number iterations has been reached,
- end the recursion because convergence has been achieved,
- enter phase 3 of the PSLP algorithm.

Phase 3: Matrix revision

If the optimal solution hasn't converged, matrix-coefficients and variable-bounds are adjusted. Coefficients are simply as given by the Taylor-approximation, so current variable-values are used to calculate the coefficient-values for the next solve. The change of bounds, however, is somewhat more delicate. It doesn't follow directly from the linearization and is only used for practical purposes: to make sure that the solution of the following solve will at least be close to the one which we have now. The actual implementation would really be just a matter of taste. Therefore, it's not so strange that the implementation of this part differs greatly in GSolve and OMNI.

In the detailed implementation, we will call the adjustment of coefficients phase 3a, and the adjustment of bounds phase 3b.

3.6 Detailed implementation PSLP

This section describes the implementation of the PSLP algorithm in full detail. Therefore, sometimes a distinction is required between the two possible recursion-guides, OMNI and GSolve. Note that the LP-solver is of no influence to the algorithm itself.

3.6.1 Definitions

During the entire description of the algorithm, the reader who's familiar with Gemms/PlanStar must keep in mind that all value-calculations, storage and matrix-updates are performed for all available right-hand-side models.

In the following definitions, variable r is a so-called 'recursion-item', which refers to a specific variable, depending on the equation in which it is present:

pooling: either the delta-variable, or any of the feasibility-variables (so $r = M_{c,d}$, $H_{c,d}$, WM_c, WU_c or $\Delta s_{c,p}$),
fuel oil: any of the variables in the linearized constraints.
cfpp: any of the variables in the linearized constraint, except for MIDP.

Define

$r^{(n)}$:= the resulting activity of variable r after the n^{th} iteration,
$U_r^{(n)}$:= the upper bound placed on variable r for the n^{th} solve,
$L_r^{(n)}$:= the lower bound placed on variable r for the n^{th} solve,
$HU_r^{(n)}$:= the number of times-in-a-row that variable r hit its upper bound (not used by GSolve),
$HL_r^{(n)}$:= the number of times-in-a-row that variable r hit its lower bound (not used by GSolve).

Only for fuel oil stability and cfpp:

$V_r^{(n)}$:= last increase of obtained value of r ($V_r^{(n)} = r^{(n)} - r^{(n-1)}$), n=1,2,3,...,
$UB_r^{(n)}$:= relative upper bound, used to calculate $U_r^{(n)}$, n=3,4,5,...,
$LB_r^{(n)}$:= relative lower bound, used to calculate $L_r^{(n)}$, n=3,4,5,... ,
$V^{(n)}_{max,r}$:= $\max\{|V_r^{(n)}|, |V_r^{(n-1)}|\}$, $n \geq 2$,
$M_r^{(n)}$:= $\max\{D_r^{(n)}, D_r^{(n-1)}\}$, $n \geq 1$,
$r_{avg}^{(n)}$:= $\dfrac{r^{(n)} + r^{(n-1)}}{2}$, $n \geq 2$,

3.6.2 Phase 1: finding an initial solution

The initialization of the PSLP algorithm is completely implemented in the OMNI modeling language and presented either directly to the solver or via GSolve. GSolve doesn't change any of the initializations.

Initialize

$r^{(0)}$:= 0, to formulate a generic recursive procedure,
$U_r^{(1)}$:= DVARBND, a parameter of the pooling-control table,
$L_r^{(1)}$:= - DVARBND,
$HU_r^{(1)}$:= 0,
$HL_r^{(1)}$:= 0, no bounds are hit yetc
$V_r^{(1)}$:=0.

In the objective function, coefficients for penalty-variables initially equal

- PEN1 for $H_{c,d}$ and $M_{c,d}$
- PEN2 for WM_c and WU_c

with PEN1 and PEN2 given by the pooling-control table.

Coefficients of the linear approximation are as described by the general implementation.

Because the initialization is exactly the same for both OMNI and GSolve, both present the exact same matrix to their solvers to obtain an initial solution. Of course, the resulting solution may very well differ because most Gemms models don't have a unique optimal solution. This has nothing to do with OMNI or GSolve though, but only depends on the LP-solver.

3.6.3 Phase 2: checking convergence criteria

Convergence checks are implemented quite differently for OMNI and GSolve. Therefore, separate discussions are required here.

3.6.3.1 OMNI + LP-solver

When OMNI is used to run the PSLP-algorithm, for each recursion-item $r^{(n)}$ after iteration n, variables are defined and evaluated depending on the type of $r^{(n)}$:

$r^{(n)}$ part of pooling (delta- or penalty-variable):

$D_r^{(n)} := |r^{(n)}|$ and $R_r^{(n)} := \frac{D_r^{(n)}}{\varepsilon}$, with ε equal to pooling-control parameter TOLZERO (default $5 \cdot 10^{-5}$).

$r^{(n)}$ part of FOST or CFPP, but no penalty-variable:

$D_r^{(n)} := |V_r^{(n)}|$, $R_r^{(n)} := \frac{D_r^{(n)}}{\theta}$, with θ the allowed approximation-error equal to pooling-control parameter TOLKTON (default 10^{-2}).

The solution is considered converged and locally optimal if $\max_r R_r^{(n)} \leq 1$. If convergence is achieved or the maximum number of iterations has been reached (n=MAXITER), the recursion is ended. Otherwise, the algorithm enters phase 3.

3.6.3.2 GSolve + LP-solver

In Gsolve, two variables and one constant are used to check convergence of the PSLP algorithm: variables PICOST and PENMAX, and constant PENTOL, which equals 10^{-6} (a simple adjustment of source code is preferred to make this constant available in the pooling-control table, either as a new or as an existing parameter (like TOLZERO), because then, experienced users can choose their own preferred convergence-settings) .

check 1: approximated errors

First of all, all feasibility-variables of all linearized constraints (and of the extra pooling constraint to ensure $\Delta s_{c,p} = s_{c,p} - s_0$) are checked. If any of these variables has an optimal value larger than PENTOL, the linear approximation isn't good enough and the recursion must enter its next iteration (after coefficient- and bound-adjustment).

The variable P1COST equals the sum of all surplus- and deficit-variables.

check 2: actual errors

Now the *actual* errors are calculated, which are the current nonlinearity-violations:
Define

$I_i^{(n)}$:= actual infeasibility of linearized row i after the n^{th} iteration-step

$$= \begin{cases} S_{c,p}^{(n)} \cdot D_{p,d}^{(n)} - X_p^{(n)} \cdot Q_{c,d}^{(n)} & \text{if } i \text{ is a pooling - row} \\ \max\{0, Pval \cdot FRMX^{(n)} - \dfrac{p0^{(n)} \cdot ASPH^{(n)}}{VOL^{(n)}}\} & \text{if } i \text{ is fuel oil stability row} \\ \max\{0, [1 + \ln(\dfrac{CFPPI^{(n)}}{VOL^{(n)}}) - \alpha] \cdot VOL^{(n)} + \beta \cdot MIDP^{(n)}\} & \text{if } i \text{ is a cfpp row} \end{cases}$$

$\text{P2COST} = \Sigma_i I_i^{(n)}$,

$\text{PENMAX} = \max_i \{ I_i^{(n)} \}$.

The solution cannot be considered converged and locally optimal if PENMAX > PENTOL. The values of P1COST and of P2COST both in the current and in the previous iteration, as well as the current value of PENMAX, are reported in the ASCII-file 'Gsolve.lst'.

Two quite relevant differences between the implemented convergence checks in OMNI and in GSolve are:

- GSolve solves the model much quicker if the nonlinear fost- or cfpp-constraints aren't constraining, because GSolve stops the recursion if the optimal solution in one iteration of a model with fost- and cfpp constraints satisfies the original nonlinear constraints. However, from Bazaraa et al. [3, p. 432], no local optimum can be guaranteed in this manner. In fact, this can only be guaranteed if and only if the optimal solution of the current iteration also solves the revised model of the next iteration. Therefore, another iteration would be required even if the current solution satisfies the original nonlinear constraints to prove local optimality.
- OMNI is more efficient if the linear approximation leads to a locally infeasible model. That is, all coefficients of the recursive constraints have converged, but some of the infeasibility-variables are still too large. Currently, in GSolve, this isn't recognized, so the recursion continues with penalty-values being the only difference between to recursion runs. If GSolve would also consider the solution converged if coefficients hardly change, this problem would be solved, since the solution is presented to OMNI which can detect the infeasibility.

3.6.4 Phase 3: matrix revision

Like the convergence checks, the actual matrix revision (bounds and coefficients) differs for OMNI and GSolve. Especially for bounds-adjustments, these implementations aren't much alike. Coefficients are updated according to the recursive equations, so are in fact the same for both OMNI and GSolve.

3.6.4.1 OMNI + LP-solver

Coefficient-adjustment

Adjustment of coefficients is straightforward: calculate the coefficients of all Taylor approximations by substituting the solution of the previous solve in the coefficient-formulae of the appropriate constraints (3.5), (3.10) and (3.12).

Vital information for developers, however, is how all this is implemented in the OMNI modeling language and in GSolve. In OMNI, various different types of modification-rows are introduced which are written, as well as the appropriate matrix row- and column-names, to an ASCII-file called 'gen.rec' which is used by OMNI to build the matrix-modifier file 'gen.rev'. The 'gen.rec' file contains static information and is written just once, whereas 'gen.rev' contains dynamic information (new coefficient and bound values) and is written after every matrix-solve.

Table 3.4 describes the files 'gen.rec' and 'gen.rev', the latter holding all evaluations of the 'modification' column. The 'Chk' column of the table is used by Gemms to check whether or not various other details are required (for example, the number of times-in-a-row a variable hit its upper bound). The name which is mentioned first in any of the rows of 'gen.rec' is the recursion-item under consideration.

Table 3.4 *OMNI table for matrix-modification*

Type	Const	Chk	Names used in 'gen.rec'				modification [1]
			first	second	third	fourth	
NE	--	--	pool-row	--	--	--	Row-type 'E'
CL	-1	--	$Q_{c,d}$	pool-row	X_p	--	Const·*third*
CL	+1	--	$D_{p,d}$	pool-row	$S_{c,p}$	--	Const·*third*
MV	+1	--	$\Delta s_{c,p}$	pool-row	$D_{p,d}$	X_p	Const·*third*·*fourth*
C1	TOLZERO	1	$\Delta s_{c,p}$	pool-row	--	--	Bounds modified
P1	TOLZERO	3	$H_{c,d}$ or $M_{c,d}$	pool-row	--	--	[2]
P2	TOLZERO	4	WM_c or WU_c	pool-row	--	--	[3]
LN	$-c_1$-1	--	*V*	cfpp-row	*CFPPI*	*V*	ln(*third*/*fourth*) + Const [4]
DV	1	--	*CFPPI*	cfpp-row	*V*	*CFPPI*	*third*/*fourth*
C2	TOLKTON	2	*V*	cfpp-row	--	--	Bounds modified
MV	1	--	*V*	fost-row	*p0*	*ASPH*	Const·*third*·*fourth*
MV	*Pval*	--	*FRMX*	fost-row	*V*	*V*	Const·*third*·*fourth*
MV	-1	--	*ASPH*	fost-row	*p0*	*V*	Const·*third*·*fourth*
MV	-1	--	*p0*	fost-row	*ASPH*	*V*	Const·*third*·*fourth*
C2	TOLKTON	2	*V*	fost-row	--	--	Bounds modified

The values of the variables in the third and fourth columns and of the modifications are assured not to become less than 10^{-6} in absolute value.

[1] Italic words used in this column refer to the consecutive columns.

[2] Penalty-coefficient $p^{(n)}$ changed only if $\{\max_r |r^{(n)}|\} \geq \{0.75 \cdot \max_r |r^{(n-1)}|\}$. Then, $p^{(n+1)} := p^{(n)} \cdot \text{RCMULT}$.

[3] Penalty-coefficient $p^{(n)}$ changed only if $\{\max_r |r^{(n)}|\} \geq \{0.75 \cdot \max_r |r^{(n-1)}|\}$. Then, $p^{(n+1)} := p^{(n)} \cdot \text{RCMULT}$

[4] If '*CFPPI*/*V*' $<10^{-6}$, then this expression is defined equal to 10^{-6}

Bounds-adjustment

An incredibly large part of the OMNI implementation determines when and how to adjust bounds on decision-variables, depending on the recurson-type.

pooling

With pooling, bounds are only changed for the delta-variables $\Delta s_{c,p}^{(n)}$, which will simply be denoted as $r^{(n)}$. Changes of the bounds on $r^{(n+1)}$ depend on the values of $r^{(n)}$ and are given by table 3.5. To prevent problems with matrix-revision, as well as to make sure that bounds can always be adjusted by multiplication or division, the upper and lower bounds of $r^{(n+1)}$ are, in absolute value, always greater than zero.

Table 3.5 *Adjusting bounds of delta-variables*

Result	Adjustment	Comment
$r^{(n)} \le 0$	$U_r^{(n+1)} := \max\{10^{-6}, U_r^{(n)}/\text{RCDIV}\}$ $HU_r^{(n)} := 0$; $HL_r^{(n)} := 0$	Pooling-control parameter RCDIV (RCDIV>1). No bound hit. Tighten upper bound if obtained value for r is less than or equal to zero.
$r^{(n)} \ge 0$	$L_r^{(n+1)} := \min\{-10^{-6}, L_r^{(n)}/\text{RCDIV}\}$ $HU_r^{(n)} := 0$; $HL_r^{(n)} := 0$	Tighten lower bound if obtained value for r is greater than or equal to zero.
$r^{(n)} = U_r^{(n)}$	$HU_r^{(n)} := HU_r^{(n-1)} + 1$; $HL_r^{(n)} := 0$ if $(HU_r^{(n)} = 3)$ then $U_r^{(n+1)} := U_r^{(n)} \cdot \text{RCMULT}$ and $HU_r^{(n)} := 0$ else no change	Upper bound hit ($U_r^{(n)} > 0$). Extend upper bound after third hit-in-a-row. Use pooling-control parameter RCMULT (>1).
$r^{(n)} = L_r^{(n)}$	$HL_r^{(n)} := HL_r^{(n-1)} + 1$; $HU_r^{(n)} := 0$ if $(HL_r^{(n)} = 3)$ then $L_r^{(n+1)} := L_r^{(n)} \cdot \text{RCMULT}$ and $HL_r^{(n)} := 0$ else no change	Lower bound hit ($L_r^{(n)} < 0$). Extend lower bound after third hit-in-a-row.

fuel oil stability and CFPP

Especially with bounds-adjustment for FOST or CFPP recursions, much code is used in OMNI. Considering the amount of required calculation time and used memory to store old variable-values, it's worthwhile to consider whether or not such an incredible cumbersome way to adjust bounds is necessary (the implementation of bounds-adjustment in GSolve much simpler, but seems to work just as fine).

In the following, the notation $r^{(n)}$ is used for the recursion-item for which bounds are adjusted. Here, recursion-items are variables *VOL, ASPH, FRMX* and *p0* and bounds are only present after the third solve.

The bound-adjustment depends on obtained values of $r^{(n-2)}$, $V^{(n-1)}$, $r^{(n)}$ and $V^{(n)}$. To try and make the following adjustments somewhat more transparent, we introduce some extreme points $A_r^{(n)}$, $B_r^{(n)}$ and $C_r^{(n)}$, with $A_r^{(n)}=r^{(n)}- V_r^{(n)}-V_r^{(n-1)}= r^{(n-2)}$, $B_r^{(n)}=r^{(n)} +V_r^{(n-1)}$ and with $C_r^{(n)}= r^{(n)} +V_r^{(n)}$, all with $n\geq 3$.

We initialize $UB_r^{(3)}:=\max\{10^{-6}, \frac{|V_r^{(3)}|+|V_r^{(2)}|}{2}\}$, $LB_r^{(3)}:=- UB_r^{(3)}$.

(1) $V_r^{(n-1)}\geq 0$ and $V_r^{(n)}\leq 0$

Note:

Situation: $A_r^{(n)}\leq r^{(n-1)}$; $r^{(n-1)}\geq r^{(n)}$; $r^{(n)}\leq B_r^{(n)}$.

The activity of r has decreased, whereas it had increased in the previous recursion. To keep control on the convergence, OMNI makes sure that $r^{(n+1)}$ won't become greater than halfway between $r^{(n-1)}$ and $B_r^{(n)}$, nor less than halfway between $r^{(n)}$ and $A_r^{(n)}$.

Change the relative upper bound $UB_r^{(n+1)}$ to $UB_r^{(n+1)}= \min\{\frac{UB_r^{(n)}}{\alpha}, \frac{M_r^{(n)}}{2}\}$, with α equal to control-parameter RCDIV and adjust the upper bound of r to $U_r^{(n+1)}:= r_{avg}^{(n)} + UB_r^{(n+1)}$. Obviously, increasing α will reduce the feasible region of the model. Increase the relative lower bound to $LB_r^{(n+1)} = \max\{\frac{LB_r^{(n)}}{\alpha}, -\frac{M_r^{(n)}}{2}\}$ and adjust the lower bound of r to $L_r^{(n+1)}:= r_{avg}^{(n)} + LB_r^{(n+1)}$. $HU_r^{(n)}:= 0$ and $HL_r^{(n)}:= 0$.

