Topics in Computational Linear Optimization

Tim Helge Hultberg

LYNGBY 2001
IMM-PHD-2000-78

IMM

ISSN 0909-3192

Trykt af IMM, DTU
Bogbinder Hans Meyer

Preface

This dissertation has been prepared at the Department of Mathematical Modelling (IMM) during the period January 1997 to November 2000 as a partial fulfilment of the requirements for obtaining the Ph.D. degree at the Technical University of Denmark (DTU).

The Ph.D. project has been supervised by my main supervisor professor Jens Clausen, professor Per Christian Hansen and associate professor Hans Bruun Nielsen.

Acknowledgements

It has been a pleasure to work in the operations research group at IMM with my supervisor Jens Clausen. After his arrival at IMM the activity of the operations research group has boosted. The group is now involved in several interesting projects (Descartes, Explain, CIAM) with industrial coorporation, resulting in a lively and stimulating environment.

I would also like to express my gratitude to my former supervisor Søren Kruse Jacobsen, who gave me good inspiration when I wrote the Ph.D plan and my co-supervisors Per Christian Hansen and Hans Bruun Nielsen, who provided valuable feed-back on the paper on direct solution of sparse unsymmetric systems.

Besides the scientific work, some of the highlights of my time at DTU have been the weekly meetings at the OR cake group and the weekly football with the guys from TELE. My warm thoughts goes to the members of the OR cake group including ex-members Pedro Borges and Michael Pilegård.

Each January for the last three years, I have been assistant teacher for Søren Nielsen at the 3-weeks course in linear programming modelling, which I enjoyed a lot. Søren also inspired me to start working on FLOPC++.

I shall never forget my six months at the University of Colorado at Denver, where I visited Harvey Greenberg who received me in the best possible way. Harvey taught me everything about integer programming (have I forgotten something, it is certainly not his fault) and was contributing to make the stay of my wife and me in Denver enjoyable. I would also like to thank Allen Holder and Markus Emsermann for helping us so much with a lot of practical things, nice company and several games of flag football and squash.

I have been privileged to be able to spend many nice relaxing weekends with my parents at Granhøjgård and holidays with my parents-in-law in Portugal. Thank you!

During the period of my Ph.D studies my family status has changed dramatically. December 27, 1997 Paula Alexandra Machado Gomes Hultberg became my wife and on October 8, 1998 our son Nuno was born. I thank you both for giving me the happiness of being a husband and a father, I could never have imagined a more wonderful wife and son. YY is still in Paulas belly, I am looking forward to meet him/her next May.

Kongens Lyngby, November 2000

Tim H. Hultberg

Summary

This dissertation adresses several topics in computational linear optimization.

Linear optimization has been an active area of research ever since the pioneering work of G. Dantzig more than 50 years ago. This research has produced a long sequence of practical as well as theoretical improvements of the solution techniques avilable for solving linear optimization problems.

Linear optimization problems covers both linear programming problems, which are polynomially solvable, and mixed integer linear programming problems, which belong to the class of NP-hard problems.

The three main reasons for the practical succes of linear optimization are: wide applicability, availabilty of high quality solvers and the use of algebraic modelling systems to handle the communication between the modeller and the solver.

This dissertation features four topics in computational linear optimization: A) automatic reformulation of mixed 0/1 linear programs, B) direct solution of sparse unsymmetric systems of linear equations, C) reduction of linear programs and D) integration of algebraic modelling of linear optimization problems in C++. Each of these topics is treated in a separate paper included in this dissertation.

The efficiency of solving mixed 0-1 linear programs by linear programming based branch-and-bound algorithms depends heavily on the formulation of the problem. In seeking a better formulation, automatic reformulation employs a range of different techniques for obtaining an equivalent formulation of a given pure or mixed integer programming problem, such that the new formulation has a tighter LP-relaxation than the original formulation.

The first paper surveys the use of automatic reformulation techniques for general mixed 0-1 linear programs.

An implementation, Spiky, of a direct method for solving large sparse unsymmetric systems of linear equations is presented in the second paper. The method is based on a matrix modification approach applied to a spiked triangular permutation of the system. The factorization consists of 3 steps. 1) a reordering of the matrix, such that only a small number, s, of spike columns reach above the diagonal is determined. 2) a block LU factorization of an augmented system is computed. This involves the solution of a sparse triangular system with s right-hand sides. Solution sparsity is exploited in the sparse triangular solves of the block LU factorization. 3) a factorization of the Schur complement matrix, of order s, is computed. The idea of this factorization method comes from Gondzio [16], but has been improved in several ways. Most importantly: a) A new fast reordering algorithm is used. b) Sparsity of the Schur complement is exploited.

Simple techniques for reducing the size of linear programs prior to the application of a solution algorithm are incorporated as important parts of most LP solvers. The usual LP reduction techniques require a very limited effort but nevertheless often result in substantial reductions. One of the reasons for this good performance is the way LP models are typically formulated using algebraic modelling languages. The third paper presents a common framework for LP reduction algorithms and shows how the usual LP reduction techniques fit into this framework. It also demonstrate how a stronger bound strengthening and the use of primal information to improve the dual bounds can be used to obtain further reductions.

In the fourth and last paper, a prototype implementation of a C++ class library, FLOPC++, for formulating linear optimization problems is presented. Using FLOPC++, linear optimization models can be specified in a declarative style, similar to algebraic modelling languages such as GAMS and AMPL. While preserving the traditional strengths of algebraic modelling languages, FLOPC++ eases the integration of linear optimization models with other software components. The class library implements a full-fledged algebraic modelling language with indexed variables and constraints, repeated sums, index arithmetic and conditional exceptions.

Besides the articles the thesis features six introductory chapters, including a description of two real-world applications of linear optimization, in which I have been involved during my Ph.D. study.

Resumé (In Danish)

Denne afhandling omhandler forskellige emner indenfor lineær optimering.

Lineær optimering har været et aktivt forskningsområde lige siden G. Dantzig's banebrydende arbejde for mere end 50 år siden. Denne forskning har resulteret i et en lang række praktiske og teoretiske forbedringer af de tilgængelige løsningsmetoder for lineære optimeringsproblemer.

Lineære optimeringsproblemer omfatter både lineære programmeringproblemer, som kan løses i polynomiel tid, og blandede heltals lineære programmerings problemer, som tilhører klassen af NP-hårde problemer.

De tre væsentligste åsager til lineær optimerings store succes i praksis er: et bredt anvendelsesområde, eksistensen af "solvere" (programmer til løsning af lineære optimeringsproblemer) af høj kvalitet samt brugen af algebraiske modelleringssystemer til at varetage kommunikationen mellem modelløren og solveren.

De fire emner indenfor lineær optimering som er behandlet i denne afhandling er: A) automatisk reformulering af blandede 0/1 lineære programmer, B) direkte løsning af "sparse" usymmetriske lineære ligningssystemer, c) reduktion af lineære programmer samt D) integration af algebraisk modellering af lineære optimeringsproblemer i C++. Hvert af disse emner er behandlet i en separat artikel, inkluderet i denne afhandling.

Det er ofte muligt af modellere et givet problem på flere forskellige måder som et blandet 0/1 lineært program. Den valgte formulering kan have meget stor betydning for den anvendte kørselstid ved løsning af problemet. I den første artikel gennemgås en række teknikker til automatisk reformulering af generelle (dvs. uden antagelser om nogen særlig struktur) blandede 0/1 lineære programmer.

I den anden artikel gennemgås en implementation, Spiky, af en direkte metode til løsning af store sparse usymmetriske lineære ligningssystemer. Metoden bygger på matrixmodifikation anvendt på en 'pigget' nedre triangulær permutation af systemet. Ideen stammer fra Gondzio [16], men er forbedret på flere punkter. Først og fremmest: a) En ny hurtig algoritme til permutation af matricen er anvendt. b) Sparsitet af Schur komplementet er udnyttet.

Simple teknikker kan ofte anvendes til at reducere lineære programmer betydeligt. Den tredje artikel opstiller en fælles ramme for LP reduktions algoritmer og demonstrerer hvorledes en større indsats til indsnævning af variablenes grænser samt brugen af primal information til styrkelse of de duale grænser kan udnyttes til at opnå større reduktioner.

Den fjerde og sidste artikel omhandler FLOPC++, et C++ klassebibliotek til algebraisk modellering af lineære optimeringsproblemer. Fordelene og de mulige anvendelser af et sådant værktøj gennemgås og FLOPC++ sammenlignes med lignende værktøjer. Desuden diskuteres de grundlæggende ideer bag implementationen samt mulige forbedringer.

Foruden de fire artikler indeholder afhandlingen seks introducerende kapitler, hvor blandt andet to anvendelser af lineær optimering, som jeg har været involveret i under mine Ph.D. studier, beskrives.

Contents

Chapter 1

Introduction

A linear program (LP) is a constrained optimization problem where both the object function and the constraints are linear. If some of the variables are required to take integral values, the problem is called a mixed integer (linear) program (MIP). We use the term *linear optimization problem* as a common denotation for LPs and MIPs.

Numerous practical problems can be modelled as linear optimization problems. Applications are found in many different areas such as engineering, management, logistics, statistics, pattern recognition, etc. A large number of the applications can be characterized as problems of "allocating limited resources among competing activities in a best possible (i.e. optimal) way".

For decades the use of linear optimization to tackle real world decision problems has been taught to students of economics, business and engineering. Typically the use of linear optimization includes at least the following three activities:

1. Formulation of the model.
2. Solving the model.
3. Interpretation of the solution.

Computational linear optimization deals with the development of software support for these three activities and their interconnections.

Formulation of the model. One of the factors behind the success of linear optimization is the relative simplicity of creating linear optimization models. Sometimes the choice of decision variables and constraints follows easily from an understanding of the problem, resulting in an algebraic description of the model. In general however, building a mathematical (algebraic) linear optimization model of a real world problem is a creative and iterative process, which depends on the insight and experience of the modeller. Comprehensive coverage of the model building aspect of linear optimization is found in [30] and [26].

An algebraic description of the model together with the required data provide a mathematically well defined specification of a linear optimization problem, which is suitable for human reading, but does not fit the input format expected by a solver. It is the role of algebraic modelling systems to bridge this gap by providing a machine readable language for expressing algebraic model descriptions together with an associated translator from this language into a suitable input for a linear optimization solver.

Different formulations of a model can lead to significant differences in the execution time required to solve the model. Therefore it is often the role of the modeller to formulate a model which is not only correct but also adapts well to the chosen solution technology. For LPs the main concern is to keep the number of rows, columns and non-zero entries of the coefficient matrix reasonably low. Paper C, "Aggressive LP Reduction", deals with automatic reduction of the size of LPs. For MIPs the tightness of the LP-relaxation is more important than the sheer size. In both cases, however, it is usually possible to improve a bad formulation automatically by submitting the problem to a presolve (reformulation) phase. Paper A, "Automatic Reformulation for Mixed 0/1 Programs", included in this thesis, discusses the use of automatic reformulation for MIPs (though the discussion is limited to problems where all the discrete variables are binary). It is shown how both changes to the coefficients of individual constraints as well as the addition of further constraints can tighten the LP relaxation of the formulation. Of course it is hard to draw a clear cut borderline between automatic problem reformulation and problem solving, but one of the consequences (and indeed one of the goals) of using techniques as described in Paper A and C is that the execution time for solving linear optimization problems becomes less vulnerable to bad (from the point of view of the solution algorithm) formulations.

Commercial algebraic modelling systems abound; some of the most prominent are AMPL[13], GAMS and XPRESS-MP. For a recent survey of LP software (including both modelling and solver software), see [12].

The C++ class library FLOPC++ for linear optimization modelling described in Paper D, "Formulating Linear Optimization Problems in C++", provides a notation similar to the languages provided by the algebraic modelling systems mentioned above.

Solving the model. Considering the practical importance of linear programming, it is not surprising that a vast number of different algorithms have been suggested for solving LPs. The currently most successful approaches are variations of the Simplex method (developed by Dantzig in the 1940s) and interior point methods. The current state of interior point methods is a result of the intensive research prompted by a paper from 1984 by Karmarkar, where he demonstrates how techniques from non-linear programming can be used to solve LPs efficiently, from a practical as well as a theoretical (that is, in polynomial time) point of view. Most LP solver packages contain implementations of both types of algorithms.

The code used to solve the unsymmetric systems of linear equations required by the simplex method is often referred to as the engine of the method, since the solution of these, typically very sparse, systems is generally the computationally most expensive part of the code. In paper B, "Direct Solution of Sparse Systems of Linear Equations using Spiked Triangular Permutation and Matrix Modification", an efficient implementation of a new method for solving these systems is presented.

Today LP-solving is a robust and fast technology and various high quality commercial solvers are available, f.ex. CPLEX, MOSEK and OSL.

Currently the preferred way to solve general MIPs is to use a linear programming based branch and bound procedure combined with generation of cutting planes. Although significant computational improvements of MIP solvers have been achieved during the last decade [6], there are still many practical examples of MIPs which are unsolvable by current MIP solvers in any reasonable amount of time.

Sometimes these problems can be solved by specialized solvers taking advantage of special structure present in the problem. However the ideal situation would be to have a general solver able to cope satisfactorily (in comparison with specialized solvers) with any kind of MIP without even the need to adjust parameter settings. Given the plethora of problems which can be formulated as MIPs (sorting for example) it is probable that this goal will never be fully achieved.

The CPLEX and OSL packages mentioned above can also solve MIPs.

Interpretation of the solution. The minimum prerequisite for interpretation of the solution of the model is that it is available in terms of the variable and constraint names used in the algebraic model formulation. It is the responsibility of the algebraic modelling systems to provide the link from the solver output to the modelling language which can therefore be used for report generation.

It might also be useful to visualize the results of the optimization, or export the results in some specific format for further processing. Generally the algebraic modelling systems does not offer much support for these tasks.

Harvey Greenberg's ANALYZE [18] is a software tool for analyzing linear programs and their solutions, including diagnostic analysis, when the LP is infeasible.

1.1 Overview of the thesis

Four different topics in computational linear optimization have been considered. My contributions to these topics are described in the four papers

Paper A. "Automatic Reformulation for Mixed 0/1 Programs"

Paper B. "Direct Solution of Sparse Systems of Linear Equations using Spiked Triangular Permutation and Matrix Modification"

Paper C. "Aggressive LP Reduction"

Paper D. "Formulating Linear Optimization Problems in C++"

included in this thesis.

The chapters 2 to 7 following this introduction are primarily meant to serve as a background and supplement for these papers.

In chapter 2, two real world applications of linear optimization (a production model for a poultry butchery and gridlock resolution for a real-time gross settlement system) in which I was involved during my Ph.D. studies are described. My experience with the implementation of the first application had a significant indirect influence on this thesis in that it raised my awareness of the complications involved in the integration of a MIP model with other software components (a GUI in this case). This awareness led to the development of the class library FLOPC++, described in Paper D, designed to incorporate algebraic modelling within a C++ program.

Chapter 3 discusses the role of algebraic modelling languages in computational linear optimization. GAMS is used as an example of an algebraic modelling language to illustrate some of the concepts.

Chapter 4 outlines the most important computational steps of both pivoting algorithms and interior point methods for solving LPs.

The parallel implementation described in paper B was implemented on the IBM SP at UNI-C (which has recently been shut down). Chapter 5 is a description of this distributed memory parallel processor and some of the issues involved in the implementation of efficient message passing code on this machine.

Chapter 6, on direct solution of sparse systems, provides some background material for paper B. Solution of sparse triangular systems is given special attention due to its extreme importance for the efficient implementation of the sparse simplex method.

And finally, in chapter 7, the conclusions are presented.

Chapter 2

Two applications of linear optimization

During my Ph.D studies I have been involved in two applications involving linear optimization, which are described in this chapter.

2.1 A production model for a poultry butchery

A production model was developed for the poultry butchery, Gedved Fjerkræslagteri (GF), which unfortunately went bankrupt in February 2000 before the model had been fully integrated in the planning procedures of the company. Before the bankruptcy GF slaughtered about 50 to 60 thousand chickens per day. Two thirds were sold as whole frozen chickens while the last third was cut into pieces and used for so called refined products.

The management of GF was uncertain if this distribution between whole chickens and refined products was the most profitable and therefore desired a tool for analyzing the consequences of eventual changes of the production capacities, besides being able to help in the short term (weekly) production planning. As it turned out the second use of the model became less important since the weekly plans were seldom followed anyway due to unforeseen

occurrences such as lack of packing, absence because of illness and urgent customer requests.

The model divides the delivered chickens into a number of weight intervals and takes as input the number of chickens in each of these weight intervals which will be delivered during the planning horizon (typically one week). The exact distribution only becomes known after the chickens have been slaughtered, but good estimates can be derived based on the age of the chickens and the age of the laying hens.

There are 5 different basic setups of the production. One of them consists in using all delivered chickens whole and the other four correspond to four different ways of cutting. For each of the four different cutting methods the capacity and preferred weight interval is given such that the number of chickens in each weight interval which are cut up in each of the four cases can be calculated before the optimization problem is solved. Since it is rather time consuming to change the basic setup, it is not allowed to do this after the daily production has started. The choice of basic production setup is modelled by a group of five binary decision variables which must sum up to one for each day. Another group of binary decision variables was introduced to model a so called joker machine which can be used for packaging of either breasts, legs or barbecued legs. The remaining variables were all continuous.

When the model was built, 19 different refined products existed. The fundamental problem was to decide how many chickens, x_{vpt}, in each weight interval, v, to use for product, p, per day, t. Most of the constraints were balance equations ensuring for example that each part of a chicken is used only once and that the total amount produced of each product corresponds to the number of chickens in each weight interval which were used for the product.

Other constraints included a capacity constraint for the joker machine and the possibility to limit the number of days used for a specific cutting method.

The objective of the model was to maximize the proceeds of the sale of the products minus the variable production costs (such as packing costs, etc.). The proceeds part of the objective function was modelled as piecewise linear functions of the quantity produced of each product. The expenses for salaries were considered a fixed cost and not included in the model. However the capacity of the cutting section is a model parameter which can be adjusted according to the number of machines and employees available.

The model had 16 different generic constraints and 10 different generic variables. For a typical week the model resulted in an LP with 3288 rows, 3064 variables (of which 40 were discrete), and 10784 non-zeroes.

Conclusion

Although in hindsight the model looks rather straightforward, the actual formulation process consisted of many trial and error steps. Most of the problems encountered arose from the difficulty to obtain all the relevant information. Often constraints had to be discovered when a solution produced by a prototype model violating the constraint was presented to the production management.

The model was formulated using the algebraic modelling language, GAMS, and data for the model were kept in text files conforming to the GAMS syntax. In the first phase, the person responsible for the production at GF was taught how to modify data in these files. It was determined that the model was a useful tool for planning and assessment of alternative production options. For example the model was used to evaluate suggestions for new products and the possibility to encourage (economically) the deliverance of heavier chickens. It was also determined that the direct modification of GAMS tables and sets was impractical for users without prior computer experience. Therefore a DELPHI programmer was hired for one month to create a menu based user interface and scenario management system. This task was significantly complicated by the need to produce syntactically correct GAMS files as output.

2.2 Gridlock Resolution

In real-time gross settlement (RTGS) systems a participating bank can place payment orders to other participating banks. Each participating bank has an account in the system. When a payment is settled the amount of the payment is debited to the account of the sending bank and credited to the account of the receiving bank. Normally a payment order is settled immediately if there are sufficient funds on the account of the sending bank to cover the payment. If a payment order can not be settled due to

insufficient funds it is placed in a queue. It might be possible to settle a queued payment order later if the account of the sending bank is credited by the settlement of payment orders from other banks. However it is possible that the queue can contain a set of payment orders which could be settled simultaneously but such that none of its payment orders can be settled individually. This situation is known as gridlock. Gridlock resolution refers to the identification and simultaneous settlement of such payment orders.

The Central Bank of Denmark is considering the implementation of a RTGS system with gridlock resolution and I was contracted to prepare an overview of gridlock resolution algorithms for this purpose.

Let n be the number of participating banks, and let m_i be the number of queued payment orders placed by the i'th bank. Each payment order placed by the i'th bank consists of an amount and an identification of the receiving bank. Let a_{ik} and r_{ik} be the amount and receiving bank, respectively, of the k'th payment order placed by bank i. The initial cover money deposited on the account of the i'th bank is denoted d_i. We assume that all amounts a_{ik} and d_i are nonnegative integers.

It is an important assumption that payments can not be split, that is, each payment order is either settled by a transfer of its total amount or not at all. We will use binary decision variables x_{ik} to indicate the status of each payment order. ($x_{ik} = 1$ if the k'th payment order placed by bank i is settled and $x_{ik} = 0$ if it is not.)

To facilitate the formulation of the models we introduce a parameter p_{ikj} defined for $1 \le i \le n$, $1 \le k \le m_i$, $1 \le j \le n$ with the value

$$p_{ikj} = \begin{cases} a_{ik} & \text{if } r_{ik} = j \\ 0 & \text{otherwise} \end{cases}$$

Now we are ready to formulate the Bank Clearing Problem. The objective is to maximize the total amount of settled payments subject to the constraint that none of the accounts of the participating banks are overdrawn.

$$\begin{array}{ll} \max & \displaystyle\sum_{i=1}^{n}\sum_{k=1}^{m_i} a_{ik}x_{ik} \\ \text{s.t.} & \displaystyle d_i - \sum_{k=1}^{m_i} a_{ik}x_{ik} + \sum_{j=1}^{n}\sum_{k=1}^{m_j} p_{jki}x_{jk} \ge 0 \quad \text{for } i = 1, .., n \end{array}$$

$$x_{ik} \in \{0,1\} \quad \text{for } i = 1,...,n, \; k = 1,...,m_i$$

The expression on the left hand side of the constraint gives the net balance of the account of bank i determined by its initial value, d_i, minus the total amount of settled payment orders send by the bank, $\sum_{k=1}^{m_i} a_{ik}x_{ik}$, plus the total amount received, $\sum_{j=1}^{n}\sum_{k=1}^{m_j} p_{jki}x_{jk}$. This net balance must be nonnegative for each of the participating banks.

As shown in [20] the Bank Clearing Problem is NP-hard because it contains the NP-hard Subset-Sum Problem as a special case. This special case occurs when $m_i = 0$ for all but a single bank. Even the problem of finding a feasible solution to the Bank Clearing Problem with a total transfer volume of at least ϵ times the optimal transfer volume is NP-hard for any $\epsilon \in]0,1]$, because a polynomial time algorithm for this approximation problem could be used to solve the NP-complete Change-Making Problem in polynomial time.

If the payment orders of each bank must be settled in a predetermined order we obtain the Sequence Constrained Bank Clearing Problem. We assume that the payment orders are presented in this desired order. That is, the k'th payment order placed by bank i can not be settled unless all the preceding (the first,...,the $k-1$'th) payment orders placed by bank i are also settled. This restriction can be formulated as an additional group of linear constraints

$$x_{ik+1} \leq x_{ik} \quad \text{for } i = 1,..,n, \; k = 1,...,m_i - 1$$

of the formulation of the Bank Clearing Problem shown above.

An example

Consider the following example with three participating banks ($n = 3$). The example contains 4 payment orders placed by Bank 1, 3 by Bank 2 and none by Bank 3 ($m_1 = 4, m_2 = 3, m_3 = 0$).

If the sequence of the payment orders must be observed, it is easily verified that only the first payment orders of Bank 1 and Bank 2 can be settled. If payment orders can be settled in any order, it is possible to settle all payments except the first placed by Bank 2. This gives a total transfer volume of 85, whereas the solution to the sequence constrained problem has a total transfer volume of only 20.

	Bank 1		Bank 2		Bank 3	
d		5		5		5
k	r_{1k}	a_{1k}	r_{2k}	a_{2k}	r_{3k}	a_{3k}
1	2	10	1	10	-	-
2	2	10	1	40	-	-
3	2	10	3	5	-	-
4	2	10	-	-	-	-

Table 2.1: Example 1

If the initial cover money deposit of Bank 2 is increased to 10 ($d_2 = 10$), it is possible to settle the first payment order placed by Bank 2 immediately. In this case the payment order would not be put in the queue of blocked payments orders and therefore the option of not settling this payment is not present in the optimization problem thereby leading to an inferior result.

This shows that it can be advantageous to queue some payment orders which could have been executed immediately. If such a blocking of a feasible payment order actually results in an increased overall transfer volume, obviously depends on the future payment orders arriving to the system. Intuitively it is seen that the probability is highest for small payment orders which would leave the account of the sending bank close to zero if settled.

The Sequence Constrained Bank Clearing Problem

This problem can be formulated as a linear optimization problem in binary variables as described above. Here we shall use a more compact non-linear formulation of the problem. As decision variables we use $0 \leq x_i \leq m_i$ to indicate the number of payments orders placed by bank i which are settled (of course x_i must be an integer).

$$\begin{array}{ll}
\max & \displaystyle\sum_{i=1}^{n}\sum_{k=1}^{x_i} a_{ik} \\
\text{s.t.} & \displaystyle d_i - \sum_{k=1}^{x_i} a_{ik} + \sum_{j=1}^{n}\sum_{k=1}^{x_j} p_{jki} \geq 0 \quad \text{for } i = 1,..,n \\
& x_i \in \{0,1,...,m_i\} \quad \text{for } i = 1,...,n
\end{array}$$

To simplify the notation we introduce the short hand notation $P_i(x)$ to denote the total amount received by the i'th bank in the solution x, i.e.

$$P_i(x) = \sum_{j=1}^{n} \sum_{k=1}^{x_j} p_{jki}$$

We are now ready to state an algorithm for solving the Sequence Constrained Bank Clearing Problem. The algorithm starts by tentatively activating all payment orders. If this is feasible, i.e. does not result in a negative net balance on any of the accounts, this is the optimal solution. Otherwise, for each bank where the tentative solution would lead to a deficit we deactivate the currently active payment order with lowest priority placed by the bank in deficit until its previous net balance plus the amount of the cancelled payment orders becomes nonnegative. This is repeated until the current solution becomes feasible.

for $i = 1, ..., n$ do $\bar{x}_i = m_i$;
repeat
  $x = \bar{x}$
  for $i = 1, ..., n$ do $\bar{x}_i = \max_k \{0 \le k \le x_i : d_i - \sum_{t=1}^{k} a_{it} + P_i(x) \ge 0\}$
  ;
until $\bar{x}=x$

Figure 2.1: An algorithm for the SCBCP

By updating the net balances after each deactivation of a payment order (not just once in each major iteration) we obtain a simplified algorithm shown in figure 2.2.

The algorithm goes as follows. Activate all payments; as long as there are deficient banks, deactivate the last payment order placed by a deficient bank.

The worst case time complexity of the algorithm is clearly linear in the total number of payment orders.

We call a settlement decision x feasible if the corresponding net settlement can be done without overdraft on the accounts of the participating banks, i.e. the total amount received from other banks minus the total amount debited from the account is less than or equal to the initial cover money

for $i = 1, ..., n$ do $x_i = m_i$;
for $i = 1, ..., n$ do $b_i = d_i - \sum_{t=1}^{x_i} a_{it} + P_i(x)$;
while not $b \ge 0$ **do**
  Let i be a bank such that $b_i < 0$
  $b_i = b_i + a_{ix_i}$
  $b_{r_{ix_i}} = b_{r_{ix_i}} - a_{ix_i}$
  $x_i = x_i - 1$
end while

Figure 2.2: A simplified algorithm for the SCBCP

for each bank. That is,

$$d_i - \sum_{k=1}^{x_i} a_{ik} + P_i(x) \ge 0 \quad \text{for } i = 1, ..., n$$

Lemma 2.1 *Suppose that y and x are two settlement decisions such that y is feasible, $x \ge y$ (i.e $\forall_{i=1}^{n} x_i \ge y_i$) and $d_j - \sum_{k=1}^{x_j} a_{jk} + P_j(x) < 0$ for some bank, j. Let*

$$\hat{x}_i = \begin{cases} x_i - 1 & \text{if } i = j \\ x_i & \text{otherwise} \end{cases} \qquad \text{for } i = 1, ..., n.$$

Then $\hat{x} \ge y$.

Proof: We only need to show that $y_j \le \hat{x}_j = x_j - 1$ since for $i \ne j$, $y_i \le \hat{x}_i = x_i$ is trivially satisfied. By hypothesis y is feasible meaning that $d_i - \sum_{i=1}^{y_i} a_{ik} + P_i(y) \ge 0$, for $i = 1, ..., n$. Since the solution x settles all payment orders settled by y (and some additional) ($x \ge y$) and all elements of p are nonnegative, we have that $P(x) \ge P(y)$. Combining this we get

$$d_i - \sum_{k=1}^{y_i} a_{ik} + P_i(x) \ge 0$$

for $i = 1, ..., n$ in particular also for $i = j$. But by hypothesis $d_j - \sum_{k=1}^{x_j} a_{jk} + P_j(x) < 0$ so we conclude that $\sum_{k=1}^{x_j} a_{jk} > \sum_{k=1}^{y_j} a_{jk}$ and therefore $x_j > y_j$. Hence, $x_j - 1 \ge y_j$. $\square$

The lemma establishes the correctness of the two algorithms as well as the uniqueness of the optimal solution to the Sequence Constrained Bank Clearing Problem.

The Bank Clearing Problem

Güntzer et al. [20] compares two heuristics for the Bank Clearing Problem, `EAF-2` and `Re-/Inactivation`. Although `EAF-2` does allow breaking the sequence given by the delivery order it is based on this order and favours settlement of the earliest delivered payment orders. This might be desirable in practice, but generally leads to solutions with an inferior total transfer volume compared to solutions obtained by heuristics, such as `Re-/Inactivation`, which does not consider the delivery order.

The `Re-/Inactivation` heuristic starts with all payment orders activated and is greedy in the sense that it tries to change the status of payment orders (from inactive to active or vice-versa) which leads to the largest decrease in the sum of deficiencies.

We show an example where this heuristic leads to a poor result.

	Bank 1		Bank 2		Bank 3	
d		0		0		100
k	r_{1k}	a_{1k}	r_{2k}	a_{2k}	r_{3k}	a_{3k}
1	-	-	1	10	1	100
2	-	-	-	-	2	10

Table 2.2: Example 2

In Example 2, activating all payments would lead to a net balance on the accounts of the three banks of 110, 0 and -10 respectively. The only way to decrease the sum of deficiencies by the inactivation of a single payment is to inactivate the payment from Bank 3 to Bank 1. This solution which is found by the `Re-/Inactivation` heuristic has a total transfer volume of 20, but as is easily seen, it is possible to obtain a solution with total transfer volume of 100.

Conclusion

Although it is possible to formulate the Sequence Constrained Bank Clearing Problem as a MIP, it does not seem practical to solve it by applying a general MIP solver to this formulation. Instead we saw how a simple linear time complexity algorithm can be used to solve it.

Chapter 3

Algebraic Modelling Languages

While an instance of a linear programming problem is specified by a coefficient matrix $A \in R^{m \times n}$, a cost vector $c \in R^n$ and a right hand side $b \in R^m$, these are hardly ever the entities in which the problem is conceived.

Real world LPs can easily have several thousand variables and constraints, hence it becomes impractical for the modeller to relate to each individual variable or constraint. The data used for the model, is typically organized as a collection of tables indexed by one or more sets of objects relevant for the problem. For a production, distribution and inventory model the fundamental sets could for example be the set of production facilities, the set of distribution centers, the set of customer zones and the set of time periods which constitutes the planning horizon. For a crew pairing model for the airline industry the fundamental sets would rather be the set of flight legs, the set of crew members, etc. Relations between the fundamental sets, represented as subsets of a Cartesian product of some of the fundamental sets, are also often relevant for the model.

Like the data, decision variables and constraints can also make use of these sets for indexing. If, for example, a model is built to find the optimal level of production on each of the available production facilities, a single generic variable indexed by the set of production facilities can be used.

We illustrate the power of indexing by a simple example, a transportation problem. Some commodity is to be transported from a number of sources to a number of destinations. Only a limited quantity of the commodity is available at each of the sources and each of the destinations must receive some minimum quantity. The transportation cost from a source to a destination is taken to be proportional to the quantity which is transported and the objective is to minimize the total transportation cost. Here is how the constraints could possibly be formulated in an algebraic modelling language, using the variable `x(i,j)` to denote the level of transportation from the source `i` and the destination `j` (`S` is the set of sources and `D` is the set of destinations) :

```
forall(i,S)
  supply(i) = sum(j,D,x(i,j)) <= CAPACITY(i);

forall(j,D)
  demand(j) = sum(i,S,x(i,j)) >= DEM(j);
```

When a model formulated in an algebraic modelling language is translated into an LP, each of the generic constraints are expanded into a number of individual constraints governed by the indexing set(s). Likewise each indexed sum construct is expanded into a sum of individual terms.