(2) $V_r^{(n-1)}\leq 0$ and $V_r^{(n)}\geq 0$

Note:

Situation: $A_r^{(n)}\geq r^{(n-1)}$; $r^{(n-1)}\leq r^{(n)}$; $r^{(n)}\geq B_r^{(n)}$.

OMNI makes sure that $r^{(n+1)}$ won't become greater than halfway between $r^{(n)}$ and $A_r^{(n)}$, nor less than halfway between $r^{(n-1)}$ and $B_r^{(n)}$.

Bounds are adjusted as under (1).

(3) $V_r^{(n-1)}\geq 0$, $V_r^{(n)}>0$ and $r^{(n)}$ is not at its upper limit

Note:

Situation: $A_r^{(n)}\leq r^{(n-1)} < r^{(n)} < C_r^{(n)}$.

OMNI makes sure that $r^{(n+1)}$ won't be greater than halfway between $r^{(n)}$ and $C_r^{(n)}$, nor less than $r^{(n-1)}$.

Decrease the relative upper bound to $UB_r^{(n+1)}= \min\{\frac{UB_r^{(n)}}{\alpha}, V_r^{(n)}\}$ and define $U_r^{(n+1)}:= r_{avg}^{(n)} + UB_r^{(n+1)}$. Increase $LB_r^{(n+1)}$ to $LB_r^{(n+1)} = \max\{\frac{LB_r^{(n)}}{\alpha}, -\frac{V_r^{(n)}}{2}\}$ and update the lower bound of r to $L_r^{(n+1)}:= r_{avg}^{(n)} + LB_r^{(n+1)}$.

Finally, of course, $HU_r^{(n)}:= 0$ and $HL_r^{(n)}:= 0$ (because no bound was hit).

(4) $V_r^{(n-1)}\geq 0$, $V_r^{(n)}>0$ and $r^{(n)}$ is at its upper limit

Note:

Situation as under (3).
OMNI makes sure that $r^{(n+1)}$ won't become less than $r^{(n-1)}$. The upper bound for $r^{(n+1)}$ is only expanded if its upper limit is hit for the third time in a row. (Various applications of sequential linear programming have shown it worthwhile to expand an upper bound only if it is reached 'several' times in a row. Again with a lack of scientific argumentation, 'several' has been implemented as '3' and cannot be changed by the user. GSolve always expands a variable-bound if it is hit.

Increase $LB_r^{(n+1)}$ to $\max\{\frac{LB_r^{(n)}}{\alpha}, -\frac{V_r^{(n)}}{2}\}$ and update the lower bound of r to $L_r^{(n+1)} := r_{avg}^{(n)} + LB_r^{(n+1)}$. Increase the number of times that this variable hit its upper bound: $HU_r^{(n)} := HU_r^{(n-1)} + 1$. Of course, $LU_r^{(n)} := 0$. If r hit its upper bound for the third time in a row ($HU_r^{(n)} = 3$), reset the number of hits to 0 and expand the relative upper bound to $UB_r^{(n+1)} := \lambda \cdot UB_r^{(n)}$, with λ equal to the value of recursion-parameter RCMULT (=1.3 by default) which can be changed by the user (but should always be greater than 1).

(5) $V_r^{(n-1)} \le 0$, $V_r^{(n)} < 0$ and $r^{(n)}$ is not at its lower limit

Note:

Situation: $A_r^{(n)} \ge r^{(n-1)} > r^{(n)} > C_r^{(n)}$.
OMNI makes sure that $r^{(n+1)}$ won't be greater than $r^{(n-1)}$, nor less than halfway between $r^{(n)}$ and $C_r^{(n)}$.

Shrink relative upper bound to $UB_r^{(n+1)} := \min\{\frac{UB_r^{(n)}}{\alpha}, \frac{V_r^{(n)}}{2}\}$ and update $U_r^{(n+1)} := r_{avg}^{(n)} + UB_r^{(n+1)}$. Shrink relative lower bound to $\max\{\frac{LB_r^{(n)}}{\alpha}, -M_r^{(n)}\}$. $HU_r^{(n)} := 0$ and $HL_r^{(n)} := 0$

(6) $V_r^{(n-1)} \le 0$, $V_r^{(n)} < 0$ and $r^{(n)}$ is at its lower limit

Note:

Situation as under (5).
OMNI makes sure that $r^{(n+1)}$ won't become greater than $r^{(n-1)}$. Expand the lower bound for $r^{(n+1)}$ only if its lower limit is hit for the third time in a row.

$UB_r^{(n+1)} := \min\{\frac{UB_r^{(n)}}{\alpha}, \frac{V_r^{(n)}}{2}\}$; $U_r^{(n+1)} := r_{avg}^{(n)} + UB_r^{(n+1)}$; $HU_r^{(n)} := 0$; $LU_r^{(n)} := LU_r^{(n-1)} + 1$.

If r hit its lower bound for the third time in a row ($LU_r^{(n)} = 3$), reset the number of hits to 0 and expand the relative lower bound to $LB_r^{(n+1)} := \lambda \cdot LB_r^{(n)}$, with λ as under (4).

OMNI writes the required matrix-changes (coefficients and bounds) to an ASCII-file called 'gen.rev' in the OMNI modeling language. Afterwards, the matrix is modified by OMNI, handed over to the solver and the algorithm enters phase 2.

A small sample of 'gen.rev' is given by the following example for a model with recursive pooling activated.

Example 'gen.rev'

```
NAME          GEMMS
ROWS
MODIFY
 E  VA108001
 E  VA108002
       ...etc...
COLUMNS
MODIFY
    YA1049GB  VA108001    -59.951633
    YA104AGB  VA108002    -59.951633
       ...etc...
    SA10801.  VA108001    2393.17041
    SA10801.  VA108009    240.127335
       ...etc...
    SA108.01  VA108001      2.447856
    SA108.01  VA108002      9.989731
       ...etc...
    MA108001  OBJ1         -1.000000
    HA108001  OBJ1         -1.000000
       ...etc...
    WA10801M  OBJ1        -10.000000
    WA10801U  OBJ1        -10.000000
       ...etc...
BOUNDS
MODIFY
 UP BOUND1    SA10801.         0.625
 LO BOUND1    SA10801.          -1.0
       ...etc...
ENDATA
```

Under the first MODIFY header, we see that the linear approximation rows VA1... obtain type 'E' (equality), so they will become an active part of the model. Under the second MODIFY, in each row the first name is the variable-name and the second is the row-name. The number at the end of the row is the new coefficient for the variable. Under the last MODIFY section, bounds are adjusted. We see that the upper bound of a delta-variable SA10801. will become equal to 0.625 (=1/RCDIV) and the lower bound stays -1.0. Obviously, this variable obtained a negative value in the initial solve.

3.6.4.2 GSolve + LP-solver

Coefficient-adjustment

Coefficients are adjusted in pretty much the same way as implemented in OMNI, since it follows directly from the linearizations, leaving hardly any room for different interpretations. Considering required calculation time, the most important difference is that GSolve changes matrix-coefficients in its internal memory, instead of writing everything to a file, which is much more efficient.

GSolve also circumvents numerical problems in another way: coefficients are simply taken equal to 0 if their evaluation is somewhere in the interval $(-10^{-6},10^{-6})$. Furthermore, if division by 0, or a logarithm of a value ≤ 0 would occur, the result of the calculation is defined as equal to 0.

Another difference between coefficients calculated by OMNI and by GSolve, is the coefficient of a penalty-variable. At the first iteration, GSolve initializes penalty-values for the FOST and CFPP deficit-variables to 10^6. Then, GSolve generates a quasi-random number between 0.99 and 1.0 which is multiplied by all penalty-values to obtain different costs for each penalty-variable. All penalty-variables are defined as equal to 0 in the first iteration.

After every iteration, all penalty-costs are multiplied by 1.5 if the value for PICOST (which equals to the sum of values of all recursion-penalty variables) in the previous iteration is less than 1.05 times the value for PICOST in the current iteration.

Bounds-adjustment

Both the implementation and the description of bounds-adjustments for GSolve is much easier than for OMNI.

Bounds are adjusted if the fact that the solution hasn't converged yet is only due to the fact that PENMAX > PENTOL (so the maximum actual infeasibility is greater than 10^{-6}, but all of the infeasibility-variables are less than 10^{-6}). For more details, see the description of phase 2 for GSolve.

Now, suppose that PICOST ≤ PENTOL < PENMAX and define $I_i^{(n)}$, $r^{(n)}$ and $V^{(n)}$ as before. Furthermore, define

$c_{r,i}^{(n)}$:= coefficient value of recursion-variable r in row i at the n^{th} iteration,
for example $c_{r,i}^{(n)} = -\{VOL^{(n-1)} \cdot p0^{(n-1)}\}$ for r=*ASPH*,

$E_{r,i} := |I_i^{(n)}|$ with r a delta-variable in pooling-row i,

$E_{r,i} := |r^{(n)} \cdot \{c_{r,i}^{(n+1)} - c_{r,i}^{(n)}\}|$ with r any of the variables *p0, FRMX, ASPH* or *VOL* in the fost or cfpp-row i.

Now, for all $r = \Delta s_{c,p}$ in any pooling-linearization i, do:

if $E_{r,i} \geq \max\{0.75 \cdot$ PENMAX, PENTOL$\}$ then
$U_r^{(n+1)} := \min\{ |r^{(n)}/2| ,$ DVARBND$\}$
$L_r^{(n+1)} := \max\{ -|r^{(n)}/2| ,$ -DVARBND$\}$
otherwise don't change this variable's bounds.

Then, for r = *p0*, *FRMX*, *ASPH* and *VOL* (with *VOL* either from a fost- or a cfpp-restriction) and for all active fost- and cfpp-linearizations i, do:

if $E_{r,i} \geq \max\{0.75 \cdot$ PENMAX, PENTOL$\}$ then
$U_r^{(n+1)} := r^{(n)} + |V^{(n)}/2,|$
$L_r^{(n+1)} := r^{(n)} - |V^{(n)}/2|,$
otherwise, don't change this variable's bounds.

3.6.5 Comments on PSLP implementation

From the description in the former sections, some comments on the PSLP implementations in OMNI and GSolve can be made, which may be interpreted as advised improvements for future releases. These comments are listed below, followed by an indication of required implementation-times.

Initialization

1. OMNI takes care of the initialization of the models, both for OMNI itself and for GSolve. For nonlinear p-value and CFPP blending, vital deficit-variables should be generated.
2. If recursive p-value or cfpp calculations aren't active, the specific quality-constraints shouldn't be activated. Currently, these constraints could be restrictive, even though they don't represent the right nonlinearities.

PSLP algorithm

3. OMNI uses much memory and many checks to derive the required adjustments of bounds. Considering the little influence of such a detailed bounds-calculation on the recursion-results, it may be worthwhile to check performance-improvement when this is simplified.
4. Following the previous comment, OMNI allows bounds to be extended when they are hit three times in a row. However, since bounds on cfpp- or fost-related variables aren't available before the fourth iteration, for such variables this can't happen sooner than after the sixth iteration. Usually, convergence will already be achieved or almost achieved by then, which makes it quite worthless with respect to the required calculation-time and data-storage.
5. OMNI allows lower and upper bounds of recursion-variables to differ only $2 \cdot 10^{-6}$. Due to the number of decimal places in the matrix file, this can lead to a fixed bound, and possibly to an infeasible matrix for a feasible model.
6. GSolve offers little flexibility to the user: bounds on feasibility-variables for CFPP or FOST cannot be set and only a few parameters of the control-table are used. Specifically, GSolve's tolerance-parameter PENTOL should become available to the experienced user.
7. With pooling, GSolve should recognize an originally infeasible model (with deactivated pooling), because then, running the recursion can never result in a feasible model.
8. GSolve should keep track of actual convergence in case of a locally infeasible model: if a model is locally infeasible, all coefficients in the linearized constraints may have converged, but some recursion-feasibility variable(s) will have a positive value, greater than the tolerance. Currently, Gemms checks this infeasibility, finds it greater than tolerated, so adjusts the matrix and runs the model again. However, since all coefficients are converged, the matrix won't change, except for penalty-coefficients, leading to the same solution. This loop continues until the maximum number of iterations is reached.

Required effort

The first remark should definitely be implemented, which is easy, because only a small error in the current OMNI source code must be corrected. The second remark is

of less importance, but may be worthwhile. Implementation and testing will be more difficult and add-up to about eight hours.

The comments made for the PSLP algorithm are all relatively easy to implement. The improvements of GSolve's recursion will take some more time than the ones for OMNI. Although making the convergence parameter available to the user is simple, both the seventh and the eighth improvement are quite time-consuming (a little more than a day, probably).

All recursion-improvements together will require about four days of implementation and testing time.

3.7 On the convergence of PSLP

The philosophy of the PSLP approach was introduced by Griffith and Stewart of the Shell Development Company in 1961 and has been widely used since then, especially in the oil and chemical industries. The principal advantage of the method is its ease and robustness in implementation for large-scale problems, given an efficient and stable linear programming solver. As shown by Bazaraa et al. [3], if the optimum is a vertex of the linearized feasible region, then a rapid convergence is obtained. Once the algorithm enters a relatively close neighborhood of an optimal solution, it essentially behaves like Newton's method with a quadratic convergence rate. Hence, highly constrained nonlinear programming problems that have nearly as many linearly independent active constraints as variables are very suitable for this class of algorithms. Real-world nonlinear refinery models tend to be of this nature and are therefore usually solved successfully with PSLP.

On the negative side, SLP algorithms exhibit slow convergence to nonvertex solutions and have the disadvantage of violating the original nonlinear constraints on their route to an optimal solution.

Bazaraa et al. [3] state that it can be shown that the PSLP algorithm either terminates finitely or leads to an infinite sequence of solutions $\{x_k\}$ which has an accumulation point under some specific conditions. Every such accumulation point is a local optimal solution. Still, convergence of PSLP cannot be proven in all cases, but has shown in practice to be a robust method. For a model with nonlinear constraints (as in Gemms), this also holds for the Generalized Reduced Gradient Method as applied by many nonlinear programming solvers such as CONOPT.

3.8 Performance test GRG and PSLP

This section describes a performance test between various combinations of modeling-system and nonlinear solving-technique: Gemms with OMNI, Gemms with GSolve and Aimms with CONOPT. To gain insight on the performance of the technique, the influence of the used LP-solver in Gemms must be eliminated as much as possible.

Therefore, if possible, different solvers are used to solve the problem. XPRESS is the only solver which can be used both by OMNI and by GSolve.

The test is based on a simple refinery-model with three pools present, which create the nonlinearities in the model. Since pooling probably is the most important and most often modeled type of Gemms' nonlinearities, it is assumed sufficient for the test, only to model this type.

Only one basic model is used for the following tests with several small modifications between runs, which indicates that a broad test wasn't the aim of the research for this report. Indeed, the goal of the test here, is to give an indication on whether or not it's recommendable to solve refinery-planning models with a nonlinear solver.

Because of the fact that the constraints of the model are mainly and that the objective function is also linear, the nonlinear solver mightn't be expected to work as efficiently as PSLP. As was described in section 3.2, nonlinear solvers normally derive their search direction from the gradient of the objective function, which, of course, is of no use here. Then, the steepest ascent method might be used, which is known to perform poorly. Still, since according to Hillier and Lieberman [7] and others, sequential linear programming is supposed to be particularly suitable for completely linearly constrained optimization problems, a comparison of efficiency is worthwhile.

3.8.1 Description of the testmodel

basic linear model

A completely linear model lies at the basis of the test. In this model, no pools are present, but they can easily be inserted to blend to various products or to go to stock. To activate pooling in Gemms, the user only has to switch on the REPO parameter. In AIMMS, the nonlinear pooling-constraints must be activated by removing commentary-marks, and the solving-technique must be set to 'nlp'. In both systems, if pooling isn't active, components which had to go to some destination via the pool, can now all go straight-run.

The modeled refinery (which looks somewhat like the actual refinery in Godorf, Germany) has only one market. The planning-scope for this refinery is two periods, with each period defined with a length of 14 days. The refinery is reasonably complex, containing three crude distillers, three high-vacuum units, two platformers, two hydro-desulfurizers, a hycon, a visbreaker and two 'TGU's.

The basics of the model as implemented in the AIMMS modeling language are taken from G. J. van Rooijen [11]. These were adjusted to coincide with Gemms functionality (for example, implement one of Gemms' internal formulae in the AIMMS model). All data available in the AIMMS model is manually converted to OMNI tables, which are presented to Gemms to generate an equivalent model.

Typical constraints in the model are:

- required qualities of final grades (e.g. minimum density),
- unit-capacities (e.g. maximum crude oil throughput on a crude-distiller),
- material-balance (total weight of produced component must equal the total weight of consumed component, for example in blending),
- market requirements (contracts with customers must be held),

Typical variables in the model are

- quantity of a specific crude-type purchased in a specific period,
- quantities of blended products in a specific period,
- unit-throughput (that is, quantities coming into each unit), running at a specific mode of operation in a specific period.

The basic model covers 2 periods. At the modeled refinery, 7 different crude-types and 1 condensate can be purchased which can be routed through 14 different units, leading to 413 possible components. The refinery can produce 14 final grades, some of which only via a pool (if active). These products are sold to only one market.

Three pools, PMG (premium mogas unleaded), HSHGO (high sulfur gasoil) and KERPL (kerosene pool) can be activated, which can blend to several products or go into inter-period or closing stock. Here, components which can blend to any of the pools, cannot blend to anything else if pooling is activated. Pools (or, in case of the basic model, their possible sources) can go into inter-period and closing stock. Inter-period stocks (end stocks of any period but the last) become attractive because purchasing-prices in the second period are assumed higher than in the first, with, just for convenience, all other data equal for both periods. Components (pools) in closing stock (end stock of the last period) are assumed to have no value. All stock-quantities are limited.

A summary of the basic model's characteristics is given by the following table.

Table 3.6 *Summary relevant characteristics linear testmodel*

	Total
Refineries	1
Markets	1
Periods	2
Products	14
Pools	-
Units	14
Components	413

The basic, 2-period, linear model contains about 1200 constraints, 2500 variables and 17500 non-zeroes. These numbers will differ slightly for both Gemms and AIMMS, since specific rows and variables for reporting-purposes or others are present.

Modeling pooling in AIMMS

The basic model can be written as

max $c^T x$
s.t.
$Ax \sim b$

with ~ meaning any of ≤, ≥ or =, and with (m x n) matrix A, n-vector x and m-vector b. Most variables must be greater than or equal to zero, but some are free.

The nonlinear constraint with pooling activated can be written as $x^d_i \cdot (\Sigma_i x^p_i) = x^p_i \cdot (\Sigma_r x^d_i)$ for all i, d and p, with x^d_i the quantity of component i going to destination d and with x^p_i the quantity of component i going into the pool.

To improve the efficiency of a non-linear solver, such a nonlinear constraint can best be written as a combination of constraints:

$$y^p = \Sigma_i x^p_i$$
$$w^d = \Sigma_r x^d_i$$
$$x^d_i \cdot y = x^p_i \cdot w,$$

because the derivatives of the nonlinear equation, which are used by CONOPT's reduced gradient method, have become much easier to compute. Therefore, this is the set of constraints which is implemented in AIMMS to represent the possible pools (see appendix C for some details).

The models are validated in section 3.8.2, and the actual performance tests and their results are presented in section 3.8.3.

3.8.2 Model validation

The basic (linear) model as described above is used for model validation. That is, to check if Gemms and AIMMS indeed model the same refinery-structure with the exact same details. To check the aspects of multiple periods and stocks, three variations of the basic model are run both with Gemms and with AIMMS:

model 1a:	1 period ; stocks are possible
model 1b:	2 periods ; stocks aren't possible
model 1c:	2 periods ; stocks are possible

The availability of stocks in a 2-period model makes it much more complex, since component-balance constraints for both periods are linked together via the stock carry-over. For a 1-period model, there's no need to switch stocks on and off, because the complexity isn't changed much.
In order to compare the models implemented in AIMMS and in Gemms, first various details (e.g. composition of component-balance constraints and the objective function) are checked in detail in the generated matrix-files from both systems.

Second, resulting objective functions and optimal purchase-quantities are examined. Tables 3.7 and 3.8 give the results for all models. Note that the second table doesn't differentiate between LP-solvers, since crude purchases were all the same when the same system was used. (The LP-solvers did differ for some other variables, so the model doesn't have a unique solution.)

Table 3.7 *completely linear model*

Modeling-tool	Solver	Model	Iterations [1]	Solving-time [2]	Objective value
Gemms/OMNI	HSLP	1a	1851	31	5821.68
Gemms/OMNI	Xpress		1422	15	5821.68
Gemms/GSolve	OSL		1589	26	5821.68
Gemms/GSolve	Xpress		2004	20	5821.68
AIMMS	Cplex		1362	6	5821.87
AIMMS	CONOPT		2022	66	5821.87
Gemms/OMNI	HSLP	1b	3587	89	4812.36
Gemms/OMNI	Xpress		3429	27	4812.36
Gemms/GSolve	OSL		3005	55	4812.36
Gemms/GSolve	Xpress		3622	29	4812.36
AIMMS	Cplex		2709	14	4816.17
AIMMS	CONOPT		4017	191	4816.17
Gemms/OMNI	HSLP	1c	5343	147	5159.09
Gemms/OMNI	Xpress		4429	33	5159.09
Gemms/GSolve	OSL		3163	61	5159.09
Gemms/GSolve	Xpress		4101	36	5159.09
AIMMS	Cplex		2547	16	5160.84
AIMMS	CONOPT		5258	299	5160.84

[1] Note that this is the number of iterations required by the Simplex method (except for CONOPT), not to be mixed-up with the number of iterations used for recursion.

[2] Time in seconds. Only the time to solve the model is reported, since matrix generation and reporting times aren't relevant here.

Table 3.8 *Optimal purchase-quantities in linear 2-period model with stocks (1c)*

Crude-type	Period	Optimal purchase (ktons)	
		Gemms	AIMMS
Arabian Heavy	1	66.9	66.9
	2	79.3	79.3
Brent	1	76.4	76.6
	2	0.0	0.0
Kuwait	1	0.0	0.0
	2	0.0	0.0
Nigerian Light	1	83.5	83.5
	2	0.0	0.0
Statfjord	1	59.5	59.2
	2	96.6	96.6
Oseberg	1	55.6	55.8
	2	54.5	54.6
Ural	1	0.0	0.0
	2	0.0	0.0

Conclusions

Very small differences were found during the first check (detailed constraint comparison), because of differences in decimal places. Therefore, some adjustments were made, but some differences in decimal places were considered irrelevant and accepted. This also explains the small differences in resulting objective values, found in table 3.7. The small differences in crude-purchase can be declared both by differences in decimal places and by the fact that the model doesn't have a unique solution.

Therefore, the Gemms and AIMMS implementations of all variations of the basic model are accepted as being equivalent and can be used for the performance test in the following section.

Note that table 3.7 shows the influence of the solver used to solve the model. First of all, the nonlinear solver CONOPT shows poor performance for a completely linear model. This isn't surprising because all intelligence within the solver is based on a nonlinear model. Still, it strengthens the expectation that this solver mightn't perform to well on a nonlinear, but mainly model. Second, the HSLP solver which is distributed with the OMNI system, obviously performs bad, relative to OSL, XPress and CPlex. Because this solver will greatly decrease the performance of a recursive run, tests are also done with the XPress solver. Still, neither HSLP nor OSL should be left out of the tests, since actual planners at refineries may use (just) one of them. An extensive report on efficient modeling and solving (including tests with CPlex, OSL and XPress) is found in the thesis of van der Tuijn [13].

3.8.3 Testresults

With the linear part accepted to be equal for the Gemms and AIMMS models, pooling can be activated to look at the performance of the nonlinear solving-methods. The added nonlinear constraints in AIMMS are straightforward (see Appendix C) and Gemms' PSLP algorithm for pooling has already been validated, so the models may still be assumed to define the same refinery-structure with present pools.

As before, three models are run, now with the three pools active:

model 2a: 1 period ; stocks are possible
model 2b: 2 periods ; stocks aren't possible
model 2c: 2 periods ; stocks are possible (pools only)

Table 3.9 shows the relevant results for these models, which are all run with various modeling systems and solvers, if possible. Tables 3.10 through 3.12 show detailed pool-related results for the largest and most interesting model, model 2c. The composition of the pools is left out of the table, in order to remain readability (there are about 80 different components which can blend to some of the pools). For OMNI, GSolve and AIMMS, the results presented in these three tables are obtained with the solvers XPress, OSL, and CONOPT, respectively. The reasons for this choice are practical: with OMNI/HSLP, model 2c can't be solved correctly, and for GSolve/XPress, no solution-output routines were available at the time of the tests. Other solvers cannot be applied by OMNI or GSolve, and for AIMMS, the only available nonlinear solver at the ORTEC premises is CONOPT.

Table 3.9 *nonlinear model: pooling activated*

Modeling-tool	Solver	Model	Recur-sions [1]	Time till feasible [2]	Solving-time [3]	Objective value
Gemms/OMNI	HSLP	2a	8	--	145	5702.72
Gemms/OMNI	Xpress		8	--	32	5702.75
Gemms/GSolve	OSL		9	--	34	5702.72
Gemms/GSolve	Xpress		8	--	28	5702.72
AIMMS	CONOPT		--	48	179	5702.87
Gemms/OMNI	HSLP	2b	8	--	240	4669.83
Gemms/OMNI	Xpress		8	--	135	4669.86
Gemms/GSolve	OSL		8	--	80	4669.79
Gemms/GSolve	Xpress		8	--	42	4669.79
AIMMS	CONOPT		--	165	442	4672.85
Gemms/OMNI	***HSLP***	2c	***12***	***--***	***1440***	***4823.53***
Gemms/OMNI	Xpress		6	--	142	4995.65
Gemms/GSolve	OSL		6	--	96	4995.55
Gemms/GSolve	Xpress		6	--	52	4995.55
AIMMS	CONOPT		--	340	1140	4996.76

[1] The number of recursions required for PSLP in OMNI and in GSolve, including initialization.
[2] Times in seconds, required for CONOPT to find a feasible solution to start its GRG method.
[3] Times are reported in seconds and refer to the total required time to solve the model.

The row of table 3.9 with a bold italic font shows a poor performance and result. However, no conclusions concerning the efficiency of the PSLP algorithm can be deduced from this result. Further research showed that the bad results are due to a bug in the HSLP-solver: in the fifth run, HSLP requires much calculation time (about 15 minutes) and comes up with the wrong solution and objective value. Of course, from then on, further recursions are worthless. The LP-matrix of the fifth run was also presented to Xpress and OSL, which did result in the correct objective function of 4995.65. Then, the recursion ends after only one more run.

Table 3.10 *Pooling results 2-period model with stocks, pool PMG*

Solving technique	Period	Total weight (kT)	Weight (kT) blended to product				Weight to stock (kT)
			BFS	SUP	BFN	SSP	
OMNI	1	63.4	36.4	4.9	5.0	4.0	13.1
Gsolve	1	63.4	34.5	4.9	6.9	4.0	13.1
AIMMS	1	63.4	34.5	4.9	6.9	4.0	13.1
OMNI	2	36.5	22.4	4.3	4.5	5.3	0.0
Gsolve	2	36.5	22.4	4.3	4.5	5.3	0.0
AIMMS	2	36.5	22.4	4.3	4.5	5.3	0.0

Table 3.11 *Pooling results 2-period model with stocks, pool HSHGO*

Solving technique	Period	Total weight (kT)	Weight (kT) blended to product		Weight to stock (kT)
			AGO	IGO	
OMNI	1	21.1	0.0	0.0	21.1
Gsolve	1	21.1	0.0	0.0	21.1
AIMMS	1	21.1	0.0	0.0	21.1
OMNI	2	33.0	17.2	15.8	0.0
Gsolve	2	33.0	17.2	15.8	0.0
AIMMS	2	33.0	17.3	15.7	0.0

Table 3.12 *Pooling results 2-period model with stocks, pool KERPL*

Solving technique	Period	Total weight (kT)	Weight (kT) blended to product			Weight to stock (kT)[1)]
			KERO	AGO	IGO	
OMNI	1	24.2	24.2	0.0	0.0	--
Gsolve	1	24.2	24.2	0.0	0.0	--
AIMMS	1	24.1	24.1	0.0	0.0	--
OMNI	2	18.7	15.0	3.7	0.0	0.0
Gsolve	2	18.7	15.0	3.7	0.0	0.0
AIMMS	2	18.7	15.0	3.7	0.0	0.0

[1)] Pool KERPL isn't allowed to enter inter-period stock.

Conclusions

In the following conclusions, the outlier of solving model 2c with HSLP, is disregarded.

First, following from table 3.9, the recursion procedures of OMNI and GSolve perform equally well, since for every model, the number of recursions required to converge is practically equal. The required calculation time, however, highly depends on the LP-solver used. Therefore, conclusions on this can only be based on comparisons of results when using the same solver, XPress. Especially for the bigger models, GSolve performs much better than OMNI.

The biggest part of GSolve's benefit relative to OMNI, is declared by the way matrices are generated and presented to the solver. GSolve handles everything in internal memory, whereas OMNI creates an ASCII-file to hold the matrix after every solve. This file is modified and presented to the LP-solver to run a second iteration. For big models (like 2b and 2c), this modification takes about 7 seconds, which should be disregarded to make solid conclusions about the actual recursive algorithms.

Still, disregarding the required time for matrix-modification in mind, GSolve seems to perform better than OMNI. This is caused by two things:

- for recursion-variables, OMNI keeps track of bounds in a cumbersome way, requiring much memory and calculation time
- GSolve doesn't check convergence of variables

The second reason is likely to have the biggest influence and is in fact missing in GSolve's implementation of PSLP.

The NLP-solver CONOPT obviously performs worst. Even finding a feasible solution is more time-consuming than for the other efficient LP-solvers to conclude their recursion, and this is only one third of the total required time. CONOPT performs relatively worse while the model-size and complexity increases.

3.8.4 Relevance of pooling

Activating pooling in a Gemms model may be quite relevant. Usually, the resulting optimal crude diet won't be affected much (because it's mostly restricted by other aspects of the model like marketing constraints) by pooling, but both the resulting objective function and details of the refinery processes will.

The objective function will of course decrease because of the extra constraints, but often no more than about 1% (which was found by running several actual refinery-models with pools). Still, this may very well be in the order of magnitude of half a million US dollars. Furthermore, the objective function shouldn't be the item to be most interested in. More important are resulting refinery-processing like crude purchases and processing, product composition and unit capacity usage. As was mentioned before, in actual refinery-models purchases usually aren't influenced much by pooling, but other processes are.

The influence on refinery-processes depends much on the possible pool-destinations. First, if a pool can be unit-feed, this unit will be strongly restricted (by specific pool feed-ratios), which is likely to lead to a decrease in the unit's capacity usage. This can be very important for planning studies. Second, if a pool can blend to a final grade, the

product composition in the optimal solution of the pooling-model will differ from the one of non-pooled model, which leads to a different demand for components.

In conclusion, for a user it's worthwhile to decide which conclusions her or she wishes to draw from the model-solution. If crude purchases are the only things in which the user is (currently) interested, pooling could be deactivated in order to speed-up the solution process. If other details are relevant and pools are present in the real-world refinery, it should be modeled.

3.9 Conclusions

Possible benefit of nonlinear solver

As it appears in the testmodel, using a nonlinear solver doesn't look much promising for Gemms models. Although the model is simpler than most which are used in practice, the basics are the same: thousands of linear rows, a few nonlinear constraints and a linear objective function.

Because of the fact that most nonlinear solvers are dedicated to problems with a nonlinear objective function (and possibly some nonlinear constraints), their implemented algorithms are also based to solve such problems with the highest efficiency. Therefore, if the model had a nonlinear objective function and/or much more nonlinear constraints, results might very well be completely different. Nonlinear solvers will benefit from the fact that they adapt a feasible solution in such a way that it should move in the direction of maximizing the objective value (using its gradient). Sequential linear programming doesn't use any smart way to derive a new solution from the current one and is likely to perform worse. Furthermore, if many more nonlinear constraints would be present, sequential linear programming may become less efficient because of the (bad) linear approximation of many nonlinear constraints in the same initial point.

In conlusion, although a nonlinear solver doesn't seem very promising for the current Gemms nonlinearities, it seems worthwhile to continue research with the AIMMS test-case, since this case has now been validated and can easily be extended to hold more nonlinearities. Once research is continued, a whole new area of modeling becomes available and many more nonlinearities might become worthwhile modeling, like:

- nonlinear influence-function of additives on properties (like octane number or viscosity), which Gemms approximates by a step-wise linear function (see the example in Appendix D).
- nonlinear unit-yields,
- nonlinear property-blending, other than CFPP or p-value

One might think of hundreds of nonlinearities which could be present in refinery-planning, but this will definitely lead to useless models due to huge calculation-times. Further research should show how far one can go with nonlinear modeling, given the current status of nonlinear solver-efficiency. It must be stressed that efficiency is of

the utmost importance for 'normal' Gemms runs, considering the little time planners have available. (This importance will be less for strategic studies or studies like cargo analysis, which is presented in chapter 4.)