In the example above, if $S = \{$ seattle, sandiego $\}$ and $D = \{$ newyork, chicago, topeka $\}$, the following constraints are generated (the identification of each constraint is shown in parenthesis):

```
(supply(seattle)):
   x(seattle,newyork)+x(seattle,chicago)+x(seattle,topeka)
      <= CAPACITY(seattle);
(supply(sandiego)):
   x(sandiego,newyork)+x(sandiego,chicago)+x(sandiego,topeka)
      <= CAPACITY(sandiego);
(demand(newyork)):
   x(seattle,newyork)+x(sandiego,newyork) >= DEM(newyork);
(demand(chicago)):
   x(seattle,chicago)+x(sandiego,chicago) >= DEM(chicago);
(demand(topeka)):
   x(seattle,topeka)+x(sandiego,topeka) >= DEM(topeka);
```

If s is the number of sources and d is the number of destinations, the coefficient matrix generated from the algebraic formulation above would have $s+d$ rows, sd columns and $2sd$ non zeros. We see that each individual variable only appears in two individual constraints, thus the generated LP becomes sparser and sparser as the number of sources and destinations increase.

It is a typical behaviour of linear optimization models that the average number of constraints in which each individual variable appears remains independent of the size of the problem. Thus large scale LPs are typically extremely sparse.

The format obtained above by expanding indexed sums and constraints is generally not suited as input to a solver. An algebraic modelling system takes as input a generic algebraic representation of the underlying model plus a set of data which instantiates the model and generates the corresponding problem instance (LP or MIP) in the format required by the solver. Normally the communication with the solver is also handled by the algebraic modelling system as illustrated below:

We now give an example of a generic algebraic model representation, an associated data set and the corresponding problem instance as required by the CPLEX solver. The model is a very simple production planning model with a set of products, `P`, each of which can be produced either 'inside' or 'outside' at known costs. Furthermore inside production consumes a certain amount of some limited resources, `R`, per unit of production, as specified by the matrix `Consumption`. The model is written as it would normally be written by a modeller using algebraic notation. Algebraic modelling languages are designed to mimic this notation as close as possible, while still providing an unambiguous and machine readable notation.

Model

$$\text{minimize} \quad \sum_{p \in P} (InCost(p) * ip(p) + OutCost(p) * op(p))$$

$$\text{subject to}$$

$$\begin{aligned} \forall_{r \in R} &: \sum_{p \in P} Consumption(p,r) * ip(p) \leq Capacity(r) \\ \forall_{p \in P} &: ip(p) + op(p) = Demand(p) \\ \forall_{p \in P} &: ip(p), op(p) \geq 0 \end{aligned}$$

Data

$$R = \{0, 1\}$$

$$P = \{0, 1, 2\}$$

	0	1	2
Demand(p)	100	200	300
InCost(p)	0.6	0.8	0.3
OutCost(p)	0.8	0.9	0.4

	Capacity(r)
0	20
1	40

Consumption(r,p)	0	1	2
0	0.5	0.4	0.3
1	0.2	0.4	0.6

The problem instance in CPLEX format should be self-explaining except for the representation of the coefficient matrix by the four arrays Cst, Clg, Rnr and Elm. This is a standard representation of sparse matrices [11] – Rnr and Elm holds a collection of sparse vectors, Rnr gives the row number and Elm the corresponding value of each non zero element, Cst(i) contains the start position of column i in the arrays Rnr and Elm, while Clg(i) contains the number of non zero elements of column i (to be found in the following positions in Rnr and Elm) as shown in the example.

Problem Instance (Solvers format)

- Number of variables and constraints, `n` and `m`
- Optimization sense (minimize or maximize)
- Objective function coefficients, `c`
- Right hand side values, `b`
- Constraint types, `sense`
- Coefficient matrix, `Cst, Clg, Rnr, Elm`
- Bounds on variables, `l` and `u`
- Variable types (continuous or integer), `ctype`

$$\begin{array}{cc} & \begin{array}{cccccc} ip_0 & ip_1 & ip_2 & op_0 & op_1 & op_2 \end{array} \\ \begin{array}{c} l_0 \\ l_1 \\ d_0 \\ d_1 \\ d_2 \end{array} & \left[\begin{array}{ccc|ccc} 0.5 & 0.4 & 0.3 & & & \\ 0.2 & 0.4 & 0.6 & & & \\ \hline 1 & & & 1 & & \\ & 1 & & & 1 & \\ & & 1 & & & 1 \end{array}\right] \end{array}$$

```
int m=5; int n=6;
int c[]={0.6,0.8,0.3,0.8,0.9,0.4};
int b[]={20,40,100,200,300};
char sense[]=('L','L','E','E','E'};
int Cst[]={0,3,6,9,10,11};
int Clg[]={3,3,3,1,1,1};
int Rnr[]={0,1,2,0,1,2,0,1,3,0,1,4,2,3,4};
double Elm[]={0.5,0.2,1,0.4,0.4,1,0.3,0.6,1,1,1,1};
```

A list of desirable features of an algebraic modelling language is found in the classic book on mathematical programming modelling by Williams [30]. We comment briefly on some of these features below:

Indexing for repetition and use of repeated sums are, as already discussed, essential for managing large models.

Relations between indices. This feature can for example be used to restrict the generation of constraints or terms of repeated sums. For example, the partial sum, $\sum_{i<j} x_i$, can be written in GAMS as

```
sum(i$(ord(i) LT ord(j)), x(i))
```

Some models have underlying sets with a natural order on the elements (typically a set of time periods). Such models often contain constraints which link variables indexed by two subsequent set elements, as in the following constraint

```
forall(i,T)
   sbalance(i) =  s(i) + p(i) - c(i) == s(i+1);
```

To be able to formulate such constraints in a natural way, as shown above, it is necessary that the algebraic modelling language used support the use of index expressions such as $i+1$ for indexing.

Separating the data from the statements of the model. A separation between the data and the model formulation is naturally achieved when generic indexed constraints are used, since coefficients appearing in a generic constraint normally depends on the indices of the constraint. If a numeric value appears in a generic constraint it is good modelling practice to represent it by a symbolic name.

Arithmetic on the coefficients. The data appearing in an algebraic model formulation is often derived from other data. It should be possible to calculate the derived data within the algebraic modelling language instead of carrying out the necessary arithmetic externally.

Solver independent model formulation. Since the translation from a model and its associated data to a problem instance in a format suitable for the solver is carried out by the modelling system, the same model can be used no matter what solver is used to solve the generated instance (provided that the solver is supported by the modelling system).

3.1 GAMS

This section gives a short overview of the first algebraic modelling language, GAMS.

GAMS was developed in the seventies by the World Bank to facilitate the development and maintenance (modification) of optimization models and is still widely used. Prior to the advent of GAMS, the translation from an algebraic description of a model to a format recognizable by the software used to solve it was often difficult and error-prone and presented a major obstacle for the practical success of optimization models. This task has been automated by GAMS. GAMS provides a machine readable notation for formulating optimization models which closely resembles the algebraic notation which most mathematically trained modellers find natural. Using this notation, an ASCII file, say model.gms, containing both data and model description must be prepared.

When a solve statement is encountered GAMS will generate a model instance based on the input file and pass it to the underlying optimization software in the required format. After the problem has been solved, information about the optimal solution is passed back to GAMS where it can be used for report generation.

We will show how to formulate and solve

$$\text{maximize} \quad c^T x \qquad \text{s.t.} \quad Ax \leq b \ , \quad x \geq 0$$

where

$$A = \begin{bmatrix} 1 & 1 & 1 & -2 \\ 0 & 1 & 0 & 1 \\ -1 & -1 & 1 & -1 \end{bmatrix} \qquad b = \begin{bmatrix} 4 \\ 3 \\ 2 \end{bmatrix} \qquad c = \begin{bmatrix} 3 \\ 2 \\ -1 \\ -1 \end{bmatrix}$$

using GAMS.

It should be emphasized that the real benefit of GAMS is its ability to generate the problem data A, b and c, given an algebraic representation of a specific LP model. This example serves solely to illustrate elements of the GAMS syntax without having to introduce details of a specific model.

The problem was stated using matrix notation. An equivalent algebraic

formulation is to maximize z subject to

$$z = \sum_{j \in J} c_j x_j$$

$$\sum_{j \in J} a_{ij} x_j \leq b_i \ , \ \forall i \in I$$

$$x_j \geq 0 \ , \ \forall j \in J$$

which can be expressed in GAMS as follows

```
$inlinecom { }
$include "data.gms"

{ ----------- Here comes the model --------------- }
variable z  'objective function value' ;
positive variable x(j)  'activity levels' ;

equations
  defz
  constraint  'lp constraints' ;

defz .. z =E= sum(j,c(j)*x(j));

constraint(i) .. sum(j,A(i,j)*x(j)) =L= b(i);

model mylp /all/;
solve mylp maximizing z using lp;
```

where data.gms is a file containing

```
{ ------------- Here comes the data -------------- }
sets  i  'row index set'  /r1*r3/
      j  'column index set'  /c1*c4/ ;

table A(i,j)  'Coefficient matrix'
           c1    c2    c3    c4
```

```
      r1     1     1     1    -2
      r2           1           1
      r3    -1    -1     1    -1

parameter b(i)  'Right hand sides'
/  r1    4,
   r2    3,
   r3    1  / ;

parameter c(j)  'Objective function coefficients'
/  c1    3,
   c2    2,
   c3   -1,
   c4   -1  / ;
```

sets Almost everything in GAMS is based on sets. Sets are used as domains of parameters, variables, equations and multi-dimensional sets. Here we have two sets, the row index set, `i=r1,r2,r3` , and the column index set, `j=c1,c2,c3,c4` . A name of a set also serves as index to the set. The elements of a set are called labels. Numbers are allowed as labels, but are potentially dangerous since they are treated as strings with no numerical value associated.

parameters Parameters are used to hold data. They can be indexed by one or more sets (like arrays in C) or not at all in which case they hold a single numerical value. GAMS has two alternative syntaxes for declaring a parameter which is not indexed. `parameter s;` or `scalar s;` The example shows how parameters can be defined (have values assigned to them) when they are declared. The table construct used above is a convenient way to declare and define two or higher dimensional parameters.

variables It is necessary to declare the decision variables of the model before they can be used in the equations. Variables can be declared as free (the default), positive (≥ 0), negative (≤ 0), binary or integer.

In GAMS we must always specify a single variable as the objective function which we want minimize or maximize. This is why we introduced the variable z above to hold the value of the objective function value. This variable must be free (no lower or upper bounds), in particular it can not be declared as a positive variable.

equations Equations are used to specify the constraints of the model. An equation which is not indexed corresponds to a single constraint (row). An indexed equation generates a number of constraints depending on the cardinality and number of sets over which it is indexed. Equations are declared by listing their names following the keyword `equations`. (Commas and/or newlines are used as separators in lists). A definition of an equation consists of the name of the equation indexed by the sets over which it is quantified followed by two dots `..` followed by the expression used to generate the individual constraints.

The same GAMS model can be used independently of the size of the problem. To solve an LP with say 3000 rows and 4000 variables we simply include an appropriate data file for the problem.

3.2 Conclusion

Algebraic modelling languages closely resembles the notation commonly used by researchers and practitioners for describing and communicating linear optimization models, yet they impose a formal syntax which makes it easy to process the model formulation by a computer program.

Manual generation of the contraints of an instance of a linear optimization model is very time consuming, even for small instances. For larger instances this task becomes impossible and some kind of automatic generation is required. Other kinds of modelling systems exist which are not based on an algebraic model description, but these are generally used only within specific application areas [30].

Another way to generate a problem instance from a model and associated data is to use some conventional programming language to express the translation. When this approach is used the burden of keeping track of row and column numbers of the generated coefficient matrix is on the shoulders of the programmer. The underlying algebraic description is not explicitly apparent in the resulting code, which becomes difficult to debug and modify. However, in situations where an optimization module is intended as an integrated part of an end-user application, the use of a traditional programming language is advantageous. The C++ class library FLOPC++, described in Paper D, is designed to provide the same functionality as traditional algebraic modelling languages directly within a C++ program.

Chapter 4

Linear Programming Algorithms

In this chapter we give a succinct introduction to the two computationally most successful families of algorithms for solving linear programs: pivoting algorithms and interior point algorithms. Our principal objective is to expose the computational effort required in each iteration of both families of algorithms. Algorithmic refinements, such as anti-cycling strategies, steepest edge pricing and higher order corrections will not be considered. More detailed descriptions of LP algorithms can be found in numerous textbooks, f.ex. [25] or [31].

We consider a linear program in standard form. Let m be the number of constraints and let n be the number of variables. Given a cost vector $c \in R^n$, a right hand side $b \in R^m$ and a coefficient matrix $A \in R^{m \times n}$ of full row rank, the problem is to find a vector $x \in \{x \in R^n : Ax = b, x \geq 0\}$ such that $c^T x = \min \{c^T x : Ax = b, x \geq 0\}$.

For any linear program (the primal LP) it is possible to pose a related dual linear program. Taking the dual of the dual linear program we get back to the original primal linear program, hence any LP gives rise to a primal-dual pair of LPs.

The generic primal-dual pair of LPs in standard form is

$$\text{P: minimize } c^T x \quad \text{subject to } Ax = b, \quad x \geq 0$$

and

$$\text{D: maximize } b^T y \quad \text{subject to } A^T y \leq c$$

The following relationship between the primal and dual problems follows immediately.

Theorem 4.1 *(Weak duality) If x is a primal feasible solution (i.e. $Ax = b$ and $x \geq 0$) and y is a dual feasible solution (i.e $A^T y \leq c$), then $c^T x - b^T y = x^T(c - A^T y) \geq 0$*

Proof. $c^T x - b^T y = x^T c - (Ax)^T y = x^T(c - A^T y) \geq 0$ (since $x \geq 0$ and $c - A^T y \geq 0$) □

The sufficiency of the following optimality conditions for solutions of P and D follows directly from the weak duality theorem.

Theorem 4.2 *(Optimality conditions (primal version)) The vector $x \in R^n$ is an optimal solution of P if and only if there exists a vector $y \in R^m$ such that the following conditions are satisfied: $Ax = b$, $x \geq 0$, $A^T y \leq c$ and $c^T x = b^T y$.*

Theorem 4.3 *(Optimality conditions (dual version)) The vector $y \in R^m$ is an optimal solution of D if and only if there exists a vector $x \in R^n$ such that the following conditions are satisfied: $Ax = b$, $x \geq 0$, $A^T y \leq c$ and $c^T x = b^T y$.*

We will prove the necessity of the optimality conditions by applying a technique for eliminating variables from a system of inequalities known as Fourier-Motzkin elimination, which we will therefore briefly describe here.

Consider an inequality system $Ax \leq b$. To eliminate a variable, say x_1, we partition the inequalities of $Ax \leq b$ into three sets, I_+, I_- and I_0, according to the sign of the coefficient of x_1 in the inequality. Our goal is to derive an inequality system in the remaining variables $x_2, x_3, ..., x_n$

which is consistent (that is, admits a solution) if and only if the original system is. For each $i \in I_+$, $a_{i1} > 0$ and we get an upper bound on x_1

$$x_1 \leq (b_i - a_{i2}x_2 - \ldots - a_{in}x_n)/a_{i1}$$

and for each $i \in I_-$ (since $a_{i1} < 0$) we get a lower bound on x_1

$$(b_i - a_{i2}x_2 - \ldots - a_{in}x_n)/a_{i1} \leq x_1$$

Clearly it is only possible for x_1 to satisfy these bounds if the largest of the lower bounds is less than or equal to the smallest of the upper bounds, i.e. if

$$\max_{i \in I_-}(b_i - a_{i2}x_2 - \ldots - a_{in}x_n)/a_{i1} \leq \min_{i \in I_+}(b_i - a_{i2}x_2 - \ldots - a_{in}x_n)/a_{i1}$$

We note that this is equivalent to the inequality system

$$(b_i - a_{i2}x_2 - \ldots - a_{in}x_n)/a_{i1} \leq (b_k - a_{k2}x_2 - \ldots - a_{kn}x_n)/a_{k1} \quad \forall i \in I_- \ \ \forall k \in I_+ \tag{4.1}$$

which also can be written as

$$\frac{1}{a_{k1}}(a_{k1} + a_{k2}x_2 + \ldots + a_{kn}x_n) + \frac{-1}{a_{i1}}(a_{i1} + a_{i2}x_2 + \ldots + a_{in}x_n) \leq$$

$$\frac{1}{a_{ik}}b_k + \frac{-1}{a_{i1}}b_i \quad \forall i \in I_- \quad \forall k \in I_+$$

and conclude that a system equivalent to $Ax \leq b$ is achieved by substituting the inequalities in which x_1 appears by 4.1.

Note that every single inequality of 4.1 is obtained by addition of positive multiples of two of the original inequalities of $Ax \leq b$. Using this property inductively as more variables are eliminated we see that inequalities resulting from repeated application of Forier-Motzkin elimination are always positive combinations of original inequalities, i.e of the form $u^T A \leq u^T b$ with $u \geq 0$.

Proof. (of optimality conditions (dual version)) Sufficiency follows from the weak duality theorem. To prove the necessity (strong duality), assume that y is an optimal solution of D. All we need to show is the existence of a vector $x \in R^n$ such that $Ax = b$, $x \geq 0$ and $c^T = b^T y$.

D can be viewed as a problem of maximizing the variable f subject to the system of inequalities given by $f - b^T y \leq 0$ and $A^T y \leq c$. Let $f \leq d_i$ for $i = 1,..,q$ and $0 \leq d_i$ for $i = q+1,..,r$ be the inequality system resulting from eliminating the y variables by Fourier-Motzkin elimination. Since we assume that D has an optimal solution, $d_i \geq 0$ for all $i = q+1,..,r$ (the problem is feasible) and $q \geq 1$ (the problem is bounded). The optimal value of f is $\max\{d_i : i = 1,..,q\}$, let s be an index for which the maximum is attained. As the inequality $f \leq d_s$ is constructed by application of Forier-Motzkin elimination it can be written as a non-negative combination of the rows of the original system $\begin{bmatrix} 1 & -b^t \\ 0 & A^T \end{bmatrix} \begin{bmatrix} f \\ y \end{bmatrix} \leq \begin{bmatrix} 0 \\ c \end{bmatrix}$, hence we get that $-b^T + x^T A^T = 0$ and $x^T c = d_s = b^t y$ for some $x \geq 0$, which proves the existence of a vector x satisfying the optimality conditions. □

The primal version of the optimality conditions can be proved by applying the dual version to the problem maximize $-c^T x$ subject to $Ax \leq b, -Ax \leq -b, -x \leq 0$ which is equivalent to P.

4.1 Pivoting Algorithms

A basis is a subset $B = \{B_1, \ldots, B_m\}$ of $\{1, \ldots, n\}$ such that $|B| = m$ and $\mathbf{B} \equiv A_{\bullet B}$ is non-singular. B is called (primal) feasible if $\mathbf{B}^{-1}b \geq 0$ and B is called dual feasible if $c_N - A^T_{\bullet N}\mathbf{B}^{-T}c_B \geq 0$, where $N = \{1, \ldots, n\} \setminus B$. If B is both primal and dual feasible then x given by $x_B = \mathbf{B}^{-1}b$ and $x_N = 0$ is an optimal solution of P, and $y = \mathbf{B}^{-T}c_B$, is an optimal solution of D, as can be verified by examining the optimality conditions.

For a given a basis B, let x_B, z_N and D_{BN} be defined as follows:

$$x_B = \mathbf{B}^{-1}b$$

$$z_N = c_N - A^T_{\bullet N}\mathbf{B}^{-T}c_B$$

$$D_{BN} = \mathbf{B}^{-1}A_{\bullet N}$$

Theorem 4.4 *Let B be a basis. Let $i \in B$ and $j \in N$. Then $B' \equiv B \cup \{j\} \setminus \{i\}$ is a basis if and only if $D_{ij} \neq 0$.*

Proof. By definition of D, $A_{\bullet B}D_{Bj} = A_{\bullet B\setminus\{i\}}D_{B\setminus\{i\}\ j} + A_{\bullet i}D_{ij} = A_{\bullet j}$. If $D_{ij} = 0$ then $A_{\bullet B\setminus\{i\}}D_{B\setminus\{i\}\ j} - A_{\bullet j} = 0$ showing the existence of a solution to $A_{\bullet B'}x = 0$ which is not zero, implying singularity of $A_{\bullet B'}$. Hence B' is not a basis.

If $D_{ij} \neq 0$ then $A_{\bullet i} = \frac{1}{D_{ij}}(A_{\bullet j} - A_{\bullet B\setminus\{i\}} D_{B\setminus\{i\}\ j})$ which shows that $A_{\bullet B'}$ can be obtained from $A_{\bullet B}$ by simple column operations, thus non-singularity of $A_{\bullet B'}$ follows from the non-singularity of $A_{\bullet B}$. Hence B' is a basis. □

Two bases B and B' are *adjacent* if $B' = B \cup \{q\} \setminus \{p\}$ for some $p \in B$ and $q \in N$.

Pivoting algorithms maintain a current basis which is replaced by an adjacent basis in each iteration. The iterations continue until the current basis becomes optimal (both primal and dual feasible) or unboundedness is discovered.

The following assignments update x_B and z_N when B is replaced by $B \cup \{q\} \setminus \{p\}$

$$\begin{aligned}
\alpha &= \frac{x_p}{D_{pq}} \\
\beta &= -\frac{z_q}{D_{pq}} \\
x_B &= x_B - \alpha D_{\bullet q} \\
x_q &= \alpha \\
z_N &= z_N + \beta D_{p\bullet} \\
z_p &= \beta
\end{aligned}$$

For sparse problems it is cheaper to compute the column $D_{\bullet q}$ and the row $D_{p\bullet}$ from scratch by

$$D_{\bullet q} = \mathbf{B}^{-1} A_{\bullet j}$$

and

$$D_{p\bullet} = e_p^T \mathbf{B}^{-1} A_{\bullet N}$$

in each iteration than to update the whole matrix D from iteration to iteration.

Replacing B by $B \cup \{q\} \setminus \{p\}$ is called a pivot. For each pair $p \in B$ and $q \in N$ such that $D_{pq} \neq 0$ a different pivot is possible. We classify the

	x_p	z_q	D_{pq}	x_q	x_p	
1	+	+	+	+	−	
2	+	+	−	−	+	
3	+	−	+	+	+	(admissible II) primal simplex
4	+	−	−	−	−	
5	−	+	+	−	−	
6	−	+	−	+	+	(admissible I) dual simplex
7	−	−	+	−	+	(admissible II)
8	−	−	−	+	−	(admissible I)

Table 4.1: Possible pivot types.

pivots according to the sign of x_p, z_q and D_{pq} as shown in Table 4.1 (for simplicity we do not consider the cases where $x_p = 0$ and/or $z_q = 0$).

Pivoting algorithms for solving linear programs are classified by the type of pivots they use. The primal simplex method uses only pivots of type 3 and the dual simplex method uses only pivots of type 6. The criss-cross method uses pivots of type 3,6,7 and 8. It can be shown [15] that it is possible to come from any basis to an optimal basis by at most $m + n$ pivots of these types.

A criss cross algorithm must be supplied with a rule for choosing the incoming and leaving variables, p and q. The least index pivot rule (Figure 4.1) guarantees finite termination of the algorithm [14]. But in practice the computational performance of the least index criss-cross method is very poor. As shown in Table 4.2, not even a prior knowledge of an optimal basis can be used to make it competitive with the simplex method.

4.1.1 The Simplex Method

Contrary to the criss cross method, the simplex method maintains primal or dual feasibility of the current basis throughout the iterations. The primal simplex method requires a starting basis which is primal feasible and the dual simplex method requires a starting basis which is dual feasible. Such starting bases can be obtained by solving a phase one linear program.

Input: A basis B, $z_N = c_N - A_N^T \mathbf{B}^{-T} c_B$ and $x_B = \mathbf{B}^{-1} b$
Let $p = min\{i \in B : x_i < 0\}$ ($p = \infty$ if $\{i \in B : x_i < 0\} = \emptyset$)
Let $q = min\{j \in N : z_j < 0\}$ ($q = \infty$ if $\{j \in N : z_j < 0\} = \emptyset$)
if $p = \infty \wedge q = \infty$ stop (solution is optimal)
if $p < q$ **then**
  Let $q = min\{j \in N : D_{pj} < 0\}$
  if $q = \infty$ stop (problem is infeasible)
else
  Let $p = min\{i \in B : D_{iq} > 0\}$
  if $p = \infty$ stop (problem is unbounded)
end if
Let $B = B \cup \{q\} \setminus \{p\}$

Figure 4.1: An Iteration of the least index Criss Cross Method

Matrix	Natural	OB first	OB last	Simplex
adlittle	85710	6640	10415	99
afiro	59	24	88	11
sc50a	178	155	257	30
scagr7	7166	1340	3490	92
stocfor1	4047	1061	1394	31

Table 4.2: Number of iterations required by the least index criss-cross method for solving some small test problems for different initial variable orderings. The natural order of the variables is the order in which they first appear in the MPS file (slack variables are first in the order). OB first refers to an ordering obtained by letting the basic variables from a known optimal basic solution appear first in the ordering. In OB last these variables appear last. For comparison, the last column shows the number of iterations required by the CPLEX primal simplex implementation. the CPLEX

Input: A primal feasible basis B, $z_N = c_N - A_N^T \mathbf{B}^{-T} c_B$ and $x_B = \mathbf{B}^{-1} b$
P1: q =argmin$\{z_k : k \in N\}$ (if $z_N \geq 0$ then Stop (Optimal))
P2: Solve $\mathbf{B} d_B = A_q$
P3: α, p =min,argmin$\{x_k/d_k : d_k > 0, k \in B\}$ (if $d_B \leq 0$ then Stop (Unbounded))
P4: $x_B = x_B - \alpha d_B$; $x_q = \alpha$
P5: Solve $\mathbf{B}^T w = e_p$
P6: $r_N = A_N^T w$
P7: $\beta = -\frac{z_p}{d_p}$, $z_N = z_N + \beta r_N$, $z_p = \beta$
P8: $B = B \setminus \{p\} \cup \{q\}$

Figure 4.2: An Iteration of the Primal Simplex Method

To ensure that primal or dual feasibility is preserved after a pivot, a so-called ratio-test must performed to choose the leaving index p in the primal method and the entering index q in the dual method. An iteration of the primal simplex method is shown in Figure 4.2 and an iteration of the dual simplex method is shown in Figure 4.3.

It is seen that the computational effort of an iteration of the dual simplex method is equal to the computational effort of an iteration of the primal simplex method.

The relative execution time of each of the individual steps of the primal simplex algorithm varies considerably from problem to problem, but, except for problems with a very big number of columns compared to the number of rows, the solves in P2 and P5 are the most time consuming steps followed by P6 [24, 7].

The sparse simplex method is notoriously difficult to parallelize. In "The 1997 Petaflops Algorithms Workshop Summary Report" [28] a set of algorithms were classified into three categories: (1) those which appear to be scalable to petaflops systems, given appropriate effort; (2) those which appear scalable, provided certain significant research challenges are overcome; (3) those which appear to possess major impediments to scalability. Four algorithms including sparse unsymmetric Gaussian elimination and the sparse simplex method were judged to belong to the last category.

Input: A dual feasible basis B, $z_N = c_N - A_N^T \mathbf{B}^{-T} c_B$ and $x_B = \mathbf{B}^{-1} b$
D1: $p =$argmin$\{x_k : k \in B\}$ (if $x_B \geq 0$ then Stop (Optimal))
D2: Solve $\mathbf{B}^T w = e_p$
D3: $r_N = A_N^T w$
D4: $\beta, q =$min,argmin$\{-z_k/r_k : r_k < 0, k \in N\}$ (if $r_N \geq 0$ then Stop (Infeasible))
D5: Solve $\mathbf{B} d_B = A_q$
D6: $\alpha = \frac{x_p}{r_q}$, $x_B = x_B - \alpha d_B$, $x_q = \alpha$
D7: $z_N = z_N + \beta r_N$, $z_p = \beta$
D8: $B = B \setminus \{p\} \cup \{q\}$

Figure 4.3: An Iteration of the Dual Simplex Method

4.2 Interior Point Algorithms

Primal-dual interior-point algorithms is currently one of the most efficient methods for solving LP's. Most existing primal-dual interior-point LP codes are based on Mehrotra's predictor-corrector algorithm, which we describe in this section.

Introducing slack variables z and using $\bullet$ to denote element-wise multiplication of vectors, the LP optimality conditions can be rewritten as:

$$F(x,y,z) \equiv \begin{bmatrix} Ax - b \\ A^T y + z - c \\ x \bullet z \end{bmatrix} = 0$$

and

$$x, z \geq 0$$

The algorithm seeks to solve the system $F(x,y,z) = 0$ by an iterative Newton-like method while keeping x and z strictly positive. Figure 4.4 shows the steps involved in an iteration of Mehrotra's predictor-corrector algorithm.

Primal-dual interior point algorithms never find an exact solution of the linear program. Typically a simple termination criteria is used to stop

Input: (x,y,z) such that $x > 0$ and $z > 0$.

1: Let $X = \text{diag}(x_1, x_2, ..., x_n)$ and $Z = \text{diag}(z_1, z_2, ..., z_n)$

2: Solve $\begin{bmatrix} X & Z & 0 \\ I & 0 & A^T \\ 0 & A & 0 \end{bmatrix} \begin{bmatrix} p_z \\ p_x \\ p_y \end{bmatrix} = \begin{bmatrix} -x \bullet z \\ c - A^T y - z \\ b - Ax \end{bmatrix}$
3: $\alpha = \max\{0 \leq \alpha \leq 1 : x + \alpha p_x \geq 0\}$
4: $\beta = \max\{0 \leq \beta \leq 1 : z + \beta p_z \geq 0\}$
5: $\sigma = ((x + \alpha p_x)^T (z + \beta p_z)/x^T z)^3$
6: $\mu = x^T z/n$

7: Solve $\begin{bmatrix} X & Z & 0 \\ I & 0 & A^T \\ 0 & A & 0 \end{bmatrix} \begin{bmatrix} c_z \\ c_x \\ c_y \end{bmatrix} = \begin{bmatrix} \sigma\mu e - p_x \bullet p_z \\ 0 \\ 0 \end{bmatrix}$
8: $\alpha = 0.99 \max\{\alpha \geq 0 : x + \alpha(p_x + c_x) \geq 0\}$, **If** $\alpha > 1$ **then** $\alpha = 1$
9: $\beta = 0.99 \max\{\beta \geq 0 : z + \beta(p_z + c_z) \geq 0\}$, **If** $\beta > 1$ **then** $\beta = 1$
10: $x = x + \alpha(p_x + c_x)$, $y = y + \beta(p_y + c_y)$, $z = z + \beta(p_z + c_z)$

Figure 4.4: An iteration of Mehrotra's predictor-corrector algorithm.

the algorithm when the current iterate is sufficiently close to fulfill the optimality conditions. For example when $\frac{||Ax-b||}{1+||b||} \leq 10^{-8}$, $\frac{||A^Ty+z-c||}{1+||c||} \leq 10^{-8}$ and $\frac{|c^Tc-b^Ty|}{1+|c^Tx|} \leq 10^{-8}$. The number of iterations required to find a solution which satisfies the termination criteria is generally between 20 and 50 irrespective of the size of the problem.

Clearly, the most time consuming tasks in each iteration are the solves in step 2 and 7. Since the coefficient matrix of the two systems are the same, a factorization of the matrix $M \equiv \begin{bmatrix} X & Z & 0 \\ I & 0 & A^T \\ 0 & A & 0 \end{bmatrix}$ can be reused for solving the second system.

One step of block LU factorization of the matrix M gives:

$$M = \begin{bmatrix} I & 0 & 0 \\ X^{-1} & I & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} X & Z & 0 \\ 0 & -X^{-1}Z & A^T \\ 0 & A & 0 \end{bmatrix}$$

Introducing the shorthand notation $D = X^{-1}Z$ and performing another block LU factorization step we get:

$$M = \begin{bmatrix} I & 0 & 0 \\ X^{-1} & I & 0 \\ 0 & -AD^{-1} & I \end{bmatrix} \begin{bmatrix} X & Z & 0 \\ 0 & -D & A^T \\ 0 & 0 & AD^{-1}A^T \end{bmatrix}$$

Thus the factorization of M can be completed by computing a Cholesky factorization of the symmetric positive semidefinite matrix $AD^{-1}A^T$. This is known as the normal equations approach. Alternatively a factorization of the symmetric indefinite matrix $\begin{bmatrix} -X^{-1}Z & A^T \\ A & 0 \end{bmatrix}$ can be used. This is known as the augmented system approach.

Most interior-point implementations are based on the normal equations approach. One of the reasons for this is that algorithms and sofisticated software for sparse Cholesky factorization are readily available. Furthermore, since the non-zero structure of $AD^{-1}A^T$ remains the same from iteration to iteration, a fill reducing reordering can be computed only once and used in all subsequent iterations. However, the additional freedom in the choice of pivot order introduced by the augmented system approach can be used to decrease the amount of fill-in considerably for some problems [27].

Several successful parallel implementations of interior-point methods for general sparse problems have been reported [22, 23, 8, 4].