Be careful when modeling nonlinearities

If more nonlinearities (e.g. using AIMMS) are modeled, this should be done very carefully and one should know how to do this as efficiently as possible. One example is the variety of ways to implement the pooling-constraints, and the fact that simplifying their resulting derivatives reduces required calculation times.
Furthermore, the user should still trust the model-outcomes. This trust may decline when more nonlinearities are modeled. First of all, the resulting solution will strongly depend on the initial solution. A simple example is the maximization of x^2, for $0 \leq x \leq 2$. Most NLP-solvers initialize variables to either zero or one of their bounds. Then, the gradient of the objective function is evaluated and if it equals 0, the resulting solution is stated locally optimal. For the example, many solvers will return a locally optimal solution $x=0$. To gain insight in the quality of the local optimum, several initial solutions should be used, requiring more calculation time. Second, standard LP-sensitivity analysis is no longer possible (although this also holds when using SLP).

Recommendations for PSLP implementation

All recommended changes to Gemms' PSLP implementations are given in section 3.6.5.

Chapter 4

Modeling cargo purchases

4.1 Introduction

In current Gemms, all variables are continuous. Therefore, variables representing the number of kilotons to be purchased of some crude-type can have any real value within specified bounds. However, crude is always delivered by fully loaded oil-tankers with a fixed capacity, which means that crude can only be purchased as fixed cargo-sizes.

Two possible ways to deal with the fixed cargo-sizes are considered here:

- using the 'enumeration approach' to evaluate the relative value of cargoes,
- modeling the process as Mixed Integer.

The goal of cargo evaluation is the same as that of current Gemms: to determine a crude diet which maximizes total refinery-profit.

The two modeling-possibilities differ much. Not only do they solve different problems, also their theoretical background differs completely. The first is based on planners' experience and describes the way some planners currently tend to make their decisions: the software system presents a large list of relative cargo values, from which the planner him- or herself must choose a cargo diet which is both profitable and possible to process. The second is based on mathematical theory and presents just one solution: a cargo diet which is optimal for the underlying model.

This chapter will show the possibility to implement both modeling techniques in PlanStar. The required adjustments to allow PlanStar to run the enumeration approach will be discussed in full detail, including required data-entries, database changes, C++ source code, and a list of advantages and disadvantages. From this list, along with the complete description of the techniques, it should become clear to readers and users whether or not they wish for the implementation of any or both of them in PlanStar.

During the development of the technical specifications, required changes to OMNI source code have been minimized, because it's likely that this modeling language will be replaced in the future. Therefore, any effort in this area is considered a regret investment.

4.2 The enumeration approach

4.2.1 Goal

Usually, the relative value of a cargo (with respect to other cargoes), calculated from the LP solution, is more important for the crude-planner than the calculated absolute cargo value (gain in the objective function due to purchase of the cargo). The enumeration approach calculates these relative cargo-values and presents a list to the user with several important results for specific cargoes or cargo-combinations. Such results are the obtained objective function, marginal value, cargo size and a so-called 'cargo value'.

The goal of the enumeration approach is to give a list of results from which the crude-planner can more or less easily produce an ordered list of attractive crude-cargoes. The results are obtained from a Gemms model of the refinery for which the cargoes are evaluated. This section discusses how the enumeration approach should be implemented in the PlanStar system. For this, first a solid functional specification of the method is required.

4.2.2 Functional specifications

This section describes the functionality of the enumeration approach.

4.2.2.1 Assumptions

First of all, we assume that an appropriate model of the refinery and its crude-processing aspects is already available (e.g. the Gemms data-dictionary or PlanStar database). The enumeration approach doesn't change this model other than by replacing possible purchases to fixed purchases for the cargoes under evaluation. Usually, the single-cargo evaluation method is used, which fixates the purchase of only one crude-type to its cargo-size. When so-called 'synergetic cargoes' or 'combi-cargoes' are present, the combi-cargo method is used, which fixates several cargoes.

The most important assumption made is based on planners' experience: *the value of a particular cargo doesn't depend on the fact that part of the remaining cargoes in that planning period also have to be processed completely.* Of course, the absolute value (value of the objective function) of a cargo may change because of differing remaining process-obligations, but the resulting sorted 'shopping-list' of attractive cargoes is assumed not to be influenced.

Synergetic and combi-cargoes

The only time when remaining process-obligations can really influence the crude-rankings, is when synergetic cargoes are evaluated. Cargoes A and B are synergetic if cargo A becomes more valuable when cargo B is also purchased or processed (e.g. a high-sulfur crude A and a low-sulfur crude B). Combi-cargoes are crude cargoes

which must be purchased together (for example, contractual obligations or two crudes on the same oil tanker). In both cases, the combi-cargo method is used, which uses the enumeration approach with all synergetic cargoes modeled as fixed crude-purchases.

4.2.2.2 Running the enumeration

To run the enumeration approach, the user should enter data in a new PlanStar window. This table must contain all relevant information for the cargoes or cargo-combinations which must be evaluated. For all crudes available in the cargo (possibly combi or synergetic), a type must be entered:

- 'main' for the main crude under evaluation
- 'combi' if this crude is in a combination with the main crude
- 'reqd': if this crude is required to purchase the main crude. That is, the main crude can only be purchased if the required crude is also purchased. See section 4.2.3.3 for an example of the PlanStar entry-screens.

In new cargo-detail tables, the user can enter which range of cargoes must be evaluated for some crude-type. That is, if a minimum number of N and a maximum number of M is entered, then the following models are run:

- purchase N-1 full cargoes (disregard if N=0),
- purchase N full cargoes,
- purchase N+1 full cargoes,
- ....
- purchase M full cargoes.

This allows the user to select multiple cargo evaluations by entering just a minimum and a maximum number of cargoes. The purchase of K (>0) cargoes is evaluated by comparing it with the results of purchasing K-1 cargoes (see section 4.2.2.4). In case of combi- or synergetic cargoes, a multiplication factor must be entered. This factor equal to j (j=0,1,2,...) implies that if K cargoes of the main crude are purchased, then ('at least' in case of a reqd-type, 'exactly' in case of combi) $j \cdot K$ cargoes of the combi- or required crude must be purchased. If this linear relation doesn't hold, the user can obtain the required relations by changing the cargo-sizes. By default, the multiplication factor for combi-cargoes equals 1.

The cargo-evaluation method is activated by choosing a new 'Run cargo evaluation' option of the PlanStar 'Run' menu. Before activating the cargo-evaluation, the user should enter all required data-entries in new entry-screens like the ones presented in section 4.2.3.3. In this table, he or she can (de)activate specific rows, which enables the user to evaluate only several of all available cargoes. Because of the fact that PlanStar has a relational database, integrity check are active while entering the cargo-specific rows (e.g. the entered crude oil must be defined in the reference tables).

For (combi-)cargo evaluation, first the original LP-model (without any cargo-data) is run and then, models are run with alterations for fixed cargo-purchases. Of course, because of the full flexibility of the original model, there's no use in fixating the purchase of some crude to 0 if it isn't even purchased in the optimal solution of the

original run. In practice, however, one tends simply to run all required evaluations, including the original run, despite of the fact that superfluous models are run. For now, we'll hold on to this simple procedure. In a later study, if the enumeration method will indeed be implemented in PlanStar, it will be worthwhile to estimate the required effort to build-in some intelligence in order to prevent useless models from being run and to investigate whether or not users see this as a relevant improvement (since the enumeration can still require hours of calculation time).

The user must select on what basis (currency and weight or volume) the resulting cargo and marginal values must be reported. The default unit for cargo value is main Gemms-model currency per metric ton.

Using only multiple runs; no multi-RHS for multiple cargo evaluations

The enumeration approach will be run as multiple LP-runs. After each LP-run, relevant results are stored and exported OMNI data-tables are updated for the next run. Another setup could have been to use Gemms' multi-RHS feature (which is currently used by some planners for cargo evaluation). The following paragraph explains why the first-mentioned setup is preferred.

By using more than one RHS to define the various cargo availabilities, more evaluations can be run after only one data-export, input reading, matrix-generation and solving step, which reduces the total amount of required time. The problem is, however, that beforehand it's unclear how many right-hand-side dependent data can be stored in Gemms' or the LP-solver's internal memory. Technically the number of right-hand-sides could be up to 36, but due to the magnitude of the models, rarely more than 6 different RHS-models can be used. Therefore, the system would have to leave it up to the user to guess beforehand how many right-hand-sides can be used. Other disadvantages of using multiple RHS for cargo evaluation are all necessary data-checks and the loss of modeling flexibility. If multiple right-hand-sides are used for cargo-evaluation, the system must check whether or not the original model doesn't contain any data for multi-RHS runs, since these should be disregarded. Again, it could be left up to the user to make sure that multiple RHS aren't used in the original LP-model. But then still, the user loses the modeling flexibility of running various scenarios (e.g. different prices and/or availabilities).

If the original model is multi-RHS, the enumeration approach allows the user to run cargo evaluations for all available right-hand-sides, just by defining cargo evaluation data for the first RHS. Gemms then copies all evaluation data for the first RHS to all other RHS automatically, if required. If the original LP-model is infeasible or unbounded, the cargo-evaluation won't be run, since there probably are some data-errors within the model which would lead to useless cargo-evaluation results.

Handling commercial constraints

The presence of commercial constraints and/or transactions adds some difficulties to the enumeration method. For example, problems occur if the original model holds a PlanStar 'transaction' restriction such as 'minimum purchase of 40,000 barrels

Northland Crude by tier 1'. Now, if this restriction would remain active during the enumeration runs, we wouldn't get any relevant results. For example, if the enumeration approach tries to evaluate the purchase of one cargo Northland Crude (say 50,000 barrels) with the restriction still active, this would lead to the evaluation of a minimum purchase of 90,000 barrels Northland Crude instead of a the evaluation of the purchase of exactly one cargo (or 50,000 barrels).

Obviously, these kind of availability-constraints must be removed, either by the user or by the system. In this chapter, it is assumed that the system removes them. Below follow some advantages and disadvantages of both options which led to this choice.

Letting the user remove irrelevant constraints

This option is currently used at some refineries. The user makes sure that the cargo evaluation models are feasible and do indeed model the purchase of the evaluated number of cargoes.

Advantages:

- obviously, it is the easiest way to implement cargo evaluation for PlanStar,
- the user can determine which availability-constraints he or she wishes to remain within the model used for cargo-evaluation and which constraints can be removed, therefore keeping the possibility to remain essential restrictions in the model.

Disadvantages:

- the user is forced to keep two models up-to-date: the original LP-model and the adjusted cargo evaluation 'basic' model
- the user might not be able to overlook all possible problems which might occur, for example the one mentioned in the introduction of this subsection. This can lead to misleading results.
- If the user wants all cargo evaluation runs to be feasible, he or she should either remove availability-restrictions for *all* evaluated crude oils together, thereby making the model less realistic, or keep separate availability-data for each cargo evaluation (which is hardly possible to maintain).

Letting the system remove constraints

Constraints which must be detected are:

- commercial transactions like in the example on Northland crude oil,
- commercial constraints which restrict the purchase of the evaluated crude. If one chooses to keep these restrictions (because they may be essential to the refinery), the model with fixed cargo purchases may become infeasible. On the other hand, if these restrictions are removed, crude-purchase becomes less restricted, possibly leading to a solution which is infeasible in practice.

  Considering the fact that most multi-crude commercial constraints are essential to the refinery, the first option is adopted here. That is, these constraints are

kept within the model and it is left up to the experience of the planner to deal with possibly resulting infeasibilities.

The best way to deal with availabilities of evaluated crudes within PlanStar is to remove all of those with the same period/refinery/crude combination as for the crude under evaluation.

Advantages of system intelligence:

- the user can remain working with his or her original refinery model, since all cargo-specific entries can be disregarded when not running the cargo evaluation.
- For each evaluated crude oil, only availabilities concerning this crude and its combination(s) must be adjusted. All others can remain active in the model.

Disadvantages of system intelligence:

- extra implementation time (and cost) required
- it isn't trivial which constraints should be removed

Considering all the above mentioned advantages and disadvantages, it is best to let the system decide which constraints should be removed. The main reason is that the user shouldn't be bothered with this matter and shouldn't be forced to keep different model-versions for both 'normal' and 'cargo' LP-runs.

4.2.2.3 Required running time

Without adding any intelligence to the enumeration approach (such as disregarding superfluous runs), an LP-model is run $2m+1$ times, with m the number of selected cargoes to evaluate (which includes both the evaluated crude-types and all cargoes for each type). If superfluous runs are disregarded by the system, between $m+1$ and $2m+1$ runs are required. The resulting running time, however, probably won't become as much as $2m+1$ times the required time of one model run, because the optimal solution of evaluation run i can be used as a basic solution for evaluation run $i+1$.

4.2.2.4 Reporting

PlanStar should report the following results of the enumeration approach:

1. scenario description
   - name of evaluated crude,
   - names of crudes in the combination (if any)
   - number of cargoes evaluated,
   - cargo sizes
2. objective value when purchasing all, say k cargoes or combinations,
3. objective value when purchasing k-1 cargoes or combinations,
4. added value,
5. cargo value,
6. marginal values for 'first ton or barrel of cargo' and 'last ton or barrel of cargo'

Here, the following definitions are used:

added value of the k^{th} (k=1,2,3,...) cargo purchased =
the objective value when purchasing the cargoes minus this value when purchasing k-1 cargoes.

cargo value of the k^{th} (k=1,2,3,...) cargo purchased =
{added value of this cargo} / {total cargo size}

The user can select the required unit of measurement for these values. This unit will also be used for the reporting of marginal values. Therefore, the cargo value of some crude oil will always be between the marginal value of the first drop of this crude cargo and the marginal value of the last drop.

For an evaluated combination of cargoes (combi or synergy), no marginal value is available. The cargo value is calculated as above, with cargo size equal to the sum of all cargo sizes in the combination. The objective function calculated when the combination isn't purchased is calculated by fixating the purchase of all cargoes in the combination to one less than in the evaluated case.

Because of the fixation of some crude purchases, the model might not be feasible for some evaluation runs. Then, marginal and cargo values will be reported as -9999.

4.2.2.5 Report layout

This section proposes the lay-out of the report generated after running the enumeration approach. The actual representation of the reports depends, however, on the implementation possibilities within Business Objects, assuming that this will become the main reports-application.

Example generated report after enumeration approach

Crude	Dens	Bbl factor	Cargo size	Nr	UoM cargo	M.v. 1st	Cargo value	M.v. last	Obj -	Obj +	Added value
ARH	0.889	6.29	30	1	Kmt	--	0.47	--	45678	45858	180
ARL	0.75	6.29	10	2	Kmt						
ARH	0.889	6.29	30	2	Kmt	--	0.07	--	45858	45883	25
ARL	0.75	6.29	10	4	Kmt						
NIL	0.73	6.29	65	1	Kmt	0.09	-0.03	-0.10	45678	45668	-10
NIL	0.73	6.29	65	2	Kmt	-0.10	-0.14	-0.17	45668	45615	-53
NIL	0.73	6.29	65	3	Kmt	-9999	-9999	-9999	45615	--	--

UoM cargo value: USD/bbl Exchange rate NLG/USD: 1.84
UoM objective value: 1000 USD

In this report, the specific gravity (dens) is reported in order to calculate the number of barrels per metric ton, from which the marginal and cargo values can be calculated by Business objects. Furthermore, 'Obj -' and 'Obj +' give the resulting objective values when the cargo isn't purchased and is purchased, respectively.

In this example, the added and the cargo value are calculated by Business Objects. All others result from Gemms' optimal solution. The cargo value (USD/bbl) for the second combination of 1 cargo ARH and 2 cargoes ARL is calculated as {added value of the second combi}/{total cargo size}=25/{(30·6.29/0.889)+(2·10·6.29/0.75)}=0.07.

For multiple-refinery and/or multiple-RHS models, refinery- and RHS-names or numbers must also be presented by the reports, because cargo values will be both refinery- and RHS-dependent.

4.2.3 Technical specifications

4.2.3.1 Default settings

The following selections are made by default and can be changed by the user:

- units for marginal and cargo values are Gemms' main currency per metric ton
- multiplication factors for combi and main crude equal 1

The following selections *cannot* be changed by the user:

- tier-number of a cargo in the Gemms model equals X
- 'component-type' of entered cargo equals 'CRUDE' in the database
- 'transaction-type' of entered cargo equals 'CRG' (newly defined type) in the database

4.2.3.2 Database

Two new tables are defined to allow for cargo evaluation data-input. Furthermore, the existing tables Comm_Transactions and Constr_Commercial are extended with possible Trans_Type 'CRG' and two new tables are created to hold all relevant cargo evaluation results. For this subsection, the reader is assumed to know the contents of available Oracle tables for PlanStar.

Below, all new Oracle tables required for the enumeration approach are given. In the notation, key-fields are underlined and new fields (not available in current PlanStar) have a bold italic font. An example is given by section 4.2.3.3.