Chapter 5

Parallel computing on the IBM SP

Technological advances have allowed the construction of faster and faster computers ever since the first computers were built. Yet the demand for increased computing power continues. Physical limits such as the speed of light is limiting the scope for further improvements of sequential computing speed, forcing computer constructors to exploit parallelism to meet this demand.

This chapter gives an introduction to some of the issues involved in writing programs for parallel computers. The purpose of writing parallel programs is normally a desire to decrease the wall clock execution time as compared with a sequential implementation. Another reason could be to solve problems which requires more memory than available on a single processor.

Different parallel computational models (paradigms) offers different conceptual views of what types of operations are available for the programmer [19]. The parallel computational model does not necessarily correspond to the physical architecture of the computer on which the program is executed.

Data parallel programming model. In this model a single program with a global name-space defines the operations (much like in a conventional sequential model). Parallelism is achieved by executing operations on array elements simultaneously across all processors. The compiler is responsible

for the actual distribution of data and communication between processors. High Performance Fortran is an extension of Fortran with constructs for guiding the compiler to generate efficient code for parallel array operations.

The shared memory parallel programming model operates with multiple processes sharing the same global name-space, thus contrary to the data parallel model, parallelism is explicitly specified by the programmer. Communication between the processes is achieved by reading and writing shared variables. Access to locations manipulated by several processes must be coordinated by some form of locking. The OpenMP standard includes a set of library supported compiler directives for specifying synchronization points, critical sections, etc.

The message passing parallel programming model. In this programming model each process executes an independent program with its own local memory space. In order to access data pertaining to another process, this data must be explicitly communicated between the two processes. For such a communication to take place both processes involved must participate actively. The Message Passing Interface (MPI) specifies a set of subroutines for communication between processes. MPI has been implemented on a vast number of platforms including both distributed and shared memory multiprocessors, networks of workstations and even single processor computers.

In Paper B a parallel message-passing program was developed for the IBM SP system at UNI-C using MPI. The message passing parallel programming model fits well on parallel computers consisting of separate processors connected by a communication network. Most modern supercomputers, including the IBM SP, and also networks of workstations fall into this category. This chapter gives an introduction to the IBM SP system at UNI-C and some of the issues involved in writing efficient parallel message-passing programs.

5.1 The IBM SP system at UNI-C

The system has a total of 100 processors, called nodes, each with its own local memory. Four of these nodes are for interactive use, 12 belong to a pool for parallel test jobs, 64 are reserved for parallel batch jobs and the last 20 are used for sequential batch jobs. All nodes run under the IBM

AIX operating system. The nodes can execute different programs simultaneously and are connected by a high-performance switch through which communication between the nodes can take place. The communication speed is essentially the same between any pair of nodes.

5.2 Hardware characteristics

The 64 nodes available for parallel batch jobs are so-called thin nodes consisting of a 120 MHz P2SC processor and a 3.5 GByte scratch disk space. Half of the parallel batch nodes are equipped with 512 MByte of main memory and the other half with 256 MByte. If a job needs to be executed on large memory nodes, this must be specified in the job command file.

Each processor has two fixed point units, two floating point units, and an instruction and branch control unit. This super-scalar architecture makes it possible to perform two floating point multiply/add operations per clock cycle, provided no dependencies exist and that data can be made available to the CPU sufficiently fast.

Each node is equipped with a 128 KB data cache and a 32 KB instruction cache. The data cache consists of 1024 cache lines of 128 bytes each, arranged in four-way associative sets. Be aware that the access of data in strides which are a power of two (or a multiple hereof) can slow down the program considerably by reducing the effective size of the cache.

Messages can be sent between the processors using Ethernet or the high-performance switch. The high-performance switch can be used with two different protocols, Internet protocol and User Space protocol. The latter is considerably faster and should always be used for production runs. The communication time can be approximated by a linear function of the message size. The linear dependency is described by two parameters: the latency, τ_S, which is the start up time to initiate a communication and the bandwidth, β, which is the rate at which data is transfered after the communication has been initiated. I have measured the values of the latency and the bandwidth for point to point communication using the high performance switch with the User Space protocol on JENSEN to:

$$\tau_S = 45 \text{ microseconds} \qquad \beta = 98.7 \text{ MB/second}$$

In practice the communication time varies a lot so the values above were found using average times of several communications of the same size.

The communication speed is essentially the same for communication between any two nodes of the machine, so in contrast to some other parallel computers the concept of nearest neighbour is not important on JENSEN.

5.3 Performance tuning

The primary reasons for parallelizing computer codes is to decrease the execution (wall clock) time and to solve problems which requires more memory than is available on a single processor. A decrease in execution time can often be achieved by tuning the code for better performance on the target machine. When writing parallel programs, performance is almost always a major concern and the issue of performance tuning should not be ignored.

A detailed description of the techniques which were used to improve performance of the ESSL BLAS on the SP nodes is given in [1]. A similar hand tuning of application codes is usually not realistic due to the big effort which would be required. Fortunately highly optimized subroutine libraries are available which covers many of the computationally intensive tasks typically encountered in applications. For the remaining parts of the code good performance can often be achieved by automatic tuning done by the compiler. Summing up, the main strategies for improving the speed of your code on the SP architecture are:

1. **Use subroutines from optimized subroutine libraries.** The subroutine libraries installed on JENSEN includes LAPACK and IBM ESSL as well as their parallel versions ScaLAPACK and PESSL. The Basic Linear Algebra Sub-programs, BLAS, are among the routines provided by ESSL.

2. **Use compiler options to optimize the code.** Compilers are generally getting better and better at optimizing codes for their target machines. The effort and strategy of this optimization is controlled by various compiler options. The best combination varies from program to program but the options

```
-O3 -qarch=pwr2 -qfloat=hssngl -qtune=pwr2
```

will produce good results in most cases. It might be necessary to add the option **-qstrict** as the **-O3** optimization otherwise has the potential of changing the semantic of the program in some special cases.

3. **Hand tune.** There exist several general techniques, such as blocking and loop unrolling, for hand tuning a program. Since hand tuning can be a cumbersome process the efforts should be concentrated on the parts of the program which consumes most time. These parts can be identified by a profiler such as **gprof** or by measuring the time consumption directly with calls to **MPI_Wtime**.

5.4 Parallel Performance

The speed of a parallel message passing program is determined by various issues such as load balancing and communication overhead, but also by the performance of the code running on each processor. The improvement of this performance was discussed in the previous section.

The communication latency of 45 microseconds corresponds to 5400 clock cycles in each of which, we recall, up to four floating point operations can be performed. It is therefore important that the number of communications is not to big or, put in other words, that the individual processors have plenty of work to do with its local data before they need to communicate again.

The blocking of algorithms is a technique which improves data locality. On sequential programs it is used to reduce the number of cache misses, similarly, on parallel programs, it can be used to reduce the number of communications. This reduction is usually achieved at the expense of an increase in idle time of the processors. Blocking is heavily used in many parallel libraries.

A good idea which works on many parallel computers is to overlap communication with computation. Unfortunately it does not work well on JENSEN because communication takes up nearly 100% of the CPU. However it is possible to overlap a send and a receive on the same node, so bidirectional send/receives should be used if possible.

Synchronization overhead occurs if a processor has to wait to receive data. To reduce the probability of such delays, messages should be send as early as possible. If data from a send arrives to a processor before the corresponding receive is issued the data must first be buffered and later copied to the memory location designated by the receive. Whereas data can be transferred directly to the desired location, if the receive has already been issued when the message arrives. It is therefore also important to post receives as early as possible, but make sure that they are non-blocking. In MPI a non-blocking receive sets up the memory location to hold a message but does not wait for the transfer to take place, which means that computation which does not depend on the arrival of the message can be performed. When computation depending on the arrival of the message must be performed it should be preceded by a call to a wait for the corresponding receive. The waits for the non-blocking receives should be posted as late as possible to diminish the risk of synchronization overhead.

Chapter 6

Direct solution of sparse systems of linear equations

This chapter introduces some of the basic concepts which are used in Paper B, "Direct Solution of Sparse Systems of Linear Equations using Spiked Triangular Permutation and Matrix Modification".

We consider the problem of solving a system of linear equations, $Ax = b$, where the matrix $A \in R^{n \times n}$ is large, sparse and unsymmetric. This is the kind of systems which are solved when the sparse simplex method is applied to solve large sparse LPs. (For an example of such a matrix, see Figure 6.2.)

Sparse simplex-type algorithms typically spend the majority of their execution time solving such systems. The use of direct solution techniques is favoured by the simplex algorithm since a factorization can be reused for several iteration by appropriate use of updating schemes.

The rows of A can be reordered (permuted) without changing the solution of the system provided that b is reordered accordingly, similarly the columns of A can be reordered if the unknowns x are reordered accordingly. A permutation matrix (of order n) is a matrix $P \in R^{n \times n}$ such that every row and

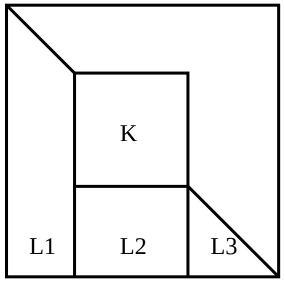


Figure 6.1: Non-zero pattern after the triangularizing reordering procedure. The matrix is divided in five parts K (the kernel matrix), L1, L2, L3 and the rest, which is all zero.

every column of P are unit vectors. Pre-multiplication by a permutation matrix corresponds to row permutation while post-multiplication by a permutation matrix corresponds to column permutation. I.e. if $[p_1\ p_2\ \ldots\ p_n]$ and $[q_1\ q_2\ \ldots\ q_n]$ are permutations of $[1\ 2\ \ldots\ n]$, then the matrix $\hat{A}$, defined by

$$\hat{A}_{p_i q_j} = A_{ij} \quad \forall i, j = 1, \ldots, n$$

can be written as

$$\hat{A} = PAQ^T$$

where P and Q^T are permutation matrices given by

$$P = [e_{p_1}\ \ldots\ e_{p_n}] \text{ and } Q = [e_{q_1}\ \ldots\ e_{q_n}]$$

Since the early days of computational linear programming [29] a triangularizing reordering procedure of the rows and columns has been used as part of the basis factorization procedure. This reordering results in a matrix, $\hat{A}$, in which the first s rows and last t columns fits a lower triangular form, i.e. such that $\forall i = 1, \ldots, s\ \forall j = i+1, \ldots, n : \hat{A}_{ij} = 0$ and $\forall j = n+1-t, \ldots, n\ \forall i = 1, \ldots, j-1 : \hat{A}_{ij} = 0$ as illustrated in Figure 6.1.

The reordered system can be solved for the first s unknowns by forward substitution, then a system of order $n-s-t$, known as the kernel, must be solved before the last t unknows are determined by forward substitution.

The triangularizing reordering is accomplished by the following procedure:

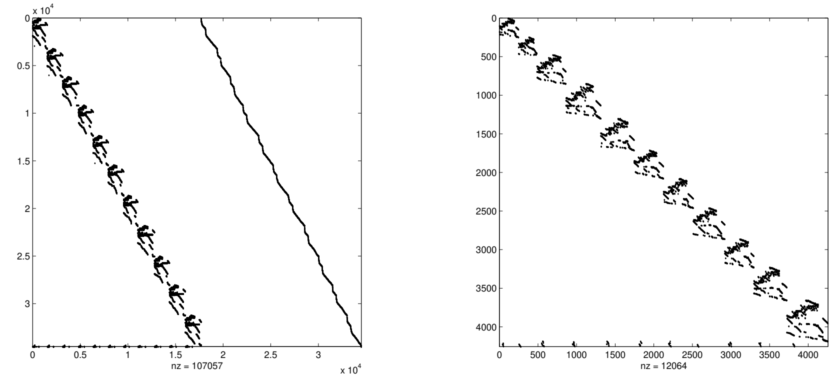


Figure 6.2: Non-zero pattern of the matrix world.bas ($n = 34506$ $nz = 107070$), an optimal basis for the LP problem world and its kernel world.bas.kernel ($m = 4256$ $nz = 12064$). World is a model for international energy planning.

1. Mark the whole matrix active.

2. If the active part contains a row singleton, unactivate this row and the corresponding column (i.e. the (only) active column which has a non zero element in the active part of the chosen row singleton) and permute them to the initial part of the matrix. Repeat this step as long as the active part contains a row singleton.

3. If a the active part contains a column singleton, unactivate this column and the corresponding row (i.e. the (only) active row which has a non zero element in the active part of the chosen column singleton) and permute them to the final part of the matrix. Repeat this step as long as the active part contains a column singleton.

An efficient implementation of this procedure needs fast column wise and row wise access to the non zero elements of the matrix.

As shown in Table 6.1, the size of the kernel is often much smaller than the size of the original matrix. The non-zero structure of the kernel of the matrix world is shown in Figure 6.2.

Problem	m	k
dbic1	43200	1665
lpl1	39951	4780
mod2	34774	4424
pds-40	66844	4772
world	34506	4256

Table 6.1: Kernel size k for five sparse real world optimal linear programming basis matrices.

6.1 The Block Triangular Form of a Sparse Matrix

A square matrix, B, is said to be reducible if there exists a permutation PBQ of B such that

$$\hat{B} = PBQ = \begin{bmatrix} B_{11} & 0 & \dots & 0 \\ B_{21} & B_{22} & \dots & 0 \\ . & . & . & . \\ B_{k1} & B_{k2} & \dots & B_{kk} \end{bmatrix}$$

with square diagonal blocks $B_{11}, ..., B_{kk}$ and $k \geq 2$.

If all the diagonal blocks are irreducible the matrix is said to be in block triangular form.

Many algorithms for solving sparse un-symmetric linear systems start by reordering the matrix into block triangular form. This form is desirable because the system $\hat{B}x = b$ can be solved by forward block substitution,

for $i = 1$ to k **do**
  $x_i = \hat{B}_{ii}^{-1} b_i$
  for $j = i + 1$ to k **do**
    $b_j = b_j - \hat{B}_{ji} x_i$
  end for
end for

which means that the factorization of the diagonal blocks can be done

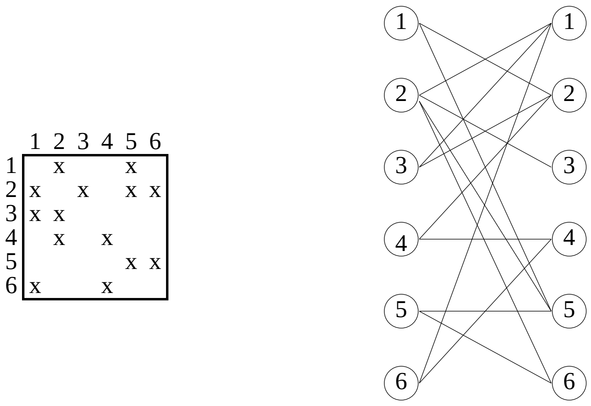


Figure 6.3: A sparse matrix and the associated bipartite graph.

independently (f.ex. in parallel) and that fill-in is limited to the diagonal blocks.

It turns out that the block triangular form of a matrix can be found by fast graph algorithms and that the block triangular form of a matrix is essentially unique. The permutation to block triangular form is generally done in two steps [11].

1. Find a reordering with non-zeros on the diagonal. Given an n by n matrix B, finding a permutation PBQ with non-zero elements along the diagonal is called finding a transversal. In graph theoretical terms this corresponds to finding a perfect matching in the bipartite graph $G = (N, E)$ with $N = \{r_1, r_2, ..., r_m\} \cup \{c_1, c_2, ...c_m\}$ and $E = \{\{r_i, c_j\} | B_{ij} \neq 0\}$ associated with B. (see Figure 6.3) As symmetric permutations preserve the diagonal elements we only need to consider row permutations PA when looking for a transversal.

   The algorithm can be seen as assigning rows to the columns such that every row is incident with the column to which it is assigned. If this is not possible the matrix is structurally singular. The following algorithm for finding a transversal

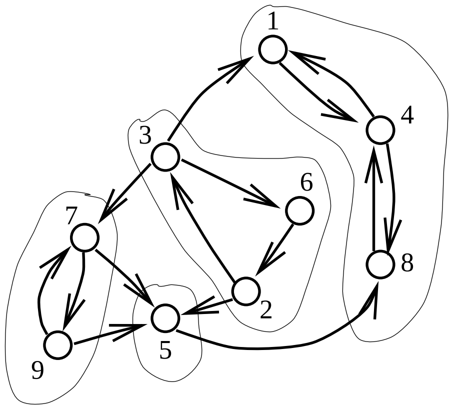
   $U = \{1, ..., n\}$
   for $k = 1$ to n **do**

   find $i_0 \in U$ $i_1, i_2, ..., i_l \notin U$ such that $B_{i_0 i_1}, B_{i_1 i_2}, ..., B_{i_s k} \neq 0$
   $p_k = j_s$
   for j=1 to s **do**
   $p_{i_j} = i_{j-1}$
   end for
   $U = U \setminus \{i_0\}$
   end for

   augments the number of rows in the transversal by one in each iteration. A simple assignment occurs in the k'th iteration if an unassigned row with a non-zero element in the k'th column exists, i.e. there exists an i_0 such that $B_{i_0 k} \neq 0$. If not, a depth-first search is initiated to find the chain $B_{i_0 i_1}, B_{i_1 i_2}, ..., B_{i_s k} \neq 0$ (which is called an augmenting path). The worst case complexity of this algorithm is $O(nt)$, where t is the number of non-zero elements of B. Complicated algorithms with better worst case complexities (as for example the algorithm by Hopcroft and Karp with worst case complexity $O(\sqrt{n}\, t)$) can be constructed, but the algorithm described above performs well in practice [11].

2. Find a symmetric reordering $Q\hat{B}Q^T$ of the matrix $\hat{B} = PB$, resulting from the first step, that puts the matrix into block triagular form. In graph theoretical terms this corresponds to finding the maximal strongly connected components and their topological order in the directed graph $G(V, A)$ with $V = \{1, ..., n\}$ and $A = \{(j, i) : \hat{B}_{ij} \neq 0\}$ associated with $\hat{B}$. A strongly connected component in a directed graph is a subset of the vertices such that a directed path exists between each ordered pair of vertices in the subset. It can be seen that the sub matrix corresponding to a strongly connected component is irreducible. By definition a path between two vertices from two different maximal strongly connected components can only exist in one of the directions (otherwise their union would constitute a strongly connected component contradicting the assumption that they were both maximal). The direction of the path defines a partial order of the maximal strongly connected components. The block triangular form is achieved by permuting the matrix such that the diagonal blocks corresponds to the maximal strongly connected components in their topological order.

The block triangular form and the intermediate permutation for a zero-free



	1	2	3	4	5	6	7	8	9
1	x		x	x					
2		x				x			
3		x	x						
4	x			x				x	
5		x			x		x		x
6			x			x			
7			x				x		x
8				x	x			x	
9							x		x

	2	3	6	7	9	5	1	4	8
2	x		x						
3	x	x							
6		x	x						
7		x		x	x				
9				x	x				
5	x			x	x	x			
1		x					x	x	
4							x	x	x
8						x		x	x

Figure 6.4: A small graph with a transversal, its associated directed graph with the connected components shown and the coressponding block triangular form

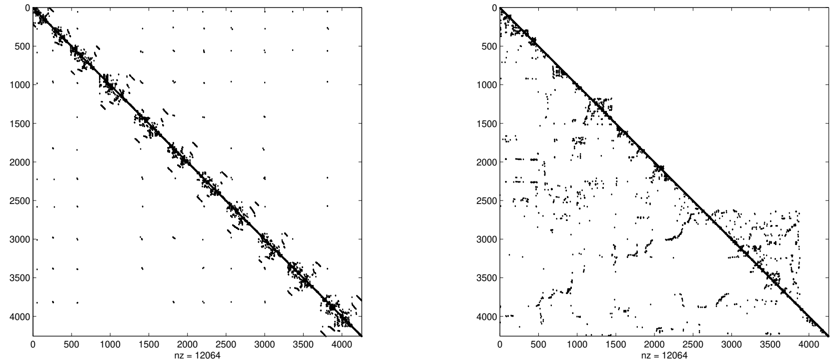


Figure 6.5: Permuted non-zero patterns of world.bas.kernel. 1) transversal 2) block triangular form

diagonal of world.bas.kernel are shown in figure 6.5.

Spiky, described in Paper B, does not permute the kernel to block triangular form prior to the factorization. This possibility was tested, using an implementation of block triangular ordering from the Harwell subroutine library, but it turned out that the ordering time exceeded the factorization time of the unordered matrices.

6.2 Exploitation of solution sparsity in sparse triangular solves

The execution time for solving large sparse LPs by a simplex algorithm is typically dominated by the solution of very sparse triangular systems [5]. Generally the right-hand sides of these systems and, more important, also the resulting solution vectors are sparse as well. A straightforward sparse implementation of the substitution algorithm as shown below solves the system in a time proportional to the number of non-zeros in the matrix.

The system $Lx = b$, where L is a sparse unit lower triangular matrix is solved by setting $x = b$ followed by sparse forward substitution.

Algorithm 1 Sparse forward substitution I

for $j = 1$ to n **do**
  for all nonzero positions i of column $L_{\bullet j}$ **do**
    $x_i = x_i - x_j * L_{ij}$
  end for
end for

However, if x is sparse many of the multiplications are unnecessary. The unnecessary multiplications are avoided by using the following variation of sparse forward substitution where we suppose that the nonzero positions $S \subseteq \{1, \ldots, n\}$ of x are known.

Algorithm 2 Sparse forward substitution II

for all $j \in S$ **do**
  for all nonzero positions i of column $L_{\bullet j}$ **do**
    $x_i = x_i - x_j * L_{ij}$
  end for
end for

The order in which the columns in S are considered in the outer loop does not necessarily have to be identical to the order that they appear in L. However all columns, k, contributing to the value of x_j (i.e. such that $L_{jk} \neq 0$) must be substituted before column j can be substituted. An order of the columns in S satisfying this requirement is called a topological order.

By using the second version of the sparse forward substitution the number of multiplications is decreased from $|L|$ to $|L_{\bullet S}|$. When the solution x is very sparse (i.e. the cardinality of S is small) the savings are big. But of course the number of operations required to determine S must also be taken into consideration. Fortunately, as we shall see, S can be determined in $O(|L_{\bullet S}|)$ time by depth-first search in the directed graph, $G(L)$, associated with L. $G(L)$ is defined as the directed graph with n vertices and an arc from vertex k to vertex i if and only if $L_{ik} \neq 0$.

Disregarding the possibility of numerical cancellation, $j \in S$ if and only if $j \in B \equiv \{i : b_i \neq 0\}$ or j can be reached from an element of B by a path in $G(L)$. This means that S can be determined by following all possible paths

emanating from the nodes in B, as illustrated by the following recursive depth-first search procedure.

Algorithm 3 Depth first search

1: $S = [\,]$
2: unmark all nodes
3: **for all** $s \in B$ **do**
4:   **if** s is unmarked **then** dfs(s)
5: **end for**
6:
7: subroutine dfs(s)
8: **begin**
9:   mark s
10:   **for all** edges $e = (s, w)$ such that w is unmarked **do**
11:     dfs(w)
12:   **end for**
13:   $S = [s, S]$
14: **end**

In line 13, when the current node s is placed in the front of the list of visited nodes all nodes which can be reached from s have already been visited and must therefore already appear in the list S. Therefore the list S contains the nodes of S in topological order after the execution of the depth-first search.

Each node in S is visited only once so the complexity of the search is only $O(|L_{\bullet S}|)$. As discussed in Paper B, the savings obtained by employing this technique can be substantial.

6.3 Parallel sparse triangular solves

Direct solution of an unsymmetric linear system, $Ax = b$, can be accomplished by first performing LU-factorization on A, giving $A = LU$, where L is lower and U upper triangular. The answer x is then obtained by solving a pair of triangular systems, first the forward solve $Ly = b$ and second the backward solve $Ux = y$. The factorization cost is generally much larger

1							
	1						
	*	1					
*		*	1				
	*	*		1			
	*				1		
		*				1	
*			*	*	*		1

Figure 6.6: Lower triangular matrix L

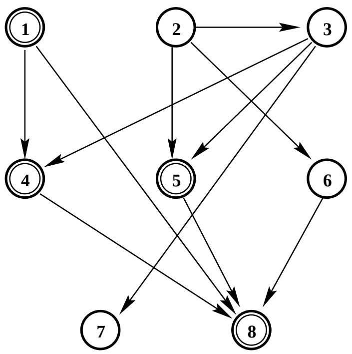


Figure 6.7: The associated graph $G(L)$

than the cost of the two triangular solves. However, a factorization is often reused for solving several different systems with the same matrix A, but with diffrent right hand sides. Thus in many contexts, including the simplex method, the aggregate cost of triangular solves can be much larger than the factorization cost.

We will consider two different approaches for exploiting parallelism when solving sparse triangular systems of linear equations. Assume that the system we want to solve, $Lx = b$, has been scaled to make L unit lower triangular.

The graph $G(L) = (V, A)$ associated with L will be useful in both cases. It is defined by

$$V = \{1, 2, ..., n\}$$
$$A = \{(j, i) : L_{ij} \neq 0 \text{ and } j < i\}$$

Parallel forward substitution

Define the level of a node of $G(L)$ to be the length of the longest path pointing into the node. If no arcs enters the node its level is 0.

The key observation is that the forward substitution of variables corresponding to nodes at the same level can be performed in parallel. But that all values of variables of lower levels must be determined before the value of a variable is found. Thus the level of the node with the highest level in the graph determines minimum number of step required by parallel forward substitution.

The partitioned inverse approach

The partitioned inverse approach for solving triangular systems in parallel was introduced by Alvarado, Pothen and Schreiber [2]. The idea is to obtain a representation of L^{-1} as a product of k unit lower matrices

$$L^{-1} = P_1 P_2 ... P_k$$

such that no fill is introduced, i.e. none of the P_i matrices has a non-zero element in a position where L has a zero. Often it will be the case that

k is smaller than the maximum level of the forward substitution approach leading to more parallelism.

The partitioned inverse approach requires preprocessing, which can only be recompensed if several solves with the same matrix are executed.

Conclusion

The danger of both approaches comes from the fact that very little work is required for a sparse triangular solve, such that a even a small communication overhead is likely to surpass the potential savings.

The solution approach presented in Paper B uses sparse triangular solves in both the factorization step as well as in the solve step. I have not tried to parallelize these sparse triangular solves.

Chapter 7

Conclusions

The four topics covered in this dissertation are all important areas of computational linear optimization, yet they are quite different from each other, which makes it difficult to draw a common conclusion of the four papers. Below I will list the conclusions and suggestions for further research within each of the areas. Further conclusions can be found within the individual papers.

A. "Automatic Reformulation for Mixed 0/1 Programs"

This paper is the result of a study of the literature on the subject and does not contain any new ideas nor implementations of the presented techniques. My aim has been to collect information from various sources and present it in a simple and succinct way. The resulting paper, I hope, can be used as a catalogue for the implementation of a MIP-solver.

B. "Direct Solution of Sparse Systems of Linear Equations using Spiked Triangular Permutation and Matrix Modification"

My work in this area started from a desire to parallelize the simplex method, which is a notoriously difficult task [21, 28]. Unless the number of rows of the problem is much larger than the number of columns, the solves with the basis matrix and its transposed represents a major part of the total execution time and is the bottleneck for

achieving good parallel performance. My main focus originally was to implement a parallel version of a basis inverse representation based on spiked triangular permutation and matrix modification, inspired by [16]. Some parts of the factorization step are inherently parallel while other parts were not considered for parallelization. These latter, sequential, parts quickly became bottlenecks for good parallel performance and my effort was shifted towards improving the sequential parts, where a new spiked triangular permutation heuristic decreased the execution time considerably.

The implementation was originally written in C, but was later encapsulated in a C++ class, which facilitated the development and experimentation with different strategies for the factorization of the Schur complement. The resulting implementation, Spiky, outperforms the package UMFPACK Version 2.2.1 by Davis and Duff [9] for most of the test-problems considered. I also ran the test-problems with the SuperLU package [10] which uses a column-column update approach and an initial column reordering based on the non-zero structure of A^TA. For these very sparse and unsymmetric matrices SuperLU is hopelessly inefficient compared to both Spiky and UMFPACK due to large amounts of fill-in.

The paper does not focus much on the parallel performance. By exploiting solution sparsity in the inherently parallel part of the factorization, the execution time of this part (and therefore also the obtainable speed-up) was decreased substantially. On the positive side it should be mentioned that the problems which takes the longest time to solve (i.e. the problems with the highest number of spikes) are also the problems which benefit the most from parallelization.

Spiky is competitive, robust and easy-to-use package for solving sparse unsymmtric systems of linear equations. However, as mentioned in the paper, it is not suited for all kinds of systems. An adaptive control of the accuracy spike threshold parameter could be incorporated in Spiky in order to always produce accurate solutions.

C. "Aggressive LP Reduction"

In this paper several experiments with new aggressive strategies for LP reduction were conducted. The experiments did not result in any major break through in LP reduction, although promising results were achieved for a few of the test-problems.

Some implementional details, to decrease the execution time, were also discussed in the paper. Nevertheless the paper does not include execution time comparisons with other LP reduction codes. My results were clearly superior compared with the published results in [3] and [17] both with respect to the number of reductions and the execution time (though off course it is difficult to compare timings from diffrent platforms) but inferior to the LP presolve of CPLEX. This makes me believe that CPLEX is using some of the same implementational techniques as presented in the paper, which are straight forward, but nevertheless have never been documented before.

The current implementation does only find possible reductions in the LP, it does not actually extract the reduced LP nor is the postsolve phase implemented. This means that some work is still required before the implementation can be used in practice.

D. "Formulating Linear Optimization Problems in C++"

Very recently, on the 11. October 2000, ILOG announced its new 'Concert Technology' (CT). In the description on the web, ILOG writes:

> ...
> **C++ model representations:** Modeling objects for representing mathematical programs without the use of matrices:
>
> - Variables
> - Constraints
> - Models and submodels
> - Model by stating constraints as expressions"
>
> ...

This might have been a description of FLOPC++. A closer examination of CT reveals that index controlled sum expressions are not supported. Such expressions can be constructed with CT by declaring an empty expression to which terms can be added in a loop. Certainly such model representation is a big step forward compared to writing code directly for the callable library of the solver, but does not enable a desirable purely declarative modelling notation within C++, as FLOPC++ does. While FLOPC++ is superior to CT in this respect, on some issues, ideas from CT can be used to improve

FLOPC++, for example by enabling column generation by stating columns as expressions corresponding to the dual problem.

The whole new aspect of object oriented optimization modelling which becomes possible with FLOPC++ opens up for new interesting research avenues to gain insight in how to best take advantage of the new possibilities. Model reuse, submodels, combination with specialized algorithms, modification of models and instances are all concepts which are facilitated by FLOPC++ and promises to become even more valuable tools for model building and algorithm development.

Bibliography

[1] R. C. Agarwal and F. G. Gustavson. Algorithm and architecture aspects of producing ESSL BLAS on Power2. In *PowerPC and POWER2: Technical Aspects of the New IBM RS/6000*, volume SA23-2737. IBM Corporation, 1998.

[2] F. L. Alvarado, A. Pothen, and R. Schreiber. Highly parallel sparse triangular solution. In J. A. George, J. R. Gilbert, and J. W. H. Liu, editors, *Graph Theory and Sparse Matrix Computation*. Springer-Verlag, 1993.

[3] E. D. Andersen and K. D. Andersen. Presolving in linear programming. *Mathematical Programming*, 71:221–245, 1995.

[4] E. D. Andersen and K. D. Andersen. A parallel interior-point algorithm for linear programming on a shared memory machine. Technical report, CORE, Catholic University of Louvain, 34 Voie Roman du Pais, B-1348 Louvain la Neuve, Belgium, January 1998.

[5] R. E. Bixby. Mip: Closing the gap, August 1999. Workshop presentation,THIRD SCANDINAVIAN WORKSHOP ON LINEAR PROGRAMMING Technical University of Denmark.

[6] R. E. Bixby, M. Fenelon, Z. Gu, E. Rothberg, and R. Wunderling. Mip: Theory and practice - closing the gap. *http://www.ilog.com/products/optimization/research/*, 1999.

[7] R. E. Bixby and A. Martin. Parallelizing the dual simplex method. Technical report, Center for Research on Parallel Computation, Rice University, Houston, 1997.

[8] T. F. Coleman, J. Czyzyk, C. Sun, M. Wagner, and S. J. Wright. pPCx: Parallel software for linear programming. *Proceedings of the Eighth SIAM Conference on Parallel Processing in Scientific Computing*, 1997.

[9] T. A. Davis and I. S. Duff. A combined unifrontal/multifrontal method for unsymmetric sparse matrices. *ACM Transactions on Mathematical Software*, 25(1):1–19, 1999.

[10] J. W. Demmel, J. R. Gilbert, and X. S. Li. *SuperLU Users' Guide*. `http://www.nersc.gov/~xiaoye/SuperLU/`.

[11] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for sparse matrices*. Oxford University Press, New York, 1987.

[12] R. Fourer. Linear programming software survey. *ORMS Today*, August 1999.

[13] R. Fourer, D. M. Gay, and B. W. Kernighan. A modeling language for mathematical programming. *Management Science*, 36:519–554, 1990.

[14] K. Fukuda and T. Terlaky. Criss-cross methods: A fresh view on pivot algorithms. *Mathematical Programming*, 79(1–3):369–395, 1997.

[15] K. Fukuda and T. Terlaky. On the existence of a short admissible pivot sequence for feasibility and linear optimization problems. Technical report, Delft University of Technology, January 1999.

[16] J. Gondzio. On exploiting original problem data in the inverse representation of linear programming bases. *ORSA Journal on Computing*, 6:193–206, 1994.

[17] J. Gondzio. Presolve analysis of linear programs prior to applying an interior point method. *INFORMS Journal on Computing*, 9(1):73–91, 1997.

[18] H. J. Greenberg. *A Computer-Assisted Analysis System for Mathematical Programming Models and Solutions: A User's Guide for ANALYZE*. Kluwer Academic Publishers, Boston, MA, 1993.

[19] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI. Portable parallel programming with the message-passing interface*. The MIT Press, 1996.

[20] M. M. Güntzer, D. Jungnickel, and M. Leclerc. Efficient algorithms for the clearing of interbank payments. *European Journal of Opeartional Research*, 106:212–219, 1998.

[21] E. L. Johnson and G. L. Nemhauser. Recent developments and future directions in mathematical programming. *IBM Systems Journal*, 31:79–93, 1992.

[22] G. Karypis, A. Gupta, and V. Kumar. A parallel formulation of interior point algorithms. Technical Report 94-20, Computer Science Department, University of Minnesota, MN 55455, 1994.

[23] I. J. Lustig and E. Rothberg. Gigaflops in linear programming. *Operations Research Letters*, 18:157–165, 1996.

[24] I. Maros and G. Mitra. Investigating the sparse simplex algorithm on a distributed memory multiprocessor. *Parallel Computing*, 26:151–170, 2000.

[25] Richard Kipp Martin. *Large Scale Linear and Integer Optimization.* Kluwer Academic Publishers, 1999.

[26] B. A. McCarl and T. H. Spreen. Applied mathematical programming using algebraic systems. `http://agrinet.tamu.edu/mccarl/regbook.htm`.

[27] C. Meszaros. *The efficient implementation of interior point methods for linear programming and their applications.* PhD thesis, Eotvos Lorand University of Sciences, 1996.

[28] National coordination offfice for computing, information and communication. The 1997 petaflops algorithms workshop summary report. `http://www.ccic.gov/cicrd/pca-wg/pal97.html`.

[29] W. Orchard-Hayes. *Advanced Linear-Programming Computing Techniques.* McGraw-Hill Book Company, 1968.

[30] H. P. Williams. *Model Building in Mathematical Programming. Third edition.* Wiley, 1993.

[31] S. Wright. *Primal-dual Interior-point Methods.* SIAM, 1997.

Appendix A

Reformulation for General Mixed 0-1 Linear Programs

Paper A: Automatic Reformulation for General Mixed 0-1 Linear Programs

Based on a version published in
Investigação Operacional 19 (1999) p. 27-41.

Abstract

The efficiency of solving mixed 0-1 linear programs by linear programming based branch-and-bound algorithms depends heavily on the formulation of the problem. In seeking a better formulation, automatic reformulation employs a range of different techniques for obtaining an equivalent formulation of a given pure or mixed integer programming problem, such that the new formulation has a tighter LP-relaxation than the original formulation. This paper surveys the use of automatic reformulation techniques for general mixed 0-1 linear programs.

A.1 Introduction

Many practical problems including several well known combinatorial optimization problems can be formulated as mixed 0-1 linear programs. However some mixed 0-1 linear programming formulations can be impossible to solve in reasonable time by using standard linear programming based branch-and-bound algorithms. On the other hand it is known that the convex hull of the set of feasible solutions to a mixed 0-1 linear program is a polyhedron. Therefore a mixed 0-1 linear program could in principle be solved by an LP-algorithm if this polyhedral description was known. Unfortunately it is, in general, NP-hard to find this polyhedral description. The goal of automatic reformulation is to find, in a short time, an equivalent formulation with an LP-relaxation which approximates the convex hull of the set of feasible solutions as well as possible.

The automatic reformulation approach as described in [13] consists of two phases: preprocessing and cut generation. The basic ideas of preprocessing are described in [11] where another related technique called probing is also described. The second phase of automatic reformulation looks for valid inequalities which are violated by some feasible point of the LP-relaxation. Adding such inequalities, also called cuts, to the formulation tightens the corresponding LP-relaxation.

Combinatorial optimization problems formulated as mixed 0-1 linear programs do often possess a special structure which can be used to characterize families of strong valid inequalities. The identification of such families together with so-called separation algorithms to find cuts among its members have recently led to substantial improvements of exact algorithms for several combinatorial optimization problems. An excellent overview of polyhedral techniques in combinatorial optimization can be found in Aardal and Hoesel [1]. Polyhedral cuts are problem specific by their nature and will not be considered here. Instead we will review the efforts to extend the success of reformulation to general mixed 0-1 linear programming.

The rest of the paper is organized as follows. In section two the mixed 0-1 linear programming problem is defined and some basic notation and concepts are introduced. Furthermore the standard LP-based branch-and-bound approach for solving this type of problem is explained to show the reader the importance of the formulation of the problem when using this kind of algorithm to solve it. Preprocessing is the subject of section three. Section four discusses lifting of cuts and 3 different types of cuts which do not assume any underlying structure of the problem: Gomory cuts, disjunctive cuts and knapsack cuts.

Throughout the paper we will use the following notation. If $S \subseteq R^n$ and $p(x)$ is a proposition defined on R^n then $S_{p(x)}$ denotes $\{x \in S : p(x)\}$.

A.2 Background and preliminaries.

A mixed 0-1 linear program, M, is an optimization problem of the form

$$\mathrm{M}: \quad \mathrm{maximize}\{cx : x \in S\}$$

where c is a row vector in R^n and S is a subset of $\{0,1\}^p \times R_+^{n-p}$ specified by a system of linear inequalities, i.e.

$$S = \{x \in \{0,1\}^p \times R_+^{n-p} : Ax \leq b\}$$

where $A \in R^{m\times n}$ and b is a column vector in R^m. The variables $x_1, ..., x_p$ are called the binary variables and the variables $x_{p+1}, ..., x_n$ are called the continuous variables. We use J to denote the index set of the variables $\{1, ..., n\}$, $D = \{1, ..., p\}$ the index set of the binary variables and $C = \{p+1, ..., n\}$ the index set of the continuous variables.

Generally the same set of feasible solutions, S, can be described by many different systems of inequalities.

Example 1 *Let* $A = \begin{bmatrix} 2 & 1 \\ -1 & 2 \end{bmatrix}$ *and* $b = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$ *and let* $\hat{A} = \begin{bmatrix} 0 & 1 \end{bmatrix}$ *and* $\hat{b} = [0]$. *Since*

$$\{x \in \{0,1\}^2 : Ax \leq b\} = \{x \in \{0,1\}^2 : \hat{A}x \leq \hat{b}\}$$

$Ax \leq b$ *and* $\hat{A}x \leq \hat{b}$ *provide two different formulations of the same set.*

Clearly the two problems $M_1 \equiv \mathrm{maximize}\{cx : x \in \{0,1\}^p \times R_+^{n-p}, Ax \leq b\}$ and $M_2 \equiv \mathrm{maximize}\{cx : x \in \{0,1\}^p \times R_+^{n-p}, \hat{A}x \leq \hat{b}\}$ are equivalent if $\{x \in \{0,1\}^p \times R_+^{n-p} : Ax \leq b\} = \{x \in \{0,1\}^p \times R_+^{n-p} : \hat{A}x \leq \hat{b}\}$. In this case we say that M_2 is a **reformulation** of M_1.

Given a particular formulation of M, we obtain a linear program, LPR_M, by relaxing the integrality requirement of the variables $x_1, ..., x_p$,

$$\text{LPR}_M : \quad \text{maximize}\{cx : x \in P\}$$

where

$$P = \{x \in [0,1]^p \times R_+^{n-p} : Ax \leq b\}$$

We will refer to LPR_M as the LP-relaxation of M (having in mind that LPR_M depends upon the formulation of M).

The optimal object function values of the two programs will be denoted z_M and z_{LPR_M}.

$$z_M = max\{cx : x \in S\} \quad z_{LPR_M} = max\{cx : x \in P\}$$

We note that $S \subseteq P$ since $S = P \cap (\{0,1\}^p \times R_+^{n-p})$. It follows directly that $z_M \leq z_{LPR_M}$. The difference $z_M - z_{LPR_M}$ is called the **integrality gap**. If an optimal solution,x^*, to LPR_M, happens to be in S, x^* is also an optimal solution to M and the integrality gap is zero .

Two different equivalent formulations of a mixed 0-1 linear program will generally have different LP-relaxations, since $\{x \in \{0,1\}^p \times R_+^{n-p} : Ax \leq b\}$ $= \{x \in \{0,1\}^p \times R_+^{n-p} : \hat{A}x \leq \hat{b}\}$ does not imply that $P_1 = \{x \in [0,1]^p \times R_+^{n-p} : Ax \leq b\}$ is equal to $P_2 = \{x \in [0,1]^p \times R_+^{n-p} : \hat{A}x \leq \hat{b}\}$. If $P_2 \subset P_1$ we say that the second formulation of the problem has a tighter LP-relaxation than the first.

Example 2 *(Example 1 continued) The two equivalent formulations have different LP-relaxations, in particular*

$$\{x \in [0,1]^3 : \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} x \leq [0]\} \subset \{x \in [0,1]^3 : \begin{bmatrix} 2 & 1 & 1 \\ -1 & 2 & -1 \end{bmatrix} x \leq \begin{bmatrix} 3 \\ 0 \end{bmatrix}\}$$

so the LP-relaxation of the second formulation is the tightest of the two.

Definition A.1 *$\alpha x \leq \beta$ is a* **valid inequality** *(for S) if $\alpha x \leq \beta$ for all $x \in S$.*

Definition A.2 *A* **cut** *is a valid inequality $\alpha x \leq \beta$ such that there exists an $\hat{x} \in P$ with $\alpha\hat{x} > \beta$.*

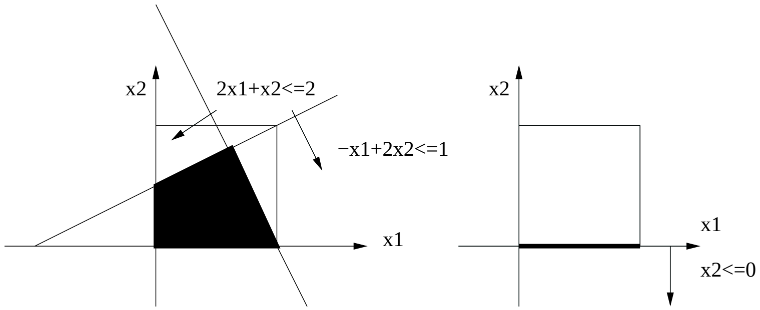


Figure A.1: LP-relaxations of the two alternative formulations

Definition A.3 *A point $\hat{x} \in P$ is* **eliminated** *by the cut $\alpha x \leq \beta$ if $\alpha\hat{x} > \beta$*

We will generally be interested in cuts which eliminate x^*, where x^* is an optimal solution to LPR_M.

We observe that if $\alpha x \leq \beta$ is a cut then

$$\begin{bmatrix} A \\ \alpha \end{bmatrix} x \leq \begin{bmatrix} b \\ \beta \end{bmatrix}$$

is a reformulation of M and that the new formulation has a tighter LP-relaxation.

Since we are maximizing a linear function, an optimal solution to M is also optimal to maximize$\{cx : x \in \text{convh}(S)\}$. The convex hull of feasible solutions to M, convh(S), is contained in P for any formulation of M. Standard results from polyhedral theory [10, 12] imply that convh(S) is a polyhedron which means that it can be described by a system of linear inequalities. In other words, there exist an $\tilde{A}$ and a $\tilde{b}$ such that

$$P = \{x \in [0,1]^p \times R_+^{n-p} : \tilde{A}x \leq \tilde{b}\} = \text{convh}(S)$$

This formulation of the feasible region, S, is the tightest possible formulation and the optimization of any linear objective function over S can be done by solving a linear program with the set of feasible solutions defined by P.

A.2.1 LP based branch-and-bound

When solving a mixed 0-1 program by a linear programming based branch-and-bound algorithm, the formulation of the problem being solved is crucial for the efficiency. In this section we shall see why this is the case.

Linear programming based branch-and-bound algorithms for solving M are based on two basic facts.

For any $F \subseteq S$

1. $\max\{cx : x \in F\} = \max\{\max\{cx : x \in F_{x_j=0}\}, \max\{cx : x \in F_{x_j=1}\}\}$
   This fact allow us to recursively divide the search for an optimal solution to M into smaller subproblems.

2. If $\max\{cx : x \in F\} < cx$ for some $x \in S$ then F does not contain an optimal solution to M.
   In this case, the subset F can be eliminated from further search. If we can find an upper bound UB on $\max\{cx : x \in F\}$ such that $UB < cx$ for some $x \in S$, it is not necessary to obtain the exact value of $\max\{cx : x \in F\}$ to be able to eliminate F from the search for an optimal solution. The object function value of the LP-relaxation of $\max\{cx : x \in F\}$ provides such an upper bound.

Figure 1 sketches a branch-and-bound algorithm for solving mixed 0-1 programs based on these principles.

During the search, the best feasible solution found so far (the **incumbent** solution) is stored in x and its object function value in z. When the algorithm terminates, x is an optimal solution of M and z is the optimal object function value, i.e. cx.

At any point of the execution of the algorithm, the elements of ActNodes are called the active nodes. P is called the root node. When examining a node, two things can happen: 1) the node is fathomed (eliminated from further search) 2) the node is substituted by two child nodes dividing the set of feasible solutions of the node by fixing a binary variable to each of its two possible values. A node is fathomed whenever a) the solution of its LP-relaxation belongs to S b) the LP-relaxation is infeasible or c)


```
z = −∞
ActNodes = {P}
while ActNodes ≠ ∅ do
  pick an element R of ActNodes
  ActNodes = ActNodes \ {R}
  (z*, x*) = (max{cx : x ∈ R}, argmax{cx : x ∈ R})
  if z* > z then
    if x* ∈ S then
      (z, x) = (z*, x*)
    else
      pick a 1 ≤ j ≤ p such that 0 < x*_j < 1
      ActNodes = ActNodes ∪ {R_{x_j=0}, R_{x_j=1}}
    end if
  end if
end while
```

Figure A.2: Linear programming based branch-and-bound algorithm for solving M.

the upper bound given by the objective function value of its LP-relaxation is smaller than the object function value of the incumbent solution.

The tree of nodes generated by the algorithm rooted by the root node is called the search tree. The **level** of a node is defined as the depth in the search tree or equivalently the number of fixed binary variables. Since binary variables can only be fixed once, the level of a node in the search tree cannot exceed p. If no nodes of level smaller than p could be fathomed, the search tree would be a complete binary tree of depth p implying the solution of $2^{p+1} - 1$ linear programs; an impossible task for problems of any realistic size.

In order to keep the search tree to a manageable size we must be able to fathom nodes of levels smaller than p (the smaller, the better). This can be achieved by having both good lower and upper bounds. The lower bound is the objective function value of the best feasible solution found so far. For this reason many algorithms include heuristics for finding good feasible solutions. The quality of the upper bound of each node can be measured by the integrality gap of the node. As we have seen, the integrality gap depends on the formulation of the problem. While the integrality gap of a descendant node can be bigger than the integrality gap of the root node, it is still true that a formulation of the root node with a tight LP relaxation will also lead to tight LP relaxations of the descendant nodes and thereby to good upper bounds.

The strategy for picking the next active node R of ActNodes to examine and the strategy for choosing a binary variable x_j for generating child nodes also affect the size of the search tree. However, no matter what strategies are used, a tighter formulation of M will still decrease the size of the search tree and thereby the execution time of the algorithm.

A.3 Preprocessing.

By using only simple bounds on the variables and a single inequality of the formulation of M at a time, preprocessing techniques can often find a much improved formulation of M without requiring a big computational effort.

A.3.1 Successive strengthening of bounds

l_j is called a lower and u_j an upper bound on x_j if $l_j \leq x_j \leq u_j$ for all $x \in S$. A lower bound is said to be stronger than another if it is higher. An upper bound is said to be stronger than another if it is lower.

The availability of strong bounds on the variables of MIP is important to many of the reformulation techniques.

It follows directly from the definition of M that $l_j = 0$, $u_j = 1$ for $1 \leq j \leq p$ and $l_j = 0$, $u_j = \infty$ for $p+1 \leq j \leq n$ are lower and upper bounds on x.

By isolating a variable in an inequality we might be able to strengthen the bounds on this variable. If $a_{ij} > 0$, isolating x_j in the i'th equality, we get

$$x_j \leq (b_i - \sum_{k \in J \backslash j} a_{ik} x_k)/a_{ij}$$

Any upper bound on the right hand side is an upper bound on x_j. Finding the lowest possible upper bound on the right hand side would involve the maximization of the right hand side subject to $x \in S$; a problem just as

difficult as the original problem. If $S \subseteq \{x \in R^n : l \leq x \leq u\}$ (that is, if l and u are lower and upper bounds on x) maximizing the right hand side subject to $x \in \{x \in R^n : l \leq x \leq u\}$ will provide an alternative upper bound on x_j which is generally not as strong, but is much faster to find. This maximum is achieved simply by setting each variable x_k for which a_{ik} is positive equal to its lower bound and each variable x_k for which a_{ik} is negative equal to its upper bound.

If $a_{ij} < 0$,

$$x_j \geq (b_i - \sum_{k \in J \backslash \{j\}} a_{ik} x_k)/a_{ij}$$

and a lower bound on x_j can be found in a similar fashion.

We note that if we are able to strengthen a bound on a binary variable, the value of this variable can be fixed.

A.3.2 Coefficient reduction.

In this subsection we shall drop the row indices when referring to a single unspecified inequality of $Ax \leq b$ and let $ax \leq b$ be a generic inequality of the formulation of M . In other words we let $ax \leq b$ represent $\sum_{k \in J} a_{ik} x_k \leq b_i$ for some $1 \leq i \leq m$.

Theorem A.1 *(Coefficient reduction) Let x_j be a binary variable. An inequality $ax \leq b$ of the formulation of M can be replaced by*

$$(a_j - \delta)x_j + \sum_{k \neq j} a_k x_k \leq b - \delta \tag{A.1}$$

without affecting the set of feasible solutions if $0 \leq \delta \leq b - max\{\sum_{k \neq j} a_k x_k : x \in S_{x_j=0}\}$

Proof (A.1) is valid for $S_{x_j=1}$ for any value of δ. For $x \in S_{x_j=0}$, (A.1) becomes $\sum_{k \neq j} a_k x_k \leq b - \delta$ which is valid for $S_{x_j=0}$ for $\delta \leq b - \max\{\sum_{k \neq j} a_k x_k : x \in S_{x_j=0}\}$ This shows that (A.1) is valid for S for $\delta \leq b - \max\{\sum_{k \neq j} a_k x_k : x \in S_{x_j=0}\}$. Now let $\bar{S}$ be the set of feasible solutions after $ax \leq b$ has been replaced by (A.1). $ax \leq b$ is valid for $\bar{S}_{x_j=1}$ for any value of δ. For $x \in \bar{S}_{x_j=0}$ we have $\sum_{k \neq j} a_k x_k \leq b - \delta$ which implies $\sum_{k \neq j} a_k x_k \leq b$ and therefore that $ax \leq b$ is valid for $\bar{S}_{x_j=0}$ for $\delta \geq 0$. $\square$

The coefficient reduction reformulation suggested by the theorem can be seen to tighten the LP-relaxation if $\delta > 0$. The maximum tightening is achieved for $\delta = b - \max\{\sum_{k\neq j} a_k x_k : x \in S_{x_j=0}\}$. Obtaining the exact value of $\max\{\sum_{k\neq j} a_k x_k : x \in S_{x_j=0}\}$ is practically as difficult as solving M itself. Instead we can use an upper bound on $\max\{\sum_{k\neq j} a_k x_k : x \in S_{x_j=0}\}$. Such an upper bound can be found quickly by using the bounds on the individual variables. Combining the bounds on the variables with a few of the remaining inequalities of M can often strengthen this upper bound without an excessive computational effort.

Example 3 *Consider the constraint*

$$10x_1 + 4x_2 + 4x_3 \leq 10$$

of M. Suppose that x_1 is a binary variable and that $0 \leq x_2, x_3 \leq 1$ are valid bounds. Then $8x_1 + 4x_2 + 4x_3 \leq 8$ can replace the old constraint since the upper bound of 8 on $\max\{4x_2 + 4x_3 : x \in S_{x_1=0}\}$ is easily obtained from the bounds on x_2 and x_3. Now suppose that

$$x_2 + x_3 \leq 1$$

is another constraint in M. Using this "side constraint" we see that $\max\{4x_2 + 4x_3 : x \in S_{x_1=0}\} \leq 4$ so the original constraint can be replaced by

$$4x_1 + 4x_2 + 4x_3 \leq 4$$

which in turn is equivalent to $x_1 + x_2 + x_3 \leq 1$

The theorem can easily be extended to simultaneous reduction of coefficients. Recall that D is the index set of the binary variables and C is the index set of the continuos variables.

Theorem A.2 *An inequality $ax \leq b$ of the formulation of M can be replaced by*

$$\sum_{j\in D}(a_j - \delta_j)x_j + \sum_{j\in C} a_j x_j \leq b - \sum_{j\in D}\delta_j \tag{A.2}$$

where $0 \leq \delta_j \leq \max\{0, b - \max\{\sum_{k\neq j} a_k x_k : x \in S_{x_j=0}\}\}$ without affecting the set of feasible solutions.

Theorem A.3 *(Subset coefficient reduction) Let x_j be a binary variable. And let $ax \leq b$ be an inequality of the formulation of M. If $K \subset J$ such that $j \notin K$ and $\{i : a_i < 0\} \subseteq K$ then*

$$(a_j - \delta)x_j + \sum_{k\in K} a_k x_k \leq b - \delta \tag{A.3}$$

is a valid inequality when $0 \leq \delta \leq b - \max\{\sum_{k\in K} a_k x_k : x \in S_{x_j=0}\}$

Proof The inequality $a_j x_j + \sum_{k\in K} a_k x_k \leq b$ is implied by $ax \leq b$. Now the theorem follows by applying coefficient reduction on the implied inequality. □

Example 4 *Let $x_1 \in \{0,1\}$ and $0 \leq x_2, x_3, x_4, x_5 \leq 1$, and consider the inequality*

$$3x_1 + x_2 + x_3 + x_4 + x_5 \leq 4$$

By choosing $K = \{2,3,4\}$ we obtain the valid inequality

$$2x_1 + x_2 + x_3 + x_4 \leq 3$$

which is not available by simple coefficient reduction.

A.4 Cut generation.

A.4.1 Lifting of cuts.

Algorithms which combine branch-and-bound and automatic reformulation are called branch-and-cut algorithms. Automatic reformulation can in principle be applied at any node of an LP based branch-and-bound algorithm. However the fact that cuts generated at one node are not generally valid at other nodes makes this approach computationally unattractive. Lifting provides a technique to overcome this drawback by obtaining a globally valid inequality from an inequality valid only at a given node.

Theorem A.4 *Let $\alpha x \leq \beta$ be a valid inequality for $S_{x_j=0}$. Then $\tilde{\alpha}_j x_j + \sum_{k\neq j} \alpha_k x_k \leq \beta$ is a valid inequality for S if $\tilde{\alpha}_j \leq \beta - \max\{\sum_{k\neq j} \alpha_k x_k : x \in S_{x_j=1}\}$.*

Proof To show that the inequality is valid for $S = S_{x_j=0} \cup S_{x_j=1}$ we must show that it is valid for both $S_{x_j=0}$ and $S_{x_j=1}$. $\tilde{\alpha_j} x_j + \sum_{k \neq j} \alpha_k x_k \leq \beta$ is valid for $S_{x_j=0}$ for any value of $\tilde{\alpha_j}$. For $x \in S_{x_j=1}$, $\tilde{\alpha_j} x_j + \sum_{k \neq j} \alpha_k x_k = \tilde{\alpha_j} + \sum_{k \neq j} \alpha_k x_k \leq \beta$ which is valid for $S_{x_j=1}$ if $\tilde{\alpha_j} \leq \beta - \max\{\sum_{k \neq j} \alpha_k x_k : x \in S_{x_j=1}\}$ $\square$

The strongest inequality is obtained for $\tilde{\alpha_j} = \beta - \max\{\sum_{k \neq j} \alpha_k x_k : x \in S_{x_j=1}\}$. But normally we will be satisfied with a value obtained from a relaxation of the maximization problem.

It is also possible to aply the theorem to obtain a globally valid inequality from a valid inequality for a node where one of the binary variables has been fixed at one. If $\alpha x \leq \beta$ is a valid inequality for $S_{x_j=1}$ we can obtain a valid inequality for S by applying the theorem with x_j complemented, since $\alpha x \leq \beta$ is valid for $S_{\hat{x_j}=0}$ where $\hat{x_j} = 1 - x_j$.

If a valid inequality is given for a subset of S with more than one variable fixed, a valid inequality for S is obtained by sequentially lifting each of the fixed variables. Generally, different valid inequalities can be obtained by varying the order in which the fixed variables are considered for lifting, as illustrated by the following example.

Example 5 *Let* $S = \{x \in \{0,1\}^4 : 13x_1 + 11x_2 + 11x_3 + 10x_4 \leq 32\}$. *We see that* $x_2 + x_3 \leq 1$ *is a valid inequality for* $S_{x_1=1,x_4=0} = \{(x_2,x_3) \in \{0,1\}^2 : 11x_2 + 11x_3 \leq 19\}$

1. *By lifting* $x_2 + x_3 \leq 1$ *for* $\hat{x}_1$ *we get* $\tilde{\alpha_1} = 1 - max\{x_2 + x_3 : S_{\hat{x_1}=1,x_4=0}\} = 1 - 2 = -1$ *so the inequality becomes* $-(1 - x_1) + x_2 + x_3 \leq 1$ *or* $x_1 + x_2 + x_3 \leq 2$. *Lifting this for* x_4 *we get* $\tilde{\alpha_4} = 2 - max\{x_1 + x_2 + x_3 : S_{x_4=1}\} = 2 - 2 = 0$ *so the valid inequality for* S *becomes*
$$x_1 + x_2 + x_3 \leq 2$$
2. *Lifting* $x_2 + x_3 \leq 1$ *for* x_4 *first, we get* $\tilde{\alpha_4} = 1 - max\{x_2 + x_3 : S_{\hat{x}_1=0,x_4=1}\} = 1 - 0 = 1$ *so the inequality becomes* $x_2 + x_3 + x_4 \leq 1$. *Lifting this for* $\hat{x}_1$ *we get* $\tilde{\alpha_1} = 1 - max\{x_2 + x_3 + x_4 : S_{\hat{x}_1=1}\} = 1 - 3 = -1$ *so the valid inequality for* S *becomes* $-2(1 - x_1) + x_2 + x_3 + x_4 \leq 1$ *or*
$$2x_1 + x_2 + x_3 + x_4 \leq 3$$

A.4.2 Gomory cuts.

Gomory [8] pioneered the idea of adding valid inequalities to a given integer linear programming formulation in order to tighten the LP-relaxation until an integer optimal solution is obtained. A Gomory cut, violated by the current solution of the LP-relaxation, can be obtained for each fractional variable within it. Although the Gomory cuts are not immediately applicable to mixed 0-1 linear programs, the idea can be extended to this type of problem [3].

Let x^* be the optimal solution of the LP-relaxation of the current formulation of M and let B be the corresponding basis-matrix. Let x_j be a binary variable such that x_j^* is fractional. Let $y = e_i^T B^{-1} N$ (i.e. the i'th row of the optimal simplex-tableau) where i is the position of x_j in the basis. The i'th row of the optimal simplex-tableau reads

$$x_j + \sum_{k \in D_N} y_k x_k + \sum_{k \in C_N} y_k x_k = x_j^*$$

where D_N denotes the non-basic binary variables and C_N denotes the non-basic continuous variables including slack-variables. Let $\lfloor a \rfloor$ denote the largest integer less than or equal to a and let f(a) denote $a - \lfloor a \rfloor$. Then the simplex-tableau rows can be rewritten as

$$x_j + \sum_{k \in D_N} \lfloor y_k \rfloor x_k + \sum_{k \in D_N} f(y_k) x_k + \sum_{k \in C_N} y_k x_k = x_j^*$$

The first two terms on the left side are integer valued for any feasible solutions to M. So we get

$$\sum_{k \in D_N} f(y_k) x_k + \sum_{k \in C_N} y_k x_k \equiv x_j^* \pmod 1$$

This relation remains true if we subtract the integer $\sum_{k \in \hat{D_N}} x_k$ on the left side, for some subset $\hat{D_N}$ of D_N

$$\sum_{k \in D_N \setminus \hat{D_N}} f(y_k) x_k + \sum_{k \in \hat{D_N}} (f(y_k) - 1) x_k + \sum_{k \in C_N} y_k x_k \equiv x_j^* \pmod 1$$

For this to be true, at least one of the following two conditions must be true. 1) the sum of the non-negative left hand side terms is greater than or equal to x_j^*, or 2) the sum of the negative left hand side terms is less than or equal to $x_j^* - 1$. To realize this, assume that both conditions are false, i.e that the sum of the non-negative left hand sides is strictly less than x_j^* and the sum of the negative terms on the left hand side is strictly greater than $x_j^* - 1$. It follows the total sum of the left hand side terms is strictly between $x_j^* - 1$ and x_j^* and therefore cannot be equal to x_j^* modulo 1.

We get

$$\sum_{k \in D_N \setminus \hat{D_N}} f(y_k)x_k + \sum_{k \in C_N^+} y_k x_k \geq x_j^*$$

or

$$\sum_{k \in \hat{D_N}} (f(y_k) - 1)x_k + \sum_{k \in C_N^-} y_k x_k \leq x_j^* - 1$$

where $C_N^+ = \{k \in C_N : y_k > 0\}$ and $C_N^- = \{k \in C_N : y_k < 0\}$.

Multiplying the last inequality by -1, dividing both inequalities by their right hand side and finally adding them yields

$$\sum_{k \in D_N \setminus \hat{D_N}} \frac{f(y_k)}{x_j^*} x_k + \sum_{k \in \hat{D_N}} \frac{1 - f(y_k)}{1 - x_j^*} x_k + \sum_{k \in C_N^+} \frac{y_k}{x_j^*} x_k - \sum_{k \in C_N^-} \frac{y_k}{1 - x_j^*} x_k \geq 1 \tag{A.4}$$

The left hand side is zero for the optimal solution to the LP-relaxation, since all the variables are non basic, hence the inequality is a cut.

For each $k \in D_N$ we can choose to put k in $\hat{D_N}$ or not. The strongest possible cut is obtained when the coefficients on the left hand side are as small as possible, so k should be put in $\hat{D_N}$ if $\frac{1-f(y_k)}{1-x_j^*}$ is less than $\frac{f(y_k)}{x_j^*}$. Thus it can be seen that the strongest cut is obtained for

$$\hat{D_N} = \{k \in D_N : f(y_k) > x_j^*\}$$

An important property of Gomory cuts for mixed 0-1 linear programming is that they can easily be lifted. Suppose, for example, that (A.4) is a Gomory-cut for $S_{x_l=0}$ where x_l is a binary variable. By setting $D_N = D_N \cup \{l\}$ and $y_l = e_i^T B^{-1} A_{.l}$ the inequality (A.4) becomes valid for S. This can be seen by considering the LP-relaxation of M with the constraint $x_l \leq 0$ added. By pivoting the slack variable of this constraint into the basis, x_l becomes non basic and the validity of the proposed cut follows.

Variables fixed at one can be treated similarly. Note that this property is not inherited by mixed integer linear programs because an integer variable which has been used to branch on might not continue to be non basic deeper down in the branch-and-bound search tree.

A.4.3 Disjunctive cuts.

Disjunctive cuts [2, 5] arise by considering valid inequalities for the set of feasible solutions of so called disjunctive programming relaxations of M.

Definition A.4 *Let d be a row vector in Z^p. Define*

$$P(d) = convh(P_{dx_D \leq 0} \cup P_{dx_D \geq 1}) \tag{A.5}$$

and

$$DPR: \quad maximize\ \{cx : x \in P(d)\} \tag{A.6}$$

DPR is called a **disjunctive programming relaxation** *of M.*

To see that $S \subseteq P(d)$ and thereby that DPR is indeed a relaxation of M, we note that dx_D is integer valued for all $x \in S$ so $S = S_{dx_D \leq 0} \cup S_{dx_D \geq 1}$.

$$S = S_{dx_D \leq 0} \cup S_{dx_D \geq 1} \subseteq P_{dx_D \leq 0} \cup P_{dx_D \geq 1} \subseteq convh(P_{dx_D \leq 0} \cup P_{dx_D \geq 1}) = P(d)$$

We also note that $P(d) \subseteq P$, since both $P_{dx_D \leq 0}$ and $P_{dx_D \geq 1}$ are subsets of P and P is a convex set.