Tables for cargo-input

table Cargo_Definition

VERSION	NUMBER(2)
CARGO_NAME	CHAR(5)
MIN_CRG	***NUMBER(1)***
MAX_CRG	***NUMBER(1)***
DESCRIPTION	CHAR(20)
ACTIVE_YN	CHAR(1)

table Cargo_Details

```
VERSION          NUMBER(2)
CARGO_NAME       CHAR(5)
MATER_TYPE       CHAR(12)        /* always equal to CRUDE */
MATER_LABEL      CHAR(4)
CARGO_SIZE       NUMBER(8,3)
MULTIPLIER       NUMBER(1)
CARGO_TYPE       CHAR(5)         /* Main, Reqd or Combi */
```

New links between tables

The only new links created are:
- tables Cargo_Details to table Cargo_Definition (i.e. all cargoes mentioned in the cargo-details, must be present in the cargo_definition)
- tables Cargo_Details to table Components, with the refinery-field of the first two tables linked to the location-field of the latter (i.e. all combinations of Mater_Type and Mater_Label specified in the first two tables must be defined by key-fields in table Materials).

Details of a crude within a cargo (e.g. prices and definition of VOL (either m^3 or barrels)) is held by the current PlanStar Comm_Transactions table, with transaction-type 'CRG'.

Then, cargoes which must be evaluated are selected from table Cargo_Definiton (those with ACTIVE_YN = 'Y'), crudes present in the cargo are obtained from Cargo_Details and refinery-information and other crude-details are obtained from Comm_Transactions and Constr_Commercial (from the first table, all crudes in the cargo with transaction-type 'CRG' are selected; from the second, the constraint is used, if present, with constraint-name equal to CARGO_NAME).

Tables for cargo-output

table Cargo_Run_Actuals

```
CASE_ID              NUMBER(2)
RUN_ID               NUMBER(2)
CARGO_ID             NUMBER(3)
CARGO_NAME           CHAR(5)
RHS                  CHAR(4)
LP_STATUS            CHAR(4)
OBJECTIVE_FUNCTION   NUMBER(15,4)
NUMBER_ITER          NUMBER(5)
NUMBER_RECUR         NUMBER(4)
NUMBER_ROW           NUMBER(5)
NUMBER_COLUMN        NUMBER(5)
```

table Cargo_Comm_Actuals

CASE_ID	NUMBER(2)
RUN_ID	NUMBER(2)
CARGO_ID	***NUMBER(3)***
CARGO_NAME	***CHAR(5)***
RHS	CHAR(4)
REFINERY	CHAR(4)
PERIOD	NUMBER(2)
MATER_LABEL	CHAR(4)
CARGOES_PURCHASED	***NUMBER(2)***
UOM_CARGO_SIZE	***CHAR(3)***
MARGINAL_VALUE	NUMBER(10,4)
DENSITY	NUMBER(8,3)
BARRELFACT	NUMBER(8,3)

4.2.3.3 Data Entry Screens (PowerBuilder)

First of all, the PlanStar 'Run' menu must be extended with an option 'Run cargo evaluation'. Secondly, the user must be enabled to enter all relevant data in new entry-windows. These windows will become available in a new subsection 'cargoes' of the current PlanStar-section 'commercial'. The cargo-section will hold two subsections: 'cargo-definition' and 'cargo crude details'. The following example will show how the user should enter the cargo-specific data and how these will be saved to the database. Finally, when removing a model-version from the database, the newly created tables must also be removed.

Example 4.1

In this example, we assume that the user wants to evaluate two cargoes. The first is simple, it's just a cargo of Northland crude oil, which can be processed by two refineries, A and D. The second is more complex: it's a combination of Arab Heavy (AH) and Arab Light (AL), which can only be processed if at least two cargoes of Northland (NL) crude are purchased. Furthermore, this cargo can only be processed at one refinery, say refinery D. The following paragraphs show all required data-entries and their conversion to the (extended) PlanStar database. The complexity of the example (both combi- and synergetic) allows the reader to see what entries would be required for more simple cargo-evaluations.

Cargo crude details

First, the user can enter all crude-related data in a cargo-availabilities screen which looks almost the same as the current crude/feedstock availabilities window. This cargo-availability screen is activated when the 'cargo crude details' subsection of the new 'cargo' section in PlanStar is selected. (PlanStar presents these sections as a tree-view.)

Cargo crude details

Period	Rhs	Location	Material label
1	1	A	NL
1	1	D	NL
1	2	D	NL
1	1	D	AL
1	2	D	AL
1	1	D	AH
1	2	D	AH

Again, as for current availabilities, for each entered row, an entry-table appears for detailed information. For the first row, this could read:

UoM Cargo size	bbl
Price	10.5
UoM Price	bbl
Platt's type	Non
Currency	DLRS
FOB price Y/N	N

Note that because of this setup, the user can still run various price-scenarios for crude evaluation by using Gemms' multiple-RHS feature.

Cargo-definition

Another subsection of the cargo section, will be 'cargo-definition'. Here, the user will enter the number of cargoes to be evaluated, the crude types present in the cargo (including its type: 'main' for the crude which is evaluated, 'combi' for a combi-cargo, and 'reqd' for a required crude). Cargo-sizes must also be entered here.

In the cargo-definition window, the user will first define a cargo-name and cargo-specifics in the following table (with entries as for our example):

Cargo-definition

Cargo_ Name	Min #	Max #	Active	Description
CG_NL	1	2	Y	Northland cargo
CG_AH	1	1	Y	Combi AH/AL

The entries under 'min #' and 'max #' define the number of cargoes to be evaluated.

In the same window, below the previous table, data can be entered which must hold all crude-labels within the cargo, including their cargo-type, the cargo-sizes and, for required (synergetic) crudes, a multiplier which tells how many cargoes of the required crude must at least be purchased in order to purchase one cargo of the main crude. Changing sizes and multipliers for required crudes gives the user full modeling-flexibility. For the main crude and for a combi-crude this multiplier should normally

equal 1 (which will be the default). This part of the window is refreshed when the user selects another row in the cargo-definition table. For the combi-cargo, it will look as follows:

Cargo-contents

Crude label	Cargo type	Cargo size	UoM cargo	Multiplier
AH	Main	100	WGT	1
AL	Combi	80	WGT	1
NL	Reqd	50	VOL	2

Refineries

The final part of the cargo-definition window, 'refineries', is required *only if, for any of the crudes within the cargo, the number of refineries which can purchase this crude is less than the number resulting from the cargo crude details.*

That is, for each crude in the cargo, possible destinations (refineries) may be specified. Then, other refineries entered in the cargo crude details are disregarded for the current cargo-evaluation. An example shows the meaning of the refineries-table. The entries at the cargo crude details suffice to allow the Northland cargo to be processed both by refineries A and D. Now, suppose we wish to evaluate this cargo if it must be processed by refinery D. Then, we don't wish to delete any of the cargo crude detail, because prices would be lost. Instead, we fill in the refinery-table as below:

Refineries

Period	Rhs	Crude label	Refinery
1	1	NL	D

Pricing-information for Northland crude at refinery D is still obtained from the cargo crude details. In further cargo evaluations, the user could deselect several runs by setting the 'active' field to 'N'. The data entered at the refineries-table is saved to the database in tables Constraints, Constraint_limits and Constr_commercial.

Cargo-specific data in the database

The following paragraphs show how all entered data of this example is saved to the extended database (note that not all entries, e.g. prices, are given above).

Italic columns cannot be changed by the user. Bold table-headers are key-fields and for table Comm_transactions, bold rows concern main crudes for crude evaluation. Below, the internal representation of the tables is given, which isn't shown to the user in this manner.

table Comm_transactions (1)

Ver	*Trans_ type*	Per	RHS	Location	*Mater_ type*	Mater_ label	*MAXMINFIX*	*Tier*
1	***CRG***	**1**	**RHS1**	**REFA**	***CRUDE***	**NL**	***MIN***	***X***
1	***CRG***	**1**	**RHS1**	**REFD**	***CRUDE***	**NL**	***MIN***	***X***
1	***CRG***	**1**	**RHS1**	**REFD**	***CRUDE***	**AH**	***MIN***	***X***
1	***CRG***	**1**	**RHS2**	**REFD**	***CRUDE***	**AH**	***MIN***	***X***
1	*CRG*	1	RHS2	REFD	*CRUDE*	NL	*MIN*	*X*
1	*CRG*	1	RHS1	REFD	*CRUDE*	AL	*MIN*	*X*
1	*CRG*	1	RHS2	REFD	*CRUDE*	AL	*MIN*	*X*

table Comm_transactions (2)

Quant	UoM_Quant	Price	UoM_Price	Plattyp	Currency	FOBprice_YN
0	**bbl**	**10.5**	**bbl**	**None**	**USD**	**N**
0	**bbl**	**10.5**	**bbl**	**None**	**USD**	**N**
0	**mt**	**97.6**	**mt**	**None**	**USD**	**N**
0	**mt**	**94.8**	**mt**	**None**	**USD**	**N**
0	bbl	9.5	bbl	None	USD	N
0	mt	96.4	mt	None	USD	N
0	mt	95.4	mt	None	USD	N

table Cargo_definition

Version	Cargo_Name	Description	Active_YN	Min_Crg	Max_Crg
1	CG_NL	Northland cargo	Y	1	2
1	CG_AH	Combi AH / AL	Y	1	1

table Cargo_Details

Version	Cargo_ Name	*Mater_Type*	Mater_label	Cargo_ Size	Multi- plier	Cargo_ Type
1	CG_NL	*CRUDE*	NL	50	1	Main
1	CG_AH	*CRUDE*	AH	100	1	Main
1	CG_AH	*CRUDE*	AL	80	1	Combi
1	CG_AH	*CRUDE*	NL	50	2	Reqd

table Constraints

Version	Constr_Name	Constype	Descript
1	CG_NL	Commer	Northland cargo

table Constraint_limits

Ver-sion	Constr_ Name	Period	RHS	MAXMINFIX	UoM_ limit	Limit type	Limitvalue
1	CG_NL	1	RHS1	FIX	VOL	TOTP	50

table Constr_commercial

Ver-sion	*Trans_ type*	Constr_ Name	Pe-riod	RHS	Loca-tion	Mater_ label	Mater_ type	*Tier*	Constr_ Coeff
1	*CRG*	CG_NL	1	RHS1	REFA	*CRUDE*	NL	*X*	1
1	*CRG*	CG_NL	1	RHS1	REFD	*CRUDE*	NL	*X*	1

When the evaluation is run, OMNI tables are created and the models are solved, as is shown by example 4.2 in the following section.

4.2.3.4 Interface modules (C++)

The existing C++ source code must be extended to run the enumeration approach. The following few paragraphs will demonstrate how the enumeration approach is run, that is, what steps must be implemented in C++ source code. Only PlanStar's export-modules are adapted. The import-modules remain unchanged because the enumeration approach won't become available for Gemms standalone (i.e. without PlanStar).

The following description mentions storage of 'required results for cargo evaluation only'. This storage will have to be implemented mostly within OMNI source code, because there's already an OMNI procedure available which stores the whole solution. This can be adjusted accordingly with only little effort (keeping the possible regret investment low). When all evaluations have ended, we wish to have results for all model runs and we wish to have them available in the database such that Business Objects can generate the appropriate report.

A small problem that arises is that the OMNI system doesn't know which evaluation is run, and after every run OMNI writes the solution to the same file, thereby always overriding the solution of the previous run. This is no problem if the solution is saved to the database after every run. However, this action is very time-consuming. Therefore, C++ code will keep track of the solution for each run and will make sure that the total cargo evaluation run ends up with only one solution-file which can be imported into the database.

The OMNI crude-availability tables AVAIL, SPAVL, TIERAV and CRDLMT, possibly with period and/or RHS indicators, will no longer be exported to data-files 'commerci.mod' and 'constrai.mod'. Instead, a new file, 'avail.mod' should be created for these tables in order to speed-up the cargo evaluation. The resulting 'switches' file

will contain an entry to make sure that OMNI reads this new data-file (decision parameter 'FILE8' will be avail.mod). Details follow below.

The C++ routines which are activated when the user selects the running of Cargo Evaluation are as follows:

1. First, some items are checked and the original LP model is run:
   1.1 Count the number of active rows in the cargo-definition table.
   1.2 If no active cargo evaluations are found, print a message like 'no active cargoes found' and *don't run any of the following steps.*
   1.3 Run the normal procedure with the adjustment that Gemms' OMNI availability-tables are saved to the data-file 'avail.mod' instead of to 'commerci.mod' and 'constrai.mod', which is currently the case.
   1.4 If no optimal solution is found for this model, *don't run any of the following steps.*
   1.5 Copy the file 'avail.mod' to 'avail_lp.mod' (overwrite if this file already exists).
   1.6 Store the objective value only.

2. Now, the actual cargo-evaluation can start. The following steps will first generate all required data-files containing the availabilities. That is, files avlXXX_Y are generated, with XXX a three-digit cargo-identification number and with Y a run-number for this cargo.
   Now, initialize XXX to '000'.
   2.1 For each defined and activated cargo, do the following :
   Initialize Y to 0. Get the minimum and maximum number of cargo-purchases of the main crude. Do the following steps for each number of cargo purchases from (minimum minus 1) to the maximum number of cargoes. If the minimum equals zero, then do the following steps from zero to the maximum number of cargoes.
      2.1.1 Generate identification numbers: XXX equals XXX+1, Y = Y+1.
      2.1.2 The data-file 'avail.mod' will now be created. It will contain the following export-results:
         - all rows at table Constr_Commercial with Trans_Typ 'AVL'
         - if the current cargo under evaluation contains a multi-crude commercial constraint (with type 'CRG'), then this constraint is exported to table (P)CRDLMT(N) (with loss of tier-number). Attention must be paid here with combinations, not to lead to a constraint which reads, e.g, 'total purchase AH + AL at refineries A + B = cargo-size', because this could lead to only purchasing crude AH or AL. Multi-refinery single-crude commercial constraints be exported to table TIERAV as a constraint for tier X.
         - all rows at table Comm_Transactions with Trans_Typ equal to 'AVL', except for those which contain a crude/period/location combination which is part of the cargo under evaluation (either as main or as dependent crude)
         - all rows at table Comm_Transactions with Trans_Typ equal to 'CRG' which contain a crude/period/location combination

which is part of the cargo under evaluation. This export will be somewhat different from the usual Comm_Transactions export. That is, purchase will always be fixed, tiers will always equal X and quantities are obtained from the cargo-specific tables (cargo_size or cargo_size·multiplier, see section 4.2.3.2).

2.1.3 Copy 'avail.mod' to 'avlXXX_Y.mod'. Overwrite it if it already exists.

2.2 Run all models without any report-steps. This is done by starting with file avl001_1.mod, copying it to 'avail.mod', running the model and storing the relevant results in a unique temporary file.

2.3 Store all solutions in the database (thereby using the values for XXX and Y to find the cargo-name and the number of purchases).

2.4 Generate Business Objects report with all relevant results.

Example 4.2 cargo-evaluation data-export

This example is an extension of example 4.1. For ease of demonstration, we'll assume that the crude oils which are used in the cargo evaluation, are available without any restrictions in the original LP-model. Otherwise, evaluation-delimiting restrictions must be removed by the system.
The following data is exported for each evaluation run:

Northland cargo evaluation

Evaluation run 1: purchase 0 cargoes Northland (Although '0' wasn't entered as the minimum purchase, this model must be run to be able to evaluate the purchase of 1 cargo Northland crude oil).
Add to file 'avl001_1.mod':

```
TABLE 1CRDLMT1
        >       CG_NL
         AREF     1
         DREF     1
         NL00     1
         BRFV     0
TABLE 1AVAIL1
        >       BMIN   BAREF BDREF
         NL00     0     10.5  10.5
```

Note that if multiple right-hand-sides are active in the original model, Northland cargo evaluation is automatically run for all right-hand-sides. There are no complications here, since if the user doesn't wish results for the other right-hand-sides, he or she should simply concentrate on the results of RHS 1. All other results could then be disregarded and taken the same as those from the original model.

Evaluation run 2: purchase 1 cargo Northland
Just create file 'avl001_2.mod' like 'avl001_1.mod', with a table 1CRDLMT1 entry BRFV = 50.

Evaluation run 3: purchase 2 cargoes Northland
Like before, but now for file 'avl001_3.mod', with BRFV = 100.

Arab Heavy combination evaluation

Evaluation run 4: purchase 0 cargoes Arab Heavy and 0 cargoes Arab Light
First, recreate availability tables to remove data from Northland cargo evaluation.
Then, generate file 'avl002_1.mod' with

```
TABLE 1TIERAV1
        >       DAVLTMX DAVLBNX    DCSTX
         AH00       0          0       97.6
         AL00       0          0       96.4
         NL00                  0       10.5

TABLE 1TIERAV2
        >       DCSTX
         AH00   94.8
         AL00   95.4
         NL00    9.5
```

Note that column headers defining maximum and minimum purchase are used, instead of a column header simply defining fixed purchase. This is because of the way Gemms implements these columns: for fixed purchases equal to 0, Gemms doesn't create any purchase-variables. Therefore, no marginal values for this purchase are obtained.