The usefulness of $P(d)$ lies in the fact that valid inequalities for $P(d)$ can easily be characterized.

Lemma A.1 *Let $P = \{x \in R_+^n : Ax \leq b\}$.*

$\alpha x \leq \beta$ is valid for P $\Leftrightarrow$ $\exists \lambda \geq 0 : \alpha \leq \lambda A$ and $\lambda b \leq \beta$

Proof
"$\Leftarrow$": Assume λ is such that $\alpha \leq \lambda A$ and $\lambda b \leq \beta$. For $x \in P$ we get $\alpha x \leq \lambda A x$, since $x \geq 0$ and $\lambda A x \leq \lambda b$, since $Ax \leq b$ and $\lambda \geq 0$. I.e. we

have $\alpha x \leq \lambda A x \leq \lambda b \leq \beta$.
"$\Rightarrow$":Suppose that $\alpha x \leq \beta$ is valid for P. Duality gives us that

$$\max\{\alpha x : Ax \leq b, x \geq 0\} = \min\{yb : yA \geq \alpha, y \geq 0\}$$

Let x^*, y^* be optimal solutions to the two LP problems above. Then $y^* \geq 0$, $y^* A \geq \alpha$ and $y^* b = \alpha x^* \leq \beta$ since $x^* \in P$. Hence $\lambda = y^*$ fulfills the conditions. □

Theorem A.5 *Let $\alpha \in R^n$ and $\beta \in R$. The inequality $\alpha x \leq \beta$ is valid for $P(d)$ if and only if there exist row vectors $u, v \geq 0$ and scalars $u_0, v_0 \geq 0$ such that*

$$\alpha \leq u\hat{A} + u_0[d \;\; 0_{n-p}] \tag{A.7}$$
$$u\hat{b} \leq \beta \tag{A.8}$$
$$\alpha \leq v\hat{A} - v_0[d \;\; 0_{n-p}] \tag{A.9}$$
$$v\hat{b} \leq \beta + v_0 \tag{A.10}$$

where $\hat{A} = \begin{bmatrix} A & \\ I_{p\times p} & 0_{p\times n-p} \end{bmatrix}$ *and* $\hat{b} = \begin{bmatrix} b \\ 1_p \end{bmatrix}$

Proof First note that

$$P_{dx_D \leq 0} = \{x \in R^n_+ : \begin{bmatrix} \hat{A} & \\ d & 0_{n-p} \end{bmatrix} x \leq \begin{bmatrix} \hat{b} \\ 0 \end{bmatrix}\}$$

and

$$P_{dx_D \geq 1} = \{x \in R^n_+ : \begin{bmatrix} \hat{A} & \\ -d & 0_{n-p} \end{bmatrix} x \leq \begin{bmatrix} \hat{b} \\ -1 \end{bmatrix}\}$$

and that $\alpha x \leq \beta$ is valid for $P(d)$ if and only if it is valid for $P_{dx_D \leq 0}$ and for $P_{dx_D \geq 1}$. Now the theorem follows by applying the lemma for $P_{dx_D \leq 0}$ and $P_{dx_D \geq 1}$. □

For a given d the theorem characterizes a family of valid inequalities for S. We would like to find a cut, $\alpha x \leq \beta$, from this family of valid inequalities which eliminates x^* i.e. such that $\alpha x^* > \beta$. To this end we consider the LP

$$\text{DISJ}: \quad \text{maximize}\{\alpha x^* - \beta : u, u_0, v, v_0 \geq 0, (A.7), (A.8), (A.9), (A.10)\}$$

which clearly has a solution with strictly positive objective function value if and only if such a cut exists. However if such a cut exists, DISJ is unbounded as any positive multiple of the cut is also a cut. Therefore, in order to use DISJ to find a cut we additionally impose the normalization constraint $\sum_{j\in J} |\alpha_j| \leq 1$. This normalization constraint can easily be linearized ($\sum_{j\in J} \bar{\alpha}_j \leq 1$, $\forall_{j\in J}\ \bar{\alpha}_j \geq \alpha_j$, $\forall_{j\in J}\ \bar{\alpha}_j \geq -\alpha_j$) so that DISJ remains an LP.

Now suppose that we have found a disjunctive cut, $\alpha x \leq \beta$. We will show how to strengthen this cut by possibly increasing some of the α_j's. Note that for $j \in D$,

$$\alpha_j = \min\{uAe_j + u_0 d_j, vAe_j - v_0 d_j\}. \tag{A.11}$$

Each d_j can be individually adjusted in order to increase the value of α_j if possible. Note that the new cut satisfies all the constraints of DISJ (without the normalization constraint) and is therefore valid.

A.4.4 Knapsack cuts.

Knapsack cuts [7] arise by considering a single inequality at a time. They can be expected to be particularly effective when the problem is sparse.

From a given inequality $a_D x_D + a_C x_C \leq b$ from the formulation of M, we can obtain a valid inequality, $a_D x_D \leq \bar{b}$, in the binary variables only, where $\bar{b}$ is a lower bound on $b - a_C x_C$. Such a bound can be found quickly from the bounds on x_C. This inequality can be turned into a knapsack constraint by complementing the variables for which a_j is negative. For example $\{x_1, x_2 \in \{0,1\} : -2x_1 + x_2 \leq 0\}$ becomes $\{x_1, x_2 \in \{0,1\} : 2(1-x_1) + x_2 \leq 2\}$ and then $\{\bar{x_1}, x_2 \in \{0,1\} : 2\bar{x_1} + x_2 \leq 2\}$ introducing $\bar{x_1} = 1 - x_1$. In the following we will assume that

$$a_D x_D \leq \bar{b} \tag{A.12}$$

is a knapsack constraint obtained from M in the way described above. A cut obtained with complemented variables can readily be expressed in the original variables.

Definition A.5 *The pair U, t with $U \subset D$ and $t \in D \setminus U$ is called a $(1,k)$-* **configuration** *with respect to $a_D x_D \leq \bar{b}$ if*

$$\sum_{j\in U} a_j \leq \bar{b}$$

and

$$a_t + \sum_{j \in Q} a_j > \bar{b}$$

for any subset Q of U with k elements and

$$a_t + \sum_{j \in R} a_j \leq \bar{b}$$

for any subset R of U with $k-1$ elements. If $|U| = k$ then $U \cup \{t\}$ is called a **minimal cover***.*

Theorem A.6 *Let U, t be a $(1,k)$-configuration with respect to $a_D x_D \leq \bar{b}$, $k \leq r \leq |U|$ and let T be a subset of U of cardinality r. Then*

$$(r-k+1)x_t + \sum_{j \in T} x_j \leq r \tag{A.13}$$

is a valid inequality.

Proof Clearly $\sum_{j \in T} x_j \leq k-1$ is valid for $S_{x_t=1}$. Lifting this inequality for $\hat{x}_t$ we obtain the lifting coefficient $\alpha_t = k-1-r$, since r is an upper bound on $\max\{\sum_{j \in T} x_j : x \in S_{\hat{x}_t=1}\}$. This shows that $(k-1-r)(1-x_t) + \sum_{j \in T} x_j \leq k-1$ is valid for S. $\square$

Example 6 *Consider the knapsack constraint*

$$13x_1 + 11x_2 + 11x_3 + 10x_4 \leq 32$$

$U = \{2,3,4\}$, $t = 1$ is a $(1,2)$-configuration with respect to this constraint since $a_2 + a_3 + a_4 = 11 + 11 + 10 \leq 32$ and $a_1 + a_{j_1} + a_{j_2} > 32$ for any $j_1 \neq j_2 \in U$. Choosing $r = 3$ and $T = \{2,3,4\}$ we obtain the valid inequality

$$2x_1 + x_2 + x_3 + x_4 \leq 3$$

In example 5 we saw how this inequality can also be generated by lifting the inequality $x_1 + x_2 \leq 1$ which is a valid minimal cover inequality for $S_{x_1=1,x_4=0}$.

To generate a cut from this family of valid inequalities for the knapsack constraint, we need a procedure to identify a minimal cover inequality violated by x^*. Such a procedure is provided by the following knapsack problem

$$\text{MC}: \quad \min\{\sum_{j \in D}(1-x_j^*)s_j : \sum_{j \in D} a_j s_j > \bar{b}, s \in \{0,1\}^p\}$$

It can be seen that the set of indices, j, such that $s_j = 1$ in the optimal solution to MC is a minimal cover inequality violated by x^* if and only if the optimal objective function value is less than one.

Another approach for generating knapsack cuts is presented in [7]. For any $u \in R_+^p, u_0 \in R_+$

$$\sum_{j \in D} \lfloor u_0 a_j + u_j \rfloor x_j \leq \lfloor u_0 \bar{b} + \sum_{j \in D} u_j \rfloor \tag{A.14}$$

is a valid inequality for S. Note that (A.14) is obtained by rounding down the coefficients of a linear combination of the source knapsack constraint and the upper bound constraints of the binary variables (i.e. it is a Chvatal-Gomory cut). A separation procedure (i.e. a procedure for finding a valid inequality violated by x^*) based on this family of valid inequalities can be stated as

$$\text{SP}: \quad \max\{\sum_{j \in D} \lfloor u_0 a_j + u_j \rfloor x_j^* - \lfloor u_0 \bar{b} + \sum_{j \in D} u_j \rfloor : u, u_0 \geq 0\}$$

This problem can be stated as an integer linear program. As in section 4.3 we need to include a normalization constraint to make SP bounded. We can use $\lfloor u_0 \bar{b} + \sum_{j \in D} u_j \rfloor = K$ for some integer K as a normalization constraint. Then the separation problem becomes

$$\begin{aligned} \text{maximize} \quad & \alpha x^* \\ \text{subject to} \quad & \alpha \leq u_0 a + u, \\ & u_0 \bar{b} + \sum_{j \in D} u_j - 1 + \epsilon \leq K, \\ & u, u_0 \geq 0, \\ & \alpha_j \text{ integer for } 1 \leq j \leq p. \end{aligned}$$

where ϵ is some small value (for example $\epsilon = 0.001$). Instead of solving this integer linear program we can use the relaxation obtained be replacing the integrality requirement on α_j by $\alpha_j \geq 1$ for all j's.

A.5 Conclusion.

Inspired by the recent success of branch-and-cut methods for several hard combinatorial optimization problems, there has been a renewed interest in the application of these methods to general mixed integer and 0-1 linear programs. Each of the automatic reformulation techniques presented in this paper has the potential to strengthen the LP-relaxation, leading to considerably reduced execution times for solving general mixed 0-1 linear programs, but the actual gains which are obtained are highly problem dependent. It therefore seems advisable to combine several of these techniques in order to achieve a robust algorithm for solving general mixed 0-1 linear programs.

Bibliography

[1] K. Aardal and S. van Hoesel. Polyhedral techniques in combinatorial optimization. Technical Report UU-CS-1997, Department of Computer Science, Utrecht University, 1997.

[2] E. Balas, S. Ceria, and G. Cornuejols. Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. *Management Science*, 42, 1996.

[3] E. Balas, S. Ceria, G. Cornuejols, and N. Natraj. Gomory cuts revisited. *OR Letters*, 19, 1996.

[4] R. Bixby, W. Cook, A. Cox, and E. Lee. Parallel mixed integer programming. Technical Report CRPC-TR95554, Center for Research on Parallel Computation, June 1995.

[5] S. Ceria and J. Soares. Disjunctive cut generation for mixed 0-1 programs: duality and lifting. Technical report, Columbia University, New York, March 1997.

[6] H. Crowder, E. L. Johnson, and M. W. Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31:803–834, 1983.

[7] F. Glover, H. D. Sherali, and Y. Lee. Generating cuts from surrogate constraint analysis for zero-one and mulitiple choice programming. In *INFORMS*. INFORMS, April 1995.

[8] R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958.

[9] R. K. Martin and L. Schrage. Subset coefficient reduction cuts for 0/1 mixed-integer programming. *Operations Research*, 33:505–526, 1985.

[10] G .L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley, New York, 1988.

[11] M. W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6:445–454, 1994.

[12] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley, Chichester, 1986.

[13] T. Van Roy and L. Wolsey. Solving mixed integer programming problems using automatic reformulation. *Operations Research*, 35:45–57, 1987.

Appendix B

Direct solution of sparse unsymmetric systems using spiked triangular permutation and matrix modification

Paper B:

Direct solution of sparse unsymmetric systems using spiked triangular permutation and matrix modification

Abstract

Solution of large sparse unsymmetric systems of linear equations is a major computational task in many applications. An implementation, called Spiky, of a direct method for solving such systems is presented. The method is based on a matrix modification approach applied to a spiked triangular permutation of the system. We are particularly interested in systems arising when large sparse linear programming problems are solved by a simplex algorithm. The factorization consists of 3 steps. 1) a reordering of the matrix, such that only a small number, s, of spike columns reach above the diagonal is determined. 2) a block LU factorization of an augmented system is computed. This involves the solution of a sparse triangular system with s right-hand sides. Solution sparsity is exploited in the sparse triangular solves of the block LU factorization. 3) a factorization of the Schur complement matrix, of order s, is computed. The idea of this factorization method comes from Gondzio [11], but has been improved in several ways. Most importantly: a) A new fast reordering algorithm is used. b) Sparsity of the Schur complement is exploited.

B.1 Introduction

Spiky is an implementation of a direct method for solving large sparse unsymmetric systems of linear equations. Though it can be used to solve any systems of this kind, we are particularly interested in systems arising when large sparse linear programs are solved by a simplex-like method.

Simplex-like algorithms require the solution of, at least, two systems of linear equations in each iteration. One of the systems involves B, the other B^T, where B, the current basis matrix, is a square non-singular submatrix of the coefficient matrix. Efficient implementation of the sparse simplex algorithm relies heavily on the routines which are used for solving these systems, since this typically is the most time consuming part of the algorithm (problems where the number of columns is much bigger than the number of rows is an important exception). In each iteration B is modified by replacing one of its columns by another column of A. It is important to exploit this fact by developing update schemes which allow the new slightly modified systems to be solved much faster than otherwise would be possible. The term basis inverse representation refers to the set of subroutines which are used to solve the sequence of related systems in a simplex-like LP solver. We use the term factorization to denote any kind of preparation which allows for faster solution of systems involving B and B^T.

For large sparse problems it is mandatory that the basis inverse representation exploits the sparsity. This is usually done by solving the systems using a sparse LU factorization of B. A reordering of B is applied to preserve the sparsity of the factors and the numerical stability. The reordering used in most modern LP codes is a dynamic scheme due to Markowitz [16],[19], where the next pivot element during the LU factorization is chosen so as to minimize the product of the number of elements in the corresponding column and row of the remaining active matrix. To enhance the stability of this procedure, pivot elements are rejected if the ratio of its absolute value to the maximum absolute value in its column is smaller than a certain threshold value.

To avoid computing a new LU-factorization in each iteration, different update schemes can be employed to accommodate for the column exchanges in B. The Schur complement update [2] and the block-LU update [6] share the advantage that neither of the LU factors are modified. In fact these

updates can be used in connection with any type of basis inverse representation, since they treat the solves with the most recently factorized basis matrix as black box routines. Let B_0 be the latest basis for which a factorization has been computed, suppose that the current basis B is obtained from B_0 by k column exchanges substituting the original columns of B_0 in positions $l_1, l_2, ..., l_k$ by $A_{p_1}, A_{p_2}, ..., A_{p_k}$. The system $Bx = b$ is equivalent with the augmented system

$$\begin{bmatrix} B_0 & A_P \\ V^T & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_P \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix},$$

where $V = [e_{l_1} e_{l_2} ... e_{l_k}]$ (e_i denotes the i'th unitvector) and $A_P = [A_{l_1} A_{l_2} ... A_{l_k}]$. The solution x is obtained from the solution to the augmented system by setting $x = x_0 + Vx_P$. The solution to the augmented system can be obtained by the three steps

1. solve $B_0 \tilde{x} = b$
2. solve $Sx_P = V^T \tilde{x}$
3. $x_0 = \tilde{x} - Yx_P$

where $Y = B_0^{-1} A_P$ and the Schur complement $S = V^T Y$. Essentially the fact that $B_0 \tilde{x} = b$ is easy to solve (because the factorization of B_0 has already been computed) is exploited to solve the system, $Bx = b$, fast. The only additional work consist in solving the $k \times k$ system $Sx_P = V^T \tilde{x}$ and matrix-vector multiply with Y of dimension $m \times k$. Of course this scheme is only advantageous when k is much smaller than m. When k becomes too big a refactorization must be performed.

The factorization step of our basis inverse representation is directly inspired by this scheme. We have seen that advantage can be taken if a system, which only differs in a small number of columns, happens to be fast to solve because the cost of factorization has already been paid. When factorizing a matrix B we can actively look for a matrix which only differs from B in a few columns and is easily invertible, not because it has already been factorized, but because it possesses an easily invertible structure. Such a structure could be lower triangularity or a block diagonal form. The principle behind this factorization scheme is called matrix modification. The matrix modification approach can be viewed as a partitioning of an equivalent augmented system. A discussion of these techniques can be found in Duff et. al. [5], Chapter 11.

Gondzio [11] used the matrix modification approach in connection with a spiked triangular reordering of the basis matrix for the basis inverse representation of an implementation of a simplex-like algorithm. Spiky follows the principles outlined by Gondzio, but we have included several improvements. A new faster heuristic for finding a spiked triangular reordering is used instead of the SPK1 heuristic [18]. This change is crucial to avoid that the reordering time dominates the total execution time. Furthermore we make sure that sparsity is exploited in every part of the factorization. Solution sparsity is exploited when solving the sparse triangular systems occurring during the block LU-factorization. This was also done in [11] but the paper is lacking in detail as regards the implementation. Our implementation of this feature follows the ideas outlined in [10] where implementation tricks to avoid that the symbolic computations dominate the solve are discussed. In [11] a dense factorization of the Schur complement was used. Experiments in this paper show that the Schur complement matrix is often sufficiently sparse to benefit from a sparse implementation of its factorization and this possibility has been included in Spiky.

In the next section we give an overview of the method and a numerical example is given in section 3. In the following three sections the 3 main steps of the factorization, ordering, computation of the block LU-factorization of the augmented system and LU-factorization of the Schur complement are discussed. Section 7 discus the solve. The numerical stability of the method is discussed in section 8. Section 9 contains computational experiments. A parallel implementation is discussed in section 10. Finally the conclusion is presented in section 11.

B.2 Overview of the method.

We wish to solve the systems $Bx = b$ and $B^T y = d$, where B is a large sparse non-singular $m \times m$ real matrix.

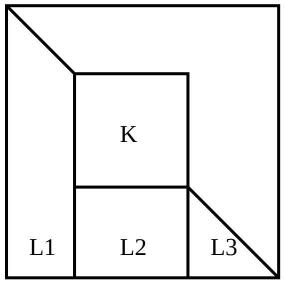


Figure B.1: Non-zero pattern after initial triangularizing reordering procedure. The matrix is divided in five parts K (the kernel matrix), L1, L2, L3 and the rest, which is all zero.

B.2.1 Identifying the kernel.

By applying a triangularizing reordering procedure to B the dimension of the matrix which must be factorized can often be reduced considerably. This is particularly true for linear programming basis matrices which typically contain many singleton columns. We refer to the kernel, K, of a matrix, B, as what remains after recursively removing column and row singletons from the matrix. The kernel of matrix and the associated reordering is illustrated in Figure 1.

A factorization of the kernel matrix K can be exploited to solve systems involving B, via the factorization

$$
\begin{aligned}
PBQ &= \begin{bmatrix} L1_1 & 0 & 0 \\ L1_2 & K & 0 \\ L1_3 & L2 & L3 \end{bmatrix} \\
&= \begin{bmatrix} L1_1 & 0 & 0 \\ L1_2 & I & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} I & 0 & 0 \\ 0 & K & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ L1_3 & L2 & L3 \end{bmatrix}
\end{aligned}
$$

Spiky includes a triangularizing reordering procedure following the principles described by Orchard-Hays [17] as its first step.

B.2.2 Factorization of the kernel.

From now on, let B be the kernel of the original matrix.

Let M_j be defined by $M_j = min\{i : B_{ij} \neq 0\}$. The columns of B can be partitioned into three sets: 1) the columns, which reaches above the diagonal, i.e such that $M_j < j$. 2) the columns, which matches the diagonal, i.e such that $M_j = j$. 3) the columns, which lies entirely under the diagonal, i.e such that $M_j > j$.

Definition B.1 *B is in spiked lower triangular form if no columns of B lies entirely under the diagonal. In this case, the columns, which reaches above the diagonal are called spike columns.*

This definition might seem counterintuitive, since by this definition any upper triangular matrix is in spiked lower triangular form (but possibly with a large number of spike columns). When the number of spikes is small, the permuted form meets the expectations of its name (in this case, the matrix is essentially lower triangular, but with a few spikes reaching above the diagonal), and is the kind of permutation we are looking for. Now we assume that B has been reordered into a spiked lower triangular form with few spikes. Heuristics for finding such reorderings are discussed in section B.4.

Let B_0 be the matrix obtained from B by replacing the columns which reach above the diagonal, say $B_{l_1}, ..., B_{l_s}$, by corresponding unit vectors matching the diagonal, $e_{l_1}, ..., e_{l_s}$, i.e.

$$B_0 = B + (V - C)V^T$$

where

$$V = [e_{l_1} e_{l_2} ... e_{l_s}], \quad C = [B_{l_1} B_{l_2} ... B_{l_s}] \quad \text{(i.e. } C = BV)$$

By construction, B_0, is lower triangular, non-singular and no new nonzeros have been introduced, so systems involving B_0 are easy to solve.

We note that $Bx = b$ can be solved by solving the augmented system

$$\begin{bmatrix} B_0 & C \\ V^T & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_C \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}$$

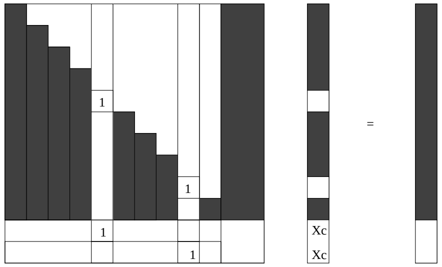


Figure B.2: Illustration of the augmented system

and setting $x = x_0 + Vx_C$. This augmented system is illustrated in Figure 2.

Using the block factorization

$$\begin{bmatrix} B_0 & 0 \\ V^T & -S \end{bmatrix} \begin{bmatrix} I & Y \\ 0 & I \end{bmatrix} = \begin{bmatrix} B_0 & C \\ V^T & 0 \end{bmatrix},$$

where $S = V^T B_0^{-1} C$ and $Y = B_0^{-1} C$, we see that the solution to (B.1) can be found by making one solve with B_0, one solve with the Schur complement matrix, S, of dimension $s \times s$ and a matrix-vector multiply by performing the following three steps.

1. solve $B_0 \tilde{x} = b$
2. solve $Sx_c = V^T \tilde{x}$
3. $x_0 = \tilde{x} - Yx_c$

Alternatively by using the block LU factorization

$$\begin{bmatrix} I & 0 \\ V^T B_0^{-1} & I \end{bmatrix} \begin{bmatrix} B_0 & C \\ 0 & -S \end{bmatrix} = \begin{bmatrix} B_0 & C \\ V^T & 0 \end{bmatrix},$$

the third step can be replaced by

- 3. solve $B_0 x_0 = b - C x_c$

The solution to the transposed system $B^T y = d$ can be found by solving the system

$$\begin{bmatrix} B_0^T & V \\ C^T & 0 \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix} = \begin{bmatrix} d \\ V^T d \end{bmatrix}$$

which can be done with one solve with S^T and two with B_0^T

1. solve $B_0^T \tilde{y} = d$
2. solve $S^T z = C^T \tilde{y} - V^T d$
3. solve $B_0^T y = d - Vz$

Again one of the solves with B_0 can be replaced by matrix-vector multiply with Y, if this matrix is retained. (By performing the solve $S^T z = Y^T d - V^T d$ and then the solve $B_0^T y = d - Vz$)

For this approach to basis inverse representation to be competitive with the traditional LU-factorization it is necessary that B_0 is easily invertible (meaning that $B_0 x = b$ and $B_0^T y = d$ can be solved efficiently) and that the number of spikes s can be kept low. Experience shows that most real world sparse linear programming bases allow nearly triangular permutations, i.e. they can be permuted to lower triangular matrices except for a small number of spike columns. Thus the easily invertible form we use for B_0 is a lower triangular matrix. (Another easily invertible form which seems particularly desirable in a parallel setting is the block diagonal form.)

In the description above we have avoided the use of permutation matrices, implicitly assuming that B is already in spiked triangular form. In practice, of course, we factorize a spiked triangular permutation of B.

By factorization we understand the part of the computation of the solution to $Bx = b$ or $B^T y = d$ which can be performed before the right-hand side is known (and therefore can be reused for different right-hand sides). The factorization part of the method described above consists of the following 3 steps: 1) **Ordering.** Identify a, hopefully small, number of spike columns and the associated ordering which puts the remaining matrix in lower triangular form. 2) **Block LU-factorization of the augmented system.**

Compute $S = V^T B_0^{-1} C$, by performing s independent sparse triangular solves. 3) **Factorization of the Schur complement matrix S.**

After this factorization has been obtained any number of systems involving B or B^T can be solved fast as described above.

B.3 A numerical example

B.3.1 Factorization

Consider the system $Ax = b$ where

$$A = \begin{bmatrix} 1 & 3 & 1 & 2 & \\ & -2 & 1 & 2 & 5 \\ 2 & & 1 & & \\ -1 & 4 & 1 & 2 & \\ 1 & & 1 & 2 & 2 \end{bmatrix}$$

and b becomes known after the factorization.

Permuting A to spiked triangular form we obtain

$$B = PAQ = \begin{bmatrix} 2 & & & & 1 \\ 1 & 2 & & 2 & 1 \\ -1 & & 4 & 2 & 1 \\ & 5 & -2 & 2 & 1 \\ 1 & & 3 & 2 & 1 \end{bmatrix}$$

with $P = \begin{bmatrix} e_5 & e_4 & e_1 & e_3 & e_2 \end{bmatrix}$ and $Q = \begin{bmatrix} e_1 & e_3 & e_5 & e_4 & e_2 \end{bmatrix}^T$

The fourth and fifth columns are spikes so

$$V^T = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

We substitute the spike columns in B by unit vectors to obtain B_0 and collect the spikes in the matrix C

$$B_0 = \begin{bmatrix} 2 & & & & \\ 1 & 2 & & & \\ -1 & & 4 & & \\ & 5 & -2 & 1 & \\ 1 & & 3 & & 1 \end{bmatrix} \qquad C = \begin{bmatrix} & 1 \\ 2 & 1 \\ 2 & 1 \\ 2 & 1 \\ 2 & 1 \end{bmatrix}$$

We perform a sparse triangular solve with B_0 for each of the two spikes to obtain Y where the rows in the positions corresponding to the spike columns are selected to get the Schur complement S

$$Y = B_0^{-1}C = \begin{bmatrix} 0 & 0.5 \\ 1 & 0.25 \\ 0.5 & 0.375 \\ -2 & 0.5 \\ 0.5 & -0.625 \end{bmatrix} \qquad S = V^T Y = \begin{bmatrix} -2 & 0.5 \\ 0.5 & -0.625 \end{bmatrix}$$

The factorization is now complete except for the LU factorization of S which we do not show.

B.3.2 Solve

We will use the factorization computed above to solve $Ay = b$ where

$$b = \begin{bmatrix} 18 \\ 32 \\ 5 \\ 18 \\ 22 \end{bmatrix}$$

The factorization is given in terms of the reordered matrix B and the associated permutation matrices P and Q. We note that $y = A^{-1}b = QQ^TA^{-1}P^TPb = QB^{-1}Pb$, so if we solve $Bx = Pb$ the solution y is given by $y = Qx$.

The first step is a solve with B_0

$$Pb = \begin{bmatrix} 5 \\ 22 \\ 18 \\ 32 \\ 18 \end{bmatrix} \qquad \tilde{x} = B_0^{-1}Pb = \begin{bmatrix} 2.5 \\ 9.75 \\ 5.125 \\ -6.5 \\ 0.125 \end{bmatrix}$$

In the second step, the relevant elements of $\tilde{x}$ are picked and a solve with S is performed

$$V^T\tilde{x} = \begin{bmatrix} -6.5 \\ 0.125 \end{bmatrix} \qquad x_c = S^{-1}V^T\tilde{x} = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$$

The third step involves a matrix-vector product Cx_c and another solve with B_0

$$Pb - Cx_c = \begin{bmatrix} 2 \\ 11 \\ 7 \\ 21 \\ 7 \end{bmatrix} \qquad x_0 = B_0^{-1}(Pb - Cx_c) = \begin{bmatrix} 1 \\ 5 \\ 2 \\ 2 \\ 0 \\ 0 \end{bmatrix}$$

Finally x is assembled from the solution to the augmented system and y is obtained by permuting x

$$x = x_0 + Vx_c = \begin{bmatrix} 1 \\ 5 \\ 2 \\ 4 \\ 3 \end{bmatrix} \qquad y = Qx = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

B.4 Ordering algorithm

We discuss how to reorder the basis matrix into a form which is lower triangular except for a number of columns, called spikes, reaching above the diagonal. We wish to keep the number of spikes in the resulting spiked triangular form as low as possible. It is an NP-hard problem to find a permutation with the minimum number of spikes, but efficient heuristics are available.

B.4.1 Spiked triangular form

Ordering algorithms to obtain a spiked triangular permutation of B have been around in linear programming for a long time. In 1971, Hellerman and Rarick [12] introduced the P^3 heuristic for permuting a sparse unsymmetric matrix into spiked triangular form with a, hopefully, small number of spikes. When Gaussian elimination without pivoting is applied to a spiked triangular matrix, the fill is confined to the spike columns. Their algorithm was used to determine a reordering of the basis matrix, before the LU factors are computed by Gaussian elimination, with the purpose of improving the sparsity of the resulting factors. However the static reorderings of both the P^3 and the P^4 [13] heuristics, which simply consists in applying P^3 to each of the diagonal blocks of the block triangular form of the matrix, suffer from severe numerical instability. In fact a breakdown of the Gaussian elimination due to a structurally zero pivot is not only possible but also frequent. The modified heuristic P^5 [7] was designed to avoid structurally zero pivots, but small pivots leading to numerical instability were not avoided. To enhance numerical stability, the static orderings obtained by the above mentioned heuristics can be combined with a dynamic column interchange strategy [1], which however increases the fill-in. In a comparison with the Markowitz strategy the authors generally discourage the use of spiked triangular orderings prior to LU factorization.

Here the situation is different. Since each spike column is replaced by a unit vector with the one element on the diagonal, structural breakdown can not occur. Furthermore since an augmented system is considered there is no need for interchanging spike columns. Numerical difficulties resulting from small pivot elements in the spike columns are mitigated to the Schur complement S where pivoting can be applied without destroying the sparsity of B_0. This, unfortunately, as will be further discussed in section 8, does not mean that the method is numerically stable. But nevertheless accurate solutions are obtained for most of the test problems considered.

Fletcher and Hall [8] showed how several well known heuristics for obtaining a spiked triangular permutation of B can be viewed as different tie-break rules of the same basic procedure. Conceptually the process can be divided into two stages. First B is row and column permuted into a lower block triangular form with fully dense **rectangular** diagonal blocks. After completing the first stage, it is a simple matter to find a spiked triangular matrix. Traverse the columns of the reordered matrix H in order; whenever a column rising above the diagonal is encountered this column is removed and put on a stack for later insertion as a spike and the remaining columns are shuffled to the left. Columns matching the diagonal are left in place. When a column is reached which lies strictly below the diagonal it is permuted to the right by inserting columns from the spike stack to the left of it until it fits the diagonal.

A high level description of the first stage algorithm was given by Fletcher and Hall:

repeat
  find a row with minimum row count
  remove all columns which intersect this row
  update row counts

$$
\begin{bmatrix}
* & * & * & & & & & & \\
x & x & x & * & * & & & & \\
x & x & x & * & * & & & & \\
x & x & x & x & x & * & & & \\
x & x & x & x & x & * & & & \\
x & x & x & x & x & x & * & * & \\
x & x & x & x & x & x & x & x & * \\
x & x & x & x & x & x & x & x & * \\
x & x & x & x & x & x & x & x & *
\end{bmatrix}
\qquad
\begin{bmatrix}
* & & & & * & & & & * \\
x & * & * & & x & & & & x \\
x & * & * & & x & & & & x \\
x & x & x & * & x & & & & x \\
x & x & x & * & x & & & & x \\
x & x & x & x & x & * & & * & x \\
x & x & x & x & x & x & * & x & x \\
x & x & x & x & x & x & * & x & x \\
x & x & x & x & x & x & * & x & x
\end{bmatrix}
$$

$*$ denotes a non-zero element, x a possible non-zero element

Figure B.3: A lower block triangular matrix with fully dense **rectangular** blocks and the corresponding spiked lower triangular matrix. The columns in positions 3, 5, 8 and 9 are spikes.

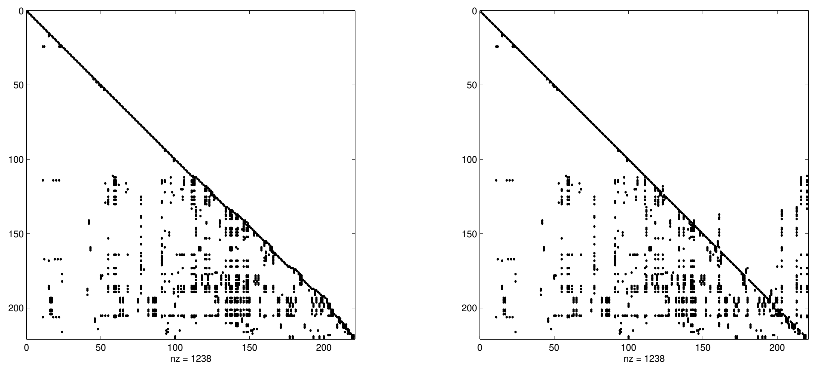


Figure B.4: Non-zero pattern of brandy. 1) after the first stage reordering 2) in spiked triangular form

remove all rows with zero row count
until all rows and columns are removed

By removing a row or a column we understand to permute it to the first position in the reordered matrix which has not yet been assigned and disregard it in the remainder of the algorithm. If the minimum row count is shared by more than one row the selection of the next row in the ordering can be based on different rules.

A more detailed description of the reordering scheme, incorporating the second stage final placement of spike columns, is shown in Figure 5. I and J are the sets of active rows and columns respectively. In each iteration the k'th row and column of the permuted matrix are determined. (We say that the row and column are paired). If the paired column is not a spike, the algorithm ensures that the diagonal element in the permuted matrix is non zero. When spike columns are identified they are removed from the set of active columns and put in the set S of unplaced spike columns. Whenever there is an active row which can not be paired with an active column such that the diagonal element becomes non zero (min $\{R_i : i \in I\} = 0$) this row is paired with a spike column from S.

The two most used tie-break rules used to select one of the rows with minimum row count in line 10 are

HR Choose a row which maximizes the number of rows removed with zero row count as a result of removing the columns intersecting the row. This is the rule used by Hellerman and Rarick in their P^3 and P^4 heuristics. Using this rule, in line 10, r must be chosen to maximize $|\{i \in I : |\{j \in J : B_{ij} \neq 0 \wedge B_{rj} \neq 0\}| = p\}|$

SPK1 Choose a row which maximizes the column count sum of the columns incident with the row. This simple rule was proposed by Stadtherr and Wood [18]. Using this rule, in line 10, r must be chosen to maximize $\sum_{j \in J : B_{rj} \neq 0} C_j$ where $C_j = |\{i \in I : B_{ij} \neq 0\}|$ is the column count of column j. Note that there is no need to update the C_j's during the reordering because the column counts of the active columns does not change.

It is seen that the HR rule is much more involved than the SPK1 rule. It is highly problem dependent, which of these two rules leads to the smallest number of spike columns. Experiments of Fletcher and Hall [8] comparing

1: $I = \{1, ..., m\}, \quad J = \{1, ..., m\}, \quad S = \emptyset$
2: **forall** $i \in I$ **do** $R_i = |\{j \in J : B_{ij} \neq 0\}|$
3: **for** $k = 1$ **to** m **do**
4: $\quad p = \min\{R_i : i \in I\}$
5: $\quad$ **if** $p = 0$ **then**
6: $\quad\quad$ choose $r \in I$ such that $R_r = 0$
7: $\quad\quad$ choose $c \in S$
8: $\quad\quad S = S \setminus \{c\}$
9: $\quad$ **else**
10: $\quad\quad$ choose $r \in I$ such that $R_r = p$
11: $\quad\quad J_r = \{j \in J : B_{rj} \neq 0\}$
12: $\quad\quad$ choose $c \in J_r$
13: $\quad\quad J = J \setminus \{J_r\}$
14: $\quad\quad S = S \cup \{j \in J_r : j \neq c\}$
15: $\quad\quad$ **forall** $j \in J_r$ **do** **forall** $i \in I$ such that $B_{ij} \neq 0$ **do** $R_i = R_i - 1$
16: $\quad$ **end if**
17: $\quad Q_r = k$;
18: $\quad P_r = k$; $\quad I = I \setminus \{r\}$
19: **end for**

Figure B.5: Spiked triangular ordering scheme the number of spikes obtained with these two tie-break rules gives no clear indication in favour of one the two rules, so they decided to use the SPK1 rule because of its lower execution time.

No matter what rule is used to choose r in line 10 of the ordering scheme, an efficient implementation needs easy access to the rows with minimum row count in each iteration. The sets of rows with a particular row count can be implemented as a collection of doubly linked lists [11]. This collection of doubly linked lists is easily initialized in $O(m)$ time when the row count of all rows are known. Since the reordering algorithm needs to have row wise access to the matrix in addition to the column wise input format, the initial row counts must be computed anyway when the row wise representation is determined. Each time a column is removed from the active part of the matrix the row count of the rows which have non zero elements in this column are decreased by one. Each time a row count is decreased the row is deleted from the set of rows with the old row count and inserted in the set of rows with the new row count in constant time. Since each non zero element can only be removed from the active part once, the overall complexity of maintaining access to rows by their row count is $O(nz)$. It is possible to exploit the fact that row counts are always decreased by one to obtain a faster implementation (though the complexity is still $O(nz)$), by using a data structure described in Zlatev [20], Chapter 6.2.

Gallivan et al. [9] used the ordering scheme described above without any tie-break rule at all and called the resulting algorithm LORA. Even if no tie-break rule is used we must have access to at least one row with minimum row count in each iteration. To ensure this it is still necessary to maintain sets of rows with the same row count as described above.

The new reordering algorithm (shown in Figure 6) was designed to avoid maintaining sets of rows with the same row count, except for rows with row count 0 or 1. Whenever the minimum row count is 0 or 1 the new algorithm works essentially in the same way as the ordering scheme described above. When the minimum row count is 2 or more a spike column must be selected. Instead of considering only columns which are incident with rows of minimum row count, all active columns are considered. We choose a column with maximum column count among the active columns. The minimum row count might not decrease immediately as a result (the algorithm is not locally optimal) but since the maximum number of row counts are decreased this choice tends to be helpful in later iterations. It is important to mention that it is not necessary to inspect all active columns


```
I = {1, ..., m},    J = {1, ..., m},    S = ∅
forall i ∈ I do  R_i = |{j ∈ J : B_ij ≠ 0}|
repeat
  p = min {R_i : i ∈ I}
  if p >= 2 then
    choose c ∈ J such that c = argmax{C_j : j ∈ J}
    J = J \ {c};    S = S ∪ {c}
    forall i ∈ I such that B_ic ≠ 0 do  R_i = R_i − 1
  else
    if p = 0 then
      choose r ∈ I such that R_r = 0
      choose c ∈ S
      S = S \ {c}
    else
      choose r ∈ I such that R_r = 1
      choose c ∈ J such that B_rc ≠ 0
      forall i ∈ I such that B_ic ≠ 0 do  R_i = R_i − 1
    end if
    P_r = k ;    I = I \ {r}
    Q_r = k ;    J = J \ {c}
    k = k + 1
  end if
until k = m
```

Figure B.6: THH spiked triangular ordering algorithm

to find one with maximimum column count. The columns can be sorted by decreasing column count up front using bucket sort in time $O(m)$. After this has been done it is a simple matter to find the next active column in this order each time a spike column must be chosen.

To check that the new fast reordering algorithm is in fact competitive with the traditional approach as regards the number of spikes, we compared the number of spikes obtained by the new algorithm and the LORA and SPK1 tie-break rules of the traditional approach. The test matrices were optimal bases of publicly available LP problems widely used for test purposes.

The results are shown in Table 1. The name refers to the name of the

Nr	Name	m	nz	Km	Knz	L	S	T
1	bas1lp	5411	81900	614	9968	220	206	246
2	baxter	27441	84454	535	1472	65	65	65
3	co9	10789	51951	1871	11837	347	292	309
4	cre-b	9648	22927	303	991	33	27	29
5	cre-d	8926	20672	210	658	14	14	13
6	dbic1	43200	129585	1665	7019	262	231	233
7	dbir1	18804	93747	300	1863	55	51	52
8	dbir2	18906	82818	487	5840	125	121	126
9	fit2p	3000	34208	20	219	10	10	10
10	ge	10099	24872	1871	5261	225	187	185
11	gen4	1537	90010	1062	63802	454	436	832
12	ken-13	28632	64066	30	61	1	1	1
13	lpl1	39951	110071	4780	11888	223	190	194
14	maros-r7	3136	62052	1350	31923	364	349	592
15	mod2	34774	108472	4424	12820	601	522	550
16	nl	7039	22803	973	3257	86	92	100
17	nsct2	23003	61234	164	1067	44	38	38
18	pds-20	33874	72779	1973	4259	27	24	28
19	pds-40	66844	144231	4772	10139	41	35	45
20	pilot87	2030	34000	1641	31500	384	318	334
21	route	20894	33030	864	3895	107	107	98
22	south31	18425	75280	112	460	12	2	2
23	stocfor3	16675	54365	1674	4321	90	84	84
24	world	34506	107070	4256	12064	540	436	474

Table B.1: Number of spikes for different ordering algorithms (L: LORA, S: SPK1 and T: THH).

LP for which the test matrix is an optimal basis. The table also includes the order and number of non zero elements of each of the test matrices (m and nz) and their kernels (Km and Knz). It is seen that THH is generally successful in keeping the number of spikes low except perhaps for the problems 11 and 14.

By preselecting a number of relatively dense columns destined to become spikes, it might be possible to cut the execution time of the ordering algorithm further down. Such columns would be removed and put on the spike-stack before the row wise representation is computed with a decrease in the number of operations proportional to the number of removed non zero elements as a result. If rows with row count zero arise as a result of the removal of these dense columns we start by pairing columns from the spike-stack with these rows. Of course the strategy of preselecting relatively dense columns as spikes might increase the number of spikes if too many columns are selected.

B.4.2 Diagonal blocks

The ordering algorithms can be modified to retain small fully dense diagonal blocks so that the number of spikes is decreased. For example in the reordered matrix in Figure B.3, the third column is a spike, but if diagonal blocks of size bigger than one is allowed in the lower triangular form this spike might be avoided. Of course this requires that the fully dense sub matrix in rows and columns 2 and 3 is not singular. The non-singularity of the whole matrix does not guarantee the non-singularity of this sub matrix because of the presence of the subsequent spike columns. Consequently such dense blocks must be explicitly checked for non-singularity if we want to exploit them to decrease the number of spikes. This option is not included in Spiky since it would complicate the implementation and it is not clear if the overall speed can be increased.

B.5 Block LU factorization

The main computational effort in computing the block LU factorization of the augmented system is the solution of $B_0 Y = C$. When solving this system we can exploit the fact that Y is typically sparse. For each sparse triangular system $B_0 Y_i = C_i$ this is done by determining the non-zero positions in Y_i by using depth-first search in the directed graph $G(B_0)$ associated with B_0, prior to the forward substitution. In this way unnecessary arithmetic operations with zero elements are avoided. The nodes in $G(B_0)$ which are reachable from nodes corresponding to non-zero elements of C_i are not necessarily found in increasing order, but as pointed out by Gilbert and Peierls [10] a topological order of the nodes is obtained from the reverse postorder of the depth-first search tree. Since we can solve $B_0 Y_i = C_i$ in any topological order of the non-zeros of Y_i the reverse postfix order found during the depth-first search can be used. When Y is sparse the savings can be huge.

The density of Y for the 24 test problems is shown in Table B.5 in the column headed $Y\delta$. The problems which benefit most from exploitation of solution sparsity as described in this section are the problems where s is large and $Y\delta$ is small, i.e. problems number 3, 6, 10, 15, 16, 23 and 24.

We mention that $S = V^T B_0^{-1} C$ alternatively can be computed by first computing $Z = V^T B_0^{-1}$ followed by $S = ZC$, which is likely to require fewer operations. But we have not tested this possibility yet.

B.6 Factorization of the Schur complement

In [11] a dense linear algebra code was used to compute the factors of the Schur complement matrix, S. S is sometimes sufficiently dense to justify the use of dense numerical linear algebra in its factorization. But it appears from Table 2 that the density of S is often low for the test problems considered. In these cases, time and storage is saved by using a sparse factorization. If the density of S is lower than a certain threshold value (the default is 0.25) we perform a sparse factorization of S, otherwise S is treated as a dense matrix.

For dense factorization of the Schur complement we have used a subroutine, dgetrf, from LAPACK, which is a free, portable library of Fortran 77 routines for solving problems in numerical linear algebra (more information on LAPACK is available at `http://www.netlib.org/lapack/`).

For sparse factorization of the Schur complement we have used UMFPACK Version 2.2.1 by Davis and Duff [4] (see also `http://www.cise.ufl.edu/`

Nr	s	B_0nz	Cnz	B_0+Cnz	Ynz	Yδ	Snz	Sδ
1	246	4637	5577	10214	35983	0.238	8829	0.146
2	65	1223	314	1537	611	0.018	95	0.022
3	309	9426	2720	12146	13281	0.023	3595	0.038
4	29	864	156	1020	1045	0.119	128	0.152
5	13	601	70	671	502	0.184	36	0.213
6	233	4879	2373	7252	24486	0.063	2189	0.040
7	52	1395	520	1915	3053	0.196	399	0.148
8	126	3635	2331	5966	11740	0.191	596	0.038
9	10	78	151	229	194	0.970	100	1.000
10	185	4416	1030	5446	12795	0.037	1569	0.046
11	832	11245	53389	64634	780996	0.884	633974	0.916
12	1	59	3	62	30	1.000	1	1.000
13	194	11414	668	12082	147486	0.159	8515	0.226
14	592	15352	17163	32515	317952	0.398	156621	0.447
15	550	11280	2090	13370	33540	0.014	4787	0.016
16	100	2760	597	3357	7414	0.076	1111	0.111
17	38	698	407	1105	1397	0.224	99	0.069
18	28	4203	84	4287	18757	0.340	208	0.265
19	45	10049	135	10184	60213	0.280	700	0.346
20	334	16721	15113	31834	179307	0.327	59085	0.530
21	98	3465	528	3993	18504	0.219	1305	0.136
22	2	274	188	462	224	1.000	4	1.000
23	84	4116	289	4405	2746	0.020	242	0.034
24	474	10662	1876	12538	30987	0.015	4220	0.019

Table B.2: Spiky statistics. s is the number of spikes. $B_0 nz$ is the number of non zero elements of B_0. Cnz is the total number of non zero elements in the spike columns C. $B_0 + Cnz$ is the sum of the number of non zeros in B_0 and C. Ynz is the number of non zeros in Y. $Y\delta$ is the density of Y. Snz is the number of non zeros of S. $S\delta$ is the density of$S$.

Factorize(B)

if the density of B ≥ 0.25 **then**
call dgetrf(B) to make a dense factorization of B
else
Find the kernel K of B. (If the kernel is empty STOP)
Determine B_0 and C and V using a spiked triangular reordering algorithm.
Compute $S = V^T B_0^{-1} C$
Factorize(S)
end if

Figure B.7: High level description of Spiky's factorization step

m	nz	δ	Matrix
16675	54365	0.0002	B the original matrix stocfor3
1674	4321	0.0015	K_1 the kernel of B
84	242	0.0343	S_1 the Schur complement of K_1
14	28	0.1429	K_2 the kernel of S_1
7	7	0.1429	S_2 the Schur complement of K_2
0	0	-	K_3 the kernel of S_2

Table B.3: Sizes of matrices involved in the factorization of stocfor3 (matrix nr. 23).

`~davis/umfpack.html`). UMFPACK implements sparse LU factorization with Markowitz pivoting.

It is also possible to use Spiky itself for sparse factorization of the Schur complement. In this way the factorization of the original matrix is broken down recursively as shown in Figure 7.

The recursion can end in two ways, either because the matrix to factorize has become so dense that we prefer to use a dense code or because it can be reordered to a triangular matrix (so the kernel is empty). The last possibility was only encountered for one of the test problems, stocfor3. In Table 3 the sizes and densities of the matrices involved in the factorization of stocfor3 are shown.

m	nz	δ	Matrix
34506	107070	0.0001	B the original matrix world
4256	12064	0.0007	K_1 the kernel of B
474	4220	0.0188	S_1 the Schur complement of K_1
267	2909	0.0408	K_2 the kernel of S_1
83	1402	0.2035	S_2 the Schur complement of K_2
75	1305	0.2320	K_3 the kernel of S_2
40	510	0.3188	S_3 the Schur complement of K_3

Table B.4: Sizes of matrices involved in the factorization of world (matrix nr. 24).

For 8 of the problems (see Table 2), the Schur complement has a density of over 25% so the recursion stops immediately. We show the behaviour of one of the remaining 15 problems, world, in Table 4. The factorization of world is completed in the third recursive call of Spiky when the density of the Schur complement exceeds 0.25.

B.7 The solve

There are two possibilities for the third step of the solve: $x_0 = \tilde{x} - Yx_c$ or $x_0 = \tilde{x} - B_0^{-1}Cx_c$. If x_c is dense the number of arithmetic operations required to compute Yx_c and $B_0^{-1}Cx_c$ is 2Ynz and 2(B_0nz+Cnz) respectively. It can be concluded from Table 2 that the latter number is nearly always the lowest and sometimes significantly lower. As a consequence the second possibility is used in Spiky.

B.8 Numerical stability

The numerical stability of block LU factorization is discussed in Highham [14]. The main result can be summarized as follows.

Suppose a block LU factorization of A

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & 0 \\ A_{21} & A_{22} - A_{21}A_{11}^{-1}A_{12} \end{bmatrix} \begin{bmatrix} I & A_{11}A_{12} \\ 0 & I \end{bmatrix} = LU$$

Nr	UMFPACK	SPIKY	Nr	UMFPACK	SPIKY
1	2.8e-13	4.9e-13	13	8.8e-14	1.7e-11
2	1.3e-12	1.3e-11	14	1.2e-14	1.4e+20
3	1.4e-11	1.9e-10	15	9.3e-13	2.2e-11
4	5.6e-12	7.1e-12	16	1.2e-12	1.7e-11
5	9.9e-13	2.8e-13	17	6.5e-11	1.1e-10
6	9.4e-8	1.1e-8	18	4.7e-15	7.1e-15
7	3.2e-10	1.3e-11	19	1.0e-14	5.7e-14
8	7.6e-11	8.2e-11	20	1.3e-11	8.9e-3
9	8.9e-14	7.5e-13	21	2.9e-13	2.0e-13
10	1.5e-13	4.7e-10	22	2.9e-14	1.0e-13
11	2.3e-11	5.0e+66	23	1.5e-13	3.7e-14
12	0	0	24	5.9e-13	5.7e-12

Table B.5: Accuracy of solutions obtained with UMFPACK and Spiky

is used to compute a solution to $Ax = b$ by solving $Ly = b$ by forward substitution and then solving $Ux = y$ by block back substitution. The stability of block LU factorization for solving $Ax = b$ as described above is determined by the ratio $||L||\ ||U||/||A||$. Unfortunately this ratio can be arbitrarily large.

As an illustration of what can go wrong consider the following matrix

$$\begin{bmatrix} 10 & 1 & 0 & 0 \\ 1 & 10 & 1 & 0 \\ 0 & 1 & 10 & 1 \\ 0 & 0 & 1 & 10 \end{bmatrix}$$

After reordering (choosing the first column as spike) the augmented matrix and its associated block LU factorization becomes

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 10 \\ 10 & 1 & 0 & 0 & 1 \\ 1 & 10 & 1 & 0 & 0 \\ 0 & 1 & 10 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 10 & 1 & 0 & 0 & 0 \\ 1 & 10 & 1 & 0 & 0 \\ 0 & 1 & 10 & 1 & 0 \\ 0 & 0 & 0 & 1 & -9791 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 10 \\ 0 & 1 & 0 & 0 & -99 \\ 0 & 0 & 1 & 0 & 989 \\ 0 & 0 & 0 & 1 & 9791 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Though Spiky is not numerically stable, in practice it computes solutions with high accuracy for most of our test problems. To measure the accuracy

we constructed a right hand side for each matrix such that the solution of the system is a vector of all ones. We then measure the accuracy as the max-norm of the error of the computed solution. In Table B.5 the accuracy of solutions computed by UMFPACK Version 2.2.1 and by Spiky are compared. For the test runs in this table, Spiky used UMFPACK for factorization of the Schur complement when the density was less than 0.25. We see that the solutions found by Spiky for two of the problems (nr 11 and 14) are useless, but that the accuracy for the remaining problems (except problem nr 20) is comparable to UMFPACK.

One way to enhance the stability is to reject columns of B_0 for which the absolute value of the diagonal element divided by the maximum absolute value in the column is smaller than a certain threshold value, θ. In this case we would simply treat those columns as though they were spikes (we refer to such columns as accuracy spikes). However, the exponential growth as illustrated in the example above can still happen. But as pointed out by Fletcher and Hall [8] similarly unfavourable situations can occur if a threshold Markowitz pivoting strategy is used.

The execution time of Spiky is highly dependent on the number of spikes, and the inclusion of accuracy spikes can slow down the factorization time. The effect of varying values of θ on the accuracy and the execution time is shown in Table B.6.

For the test runs used to produce Table B.6, Spiky itself was used recursively to factorize the Schur complement as long as the density was lower then 0.25. If the ratio of the number of spikes to the number of kernel columns exceeded 0.9 the recursion was ended by reverting to a dense factorization of the Schur complement even if the density was not over 0.25.

We see that most of the test problems are rather insensitive to the choice of θ, and that the accuracy of the solutions to the problematic problems can be improved by choosing θ sufficiently large.

B.9 Performance.

Spiky is written i C++. For the computational tests in this paper the program was compiled with the GNU C++ compiler with optimization flag `-O3` and executed on a Intel Pentium III 450MHz running under Linux 2.71. The Unsymmetric-pattern Multifrontal Package UMFPACK version

	e1	t1	e2	t2	e3	t3
bas1lp	3.9e-12	0.08	1.8e-11	0.08	2.8e-12	0.10
baxter	1.4e-11	0.06	6.7e-12	0.05	1.5e-12	0.06
co9	1.2e-09	0.07	2.6e-10	0.13	1.8e-11	0.16
cre-b	7.2e-12	0.02	7.2e-12	0.01	7.1e-12	0.02
cre-d	2.8e-13	0.01	2.8e-13	0.01	3.2e-13	0.01
dbic1	5.5e-07	0.30	5.5e-07	0.29	5.5e-07	0.29
dbir1	1.2e-11	0.05	1.2e-11	0.05	1.2e-11	0.06
dbir2	3.2e-10	0.05	3.2e-10	0.05	3.2e-10	0.05
fit2p	2.2e-13	0.02	2.2e-13	0.02	2.2e-13	0.02
ge	7.9e-11	0.03	1.7e-13	0.03	1.8e-13	0.04
gen4	4.4e+46	3.75	1.9e+30	3.95	5.9e-04	4.71
ken-13	0	0.05	0	0.05	0	0.05
lpl1	6.6e-11	0.15	4.7e-11	0.15	7.8e-13	0.16
maros-r7	1.0e+20	1.41	3.0e+09	1.93	5.2e-14	9.09
mod2	1.5e-11	0.15	2.7e-11	0.16	2.3e-12	0.33
nl	1.2e-09	0.02	5.5e-10	0.02	6.0e-12	0.02
nsct2	1.3e-10	0.04	1.3e-10	0.04	1.3e-10	0.04
pds-20	6.6e-15	0.07	6.6e-15	0.07	6.6e-15	0.08
pds-40	3.7e-14	0.16	3.7e-14	0.16	3.7e-14	0.16
pilot87	4.7e-03	0.34	2.1e-08	0.40	1.1e-11	0.72
route	1.7e-13	0.03	1.7e-13	0.03	1.7e-13	0.03
south31	1.7e-14	0.04	1.7e-14	0.04	1.6e-14	0.04
stocfor3	2.1e-14	0.04	2.1e-14	0.04	2.1e-14	0.04
world	5.8e-10	0.13	2.8e-09	0.14	1.1e-11	0.22

Table B.6: Accuracy of solutions and execution time of Spiky for varying values of θ. 1: $\theta = 0.001$, 2: $\theta = 0.01$, 3: $\theta = 0.1$

2.2.1 is publicly available and written in Fortran, it was compiled with the GNU Fortran compiler with optimization flag `-O` (which for this code is faster than `-O3`) and linked with optimized Pentium BLAS provided by Intel.

It can be seen from Table B.7 that the factorization of Spiky is generally 5 to 10 times faster than UMFPACK. Only on one of the problems, nr 14, UMFPACK is fastest, but Spiky can not be used to solve problem 11 and 14 anyway due to numerical unstability. UMFPACK was executed with default parameters which means that a block triangular reordering is enabled. This was indeed the fastest choice for most of the problems. As part of the block triangular reordering the initial triangular factors are removed just like in Spiky, but at a much higher cost. We therefore also compared the execution time for factorizing the kernel of the test matrices of the two codes. Spiky still comes out as a clear winner, but the difference is not as big as for the full matrices.

The time for solving systems using the computed factorizations is also important, especially in a linear programming context where the same factorization is usually used for solving many systems before a refactorization is computed. The solve time for the kernel of the test matrices for the two codes are shown in the last two columns of Table B.7. For most problems Spiky wins with about a factor 2. The solve times were measured for dense right-hand sides. In linear programming applications the right-hand sides will typically be very sparse, therefore Spiky provides an overloaded solve procedure for sparse right-hand sides. If the solution is sparse, the execution time of the solve can be more more than halved by using the sparse version.

B.10 Parallel Implementation.

There have been various attempts to adapt the simplex method to new parallel computer architectures, but apart from special cases such as dense problems [15] or problems with a big ratio of columns to rows [3], very limited speed-ups have been achieved. While other parts of the algorithm, such as pricing, can easily be parallelized, the parallelization of the basis inverse representation offers more challenges.

We implemented a parallel version of the factorization method described in this paper for the IBM SP distributed memory parallel computer. The

Nr	U	S	UK	SK	UKS	SKS
1	21	10	7	6	0.14	0.13
2	68	5	1	0	0.07	0.02
3	36	7	8	4	0.26	0.18
4	11	1	1	0	0.03	0.02
5	9	1	1	0	0.02	0.01
6	96	11	6	3	0.20	0.11
7	28	5	1	0	0.04	0.02
8	28	6	6	1	0.09	0.05
9	5	1	0	0	0.00	0.00
10	17	3	4	1	0.17	0.10
11	345	230	281	226	1.86	3.49
12	27	6	0	0	0.01	0.00
13	141	17	29	9	0.81	0.29
14	12	94	7	92	0.34	1.84
15	60	15	13	8	0.56	0.29
16	13	2	3	1	0.09	0.05
17	17	4	1	0	0.02	0.01
18	77	8	6	1	0.22	0.07
19	205	16	17	3	0.59	0.18
20	49	26	48	25	0.67	0.78
21	17	4	4	1	0.12	0.06
22	16	3	0	0	0.00	0.01
23	19	4	2	1	0.13	0.07
24	60	13	12	7	0.50	0.26

Table B.7: Execution times in hundredth of seconds. U: factorization time using UMFPACK, S: factorization time using Spiky, UK: kernel factorization time using UMFPACK, SK: kernel factorization time using Spiky, UKS: kernel solve time using UMFPACK, SKS: kernel solve time using Spiky

parallel implementation treats Y and S as dense matrices, thus it is not efficient for the problems where Y and S are very sparse.

The heuristics for finding a spiked triangular reordering are difficult to parallelize since the amount of work in each iteration, which depends directly on the previous iteration, is very small. Therefore we have chosen not to parallelize this step, but concentrate instead on an efficient sequential implementation which is executed simultaneously on all processors. Thus, in order for the overall algorithm to be scalable, it is necessary that the fraction of the total time which is spent in the reordering phase decreases as the problem size increases. Since this fraction is highly problem dependent, it is hard to say if this is the case. The reordering algorithm runs in $O(nz)$ time. The block LU-factorization of the augmented system requires $O(s * nz)$ time when the sparsity of Y is not exploited. A dense factorization of the Schur complement is accomplished is $O(s^3)$ time. It is not unreasonable to assume that s grows linearly with nz, in these circumstances we see that the sequential fraction of the algorithm goes towards zero as nz increases.

B.10.1 Data distribution

Because the computation of the matrix Y is achieved by s independent sparse triangular solves with B_0, we need a copy of B_0 on each processor. Also since the reordering is not parallelized we need the whole B on at least one of the processors. Therefore we choose to keep B on all processors.

Each sparse triangular solve with B_0 computes a column of Y. To avoid redistribution we adopt a one dimensional (column) distribution of Y. Since S is a submatrix of Y we let this distribution carry over to S. To ensure load balancing in the construction of the Schur complement the columns of Y must be evenly distributed over the processors. On the other hand the block size of the partitioned LU factorization of S should not be to small since this would mean that too much time is spend on communication. These two goals are partially conflicting; for small s it is fastest to use only one processor to do the LU factorization of S but without redistribution this would mean no parallelism in the construction of Y. In the current parallel implementation we use a block cyclic column distribution of Y (and S) with block-size $\min(s/p, 70)$, where p is the number of processors.

B.10.2 Block LU factorization

This step parallelizes perfectly. Recall that Y is obtained by solving the sparse triangular system, $B_0Y = C$, with s right-hand sides. This step can be parallelized by distributing the columns of C among the processors. Actually since C is the matrix consisting of the spike columns and thereby a sub matrix of B, all right-hand sides are already available on all processors (since B was broadcast to all processors), and only the work is distributed. On each processor a sparse triangular system with approximately s/p right-hand sides must be solved.

B.10.3 LU factorization of the Schur complement

Dense LU factorization on distributed memory parallel computers has received much attention and many efficient and scalable implementations have been constructed. We use one such routine from the parallel ESSL subroutine library provided by IBM. This routine has been specifically tuned for the IBM SP architecture and was written in Fortran and MPI. The implementation makes use of a partitioned algorithm in order to increase the granularity, i.e. the amount of work performed between communications.

Reasonable speed-up can be achieved for dense LU factorization if the size of the matrix is big compared to the number of processors. However, for many of the test problems the number of spikes is too low to benefit from the parallelization of the factorization of the Schur complement.

B.11 Further computational experiments

This section contains the results of further computational experiments which were conducted at a later stage than the experiments leading to the results shown so far. The aim of the results shown in this section is to support some of the previous conclusions which were not adequately documented by computational results. During the development of Spiky many different implementation variants were tested and a long sequence of improvements were obtained. However we did not always bother to keep the old implemetation and the results of the tests showing the superiority of a new variants once an improvement was identified. For this section a

number of previously discarded algorithmic variants were reconstructed in order to quantify the effect of each individual improvement.

The computational experiments in this section were performed on a Pentium III 800Hmz with 256MB RAM.

B.11.1 Ordering heuristics

In section 4 some different reordering heuristics were compared with regard to their ability to keep the number of spikes low. However no execution times were shown to support the choice of one heuristic over the others. In this subsection, 4 different reordering heuristics are compared with respect to their overall impact on the execution speed. The comparison includes 3 reordering heuristics described in section 4, SPK1, LORA and THH (HR was not implemented), plus a variant of SPK1, called SPK1(5), where the maximum number of min count rows considered in the tie-break is limited to 5.

Each of the heuristics were considered in combination with two different settings of the accuracy threshold parameter θ ($\theta = 0.1$ and $\theta = 0.001$) for a total of 8 different combinations reported in Table B.8 to Table B.15. For each combination the overall execution time of the factorization, **t**, the number of spikes (including accuracy spikes), **s**, the number of accuracy spikes, **as**, the accuracy of the solution, **acc**, as well as the execution time for each of the 4 steps of the factorization, **t0, t1, t2, t3**, explained below.

t0 = Kernel extraction The time for identifying the kernel and setting up datastructures for solving the full system using a factorization of the kernel.

t1 = Reordering heuristic

t2 = Block factorization

t3 = Factorization of the Schur complement

The execution times are all in milliseconds.

The kernel extraction is done before any reordering thus **t0** does not depend on the reordering heuristic nor on the accuracy threshold parameter. However since the measured execution times are wall clock times which can be affected by processes running in the background, small deviations in the values of **t0** can be observed in the tables B.8 - B.15.

The number of spikes selected by the reordering heuristic, s - as, depends only on the ordering heuristic (not on the accuracy threshold parameter), so they are the same in table B.N and B.N+4, N=8 to 11.

It can be seen that the fraction of the execution time which is spend in the reordering step varies considerably, but is often sufficiently high to warrant the use of a very fast heuristic. The overall best reordering heuristic is higly problem dependent, but with a clear tendency in favour of SPK1(5) and THH.