Evaluation run 5: purchase 1 cargo AH, 1 cargo AL and a minimum of 2 cargoes NL
Change the availability-entries at table 1TIERAV1 to 100, 80 and 100 for crude oils AH00, AL00 and NL00 respectively. Generate the unique filenames.

The system will now run all models and provide the user with the results as described by section 4.2.2.5.

4.2.3.5 Output (Business Objects)

Business Objects must read the new output tables, calculate added and cargo values, adapt values to the reporting unit of measurement and generate a report. This report must be created and look like the one presented in section 4.2.2.4.

4.2.3.6 Documentation

New Help-screens must be created in which the enumeration approach is described.

4.3 Mixed Integer Programming

In principle, modeling fixed cargo sizes is straightforward Mixed Integer programming: integer variables are used for cargo-purchases and continuous variables represent numerous other model-details such as resulting quantity of blended products.

4.3.1 Goal

The goal of a Mixed Integer model is to find a crude diet which maximizes expected total refinery profit. Opposite to the enumeration approach, which only evaluates one cargo or a small combination of cargoes at a time, MIP in fact evaluates all available cargoes together.

Because of the many restrictions introduced into the MIP model, it might become infeasible. Gemms, however, is able to show the user (some of) the infeasible rows, which will help him or her to gain much insight in the model-problems.

4.3.2 Functional specifications

The functionality of the Mixed Integer model is straightforward: the MIP solver searches for a cargo package which maximizes the refinery margin, subject to the familiar linear constraints.

4.3.2.1 Assumptions

Apart from the fact that cargo-purchases are taken into account, no assumptions other than those for the original LP-model are made.

4.3.2.2 Running the Mixed Integer model

Before running the model as mixed integer, the user selects which and how many crude-cargoes are available and enters the cargo sizes. The solver then searches for a crude package which maximizes total profit. If for an available crude-type no cargo-size is active, its availability is deduced from the original model and its purchase is modeled as a rational positive variable.

4.3.2.3 Required running time

The required time for solving a Mixed Integer programming model is larger than for an LP and depends on many things: the distance between the optimal solution of the LP-relaxation and the MIP, the intelligence of the solver (preprocessing, branch-and-bound algorithm), and the number of integer variables and their bounds.

Even if the problem were just IP (just integer variables representing the number of cargoes purchased, no continuous variables) then, with n the number of available cargoes, 2^n possible solutions might have to be evaluated (that is, checked for feasibility and if so, then calculate the objective value). For example, if n=20, then more than 1,000,000 possible solutions might have to be evaluated. Even the fastest computers are incapable of doing this within an acceptable amount of time. In practice, due to smart solution algorithms, this situation will never occur, but still, no guarantee on the actual amount of time required can be given.

When finishing the MIP, several other LP- or MIP-models should be run in order to increase planner's 'feel' for the refinery's situation.

4.3.2.4 Results

PlanStar results from MIP will be practically the same as from the current LP-model. The only things which must be added to the (Business Objects) reports are the cargo-sizes and, for each cargo under consideration, the number of cargoes to be purchased. Because of the integral aspect of the model, marginal values cannot be calculated and reported. Note, however, that marginal values obtained from an LP-model also have little meaning if the rational variables in fact represent integer cargoes. (A marginal value reflects the gain or loss from a 'small' change in cargo purchase. This change cannot be 'small' if only cargoes are available.)

4.3.3 Technical specifications

The user can enter MIP-specific data in a new section of the current 'commercial' section of the PlanStar system. A subsection called 'MIP crude availabilities' will become available as a part of 'commercial'. This is discussed in more detail in the following sections.

4.3.3.1 Database

The database must be extended to hold all relevant cargo-information. For MIP this extension is much smaller than for the enumeration approach. Considering the fact that it is still required to be able to run the original LP model, as well as the enumeration approach at all times, MIP data mustn't override any of the existing tables, so the database is extended. The following sections define the new table, the new data-entry screens and a description of how the entries could be converted to a mixed integer OMNI model. Large changes to OMNI code would be required to be able to convert the model to an appropriate matrix-file. The format of this file will depend on the solver which is used. The required OMNI changes won't be described here, since it is considered to be too much of an effort to investigate this at the moment because of the expected OMNI replacement. Should OMNI remain as the used modeling tool, more effort can be invested in an exact representation of required OMNI changes.

table MIP_Comm_Transactions

```
VERSION          NUMBER(2)
MATER_TYPE       CHAR(12)          /* always equal to CRUDE */
MATER_LABEL      CHAR(4)
PERIOD           NUMBER(2)
TIER             NUMBER(1)
REFINERY         CHAR(4)
RHS              CHAR(4)
MIN_PUR          NUMBER(1)
MAX_PUR          NUMBER(1)
CARGO_SIZE       NUMBER(8,4)
UOM_CARGO_SIZE   CHAR(3)
```

4.3.3.2 Data Entry Screens (PowerBuilder)

First of all, the PlanStar 'Run' menu must be extended with an option 'Run MIP'. When this option is selected, the same sub-menu as for Run LP will be shown, using a text reflecting the selection of Mixed Integer Programming.

Second, at a specific 'MIP crude availabilities' window, the user will be able to enter all required data in almost the same way as is currently done for crude/feedstock availabilities. The window from the current crude/feedstock availabilities is copied, but the window in the lower-right corner will be somewhat adjusted. That is, an extra row-name 'cargo-size' will be present, including its data-entry field. Now, the entry at the 'quantity' field of the lower-left table will define the minimum, maximum or fixed number of cargoes which can be purchased.

Specific constraints can be entered in the same way as for the current LP implementation. Therefore, no adjustments are made here.

Finally, the PowerBuilder parts must be extended to make sure that the new MIP-tables are also removed when the user removes a model-version from the database.

4.3.3.3 Interface modules (C++)

The export-modules must be adjusted in about the same way as described by section 4.2.3.4, though very much simplified. If mixed integer modeling is activated, the only change to current data-export is the exporting of table Comm_Transactions: first, all commercial transactions with Trans_Typ equal to MIP must be exported to OMNI-tables. Then, all other commercial transactions can be exported, except for records with Trans_Typ equal to 'AVL' and with all other entries at the key-fields (e.g. tier-number) the same as for the exported MIP-records.

Note that a full research on the usage of Mixed Integer refinery-modeling is beyond the scope of this report. Here, just the aspects concerning cargo-modeling are discussed. At a later stage, one might consider how to enable the user to specify 'if-

then' constraints, which are typical for MIP and may be very useful for refinery modeling.
Mixed integer data-export isn't supported by the current OMNI source code. Probably the easiest way to export the MIP-tables is to add two columns to the tables AVAIL, TIERAV and/or SPAVL: 'MIP' and 'SIZE'. An entry at a row/column intersection at column 'MIP' will define the crude-availability defined by this row as mixed integer. The entries at the minimum, maximum and fixed availability-columns will then be interpreted as cargo-numbers, with the entered unit of measurement defining the cargo-size's unit of measurement. The entry at this row for column 'SIZE' will be used as cargo-size (zero if blank).

Example

```
TABLE 1TIERAV1
      >       AAVLTMX AAVLBNX    ACSTX MIP SIZE
       AH00       2         0     97.6   1   100
       AL00       2         0     96.4   1    80
       NL00                100     10.5
```

Constraints like 'purchase of AL00 must be equal to the purchase of AH00' can be entered at the commercial constraints in the same way as is currently done for the LP model.

OMNI

If MIP is implemented in the current modeling system OMNI, many changes are required here. First of all, the availability tables must be able to handle new column-headers 'MIP' and 'SIZE'. Second, the OMNI source code must be extended in order to generate a matrix-file which deals with integer variables, if required. This extension will be large and very time-consuming.

4.3.3.4 Output (Business Objects)

There's no need to adjust any of the existing report-possibilities.

4.4 Enumeration *vs* MIP

This section produces a list of advantages and disadvantages of both the enumeration approach as described by section 4.2 and the Mixed Integer model of section 4.3.

An advantage which holds for both approaches is the fact that the setup described here, allows the user to change cargo data within the same case and version number as the original model. Therefore, cargo evaluation, linear programming and mixed integer programming can all be run within the same case and version without changing any data for each model.

4.4.1 Enumeration

4.4.1.1 Advantages

- Models remain LP and are therefore fairly easy to solve.
- Approach exactly represents all actual planner's activities.
- Total required time can be estimated well.
- Hardly any OMNI adjustments are required, which means little regret investment.

4.4.1.2 Disadvantages

- Many models may have to be run, which requires much running time.

  An advantage of this fact, however, is the fact that many results are presented to the planner. Therefore, he or she has full flexibility in selecting a cargo diet which can actually be processed by the refinery. Furthermore, the planner can provide the crude-traders (who actually have to buy the cargoes) with several possibilities.
- New reports must be generated.

  The new reports won't be implemented in OMNI, but only in Business Objects, which will probably become the standard PlanStar output source. Therefore, still no regret investments are made.
- Valuable cargoes may be missed because the planner beforehand doesn't consider them to be of any relevance. Therefore, they won't be taken into account during the cargo evaluation.

  To overcome this problem, the planner could simply run the cargo evaluation for *all* available cargoes.

4.4.2 Mixed Integer Programming

4.4.2.1 Advantages

- MIP does in fact contain the complete cargo evaluation process of the enumeration approach: introduce integer variables only for the evaluated crude oil and its dependent crude oils. By binding the integer variables to different numbers of cargoes to be purchased, the enumeration approach is exactly represented.
- MIP offers a whole new scope of possible constraints which can be used for refinery-modeling.
- The planner may gain much insight from the MIP solution. Obviously, if the optimal solution is feasible for scheduling as well, it is indeed the package which the traders should purchase.
- Only little implementation time required for extensions other than OMNI, therefore easy to use for any other modeling tool.
- The solution of the MIP supplies the planner with a combination of cargoes which is most attractive. This gives the planner much insight in which LP's are worth

running after the MIP, instead of running any LP he can think of himself. Therefore, valuable cargo-combinations which the planner might have overseen may come to light when using mixed integer programming.
- If the number of integer variables (possible cargo-purchases) is relatively small, the calculation time should be within reasonable bounds. Because of new insights as a result from the MIP-solution, probably not too many LP-models have to be run afterwards, which should keep the total time required below the required time for the enumeration approach.
- Stocks are much more actively present in MIP than in the enumeration. If stocks are small, MIP takes into account that a purchased crude diet has to be processed (almost) in full. Large closing stocks for a specific crude can represent the fact that this particular crude can be processed (partly) at after the end of the current planning period.
- Very little changes in the report-section are required. Some existing PlanStar (Business Objects) reports may have to be extended to express the required crude-purchase in number of cargoes.
- After solving the first Mixed Integer model, the user can interactively find good diets of cargo-purchases by fixating the purchase of a number of cargoes to a number other than the value in the optimal solution.

4.4.2.2 Disadvantages

- MIP tends to come up with a cargo diet which is optimal *if it could be processed within the planning period.* However, the solution is indeed a *planning* solution. The next step (after purchasing the crudes) would be scheduling. Here, several time aspects which aren't modeled for planning may prevent the given optimal solution from being feasible in practice. For example, the cargo diet selected by the MIP model may produce enough LPG for the planning period as a whole, but the planner knows that in the second week, for a period of three days, an enormous amount of LPG is required. The crudes which should be purchased according to the MIP solution may not be processed in such a way that this demand is met, which is unacceptable. Note, however, that this situation also occurs when running the 'normal' LP-model.

  Therefore, the MIP solution isn't the only solution the planner should look at. He or she should combine this solution with the solution of various LP-runs and possibly some other Mixed Integer models (with some other purchase restrictions to find another solution).
- In practice, the planner hands over a list of most desirable crudes to a trader, who then tries to buy these. If this trader is only handed the crude diet which resolved from the Mixed Integer model, he or she has no clear view in which cargoes to purchase in case the package as a whole cannot be purchased. Therefore, the trader might want something like a 'second best' package, which is rarely presented by MIP-solvers themselves.

  This problem can be solved in the same way as was suggested for the previous one. Note that currently planners also run various LP-models, so not much extra effort is required to gain more insight.

- Very large adjustments to the modeling system (OMNI) are required to allow for MIP. Different adjustments are required for all possible solvers. Since any OMNI change is seen as a regret investment, it may be worthwhile to implement MIP only in the (possible) new modeling system. If OMNI remains the Gemms modeling system, it may be considered to implement MIP for just one solver, for example Gsolve/OSL.
- The required amount of time depends on too many things (such as the position of the optimal solution of the LP-relaxation with respect to the optimal mixed integer solution) to be able to give any practically relevant estimate.

  Of course, since for a specific refinery the models usually won't change too much in size, a good estimate of the amount of time required becomes evident after running MIP for the first time.

4.5 Conclusions

In order to assist the planner as much as possible in his or her functioning, the cargo evaluation both by enumeration and by Mixed Integer programming should become available. Then, the user can choose to run the standard LP, the enumeration, or the MIP model. If the MIP has a satisfactory solution time, it can be used quite effectively for strategic planning and further research. Both approaches are a great extension of the current PlanStar functionality.

If OMNI is really due to be replaced, then the required investment in Mixed Integer programming might not be worthwhile yet. However, it could also be seen as an investment in research for future developments. If the MIP appears to be effective (for example if only several cargoes are under consideration), it's worthwhile to implement it in the new system. Based on the technical specifications as well as on the implementation in OMNI (especially the most efficient way to specify the constraints), the implementation-time should be greatly reduced.

The enumeration approach seems ready to be implemented. It represents current planners' activities in great detail, but makes them much easier to do. Therefore, it will make PlanStar more applicable for any crude oil planner.

More research on Mixed Integer modeling

Before investigating the possibility to implement all nice features of mixed integer modeling into PlanStar, more research is required to see in which part of the model users might like, for example, 'if-then' constraints and to see how much more effort would be required for implementation. Constraints which could easily be implemented once MIP is available are things like 'the number of components blended to this product is less than 5', and 'the number of units active for more than 30 days is greater than 4'.

Cargo evaluation modeling note

Some refinery-planners tend to use minimum availabilities to represent opening stocks. This cannot be used when activating cargo evaluation. If this kind of modeling is, the planner should (only once) update the model by removing the minimum availabilities and replacing them by opening stock, and costs entered for these availabilities must be replaced by costs of opening stock. The model then deals with them in exactly the same way as when modeling them as minimum availabilities. Considering the required removal of (some) existing availabilities for cargo evaluation (see section 4.2.2.2), this change of the model is essential.

Chapter 5

Conclusions

This chapter gives a summary of the most important results, conclusions and recommendations found during the research for this report. Details are found in sections 3.6.5, 3.9, 4.4 and 4.5.

5.1 PSLP implementation

A vital error in the implementation of current Gemms releases, is the fact that deficit-variables for recursive FOST- and CFPP-constraints aren't generated. All other aspects of PSLP are implemented well, but several improvements are recommended (details in section 3.6.5). The most important improvements are:

- increase OMNI's minimum difference between lower and upper bounds of recursion-variables from $2 \cdot 10^{-6}$ to $2 \cdot 10^{-3}$, to avoid unnecessary infeasibility,
- define linearized CFPP- and FOST-constraints as inactive if the respective switches are turned off,
- enable GSolve to detect an initially infeasible model, to avoid running a (time-consuming) recursive loop which can never lead to a feasible model,
- enable GSolve to detect local infeasibility of a recursive model, to avoid running useless iterations after convergence of recursive coefficients.

All improvements can be implemented quite easily both in OMNI and in GSolve (FORTRAN).

5.2 Nonlinear solvers

For current Gemms models, there's no need to solve them with a nonlinear solver. Probably due to the mainly linear models and the linear objective function, such a solver requires about 10 times as much calculation time than sequentially solving linear models, without resulting in a better solution.

On the other hand, the benefit of using nonlinear solvers is the fact that a whole new scope of refinery-modeling becomes available. Models can quite easily be extended to hold many other aspects of the refinery which are typically nonlinear. The problem is, however, that it's likely that actual refinery-planning models are too large to actually use all available nonlinearities and to be able still to solve the model within reasonable time.

5.3 Cargo analysis

For cargo-analysis, Mixed Integer programming doesn't seem to be the desired tool. The planner will gain much more insight in the relative values of crude cargo purchases if a sequence model is run with fixed crude purchases. This sequence is represented by the enumeration approach described in chapter 4.

The enumeration approach can be implemented in the PlanStar system, which should run a sequence of Gemms model. To enable the user to run this approach with PlanStar, the database used for Gemms models must be extended. Furthermore, some data-entry windows and menu-items must be added to the PlanStar system, and adjustments to C++ source code are required.