B.11.2 Block factorization

In this subsection we quantify the computational savings resulting from the improved exploitation of solution sparsity in the sparse triangular solves of the block factorization step. As noticed in the previous subsection many of the test problems spend very little time in the factorization of the kernel. To assess algorithmic changes to the factorization of the kernel better, we use the 10 test problems with the biggest kernels which are also the problems which spend most time in the factorization of the kernel.

In Table B.16 the block factorization time using the technique of Gilbert and Peierls is compared with traditional forward substitution (avoiding multiplication with zero).

The parameter settings for the case using the technique Gilbert and Peierls corresponds to the settings used for Table B.11, thus ideally the values of "GP t" and "GP t2" should be found in this table as well, however since the tests were executed on two different days some small differences are present.

As expected, the problems which benefit most from this technique are the ones where Y is very sparse and the number of spikes is relatively high. The number of spikes can be found in Table B.11, and the density of Y is included in the column Yδ (Y is generally more sparse than originally reported in Table B.2 due to the inclusion of accuracy spikes). For problems with a high density of Y, the overhead introdued by the GP technique can not be compensated for by subsequent savings. However even for a problem, gen4, with a density of Y as high as 0.788, the execution time of the block

	t0	s	as	t1	t2	t3	acc	t
bas1lp	21	221	1	3	15	25	4.1e-12	67
baxter	38	117	52	0	0	0	9.2e-12	41
co9	18	682	335	5	22	68	1.2e-11	115
cre-b	9	105	72	0	0	0	7e-12	11
cre-d	8	64	50	0	0	0	4e-13	10
dbic1	60	863	601	3	37	33	1.1e-06	136
dbir1	35	55	0	0	0	0	9.8e-11	37
dbir2	30	125	0	2	4	0	1e-10	38
fit2p	8	10	0	0	0	0	1.6e-13	9
ge	10	243	18	2	3	5	2.9e-12	23
gen4	21	850	396	23	256	3144	1.5e+13	3446
ken-13	38	1	0	0	0	0	0	39
lpl1	54	320	97	8	11	15	1.5e-13	90
maros-r7	15	1022	658	11	154	271	7.4e-14	454
mod2	45	780	179	8	31	129	4.6e-12	215
nl	8	143	57	1	1	2	2.8e-12	14
nsct2	26	44	0	0	0	0	2.4e-08	27
pds-20	48	27	0	2	1	0	0	54
pds-40	98	41	0	8	6	0	7.3e-15	114
pilot87	7	558	174	12	40	513	4.1e-11	575
route	16	109	2	1	2	1	1.2e-13	21
south31	27	12	0	0	0	0	4e-14	28
stocfor3	23	90	0	2	0	0	1.5e-14	27
world	44	685	145	8	25	60	1e-10	139

Table B.8: LORA reordering heuristic. $\theta = 0.1$

	t0	s	as	t1	t2	t3	acc	t
bas1lp	20	207	1	3	11	19	7.2e-11	56
baxter	36	120	55	1	0	0	9.2e-12	40
co9	17	618	326	12	19	94	5.8e-11	145
cre-b	9	103	76	0	0	0	7e-12	11
cre-d	8	67	53	0	0	0	3.5e-13	9
dbic1	59	874	643	11	37	31	7.1e-07	140
dbir1	33	51	0	0	0	0	3.5e-11	36
dbir2	29	121	0	2	4	0	3.9e-10	38
fit2p	8	10	0	0	0	0	1.1e-13	9
ge	10	198	11	11	2	3	1.5e-13	29
gen4	20	842	406	23	336	3056	7.5e+19	3437
ken-13	38	1	0	0	0	0	0	38
lpl1	51	291	101	32	14	16	2.7e-13	115
maros-r7	14	1010	661	11	153	241	1e-13	422
mod2	43	696	174	58	24	116	2.9e-12	244
nl	8	147	55	3	1	3	1.9e-11	17
nsct2	25	38	0	0	0	0	2.8e-09	26
pds-20	47	24	0	4	1	0	1.4e-14	53
pds-40	97	35	0	16	5	0	1.2e-14	121
pilot87	7	493	175	18	37	381	1.7e-11	446
route	15	109	2	3	4	1	1.4e-13	25
south31	26	3	1	0	0	0	3.2e-14	27
stocfor3	22	84	0	3	0	0	4.1e-14	28
world	43	582	146	49	18	33	3.5e-11	146

Table B.9: SPK1 reordering heuristic. $\theta = 0.1$

	t0	s	as	t1	t2	t3	acc	t
bas1lp	20	209	1	3	11	20	9.8e-11	57
baxter	36	113	48	0	0	0	9.2e-12	39
co9	18	656	338	5	20	91	2.1e-11	136
cre-b	9	108	77	0	0	0	7.1e-12	11
cre-d	8	64	51	0	0	0	3.2e-13	9
dbic1	60	862	630	4	36	31	9.4e-07	133
dbir1	33	51	0	0	0	0	6e-11	36
dbir2	29	122	0	2	4	0	9.3e-11	37
fit2p	8	10	0	0	0	0	1.1e-13	9
ge	10	209	11	3	3	4	4.5e-13	21
gen4	20	850	394	23	256	3124	2.2e+16	3426
ken-13	38	1	0	0	0	0	0	38
lpl1	51	300	93	8	10	12	1.9e-13	84
maros-r7	14	1006	657	12	156	234	6.3e-14	418
mod2	43	733	174	9	29	116	1.1e-11	199
nl	8	146	59	1	1	2	2.9e-12	14
nsct2	26	41	0	0	0	0	2.8e-09	27
pds-20	48	30	0	2	1	0	0	54
pds-40	99	42	0	8	5	0	3.1e-14	114
pilot87	7	503	179	12	42	438	3.8e-11	501
route	16	102	2	1	2	1	8.1e-14	21
south31	26	8	0	0	0	0	4.2e-14	27
stocfor3	22	85	0	2	0	0	1.2e-14	26
world	43	604	138	8	20	37	1.6e-10	110

Table B.10: SPK1(5) reordering heuristic. $\theta = 0.1$

	t0	s	as	t1	t2	t3	acc	t
bas1lp	21	246	0	2	10	30	3.3e-12	66
baxter	37	129	64	0	0	0	9.2e-12	40
co9	18	619	310	4	19	45	1.1e-11	88
cre-b	9	105	76	0	0	0	7e-12	11
cre-d	8	67	54	0	0	0	2.9e-13	9
dbic1	61	1093	860	3	56	54	6.5e-07	176
dbir1	34	52	0	0	0	0	1.3e-11	36
dbir2	30	126	0	1	3	0	1.2e-10	36
fit2p	8	10	0	0	0	0	7.5e-13	9
ge	10	193	8	2	3	3	1.6e-13	20
gen4	21	926	94	18	204	4123	0.021	4368
ken-13	38	1	0	0	0	0	0	39
lpl1	54	336	142	7	19	16	1.1e-12	98
maros-r7	15	1220	628	10	74	2588	3.2e-13	2689
mod2	45	795	245	7	33	95	7.3e-13	182
nl	8	162	62	1	2	2	6.1e-12	14
nsct2	26	38	0	0	0	0	2.8e-09	27
pds-20	48	28	0	2	2	0	1.5e-14	54
pds-40	101	45	0	6	7	0	1.9e-14	116
pilot87	7	494	160	10	38	380	6.6e-11	438
route	16	98	0	1	3	0	9.6e-14	22
south31	27	4	2	0	0	0	3e-14	28
stocfor3	22	85	1	2	0	0	1.6e-14	26
world	45	716	242	7	26	43	8.6e-12	123

Table B.11: THH reordering heuristic. $\theta = 0.1$

	t0	s	as	t1	t2	t3	acc	t
bas1lp	21	220	0	3	15	10	1.8e-11	51
baxter	38	66	1	0	0	0	9.2e-12	40
co9	18	358	11	5	7	13	1e-08	45
cre-b	9	33	0	0	0	0	7.2e-12	10
cre-d	8	14	0	0	0	0	2.8e-13	9
dbic1	61	863	601	3	37	33	1.1e-06	137
dbir1	34	55	0	0	0	0	9.8e-11	37
dbir2	30	125	0	2	4	0	1e-10	38
fit2p	8	10	0	0	0	0	1.6e-13	9
ge	10	233	8	3	3	5	2.4e-10	22
gen4	21	497	43	23	645	634	2.8e+144	1325
ken-13	38	1	0	0	0	0	0	39
lpl1	53	223	0	8	10	5	4e-10	78
maros-r7	15	364	0	12	162	247	6.3e+151	437
mod2	45	601	0	8	22	23	1.3e-08	100
nl	8	86	0	1	1	0	1.2e-10	12
nsct2	26	44	0	0	0	0	2.4e-08	27
pds-20	48	27	0	2	1	0	0	54
pds-40	98	41	0	8	6	0	1.2e-14	114
pilot87	7	387	3	12	51	272	1.3e+07	344
route	16	107	0	1	2	1	1.6e-13	22
south31	27	12	0	0	0	0	4e-14	28
stocfor3	22	90	0	2	0	0	1.5e-14	27
world	44	540	0	8	18	16	9e-09	89

Table B.12: LORA reordering heuristic. $\theta = 0.001$

	t0	s	as	t1	t2	t3	acc	t
bas1lp	21	206	0	3	11	11	9.4e-10	48
baxter	38	66	1	1	0	0	9.5e-12	40
co9	18	306	14	12	7	11	2.2e-08	50
cre-b	9	27	0	0	0	0	7.2e-12	11
cre-d	8	14	0	0	0	0	2.2e-13	9
dbic1	60	874	643	11	35	31	7.1e-07	140
dbir1	33	51	0	0	0	0	1.3e-11	36
dbir2	29	121	0	2	4	0	3.9e-10	38
fit2p	8	10	0	0	0	0	1.1e-13	9
ge	10	188	1	11	2	2	4.7e-11	28
gen4	20	476	40	23	697	569	2.6e+190	1311
ken-13	38	1	0	0	0	0	0	39
lpl1	53	190	0	32	13	3	3.5e-12	104
maros-r7	14	349	0	12	161	219	7.5e+175	409
mod2	43	522	0	57	16	14	1e-09	132
nl	8	92	0	3	1	1	2.6e-11	14
nsct2	26	38	0	0	0	0	2.8e-09	27
pds-20	48	24	0	4	1	0	1.4e-14	55
pds-40	98	35	0	16	5	0	1.2e-14	121
pilot87	7	326	8	18	50	160	53	238
route	15	107	0	3	4	1	2.5e-11	25
south31	26	2	0	0	0	0	3.2e-14	27
stocfor3	22	84	0	3	0	0	4.1e-14	28
world	43	436	0	49	12	10	2.4e-07	116

Table B.13: SPK1 reordering heuristic. $\theta = 0.001$

	t0	s	as	t1	t2	t3	acc	t
bas1lp	20	208	0	3	11	11	6.1e-11	48
baxter	36	65	0	0	0	0	9.2e-12	38
co9	18	328	10	5	6	14	1.9e-08	45
cre-b	9	31	0	0	0	0	7e-12	10
cre-d	8	13	0	0	0	0	2.8e-13	9
dbic1	58	862	630	4	37	32	9.4e-07	133
dbir1	33	51	0	0	0	0	1.4e-10	36
dbir2	29	122	0	2	4	0	9.3e-11	37
fit2p	8	10	0	0	0	0	1.1e-13	9
ge	10	202	4	3	3	3	1.2e-11	20
gen4	20	492	36	23	662	609	9.9e+166	1317
ken-13	38	1	0	0	0	0	0	38
lpl1	51	207	0	8	11	4	7.4e-12	77
maros-r7	15	349	0	12	171	220	1.6e+173	419
mod2	43	559	0	8	19	17	9.9e-09	90
nl	8	87	0	1	1	0	1.1e-10	12
nsct2	25	41	0	0	0	0	2.8e-09	26
pds-20	47	30	0	2	1	0	0	52
pds-40	97	42	0	8	5	0	4.9e-14	112
pilot87	7	332	8	12	62	171	9.7e+09	255
route	15	100	0	1	2	1	5.3e-12	21
south31	26	8	0	0	0	0	4.2e-14	27
stocfor3	22	85	0	2	0	0	1.2e-14	26
world	43	466	0	8	14	12	3.9e-08	79

Table B.14: SPK1(5) reordering heuristic. $\theta = 0.001$

	t0	s	as	t1	t2	t3	acc	t
bas1lp	20	246	0	2	10	9	4.9e-12	43
baxter	37	66	1	0	0	0	9.2e-12	38
co9	18	311	2	4	6	11	1.3e-09	41
cre-b	9	29	0	0	0	0	7.2e-12	10
cre-d	8	13	0	0	0	0	2.8e-13	9
dbic1	59	1093	860	3	55	53	6.5e-07	173
dbir1	33	52	0	0	0	0	1.3e-11	35
dbir2	29	126	0	1	3	0	1.2e-10	35
fit2p	8	10	0	0	0	0	7.5e-13	9
ge	10	187	2	2	3	2	1.7e-11	19
gen4	20	840	8	18	266	2998	1.3e+51	3304
ken-13	37	1	0	0	0	0	0	38
lpl1	51	194	0	7	22	7	6.4e-11	89
maros-r7	14	592	0	9	123	1058	3.8e+20	1208
mod2	43	550	0	7	18	17	1.2e-11	87
nl	8	100	0	1	1	1	1.5e-09	12
nsct2	25	38	0	0	0	0	2.8e-09	26
pds-20	47	28	0	2	2	0	1.5e-14	52
pds-40	98	45	0	6	7	0	1.9e-14	113
pilot87	7	337	3	10	49	178	0.0042	246
route	15	98	0	1	3	0	9.6e-14	22
south31	26	2	0	0	0	0	5.7e-14	27
stocfor3	22	84	0	2	0	0	1.6e-14	26
world	43	474	0	7	14	13	1.1e-09	79

Table B.15: THH reordering heuristic. $\theta = 0.001$

	GP t	t	GP t2	t2	Yδ
bas1lp	64	57	10	5	0.238
co9	85	105	18	33	0.013
dbic1	170	190	54	74	0.008
gen4	4613	4490	201	121	0.788
lpl1	94	107	19	33	0.064
maros-r7	2754	2749	72	97	0.038
mod2	174	225	31	81	0.008
pds-40	111	108	7	5	0.280
pilot87	396	395	37	33	0.159
world	119	162	26	69	0.008

Table B.16: Execution times in milliseconds with (GP) and without Gilbert and Peierls technique for exploitation of solution sparsity. t refers to the total factorization time and t2 refers to the time spend in the block factorization step. Yδ is the density of Y. ($\theta = 0.1$, THH reordering heuristic)

factorization step is less than the double of the corresponding execution time using the traditional approach.

B.11.3 Factorization of the Schur complement

In this subsection we highlight the effect of using sparse factorization of the Schur complement. As we have already seen in Table B.2, the density of the Schur complement of the test problems varies a lot. Obviously only the problems with relatively sparse Schur complements benefit from the possibility to apply Spiky recursively to the Schur complement.

In Table B.17 the effect on the execution time of restricting the number of recursive applications of Spiky is shown. As before the Schur complement is factorized using dense LU factorization if it's density exceeds 0.25.

It can be seen that the factorization speed is improoved by upto a factor 7 by using recursive sparse factorization of the Schur complement, constituing the single most important improvement relative to the method of Gondzio.

	D	R1	R5	level
bas1lp	98	61	64	1
co9	245	147	85	5
dbic1	1195	186	170	2
gen4	4902	4598	4613	0
lpl1	202	99	94	2
maros-r7	2755	2791	2754	0
mod2	888	301	174	4
pds-40	110	110	111	0
pilot87	528	418	396	2
world	675	183	119	5

Table B.17: Effect of varying the maximum number of recursive applications of Spiky. D : total factorization time using dense factorization of the Schur complement. R1 (R5) : total factorization time using maximally 1 (5) recursive applications of Spiky. The last column, level, indicates the number of actual recursive applications in the last case (max 5 allowed).($\theta = 0.1$, THH reordering heuristic)

B.12 Conclusion

Spiky is inspired by Gondzio [11] and Gilbert and Peierls [10]. By combining ideas from these papers and a new fast spiked triangular reordering algorithm, an efficient implementation of a sparse unsymmetric linear system solver has been achieved.

The method was shown to be very competitive in speed compared to a state-of-the-art implementation of a more traditional approach to sparse matrix factorization. Furthermore the method has the advantage that the second step, the computation of Y, is trivially parallelizable. We tried to exploit this fact in a parallel implementation. Much of the work on the parallel implementation consisted in work on the sequential part of the method, the reordering algorithm, which we managed to make much faster. As a result we obtained nearly linear speed-up on 4 processors for the test problems with most spike columns. Later however we realized that the sequential version could be improved considerably by exploiting solution sparsity in the computation of Y, initial kernel extraction, etc. This was not incorporated in the parallel implementation, but if this had been done the speed-ups would have become smaller, because the time spend in the parallelizable second step is much smaller. The tables B.8 to B.15 give insight in the obtainable speed up by showing the fraction of the execution time spend in each of the four parts of the factorization step. The first two steps, i.e. the kernel extraction and the reordering heuristic can not be parallelized, the third step is embarrassingly parallel while the last step, the factorization of the Schur complement offers some limited possibilties for parallel speed-up depending on the size of the Schur complement. A (slightly pessimistic) estimate of the execution time, T_p, on a p processor system can be obtained by the formula $T_p = t0 + t1 + t2/p + t3$, where $t0, t1, t2$ and $t3$ refers to the partial execution times as explained in section B.11. Thus the best speed up can be obtained for the problems where $t2$ is biggest in relation to the total execution time, for example dbic1. For this problem using the THH reordering heuristic and $\theta = 0.1$, the estimates of the execution time (in milliseconds) is $T_1 = 189, T_2 = 159$ and $T_3 = 149$, that is, a speed up of only 1.19 and 1.27 using two and three processors respectively in the best case.

The results of the paper illustrates that the factorization method used in Spiky, does not fulfill the hopes of being more amenable to parallelization than the traditional sparse LU factorization approach. However, when efficiently implemented, the method is superior to a state of the art implementation of the traditional approach.

Bibliography

[1] M. Arioli, I. S. Duff, N. I. M. Gould, and J. K. Reid. Use of the P^4 and P^5 algorithms for in-core factorization of sparse matrices. *SIAM Journal on Scientific and Statistical Computation*, 11(5):913–927, 1990.

[2] J. Bisschop and A. Meeraus. Matrix augmentation and partitioning in the updating of the basis inverse. *Mathematical Programming*, 13:241–254, 1977.

[3] R. E. Bixby and A. Martin. Parallelizing the dual simplex method. Technical report, Center for Research on Parallel Computation, Rice University, Houston, 1997.

[4] T. A. Davis and I. S. Duff. A combined unifrontal/multifrontal method for unsymmetric sparse matrices. *ACM Transactions on Mathematical Software*, 25(1):1–19, 1999.

[5] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for sparse matrices*. Oxford University Press, New York, 1987.

[6] S. K. Eldersveld and M. A. Saunders. A block-LU update for large-scale linear programming. *SIAM J. Matrix Anal. Appl.*, 13(1):191–201, 1992.

[7] A. M. Erisman, R. G. Grimes, J. G. Lewis, and Jr. W. G. Poole. A structurally stable modification of Hellerman-Rarick's P^4 algorithm for reordering unsymmetric sparse matrices. *SIAM Journal of Numerical Analysis*, 22(2):369–385, 1985.

[8] R. Fletcher and J. A. J. Hall. Ordering algorithms for irreducible sparse linear systems. *Annals of O.R.*, 43:15–32, 1993.

[9] K. Gallivan, P. C. Hansen, T. Ostromsky, and Z. Zlatev. A locally optimized reordering algorithm and its application to a parallel sparse linear system solver. *Computing*, 54(1):39–68, 1995.

[10] J. R. Gilbert and T. Peierls. Sparse partial pivoting in time proportional to arithmetic operations. *SIAM Journal of Sci. Stat. Comput.*, 9(5):862–874, 1988.

[11] J. Gondzio. On exploiting original problem data in the inverse representation of linear programming bases. *ORSA Journal on Computing*, 6:193–206, 1994.

[12] E. Hellerman and D. C. Rarick. Reinversion with the preassigned pivot procedure. *Mathematical Programming*, 1:195–216, 1971.

[13] E. Hellerman and D. C. Rarick. The partitioned preassigned pivot procedure (P^4). In D. J. Rose and R. A. Willoughby, editors, *Sparse Matrices and Their Applications*, pages 67–76. Plenum Press, 1972.

[14] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 1996.

[15] G. Karypis and V. Kumar. Performance and scalability of the parallel simplex method for dense linear programming problems. An extended abstract. Computer Science Department, University of Minnesota, May 1994.

[16] H. M. Markowitz. The elimination form of the inverse and its application to linear programming. *Management Science*, 3:255–269, 1957.

[17] W. Orchard-Hayes. *Advanced Linear-Programming Computing Techniques*. McGraw-Hill Book Company, 1968.

[18] M. A. Stadtherr and S. E. Wood. Sparse matrix methods for equation based chemical process flowsheeting - I. Reordering phase. *Computers and Chemical Engineering*, 8:9–18, 1984.

[19] U. H. Suhl and L. M. Suhl. Computing sparse LU factorizations for large-scale linear programming bases. *ORSA Journal on Computing*, 2(4):325–335, 1990.

[20] Zahari Zlatev. *Computational Methods for General Sparse Matrices*. Kluwer Academic Publishers, 1991.

Appendix C

Aggressive LP reduction

Paper C:

Aggressive LP reduction

Abstract

Simple techniques for reducing the size of linear programs prior to the application of a solution algorithm are incorporated as important parts of most LP solvers. The time spent to reduce the problem size is often more than regained by subsequent savings in the solution time. There is a delicate trade off between the computational effort invested in the problem reduction and the actually obtained reductions. The usual LP reduction techniques require a very limited effort but nevertheless often result in substantial reductions. One of the reasons for this good performance is the way LP models are typically formulated using algebraic modelling languages. We present a common framework for LP reduction algorithms and show how the usual LP reduction techniques fit into this framework. We also demonstrate how a stronger bound strengthening and the use of primal information to improve the dual bounds can be used to obtain further reductions.

C.1 Introduction

We are interested in solving linear programs of the form

$$\text{minimize } \{c^T x : \ Ax = b, \ l \leq x \leq u\}$$

for $x \in R^n$, where $A \in R^{m \times n}, b \in R^m$ and $c, l, u \in R^n$ are given data (elements of l and u are allowed to be $-\infty$ and ∞ respectively). No matter whether an interior point or a simplex method is used, it is common to submit the problem to a presolve phase before attempting to solve it. Besides problem reduction, the presolve phase can be used to improve the scaling of the problem and to detect linear dependent rows of A. This paper is about the problem reduction exclusively and we assume that A is full row rank and well scaled.

LP reduction algorithms were already incorporated in some LP solvers when the first paper [3] on the subject appeared in 1975. Since then, its practical importance has been confirmed by various authors, [1, 8, 12]. In these papers it is shown how the size of various “real-life” test-problems can be reduced considerably by applying variants of the simple and fast techniques described in the first paper. The reduced problems are often solved much faster than the original problems. The importance of problem reduction was emphasized by Lustig, Marsten and Shanno [12] in the following statement: “The most important issue in creating a fast implementation of an interior point algorithm is clearly a fast implementation of the numerical linear algebra. For large linear programs, the second most important issue is preprocessing to reduce problem size.” This is also true if a simplex method is used to solve the model, although interior point methods seem to benefit even more from problem reduction since they work with a global view of the model as opposed to the local view of the simplex method.

The motivation behind this work is to investigate if it is possible to find extensions of traditional LP reduction algorithms which are computationally attractive. We can not expect all problems to benefit from a new aggressive LP reduction scheme, but to be used as a default reduction procedure we must comply with the philosophy that “when it does not help, it does not hurt (much)”, meaning that the time wasted in fruitless search of non-existing reductions should be kept at a minimum.

The rest of the paper is organized as follows. Section 2 contains an exposition of traditional LP reduction techniques viewed as relaxations of what

we have called extreme LP reduction. Section 3 gives a partial explanation of the easy success of LP reduction. In section 4 some possibilities for enhancing the bound strengthening are discussed and section 5 takes a closer look on primal-dual interaction. Some implementation issues are discussed in section 6. In section 7 we show how Fourier-Motzkin elimination can be used to reduce some linear programs. In section 8, LP reduction is illustrated by a small numerical example. Finally computational experiments and conclusions are given in sections 9 and 10.

For notational convenience we shall use the convention that zero multiplied with infinity or minus infinity is zero ($0*\infty = \infty*0 = 0*-\infty = -\infty*0 = 0$) throughout the paper. The i'th row of a matrix A will be denoted A_i. If I is a subset of row indices, A_I denotes the corresponding sub matrix. If v is a vector, v_+ is a vector with elements $v_{+_i} = v_i$ if $v_i > 0$ and 0 if $v_i <= 0$ and v_- is a vector with elements $v_{-_i} = v_i$ if $v_i < 0$ and 0 if $v_i >= 0$.

C.2 Traditional LP reduction

LP reduction techniques are usually presented as a collection of simple rules which, if they can be applied, reduces the size of the problem. A reduction might make it possible to apply one of the rules to the new reduced problem to make further reductions which could not have been made directly in the original problem. In this way the application of the rules continues until no more reductions can be made or the reductions obtainable are no longer worth the effort to find them.

Here we will regard the traditional LP reduction rules as relaxations of the following **extreme** LP reduction algorithm based on the optimality conditions for the linear program

$$\text{P:} \quad \text{minimize } \{c^T x : \ Ax = b, \ l \leq x \leq u\}$$

and its associated dual program

$$\text{D:} \quad \text{maximize } \{b^T y + l^T z - u^T w : \ A^T y + z - w = c, \ z \geq 0, \ w \geq 0\}.$$

These can be stated as follows: (x, y, z, w) are optimal solutions for P respectively D if and only if

$$Ax = b \qquad\qquad A^T y + z - w = c$$

$$l \le x \le u \qquad\qquad z, w \ge 0$$
$$c^T x = b^T y + l^T z - u^T w$$

The optimality region will be denoted $S^*(A, b, c, l, u)$ or simply S^* , i.e. S^* is the set of primal-dual solutions $(x, y, z, w) \in R^n \times R^m \times R^n \times R^n$ satisfying the optimality conditions stated above. In any optimal solution x_j belongs to the interval $[\underline{x}_j^*, \overline{x}_j^*]$ where

$$\underline{x}_j^* = \min\{x_j : (x, y, z, w) \in S^*\} \tag{C.1}$$
$$\overline{x}_j^* = \max\{x_j : (x, y, z, w) \in S^*\} \tag{C.2}$$

Each of these bounds can be obtained by solving a linear program in (x, y, z, w). The extreme algorithm simply consists in solving these $2n$ linear programs. If $\underline{x}_j^* = \overline{x}_j^*$, x_j can be eliminated by substituting it with this value. If the optimal solution is unique, all variables can be eliminated in this way and the resulting reduced problem is empty. Otherwise the remaining constraints describe the optimal face.

Of course one never wants to use extreme LP reduction. The time required to determine just one of the $2n$ bounds exceeds the time required to solve the original un-reduced LP. In practice we need to consider relaxations of (C.1) and (C.2) in order to obtain lower and upper bounds on $\underline{x}_j^*$ and $\overline{x}_j^*$. Such bounds will be called *implied bounds*, i.e. the bounds $\underline{x}_j \le x_j \le \overline{x}_j$ are implied if and only if $\underline{x}_j \le \underline{x}_j^*$ and $\overline{x}_j \ge \overline{x}_j^*$. We shall also use implied bounds on the dual variables y, z and w defined analogously.

By isolating a variable in an equality we might be able to strengthen the bounds on this variable. If $a_{ij} \ne 0$, isolating x_j in the i'th equality, we get $x_j = (b_i - \sum_{k \ne j} a_{ik} x_k)/a_{ij}$ Any upper bound on the right hand side is an upper bound on x_j. Finding the lowest possible upper bound (based on primal information) on the right hand side would involve the maximization of the right hand side subject to $Ax = b$, $l \le x \le u$; a problem just as difficult as the original problem. Instead we maximize the right hand side subject to $x \in \{x \in R^n : \underline{x} \le x \le \overline{x}\}$ where $\underline{x}$ and $\overline{x}$ are the strongest currently known implied bounds on x. This provides an alternative, generally weaker, upper bound on x_j which is much faster to find. If $a_{ij} > 0$, this maximum is achieved simply by setting each variable x_k for which a_{ik} is positive equal to $\underline{x}_k$ and each variable x_k for which a_{ik} is negative equal to $\overline{x}_k$; and the opposite if $a_{ij} < 0$. Lower bounds are treated similarly. For sparse problems, using only bounds on the variables and a single constraint

of P at a time, usually quite strong implied bounds can be computed efficiently. This technique is called myopic bound strengthening and can be summarized as shown in Figure 1.

Initialize $\underline{x}_j = l_j$ and $\overline{x}_j = u_j$ for all $j \in \{1, ..., n\}$
repeat
  Let x_j be a variable and $A_i x = b_i$ a constraint such that $A_{ij} \ne 0$.
  if $\min\{x_j : A_i x = b_i, \underline{x} \le x \le \overline{x}\} > \underline{x}_j$ **then** $\underline{x}_j = \min\{x_j : A_i x = b_i, \underline{x} \le x \le \overline{x}\}$
  if $\max\{x_j : A_i x = b_i, \underline{x} \le x \le \overline{x}\} < \overline{x}_j$ **then** $\overline{x}_j = \max\{x_j : A_i x = b_i, \underline{x} \le x \le \overline{x}\}$
until no more bounds can be strengthened

Figure C.1: Myopic bound strengthening (primal variables)

Myopic bound strengthening can also be applied to the dual constraints to obtain implied bounds on the dual variables in a similar way. Finally it should not be forgotten that the zero-gap condition $c^T x = b^T y + l^T z - u^T w$ might also be used to strengthen the bounds of some variables.

A bound on a variable of a linear program is said to be *redundant* if it can be removed without affecting the set of solutions satisfying the constraints of the LP. A bound on a variable is *weakly redundant* if it is redundant and there exist a feasible solution with the value of the variable at its bound.

The implied bounds are used to detect conditions for reducing P in three ways.

1. If a variable x_j is forced by the primal constraints to take a specific value x_j^* then the variable can be eliminated by substituting it with this value.

2. If $z_j > 0$ ($w_j > 0$) is implied by the dual constraints then x_j must be at its lower bound $x_j = l_j$ (upper bound $x_j = u_j$) in any optimal solution. In this case x_j can be eliminated as above.

3. If the bounds on a variable $l_j \le x_j \le u_j$ are redundant then x_j may be treated as free and hence can be eliminated by Gaussian elimination. This also eliminates one of the equations of the LP in which x_j appears.

The first case occurs if $\underline{x}_j = \overline{x}_j$. The optimal value of x_j is implied by the problem constraints only and is not due to economical preferences; there are simply no other feasible alternatives. This distinction of the cause of the optimal value of a variable can be important for the interpretation of the solution [9].

The second case is detected if $\underline{z}_j > 0$ ($\underline{w}_j > 0$) which is based on pure dual information. It is easy to see that an optimal solution with $x_j = l_j$ ($x_j = u_j$) must exist if $z_j \geq 0$ ($w_j \geq 0$) is redundant in D, but of course alternative optimal solutions with other values of x_j might exist if the redundancy is weak. In section 5, we will discuss how weakly redundant bounds (primal or dual) should be handled.

In the third case, x_j is said to be implied free. Depending on the non-zero structure of the coefficient matrix, the use of Gaussian elimination to remove a row and a column might result in new non-zero elements (fill-in) in the reduced coefficient matrix. For this reason some implementations restrict the use of Gaussian elimination to variables which appears in at most a certain number of rows. The easiest implementation is achieved by considering only variables corresponding to singleton columns, because in this case no new non-zero elements are generated when the corresponding row is removed.

Some cases of infeasibility and unboundedness can trivially be detected during the reduction phase. This occurs when incompatible implied lower and upper bounds on a variable (primal or dual) are detected. For ease of presentation we shall not mention this possibility again and simply assume that the problem at hand is feasible and bounded.

C.3 Why does it work?

It might come as a surprise that the fast LP reduction techniques described in the preceding section are often so successful. The key issue to explain this fact is the modelling practice typically observed when formulating large LP models. We shall not attempt to explain this fact in detail, but we show some typical situations in practical model formulation which leads to easy reductions.

An equation such as $\sum_j a_{ij}x_j = b_i$ where $b_i = 0$ and $x_j, a_{ij} \geq 0$ allow us to fix all variables occurring in this equation to zero. Such an equation can

appear in a problem, for example if b_i represents the available amount of a resource consumed by some of the activities x_j (the ones with $a_{ij} > 0$). The equation is part of a model class which is used to generate several model instances with varying data. In other words, b_i is not always zero so the participating variables can not generally be removed from the model class formulation. With some effort it should be possible to formulate the model class in most algebraic modelling languages in such a way that the variables in question are not generated for model instances with $b_i = 0$. However this is normally a bad idea to do, since it obscures the model and the same job is done nicely by the problem reduction, if available.