5.4 Future research

Chapters 3 and 4 lead to a recommended continuation of research. Both for Mixed Integer and for Nonlinear programming, benefits are yet to be discovered for refinery planning. If a modeling-system like AIMMS or PIMS is used for such research, both types can be investigated at the same time, because both can quite easily be modeled, based on the test-model built for this report.

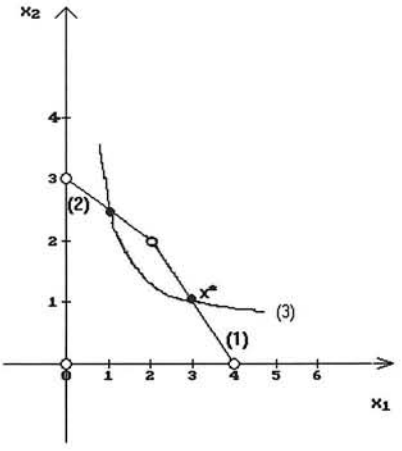
Probably the biggest advantage of Mixed Integer and/or Nonlinear modeling, and the best reason for future research, is the whole new area of modeling-possibilities which is opened, such as 'if-then' constraints with MIP and nonlinear unit-yields with NLP.

Appendix A *Example Gemms' PSLP algorithm*

This example demonstrates the principle of the PSLP algorithm, as implemented both in Gemms/OMNI and in GSolve. The example follows the phases of this algorithm: initialization and finding a feasible solution, check convergence and possibly revise the optimization problem.

To demonstrate the dependence on the initial solution, this example is constructed in such a way that the first PSLP-maximization will have a whole area of optimal solutions. Then, the resulting initial solution depends on the LP-solver used.

The problem under consideration is described below. Note that the nonlinear constraint is a quadratic equality, leading to about the same complexity as a pooling-constraint.



problem:

$$\max f(x) = x_1 + x_2$$

s.t.

$$(1) \quad x_1+x_2 \le 4$$

$$(2) \quad 0.5x_1+x_2 \le 3$$

$$(3) \quad g(x) = x_1x_2 - 3 = 0$$

global optimal solution:

$$x^* = (3\ ,\ 1)$$

$$f(x^*) = 4$$

$$g(x^*) = 0$$

Linearization of *g(x)*

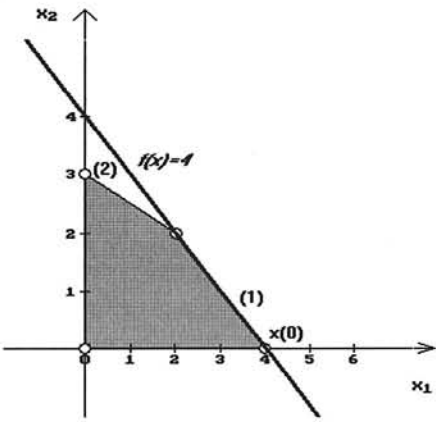
The recursive equation used to approximate $g(x)$ is obtained from the first-order Taylor series:

$$g(x^{(n)}) \approx g(x^{(n-1)}) + \sum_i (x_i^{(n)} - x_i^{(n-1)}) \left.\frac{\partial g(x)}{\partial x_i}\right|_{x^{(n-1)}} = x_2^{(n-1)}x_1^{(n)} + x_1^{(n-1)}x_2^{(n)} - x_1^{(n-1)}x_2^{(n-1)} - 3$$

In this example, we'll apply the PSLP-algorithm with convergence-parameter ε=0.01 and consider the solution converged and locally optimal after n (=1,2,3,...) iterations, if the absolute error of the linearized constraint after n iterations is less than ε. For ease of demonstration, we'll leave out Gemms' bound-adjustments.

Example A.1 convergence to the global optimum

Phase 1: finding an initial solution



problem:

max $f(x) = x_1 + x_2$

s.t.

(1) $x_1+x_2 \leq 4$

(2) $0.5x_1+x_2 \leq 3$

initial solution:

$x^{(0)} = (4, 0)$

[a whole range of optimal solutions is available]

$f(x^{(0)}) = 4$

$g(x^{(0)}) = -3$

Phase 2: convergence checks

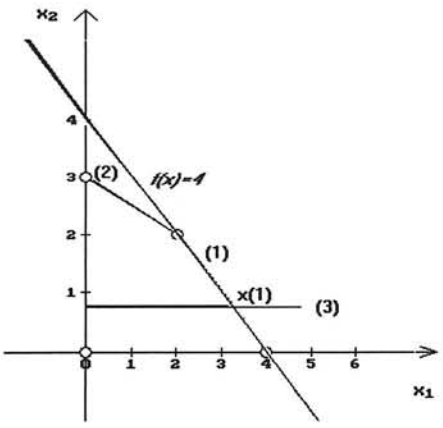
$|g(x^{(0)})| > \varepsilon$, so phase 3 is entered.

Phase 3: matrix revision

Now, $g(x)$ is approximated in $x^{(0)}$ and feasibility-variables S and D enter both the equality and the objective function. A penalty coefficient equal to 1000 is assumed.

This phase is repeated until convergence has been achieved or the maximum number of iterations reached.

iteration 1



problem:

max $f(x) = x_1 + x_2 - 1000(S+D)$

s.t.

(1) $x_1+x_2 \leq 4$

(2) $0.5x_1+x_2 \leq 3$

(3) $4x_2 - 3 + S - D = 0$

'optimal' solution:

$x^{(1)} = (3\frac{1}{4}, \frac{3}{4})$

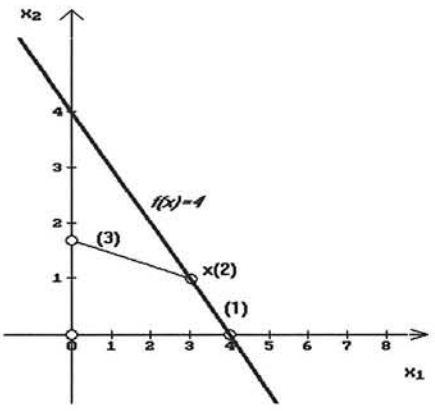
$f(x^{(1)}) = 4$

$g(x^{(1)}) = -\frac{9}{16}$

$S = D = 0$

Again, $|g(x)| > \varepsilon$, so the model is revised and solved once more.

iteration 2



problem:

max $f(x) = x_1 + x_2 - 1000(S+D)$

s.t.

(1) $x_1+x_2 \leq 4$

(2) $0.5x_1+x_2 \leq 3$

(3) $\frac{3}{4}x_1 + \frac{13}{4}x_2 - 5\frac{7}{16} + S - D = 0$

'optimal' solution:

$x^{(2)} = (3\frac{1}{40}, \frac{39}{40})$

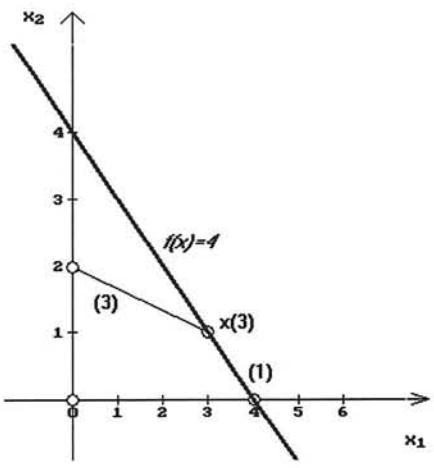
$f(x^{(2)}) = 4$

$g(x^{(2)}) \approx -0.051$

$S = D = 0$

Another iteration is required.

iteration 3



problem:

$$\max f(x) = x_1 + x_2 - 1000(S+D)$$

s.t.

(1) $x_1+x_2 \leq 4$

(2) $0.5x_1+x_2 \leq 3$

(3) $\frac{39}{40}x_1+\frac{121}{40}x_2 - 5.9494 + S - D = 0$

'optimal' solution:

$x^{(3)} \approx (3.000 , 1.000)$

$f(x^{(3)}) = 4$

$g(x^{(3)}) \approx 10^{-5}$

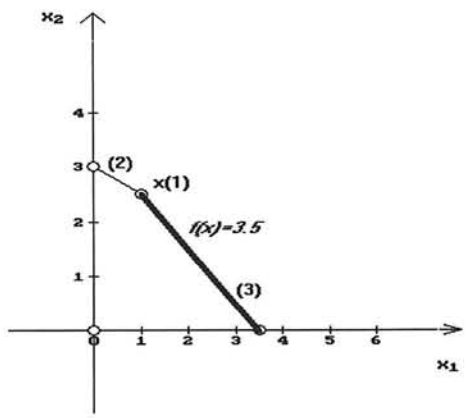
$S = D = 0$

The solution has converged and the local optimum of $x = (3,1)$ with $f(x)=4$ has been found. Because of the simplicity of the example, we know that this is in fact a global optimum. With nonlinear problems, however, this can never be guaranteed beforehand. The following part shows how the algorithm could easily find another local optimum.

Example A.2 Convergence to local optimum

Now, suppose that our LP-solver returned an initial solution $x^{(0)} = (2,2)$, with the same objective value as before: $f(x)=4$. The resulting situation differs completely from the one described in part I:

iteration 1



problem:

$$\max f(x) = x_1 + x_2 - 1000(S+D)$$

s.t.

(1) $x_1+x_2 \leq 4$

(2) $0.5x_1+x_2 \leq 3$

(3) $2x_1 + 2x_2 - 7 + S - D = 0$

'optimal' solution:

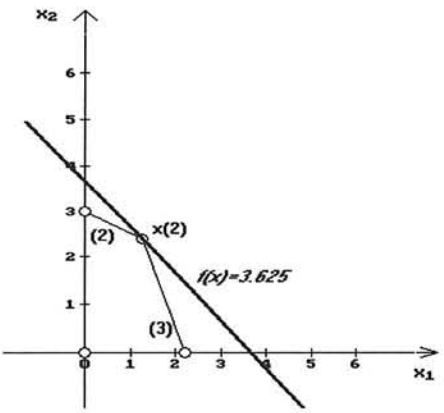
$x^{(1)} = (1 , 2.5)$

$f(x^{(1)}) = 3.5$

$g(x^{(1)}) = 1$

$S = D = 0$

iteration 2



problem:

$$\max f(x) = x_1 + x_2 - 1000(S+D)$$

s.t.

(1) $x_1+x_2 \leq 4$

(2) $0.5x_1+x_2 \leq 3$

(3) $2.5x_1 + x_2 - 5.5 + S - D = 0$

'optimal' solution:

$x^{(2)} = (1.25 , 2.375)$

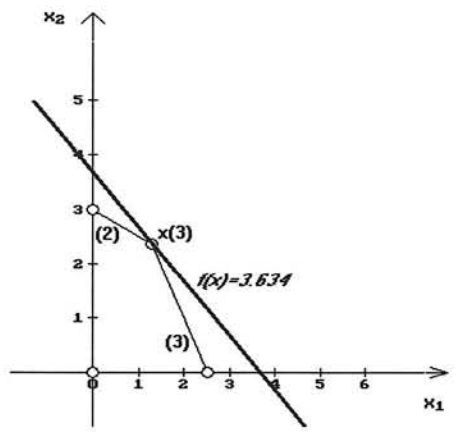
$f(x^{(2)}) = 3.625$

$g(x^{(2)}) = -0.03125$

$S = D = 0$

Again, $|g(x)| > \varepsilon$, so we can forget about other convergence checks and continue.

iteration 3



problem:

$$\max f(x) = x_1 + x_2 - 1000(S+D)$$

s.t.

(1) $x_1+x_2 \leq 4$

(2) $0.5x_1+x_2 \leq 3$

(3) $\frac{19}{8}x_1+\frac{5}{4}x_2 - 5\frac{31}{32} + S - D=0$

'optimal' solution:

$x^{(3)} \approx (1.268 , 2.366)$

$f(x^{(3)}) \approx 3.634$

$g(x^{(3)}) \approx -0.0002$

$S = D = 0$

Now, $|g(x)| < \varepsilon$, so the locally optimal solution $x = (1.268 , 2.366)$ is returned, which fulfills the pooling-like nonlinear equation. This point is an approximation of the intersection between the graphs of (3) and (2) in the first figure of this example:

$$x = \left(\frac{6}{3+\sqrt{3}}, \frac{3+\sqrt{3}}{2}\right) \approx (1.268 , 2.366)$$

$f(x^*) \approx 3.634$

$g(x^*) = 0$.

The recursion is now ended and the best solution isn't found. This is, however, a known problem with nonlinear optimization: a global optimum can hardly ever be assured. In the case of FOST- and CFPP-constraints, however, a global optimum may very well be found: if the initial solution satisfies the nonlinear inequalities, it doesn't matter that the linear approximation isn't good yet, a better solution is simply not

available and the initial optimum is a global optimum. For example, if the nonlinear equation in the example would be an inequality (like a fuel oil stability constraint), so $x_1x_2\text{-}3 \leq 0$, and the LP-solver found an initial solution $x = (4,0)$, then there's no need to approximate the nonlinear function. The initial point satisfies the original nonlinear condition, so x is a local optimum. In fact, in this situation, we know that there's no other solution which results in a higher objective value, because the initial solution is the optimal solution of the original, linear model.

Appendix B *Entering pooling-data in Gemms*

Introduction:

Fairly often components will be put together ('pooled') for practical or technological reasons before they are used in blending, as unit feed, or in transfer to other refineries. The limited availability of tanks is one of the main practical reasons.

Data-entries for the:

To define a pool, adjust the data-tables in the following way (table-names refer to the OMNI-tables, adapt for PlanStar tables if required):

- define the pool as a component (table COMPS)
- define the pool as a product (table PRODS)
- enter the pool-component (CRCP) name in table PRODS, at the intersection of the product row-name and column POOL, between apostrophes.

Then, to allow components to blend to the pool, fill table BPOT as for 'normal' blending products: the product-name of the pool as column-header, the components in the rows, and a '1' entry at the specific intersections. To allow the pool to blend to a product, use its CRCP name as the component which can blend to the product.

The following entries give an example for pools 'PLLS' and 'PLMC'.

```
Table (R)COMPS
                        >      PL      BS      EK      JK      MH      AR
LS Crude tank           LS     1
Mogas comp tank         MC     1
Crude                   00             1       1
Platformate             PF             1       1       1       1       1

Table PRODS
                        >       POOL
LS Crude tank           POOL1  'PLLS'
Mogas comp tank         POOL2  'PLMC'
Premium                 PREM

Table (R)BPOT
                        >       PREM    POOL1   POOL2
Mogas comp tank         PLMC     1
Brent                   BS00             1
Ekofisk                 EK00                     1
Platformates            **PF                     1
```

***Other destinations of the Pool*:**

The above example showed one destination of a pool: blending to a product (here premium gasoline). Note that a pool may *not* blend into another pool. Other possible destinations for a pool are:

1. unit feedstock,
2. transfer to another refinery,
3. intermediate transfer (e.g. for renaming conventions),
4. inter-period or closing stock.

1. When the Pool acts as **unit-feed** (one crude tank may be present in which two crudes are pooled), the pool's source-components must be indicated as the feeds in the unit (UN)-tables. In these tables, also a row-name equal to POOL must be present, with the pool's CRCP-name between apostrophes in all columns with the feedstock one of the pool's source components. Otherwise, the pool is unable to get rid of all its constituent components (or the ones not present must have zero weight in the pool). For example, consider the following table added to the previous ones.

```
Table (R)CD
                        >                BS00LC  EK00LC
Crude                    **00            -100    -100
Gasoil                   **GO              20      35
Naphta                   **NA              50      35
Long Residu              **LR              30      30
Pooled Feed              POOL           'PLLS'  'PLLS'
```

The procedure will ensure that a pool's possible source components will be used in the various unit-modes of operation according to their proportion of the pool. For example, when the LP determines that the optimal composition of the previously defined pool equals 60% Brent and 40% Ekofisk, then the Brent intake of the CD unit at its low-cut mode will be 60/40 times the intake at this mode of Ekofisk.

2. When the Pool can be transferred to **another refinery**, the component-name of the pool must be indicated in table TRANS, FERS or TRNS(TRM)

3. The pool can also be **transferred/renamed** into other materials via tables INTER and FDPOOL.

4. Finally, the pool can also be 'transferred' to another **period** by defining it with the component-name in the IPSTK-table.

Possible pool source-components are also allowed to go straight-run to a destination, as long as it's not one of the pool's possible destinations. (Because then, the whole nonlinear aspects (proportions in the source and proportions in the destination) of the pool will be lost.)

***(De)activate recursive Pooling*:**
In Table Decision, the user can select to deactivate recursive pooling by switching on the REPO-parameter. If not activated, segregation of the pool-components is allowed; in that case the user-given properties of the separate components will be used in blending.

Users can also decide only to disregard several of all defined pools. In that case, an entry in table PRODS must be given at the intersection of the specific pool-rows and the UNPOOL header. These pools are deactivated and all their source components can be routed to the original pool's destinations straight-run.

Reporting:

Suppose a pool will be used as a component in the blending of grade A. Then, in the reports of GEMMS the composition of grade A will be reported in terms of the constituent components of the pool, independent of the REPO-variable. The composition of all pools can be asked for by running report 4003. (Gemms versions lower than 5.2 always generate this pool-composition report.)

Appendix C *implementation pooling in AIMMS*

The following code is the most relevant part of the code added to the Godorf-like model (after some further adjustments) to cope with pooling. Comments between '/*' and '*/' may refer to Gemms-row names as described by table 3.1. Equation-names and conditions for them to hold are removed. Variables starting with 'x_' are the most relevant decision variables (e.g. `x_mblend` is the total weight of a component blending to a product (here: via a pool).

Index-definition:

`pool`	:=	any of the three available pools
`c`	:=	component-type
`cm`	:=	component-label
`fp`	:=	final product
`t`	:=	time-period

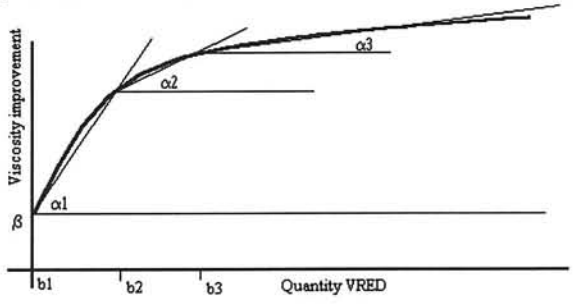
```
/* Gemms B(R)(P)(PL):  */
 total_pool_mass(pool,t)
       = sum [(fp), x_pool_fp(pool,fp,t)]  + x_pool_stock(pool,t)],

/* Gemms S(R)(P)(PL)V: */
 total_pool_vol(pool,t)
       = sum [(c,cm), x_comp_pool(c,cm,pool,t) /comp_prop(c,cm,"DENS")],

/* Gemms V(R)(P)(PL)(01)., destinations are final product (fp) and stock */
 x_comp_pool(c,cm,pool,t)
               = x_comp_blend_pool(c,cm,pool,t) + x_comp_stock_pool(c,cm,pool,t),

/* Gemms V(R)(P)(PL).(01)  */
 x_pool_fp(pool,fp,t)
       = sum[(c,cm)$(comp_blend_pool(c,cm,pool), x_mblend(c,cm,fp,t)],
 x_pool_stock(pool,t)
       = sum[(c,cm)$( comp_blend_pool(c,cm,pool)), comp_stock(c,cm,t)],

/* Nonlinear pooling */
/*  Equalities are replaced by '≤' and '≥' with constant 'accuracy' added or
subtracted, to circumvent numerical problems. */
 x_mblend(c,cm,fp,t)*total_pool_mass(pool,t)
               <= x_comp_pool(c,cm,pool,t)*x_pool_fp(pool,fp,t) + pooling_accuracy,
 x_mblend(c,cm,fp,t)*total_pool_mass(pool,t)
               >= x_comp_pool(c,cm,pool,t)*x_pool_fp(pool,fp,t)- pooling_accuracy,
 comp_stock(c,cm,t)*total_pool_mass(pool,t)
               <= x_comp_pool(c,cm,pool,t)*x_pool_stock(pool,t) + pooling_accuracy,
 comp_stock(c,cm,t)*total_pool_mass(pool,t)
               >= x_comp_pool(c,cm,pool,t)*x_pool_stock(pool,t) - pooling_accuracy,
```

Appendix D *Example step-wise linearization for additives*

Here, an example is demonstrated of modeling a nonlinear additive-response in Gemms.

Suppose that at some refinery, an additive called 'VRED' can be used to reduce the viscosity of standard liquid refinery fuel, called LREFU. The nonlinear shape of the viscosity-reduction is given by figure D.1. The user can present a data-table to Gemms which approximates the viscosity-reduction with (here: three) separate linear parts, as shown by the figure. In this table, values for β, the slopes α_1, α_2, α_3 as well as the starting-points of the linear parts (b_1, b_2 and b_3), must be present. Possibly some costs and a maximum of additive-usage may also be presented.

Figure D.1 LREFU viscosity reducer



The initial improvement in viscosity-reduction (with the first drop of VRED) is given by β (usually zero). All alphas represent the additive's response. Suppose the table entered by the user looks as follows (with alphas under header LREFU and with β and b's under QUANT).

```
TABLE VRED
        >           QUANT   LREFU
         BETA       0.50
         ALPHA1     0.0     .50
         ALPHA2     0.2     .42
         ALPHA3     0.5     .35
```

Then, Gemms generates constraints for product PR (here LREFU) which reads "total blended product viscosity - reduction ≤ maximum specification}, or

$$\Sigma_c \, visc_c \cdot \{Y_{c,PR}\} - maxvisc_{PR} \cdot \{X_{PR}\} - \{reduction\} \leq 0$$

with c a component which can blend to product PR, $Y_{c,PR}$ the total weight of blended component c to PR, X_{PR} the total blended weight of the product, $visc_c$ the viscosity property-value of c, $maxvisc_{PR}$ the specified maximum viscosity-value, and with {reduction} the total quantity of reduced viscosity-value. The viscosity-reduction is approximated with the step-wise linear function, using slopes and starting-points.

Bibliography

[1] J. Abadie, 'Application of the GRG algorithm to optimal control problems', in J.Abadie (ed.), Integer and nonlinear programming, North-Holland publishing company, Amsterdam 1970

[2] J. Abadie and J. Carpentier, 'Generalization of the Wolfe Reduced Gradient Method to the case of nonlinear constraints', in R.Fletcher (ed), Optimization, Academic Press, New York 1969

[3] M.S. Bazaraa et al., 'Nonlinear programming, theory and algorithms', John Wiley & Sons, Inc., New York 1993 (first edition 1979)

[4] J. Bisschop, 'AIMMS, the modeling system', Paragon decision technology bv, Haarlem 1993

[5] W. Brussaard, 'Pooling in GEMMS', ORTEC Consultants bv, Gouda 1998

[6] W.H. Greene, 'Econometric analysis', Macmillan publishing company, New York 1993 (2nd edition)

[7] F.S. Hillier and J. Lieberman, 'Introduction to operations research', McGraw-Hill book company, Singapore, 1995 (6th edition)

[8] M.R.B. Kreuk, 'GEMMS Linear Programming Model - a detailed description of the data-conversion', ORTEC Consultants bv, Gouda 1998

[9] G. Pierce, 'PSLP formulation in Gemms/OSL', draft version of note, Shell Oil Company, 1995.

[10] M.J.D. Powell, 'Variable Metric Methods for Constrained Optimization', in A. Bachem, M. Grotschel, and B. Korte (eds), Mathematical programming: the state of the art, Springer-Verlag, Berlin, 1983, pp.288-311. Reference taken from Hillier [5].

[11] G.J. van Rooijen, 'Robust optimization in crude supply planning', thesis, Free University of Amsterdam, 1996

[12] C. Schweigman, 'Constrained minimization: handling of linear and nonlinear constraints', dissertation Delft University of Technology, Stichting Studentenpers, Nijmegen 1974

[13] I.M. van der Tuijn, 'Efficiently modeling and solving the crude supply planning problem', thesis, Delft University of Technology, 1998.

Summary

Many techniques developed in the field of operations research are used to solve logistic problems at an oil-refinery. One of the most widely used techniques is linear programming, which has proven to be very useful within the oil business because of it's easy understanding and little solving time. This report deals with the application of sequences of linear programming models to solve nonlinear or mixed integer problems.

The Gemms refinery-planning system is based purely on linear models. Therefore, if nonlinear relationships must be modeled and solved, there are only three ways to deal with them: disregard the nonlinearity and assume a linear relationship, apply the nonlinear relationship to calculate coefficients of a linear constraint, or approximate the nonlinearity by a sequence of linear models.

Of course, the first mentioned method usually doesn't give a very good representation of reality, but it may suffice if the actual relationships aren't too relevant for the decision process. For example, in Gemms, unit-yields are assumed to increase or decrease linearly with the unit's input. In the real world, however, twice as much input doesn't necessarily double all yields, because of reduced reaction-times per molecule.

The second way, calculation of coefficients with a nonlinear formula, can be used by Gemms in blending. This is required when a property-value of a resulting product isn't a linear combination of the property-values of the blending components. If the formulae give a sufficiently close approximation of resulting product-properties, they should be used instead of recursive techniques, because their little calculation time.

The third method, a (recursive) sequence of linear models, lies at the basis of chapter 3. In this chapter, both the theory and the implementation within Gemms of this method are discussed in all detail. Furthermore, the performance of the recursion with respect to that of a NLP-solver is tested by creating a Gemms model in the AIMMS modeling language.

Chapter 4 describes a possible method for cargo-analysis, which can be used to determine the relative value of a specific crude-type for a specific refinery. Because this crude is purchased in batches (e.g. a full oil-tanker), the variable which represents this purchase shouldn't be continuous (which is the case in the linear model), but should be integer. Another possibility is to run a sequence of linear models in which the purchase is fixed to a cargo-size. The whole sequence of outputs will give the user more insight in cargo-values than the result from a mixed integer model. Therefore, the possibility to implement this sequence ('the enumeration approach') in PlanStar (used for refinery-planning in combination with Gemms) is discussed in great detail, including functional and technical specifications.

The last chapter of this report gives a summary of the most important results, conclusions and recommendations.

Samenvatting

Om de logistieke problemen van een olieraffinaderij op te lossen, worden vele technieken uit de operations research toegepast. Voornamelijk lineaire programmerings modellen zijn zeer populair, met name vanwege het feit dat ze vrij makkelijk te gebruiken en over het algemeen snel oplosbaar zijn. Dit rapport behandelt de toepassing van reeksen van lineaire modellen om niet-lineaire of Mixed Integer problemen op te lossen.

Het Gemms raffinaderij-planning systeem is puur gebaseerd op lineaire modellen. Indien niet-lineaire relaties toch gemodelleerd en opgelost moeten worden binnen dit systeem, zijn er drie mogelijke oplossingsmethoden: veronderstel simpelweg een lineair in plaats van een niet-lineair verband, bereken coëfficiënten in een lineaire nevenvoorwaarde met behulp van de niet-lineaire relatie, of benader de niet-lineariteiten met een reeks lineaire modellen.

De eerstgenoemde oplossing zal in het algemeen geen goede benadering van de realiteit zijn, maar indien de niet-lineaire relaties niet echt relevant zijn voor de beslissingsprocessen op de raffinaderij kan het goed genoeg zijn. In Gemms modellen, bijvoorbeeld, wordt aangenomen dat unit-yields (of 'opbrengsten') lineair met de doorzet toe- of afnemen. Echter, in werkelijkheid betekent een twee maal zo grote doorzet niet altijd exact twee maal zoveel van dezelfde opbrengsten, doordat de reactietijd per molecuul verminderd is.

De tweede methode, berekening van coëfficiënten via de niet-lineaire formule kan in Gemms worden toegepast bij blending indien resulterende kwaliteiten van een product geen lineaire relatie tonen met de kwaliteiten van de blendende componenten. Indien de ontstane lineaire vergelijking een voldoende benadering geeft van de werkelijkheid, dan heeft deze methode de voorkeur boven de derde methode vanwege de grote tijdwinst.

De laatste methode, een (recursieve) rij van lineaire modellen ligt aan de basis van hoofdstuk 3. Hierin worden zowel de theorie als de Gemms-implementatie gedetailleerd besproken. De geleverde prestaties ten opzichte van die van een speciale solver voor niet-lineaire modellen worden eveneens onderzocht.

Hoofdstuk 4 beschrijft een mogelijke methode voor zogenaamde cargo-analyse, dat gebruikt kan worden om de relatieve waarde van een bepaald type ruwe olie voor een raffinaderij te bepalen. Omdat deze olie altijd in grote ladingen tegelijk wordt ingekocht (bv. een hele olietanker), dient een variabele die de hoeveelheid in te kopen ruwe olie aangeeft niet continu te zijn. Dit is wel het geval in de modellen van Gemms. Een betere modellering zou het gebruik van integer variabelen zijn, die bijvoorbeeld het aantal aangevraagde tankers met de ruwe olie aangeven. Een andere mogelijkheid is het oplossen van een reeks lineaire modellen waarbij in elk model de ingekochte olie vastgezet wordt op de inhoud van een aantal tankers. Nadat de hele reeks is opgelost, krijgt de gebruiker een samenvatting van alle relevante output,

waardoor veel meer informatie wordt verkregen dan van één Mixed Integer oplossing. Daarom is er in dit rapport uitgebreid onderzoek gedaan (inclusief functionele en technische specificaties) naar de mogelijkheid om cargo-analyse volgens deze laatste methode (de enumeratie-methode) te implementeren in het PlanStar software systeem. Met dit systeem dient dan een reeks van Gemms modellen te worden opgelost en de resultaten moeten kunnen worden gerapporteerd en opgeslagen in een database.

Het laatste hoofdstuk van dit rapport vat de belangrijkste resultaten, conclusies en aanbevelingen uit eerdere hoofdstukken samen.

The following reports appeared in the WBBM Report Series:

1 W. Osman, A Design of an MSS for Integrated Program & Project Management

2 J.J. de Jonge, Maintenance Management and Modelling

3 B. Meima, Expert Opinion and Space Debris

4 K. Wouterse, Design of a Management Supporting System for Distributed Production Planning

5 P.J.M. Waasdorp, Forecasting and Inventory Replenishment in a Distribution Chain

6 R.A.L. Slaats, Planningsalgoritmes voor MRP

7 H.C. Vos, A Multi-Component Maintenance Model with Discounts

8 G. Dijkhuizen, Inventory Management with Integrated Regular and Express Ordering

9 R. van Dorp, Dependence Modelling for Uncertainty Analysis

10 D. van Schooneveld, Fractal Coding of Monchrome Digital Images

11 D. Roeleven, Modelling the Probability of Accident for Inland Waterway Transport

12 W. van der Sluis, The Coordination of Production and Distribution in a Decentralized Distribution Chain: A Contractual Approach

13 A.G. Chessa, Object-based Modelling of Hydrocarbon Reservoirs

14 M. Kwak, Planning and Replenishment for a Seasonal Demand Pattern

15 P. van Kampen, Maintenance Management of Multi-Component Objects

16 C. de Blois, Dynamic Pollutant Transport Modeling for Policy Evaluation in the Rhine Basin

17 G. van Acken, Hoe goed is een klant?

18 J.B. Molenkamp, Matching of assets and liabilities for pension funds

19 A.J. Bomans, Validation of a Model for Optimising Decisions on Maintenance

20 M.A. Odijk, Performance Evaluation of Railway Junction Track Layout Designs

21 S.M. Geervliet, Modellering van de Faalkans van Ondergrondse Transportleidingen

22 R.R. Witberg, A Neural Network Solution to Wireline-log Recognition Applications

23 H. de Blank, Beslissingsondersteunend Systeem voor het Persproces van Mengvoeders

24 M.C. Rozema, RailEase, Ondersteuning bij het Specificeren van Railinfrastructuur in Knooppunten

25 J. Dorrepaal, Analysis Tools for Reliability Databases

26 G.M. te Brake, Automated Detection of Stellate Lesions and Architectural Distortions in Digital Mammograms

27 S.T. van Houwelingen, Petrophysical Conductivity Modelling – Determination of Effective Conductivity and Electro-Type Tool Response Modelling

28 R.P.M. Goverde, Civil Aircraft Autopilot Design Using Robust Control

29 W.W.J. Götz, Influence Diagrams and Decision Trees in Severe Accident Management

30 J.P.A. van der Vliet, Assets Liability Matching for Life Insurers

31 M.P.C. Alders, Spatio-spectral Analysis of Wireline Logs

32 J.W.P. Karelse, Risicomanagement bij een Woningcorporatie met Monte Carlo Simulaties

33 F.L. Härte, Efficiency Analysis of Packaging Lines

34 H.L. Liem, Gedragsanalyse van Luchtvaartmaatschappijen met Betrekking tot Geluidsbelasting op Schiphol

35 N.P. van Elst, Betrouwbaarheid van het Sluitproces van Beweegbare Waterkeringen

36 L. Lam, The Analysis of Doubly Censored Survival Data

37 F. Phillipson, Lokale Treinverkeersregeling

38 L.B. Gerlagh, Efficiënte Routering van Strooiwagens ten behoeve van de Gladheidsbetrijding op de Wegen

39 F.P.M. Schouten, Analytic Techniques for Business Modeling: Opportunities for Advance

The Departmens of Mathematics and Computer Science at Delft University of Technology offers a selected group of people a two-year post-Master's program featuring specialized instruction in practical mathematical modeling and consultancy skills (in Dutch: Wiskundige Beheers- en Beleidsmodellen). Participants follow courses in the first year of the program and apply their skills in a project in the second year. Projects are carried out under contract with a company or independent research institute. The WBBM Report Series contains the final reports of these projects.

TU Delft

Delft University of Technology
Wiskundige Beheers- en Beleidsmodellen

ISBN 90-407-1268-5