The use of so called accounting variables to help model formulation and maintenance is a common and sensible practice [13]. Generally speaking, accounting variables are redundant variables introduced in the model with the purpose of denoting some expression used in the model. By associating a name with a possibly complicated expression, the model can become much easier to formulate and understand. Furthermore the risk of modelling errors is decreased if a variable is used to denote a repeatedly occurring expression. It is clear that the bounds (if any) on accounting variables must be redundant. This is easily detected by the problem reduction and therefore these variables can be eliminated.

It can be convenient to formulate a model using some explicitly fixed variables. It is also surprisingly frequent for practical problems to contain variables which are implicitly fixed by singleton rows. Sometimes the removal of the implicitly fixed variables results in new row singletons and thereby the detection of more implicitly fixed variables.

Based on my experience with publicly available test problems, it is my impression that more than 95% of all reductions made by traditional LP reduction can be attributed to one of the cases described above.

C.4 Aggressive bound strengthening

Myopic bound strengthening might lead to an infinite sequence of bound improvements. For example, the equations

$$-x_1 + 2x_2 = 2 \tag{C.3}$$
$$x_1 + x_2 = 4 \tag{C.4}$$

with the initial bounds $1 \leq x_1 \leq 4$ and $0 \leq x_2 \leq 3$ lead to the following infinite sequence of implied bounds

From $x_2 = 1 + 1/2x_1$ we get $x_2 \geq 3/2$
From $x_1 = 4 - x_2$ we get $x_1 \leq 5/2$
From $x_2 = 1 + 1/2x_1$ we get $x_2 \leq 9/4$
From $x_1 = 4 - x_2$ we get $x_1 \geq 7/4$
From $x_2 = 1 + 1/2x_1$ we get $x_2 \geq 15/8$
and so on ...

It can be seen that the implied bounds on x_1 and x_2 converges to $x_1 \in [2, 2]$ and $x_2 \in [2, 2]$. As pointed out by Fourer and Gay [5] this amounts to solving the above system iteratively with the Gauss-Seidel method.

The myopic bound strengthening works surprisingly well on real world sparse LP's. However it is tempting to try the possibility to strengthen bounds further by considering more than one equation at a time together with the currently best known implied bounds. This gives rise to the a general bound strengthening algorithm shown in Figure 2, where $I_{ij} \subseteq \{1, ..., m\}$ can be chosen according to different strategies, $I_{ij} = \{i\}$ corresponds to normal myopic bound strengthening. Some other possible strategies are:

Initialize $\underline{x}_j = l_j$ and $\overline{x}_j = u_j$ for all $j \in \{1, ..., n\}$
repeat
  Let x_j be a variable and $A_i x = b_i$ a constraint such that $A_{ij} \neq 0$.
  if $\min\{x_j : A_{I_{ij}} x = b_{I_{ij}}, \underline{x} \leq x \leq \overline{x}\} > \underline{x}_j$ **then** $\underline{x}_j = \min\{x_j : A_{I_{ij}} ix = b_{I_{ij}}, \underline{x} \leq x \leq \overline{x}\}$
  if $\max\{x_j : A_{I_{ij}} x = b_{I_{ij}}, \underline{x} \leq x \leq \overline{x}\} < \overline{x}_j$ **then** $\overline{x}_j = \max\{x_j : A_{I_{ij}} x = b_{I_{ij}}, \underline{x} \leq x \leq \overline{x}\}$
until no more bounds can be strengthened

Figure C.2: General bound strengthening (primal variables)

Slightly myopic bound strengthening $I_{ij} = \{i, k\}$ where k is chosen to be among the rows which maximize the number of columns with non-zero elements in both row i and k. This strategy has the advantage that the number of rows (two) in the LP's we are solving are fixed and known in advance.

Egocentric bound strengthening $I_{ij} = \{i : A_{ij} \neq 0\}$ An advantage of this scheme is that the set of equations, I_{ij}, to include in the bound strengthening of x_j does not depend on i, so the total number of LP's involved in the bound strengthening is limited to two for each variable. If there happens to be few variables (besides x_j) with non-zero elements in more than one of the equations determined by the set I_{ij}, the resulting bounds will not be much stronger than the bounds which could have been achieved by doing myopic bound strengthening on each of the individual equations. In this case, however, the resulting two LP's will have nearly diagonal bases and the computational effort to solve them is not much bigger than the effort of performing myopic bound strengthening. If the equations on the contrary have many non-zero elements in common columns, more work can be expected, but generally we will also obtain stronger bounds.

The average number of non zero elements in each column is typically reasonably small for large sparse real world LP's so the size of the LP's which must be solved by the bound strengthening procedure with this strategy is small on average. Sometimes an otherwise sparse problem might contain a small number of dense columns. Strengthening the bounds on a variable corresponding to a column with a number of non zero elements much higher than the average with this strategy involves solving a big LP (potentially as big as the original if the column in question is fully dense). Therefore such dense columns must be excluded from this strategy and myopic bound strengthening used for the corresponding variables.

Full bound strengthening $I_{ij} = \{1, ..., m\}$ With full bound strengthening we are approaching extreme LP reduction. The difference is that while extreme LP reduction consider all equations of the optimality conditions simultaneously, in full bound strengthening primal and dual constraints are considered separately.

Anything beyond myopic bound strengthening certainly deserves the adjective aggressive. Even carrying on myopic bound strengthening until convergence is sometimes considered too aggressive, but as we shall see in the section on implementational issues, the computational effort of continuing bound strengthening until no bound can be moved more than a small value EPS (equal to 1e-08 in the computational tests) is not necessarily prohibitive.

C.5 Primal-dual interaction

The primal-dual connection of the optimality conditions can be expressed in two alternative ways. As the complementary slackness condition, $(x_j - l_j)z_j = 0,\ (u_j - x_j)w_j = 0 \quad \text{for } j = 1, ..., n$ or as $c^t x = b^t y + l^t z - u^t w$. Although they are equivalent if the whole set of optimality conditions is considered simultaneously, differences are possible in the bounds which can be achieved by myopic bound strengthening on each of the alternative formulations. The problem reduction can take advantage of both these formulations at the same time.

The constraint $c^t x = b^t y + l^t z - u^t w$ can be used for bound strengthening ($c^t x \leq \overline{f} \equiv b^t_+ \overline{y} + b^t_- \underline{y} + l^t_+ \overline{z} + l^t_- \underline{z} - u^t_+ \overline{w} - u^t_- \underline{w}$ and $b^t y + l^t z - u^t w \geq \underline{f} \equiv c^t_+ \underline{x} + c^t_- \overline{x}$). Here $\underline{f}$ and $\overline{f}$ are the currently best known bounds on the optimal objective function value. If for some reason an upper or lower bound on the objective function is known it can be used instead of $\overline{f}$ or $\underline{f}$ if it is stronger.

Implied bounds on the dual variables can have consequences for the primal problem and vice-versa. We have already discussed how a strictly positive implied bound on a variable z_j allow us to fix the corresponding primal variable at its lower bound. Conversely it is also possible to fix a dual variable z_j at zero if the corresponding primal variable x_j can be shown, by an implied bound, to be strictly greater than its lower bound. This does not immediately result in any reductions but might open up for further strengthening of the bounds on the dual variable possibly leading to the fixing of primal variables. Curiously, the use of primal implied bounds to fix dual variables was already exploited by Brearly et. al. [3] (although the use was limited to slack columns) but this important option seems to have been forgotten by later authors [1, 8].

Instead these authors focus on the ability to fix primal variables as a consequence of weakly redundant dual bounds. Clearly the removal of a redundant dual bound restricts the primal feasible region by fixing the associated primal variable, but since the dual feasible region is not changed the optimal objective function value does not change either. In other words, if $z_j \geq 0$ is weakly redundant there exists an optimal (primal) solution with $x_j = l_j$. But by fixing x_j we might eliminate some alternative optimal solutions. Although we are sometimes only interested in any single optimal solution, in general it possible that the user desires to retain the possibility to determine the whole set of optimal solutions. For this reason we choose

not to use weak dual redundancy to fix the corresponding primal variable unless this option is specifically enabled. Like in the primal case (discussed below) care must be taken to ensure that only simultaneously redundant bounds are removed in this way.

C.5.1 Implied free variables

A variable is said to be implied free if its bounds are redundant. Whenever a redundant bound on a primal variable is detected we can fix a dual variable at zero. If both of the bounds are redundant the variable can be eliminated by substitution. This option is most advantageous for columns with few non-zeros.

In practice, LP's tend to have many weakly redundant bounds and it is important for the reduction algorithm to detect as many as possible of these; however care must be taken. The removal of one weakly redundant inequality from a system of inequalities might negate the redundancy of other inequalities. Therefore a set of weakly redundant inequalities can not generally be removed simultaneously. This was (re)discovered by Tomlin and Welch [16] in "the hard way", when their procedure for detecting implied free LP variables resulted in an unbounded problem.

The question, when is it feasible to simultaneously remove a set of redundant bounds, is answered by the theory of dependency sets, see Greenberg [10]. Let $B = \{l_j \leq x_j : l_j > -\infty\} \cup \{x_j \leq u_j : u_j < \infty\}$ be the set of finite bounds on the primal variables. A dependency set for a redundant bound is a set of constraints which renders the bound redundant. Here we consider the equations $\{Ax = b\}$ as given so $d \subset B$ is a dependency set for $e \in B$ iff $e \notin d$ and e is implied by $\{Ax = b\} \cup d$. Suppose $\{Ax = b,\ l \leq x \leq u\}$ is consistent and that $R = \{l_j \leq x_j : i \in I_1\} \cup \{x_j \leq u_j : i \in I_2\} \subseteq B$ are all redundant. It is easily seen that the redundant bounds in R can be simultaneously removed if and only if B contains a dependency set common for all elements of R.

This does not, however, provide an algorithm for finding a maximum cardinality set of simultaneously redundant bounds, although this problem could be formulated as a max-satisfiability problem if all (minimal) dependency sets for each redundant bound were known. The max-satisfiability problem is NP-hard and even the task of determining all dependency sets let alone all redundant bounds is a challenging task. Most of the practical methods to determine redundant constraints are based on some kind of pivoting algorithms [6]. Clearly the time required to solve this problem to optimality is prohibitive.

Fortunately all strictly redundant bounds can be removed simultaneously without further ado. This is exploited when a strictly redundant bound is determined during the bound strengthening. When a weakly redundant bound is found by the bound strengthening we cannot determine the corresponding dependency set since implied bound can have been used in the bounding.

However, to strengthen the dual problem, it is important to determine as many simultaneously redundant bounds as possible in reasonable time. We have used the following heuristic procedure for determining a set of simultaneously weakly redundant bounds. To simplify the discussion we consider an LP with lower bounds only.

Let J be the set of variables whose lower bound can be seen to be weakly redundant by considering only a single equation and the original bounds:

$$J = \{j : \exists i \text{ such that } l_j = \min\{(b_i - \sum_{k \neq j} a_{ik} x_k)/a_{ij} : l \leq x\}\}$$

And let I be the set of equations which together with the original bounds can be used to determine a weakly redundant bound.

$$I = \{i : \exists j \text{ such that } l_j = \min\{(b_i - \sum_{k \neq j} a_{ik} x_k)/a_{ij} : l \leq x\}\}$$

Now we consider the sub-matrix A_{IJ} of the constraint matrix. Note that A_{IJ} is not necessarily quadratic, since each lower bound might be determined weakly redundant in more than one row and a doubleton row can render two lower bounds weakly redundant. If $A_{\tilde{I}\tilde{J}}$ is a quadratic sub matrix of A_{IJ} which can be reordered to lower triangular form then all the redundant bounds in $B = \{x_j \geq l_j : j \in \tilde{J}\}$ can be simultaneously removed, because these bounds does not have any circular dependencies. The problem of determining a maximum size (permuted) triangular sub matrix of a sparse matrix have other applications as well, see for example [11, 4], and efficient heuristics have been developed for it. In the implementation, a modification of the SPK1 heuristic, due to Stadtherr and Wood [15], have been used.

C.5.2 Bound replacements

Often a weak redundancy is detected only with the help of implied bounds on the other variables and might in fact be dependent on these implied bounds as the following example shows. For the system

$$\begin{aligned} x_1 + x_2 &= 10 \\ x_1, x_2 &\geq 0 \end{aligned}$$

none of the lower bounds are redundant, however it is easy to derive the following implied bounds $x_1 \in [0, 10], x_2 \in [0, 10]$. Using the implied bound $x_2 \leq 10$ the lower bound on x_1 becomes redundant and we arrive at the equivalent system

$$\begin{aligned} x_1 + x_2 &= 10 \\ 0 \leq x_2 &\leq 10 \end{aligned}$$

If this was a part of a larger system, it could be used to eliminate x_1 which has now become implied free. Another use of the new equivalent system is to exploit the corresponding changes of the dual bounds. In the first formulation the bounds of the dual variables corresponding to x_1 and x_2 are $z_1 \in [0, \infty]$, $z_2 \in [0, \infty]$, $w_1 \in [0, 0]$ and $w_2 \in [0, 0]$, but for the alternative formulation we get $z_1 \in [0, 0]$, $z_2 \in [0, \infty]$, $w_1 \in [0, 0]$ and $w_2 \in [0, \infty]$. It is possible that the second set of dual bound can be beneficial for the dual bound strengthening.

C.6 Implementation Issues

The engine of LP reduction is the bound strengthening. We will discuss how the myopic bound strengthening can be efficiently implemented. Our discussion is focused on the primal bound strengthening, but it is valid for the dual bound strengthening as well.

The fixing of variables in singleton rows and in forcing rows both come out as special cases of general myopic bound strengthening. Nevertheless we include tests for these special cases because such rows can be treated more efficiently and with less exposure to rounding error. The algorithm for strengthening the variables appearing in a single row is outlined below.

if Rowlength[i]=0 **then**
  mark this row as eliminated
else
  if Rowlength[i]=1 **then**
    mark this row as eliminated and fix the corresponding variable (updating Rowlength)
  else
    L = $\sum_{j\in\{j:A_{ij}>0\wedge\underline{x}_j>-\infty\}} A_{ij}\underline{x}_j + \sum_{j\in\{j:A_{ij}<0\wedge\overline{x}_j<\infty\}} A_{ij}\overline{x}_j$
    Linf = $|\{A_{ij}>0\wedge\underline{x}_j=-\infty\}|+|\{A_{ij}<0\wedge\overline{x}_j=\infty\}|$
    U = $\sum_{j\in\{j:A_{ij}>0\wedge\overline{x}_j<\infty\}} A_{ij}\overline{x}_j + \sum_{j\in\{j:A_{ij}<0\wedge\underline{x}_j>-\infty\}} A_{ij}\underline{x}_j$
    Uinf = $|\{A_{ij}>0\wedge\overline{x}_j=\infty\}|+|\{A_{ij}<0\wedge\underline{x}_j=-\infty\}|$
    if L = b[i] $\wedge$ Linf =0 **then**
      forall variables j of row i **do** fix j at the appropriate bound (updating Rowlength)
    else
      if U = b[i] $\wedge$ Uinf = 0 **then**
        forall variables j of row i **do** fix j at the appropriate bound (updating Rowlength)
      else
        if Linf = 1 $\wedge$ Uinf $\geq$ 1 **then**
          find j such that $j\in\{j:(A_{ij}>0\wedge\underline{x}_j=-\infty)\vee(A_{ij}<0\wedge\overline{x}_j=\infty)\}$
          if $a_{ij}>0$ **then**
            upper=$(b[i]-L)/a_{ij}$; **if** upper $<\overline{x}_j$ **then** $\overline{x}_j$ = upper ;
          else
            lower=$(b[i]-L)/a_{ij}$; **if** lower $>\underline{x}_j$ **then** $\underline{x}_j$ = lower ;
          end if
        end if
        if Uinf = 1 $\wedge$ Linf $\geq$ 1 **then**
          find j such that $j\in\{j:(A_{ij}>0\wedge\overline{x}_j=-\infty)\vee(A_{ij}<0\wedge\underline{x}_j=\infty)\}$
          if $a_{ij}>0$ **then**
            lower=$(b[i]-U)/a_{ij}$; **if** lower $>\underline{x}_j$ **then** $\underline{x}_j$ = lower ;
          else
            upper=$(b[i]-U)/a_{ij}$; **if** upper $<\overline{x}_j$ **then** $\overline{x}_j$ = upper ;
          end if
        end if
        if Linf = 0 **then**

          for all variables j of row i **do**
            if $a_{ij}>0$ **then**
              upper=$(b[i]-L)/a_{ij}+\underline{x}_j$; **if** upper $<\overline{x}_j$ **then** $\overline{x}_j$ = upper ;
            else
              lower=$(b[i]-L)/a_{ij}+\overline{x}_j$; **if** lower $>\underline{x}_j$ **then** $\underline{x}_j$ = lower ;
            end if
          end for
        end if
        if Uinf = 0 **then**
          for all variables j of row i **do**
            if $a_{ij}>0$ **then**
              lower=$(b[i]-U)/a_{ij}+\overline{x}_j$; **if** lower $>\underline{x}_j$ **then** $\underline{x}_j$ = lower ;
            else
              upper=$(b[i]-U)/a_{ij}+\underline{x}_j$; **if** upper $<\overline{x}_j$ **then** $\overline{x}_j$ = upper ;
            end if
          end for
        end if
      end if
    end if
  end if
end if

By treating all variables of a row consecutively, computational savings are achieved by first computing the minimum and maximum possible level, L and U, of the row subject to the current bounds. Even if L=$-\infty$ and U=∞ it might still be possible to find a new finite bound based on this row. A simple example of this is the row $x_1-x_2=10$ together with the bounds $0\leq x_1\leq\infty$, $0\leq x_2\leq\infty$ which can be strengthened to $10\leq x_1\leq\infty$, $0\leq x_2\leq\infty$. To use this kind of bound strengthening, it is necessary to count the number of infinite bounds appearing in the expression for calculating L and U. In the algorithmic sketch above, L and U is used for the finite part of the minimum and maximum level of the i'th row and Linf and Uinf are used for counting the infinities. When the upper or lower bound of a row becomes finite (Linf=0 or Uinf=0) the other bound will also become finite after using this row for bound strengthening. When this happens we mark the row as 'finite' and are able to use a slightly faster procedure next time the row is used for bound strengthening, since we do not have to count the infinities anymore.

In [1, 8, 5] a number of passes are performed where in each pass all rows are submitted to this procedure. Typically the passes continue until no more progress is made or a predetermined maximum number of passes is reached. The importance of limiting the number of passes comes from the fact that all rows are considered in each pass, even though typically only a small number of rows contribute to the progress in the later passes. In our implementation this is avoided by only considering rows which contain variables with bounds that have been strengthened since the row was last visited. (We refer to these rows as the active rows.) This can be achieved by marking the rows in which a variable appears as being active each time one of its bound is strengthened (which can be done efficiently since we store the coefficient matrix both by rows and by columns). In the initial passes the number of active rows is typically very high so the time spent in marking rows is not compensated by an eventual small saving in the next pass. Therefore we omit marking active rows in the first pass. In the second pass all rows are examined but we start to mark active rows such that in the third and all following passes only active rows are examined. The bound strengthening stops when there are no more active rows.

Perhaps the most important new feature of our LP reduction algorithm is the use of full primal-dual interaction. By this we mean that redundant bounds in both the primal and dual problems are used to fix variables in its dual problem and that the bounds which are changed in this way are used to strengthen other bounds further, if possible. This can be implemented efficiently since we avoid visiting all the rows which are not affected by the fixed variables.

It might be possible to obtain further improvements by updating the values of L and U for each row in the later passes instead of recalculating them each time the row is visited. However the improvements, if any, will be marginal since the overwhelming majority of the work is carried out in the first few passes where the high number of strengthened bounds makes the updating scheme unattractive.

The description of the algorithm contains a lot of tests on the sign of matrix elements. By reorganizing the storage of the coefficient matrix such that the positive non zero elements precede the negative non zero elements in each row, these tests are avoided in the implementation. By replacing loops like


```
for (j=jb; j!=je; j++) {
   // here we must check the sign of TElm[j]
}
```

by two consecutive loops like

```
for (j=jb; j!=jm; j++) {
   // here we know that TElm[j] > 0
}
for ( ; j!=je; j++) {
   // here we know that TElm[j] < 0
}
```

the tests are avoided without any overhead except for the initial reorganization.

We have described LP reduction in terms of a problem in standard form, which eases the presentation considerably since we do not constantly have to refer to the type of each constraint. However this does not mean that we have to add slack variables in the actual implementation. Time is saved by noting that if bound strengthening is performed on a '$\leq$ row', only the lower bound of the row is relevant and for a '$\geq$ row' only the upper bound is relevant.

C.7 Fourier-Motzkin Reduction

Fourier-Motzkin Elimination is an old technique for eliminating variables from a system of linear inequalities. It is well known that repeated application of Fourier-Motzkin elimination can be used to solve linear programs. This procedure is generally not practical since the number of intermediate inequalities can grow exponentially. However, depending on the non-zero structure and sign pattern of the constraint matrix, the elimination of some variables can lead to genuine reductions. To make make this paper self contained we will now give a brief exposition of Fourier-Motzkin elimination.

Consider the inequality system $Ax \leq b$. To eliminate a variable, say x_1, we partition the inequalities of $Ax \leq b$ into three sets, I_+, I_- and I_0, according to the sign of the coefficient of x_1 in the inequality. Our goal is to derive an inequality system in the remaining variables $x_2, x_3, ..., x_n$ which is consistent (that is, admits a solution) if and only if the original system is. For each $i \in I_+$, $a_{i1} > 0$ and we get an upper bound on x_1

$$x_1 \leq (b_i - a_{i2}x_2 - ... - a_{in}x_n)/a_{i1}$$

and for each $i \in I_-$ (since $a_{i1} < 0$) we get a lower bound on x_1

$$(b_i - a_{i2}x_2 - ... - a_{in}x_n)/a_{i1} \leq x_1$$

Clearly it is only possible for x_1 to satisfy these bounds if the largest of the lower bounds is less than or equal to the smallest of the upper bounds, i.e. if

$$\max_{i \in I_-}(b_i - a_{i2}x_2 - ... - a_{in}x_n)/a_{i1} \leq \min_{i \in I_+}(b_i - a_{i2}x_2 - ... - a_{in}x_n)/a_{i1}$$

We note that this is equivalent to the inequality system

$$(b_i - a_{i2}x_2 - ... - a_{in}x_n)/a_{i1} \leq (b_k - a_{k2}x_2 - ... - a_{kn}x_n)/a_{k1} \quad \forall i \in I_- \ \forall k \in I_+$$

and conclude that $Ax \leq b$ is consistent if and only if the system consisting of these inequalities together with the original inequalities not including x_1,

$$a_{i2}x_2 + ... + a_{in}x_n \leq b_i \quad \forall i \in I_0,$$

is consistent.

In general the elimination of a variable can increase the number of inequalities considerably. If for example the original system consists of 25 inequalities with $|I_+| = |I_-| = 10$ and $|I_0| = 5$ the elimination will result in a system with $|I_0| + |I_+| * |I_-| = 105$ inequalities (but of course many of these might turn out to be redundant). If on the other hand $|I_+|$ or $|I_-|$ is equal to one the number of inequalities of the system will decrease by one as a result of the elimination of the variable (we go from $|I_0| + |I_+| + |I_-|$ to $|I_0| + |I_+| * |I_-|$ inequalities).

To apply Fourier-Motzkin reduction we must consider P as a system of inequalities:

$$\begin{aligned} -z + c^T x &\leq 0 \\ Ax &\leq b \\ -Ax &\leq -b \\ -x &\leq 0 \end{aligned}$$

where z is a new variable to be minimized (and is not subject to elimination). If the original formulation of P contains inequalities, we obtain a smaller system by using these inequalities directly in the inequality formulation of P instead of adding slack variables first. We do this to save time although it is easy to see that Fourier-Motzkin reduction would eliminate the added slack variables and extra inequalities again. (That is, eliminating the variables, s, from the system $\{-z + c^T x \leq 0, -Ax - s \leq -b, Ax + s \leq b, -x \leq 0, -s \leq 0\}$ results in the system $\{-z + c^T x \leq 0, Ax \leq b, -x \leq 0\}$)

We use the term Fourier-Motzkin reduction to refer to Fourier-Motzkin elimination when applied to a variable for which $|I_+|$ or $|I-|$ is one. An iteration is summarized in Figure 3, where

$$C_j^+ = \{i : 1 \leq i \leq m, A_{ij} > 0\} \text{ for } \ j = 1, ..., n$$

and

$$C_j^- = \{i : 1 \leq i \leq m, A_{ij} < 0\} \text{ for } \ j = 1, ..., n$$

are used to denote the set of rows where the j'th variable appears with a positive and negative coefficient respectively.

chose a variable q such that $|C_q^+| = 1$ or $|C_q^+| = 1$
if $|C_q^+| = 1$ **then**
  let p be the row where $A_{pq} > 0$
  for $k \in C_q^-$ **do**
    "$A_k x \leq b_k \mathrel{+}= \frac{-A_{kq}}{A_{pq}}(A_p x \leq b_p)$"
  end for
end if
if $|C_q^-| = 1$ **then**
  let p be the row where $A_{pq} < 0$
  for $k \in C_q^+$ **do**
    "$A_k x \leq b_k \mathrel{+}= \frac{-A_{kq}}{A_{pq}}(A_p x \leq b_p)$"
  end for
end if
remove $A_p x \leq b_p$ from the inequality system

Figure C.3: An iteration of Fourier-Motzkin reduction

The notation used within quotes above is slightly non-standard, but should be straight forward to understand - the k'th inequality is modified by adding a constant $(\frac{-A_{kq}}{A_{pq}})$ times the p'the inequality.

Inequalities which are removed from the system by Fourier-Motzkin reduction are used to reconstruct the value of the eliminated variables.

We incorporate a simple test for redundancy of the modified inequalities based on the original bounds of the variables. If a (non redundant) singleton inequality results we can use it for strengthening the bound on the variable, but otherwise no bound strengthening is used.

The number of fill ins generated in an iteration is bounded by $(R_p - 1)|C_q^+||C_q^-|$, where $R_p = |\{j : 1 \leq j \leq n, A_{pj} \neq 0\}|$. This bound is used for selecting the variable, q, to eliminate and can also be used to refuse to eliminate a variable if to many new non zero elements could be generated as a result.

C.8 A numerical example

Consider the following LP and its associated dual problem.

$$\begin{aligned} \textbf{P:}\ \text{minimize}\ & -x_1 - 2x_2 \\ x_1 + x_2 + x_3 &= 4 \\ -x_1 + 2x_2 + x_4 &= 2 \\ x &\geq 0 \end{aligned}$$

$$\begin{aligned} &\textbf{D:}\ \text{maximize}\ 4y_1 + 2y_2 \\ &y_1 - y_2 + z_1 = -1 \\ &y_1 + 2y_2 + z_2 = -2 \\ &y_1 + z_3 = 0 \\ &y_2 + z_4 = 0 \\ &z \geq 0 \end{aligned}$$

with primal and dual feasible regions as illustrated in Figure 1.

Myopic bound strengthening on the primal problem provides the following implied bounds: $x_1 \in [0,4], x_2 \in [0,3], x_3 \in [0,4], x_4 \in [0,6]$. For the dual problem the following implied bounds are obtained by myopic bound strengthening: $y_1 \in [-\infty,-1], y_2 \in [-\infty,0], z_1 \in [0,\infty], z_2 \in [0,\infty], z_3 \in [1,\infty], z_4 \in [0,\infty]$.

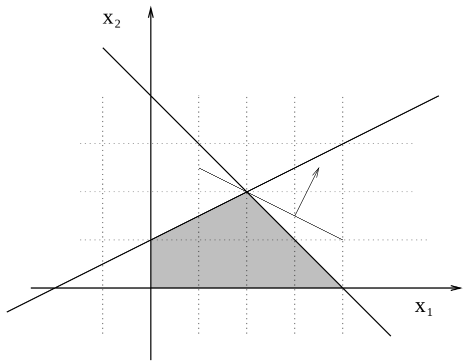


Figure C.4: Primal feasible region (projected onto x_1, x_2 space)

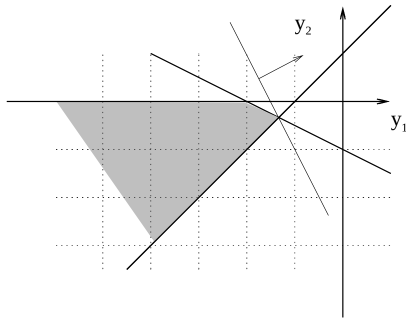


Figure C.5: Dual feasible region (projected onto y_1, y_2 space)

Using $z_3 \in [1, \infty]$ which implies that $z_3^* > 0$, the associated primal variable x_3 can be fixed at its lower bound, i.e. $x_3 \in [0, 0]$. Since a primal bound was changed as a result of the dual bound strengthening we re-apply primal bound strengthening according to our full primal-dual interaction strategy. Using the new upper bound on x_3 we are able strengthen the bounds on the remaining primal variables. From $x_1 = 4 - x_2$ we get new implied bounds on x_1, $x_1 \in [1, 4]$. As a consequence z_1 can be fixed at zero. Although this in fact has consequences for other dual bounds, these strengthenings can not be made by using only myopic bound strengthening. Since no more bounds are changed the bound strengthening terminates.

Noting that the original bounds on x_1, $[0, \infty]$ are redundant, we eliminate x_1 using the substitution $x_1 = 4 - x_2 - x_3$ resulting in the following reduced problem:

$$\begin{array}{rrl} \textbf{P':} & \text{minimize} & -4 - x_2 \\ & & 3x_2 + x_4 = 6 \\ & & x \geq 0 \end{array} \qquad \begin{array}{rl} \textbf{D':} & \text{maximize } 6y_2 \\ & 3y_2 + z2 = -1 \\ & y_2 + z_4 = 0 \\ & z \geq 0 \end{array}$$

which in fact is trivially solved by problem reduction.

For comparison, we submitted this tiny problem to CPLEX 6.5. Its reduction algorithm did not find any reduction of this problem. Because there is no description of the CPLEX reduction algorithm available, it is hard to say precisely why this is the case. But since the LP reduction of CPLEX does use Gaussian elimination of implied free variables we conclude that it fails to detect the redundancy of $x_1 \geq 0$. We have two possible explanations of this:

1. Bound strengthening in CPLEX may only be performed on rows with finite lower and upper bounds. In the dual bound strengthening of our problem we used $y_1 - y_2 + z_1 \geq -1$ together with $y_1 \leq 0$, $y_2 \leq 0$, $z_1 \geq 0$ to obtain a new upper bound, -1, on y_1 (which in turn was used to determine the implied bound $z_3 \geq 1$) although both the lower and upper bound of the row are infinite.

2. CPLEX may not use full primal-dual interaction. This means that primal bound strengthening is not re-applied after the slack variable x_3 is fixed to zero, so the indirect implications of fixing x_3 are never harvested.

Only the implementors of CPLEX know which, if any, of these two possibilities is the correct.

Alternatively, Fourier-Motzkin reduction can be applied to the problem of the example, as illustrated in the table below. The variable x_1 appears with a positive coefficient in only one inequality and can therefore be eliminated without introducing new inequalities. The result of projecting out x_1 is shown in column 1 of the table. After simplifying and removing redundant simple bounds we get column 1', which shows us that x_2 can now be eliminated and we obtain column 2 and 2'..

Initial system	1	1'	2	2'
$-z - x_1 - 2x_2 \leq 0$	$-z - x_2 \leq 4$	$-z - x_2 \leq 4$	$-z \leq 6$	$-z \leq 6$
$x_1 + x_2 \leq 4$	removed			
$-x_1 + 2x_2 \leq 2$	$3x_2 \leq 6$	$x_2 \leq 2$	removed	
$-x_1 \leq 0$	$x_2 \leq 4$	redundant		
$-x_2 \leq 0$	$-x_2 \leq 0$	$-x_2 \leq 0$	$0 \leq 2$	redundant

Table C.1: Progression of Fourier-Motzkin reduction of the example problem

Working backwards in the table we get the optimal solution $z = -6$, $x_2 = 2$ and $x_1 = 2$.

C.9 Computational Experiments

With the impressive speed of todays LP solvers, it has become more difficult to come up with new reduction algorithms which are able to decrease the total solution time. To be used advantageously in connection with a modern fully tuned solver code, the code for reduction must be equally well polished. The main purpose of this section is to reveal if it is possible to obtain substantial further reduction by applying some of the aggressive strategies which have been described. Further investigations must then decide if the execution time can be made sufficiently small to be worthwhile.

The Netlib [7] set of LP test problems has served for benchmark purposes in the LP community for a long time and by todays standards the problems are very small. The impressive improvements of LP solvers since the compilation of the Netlib LP test problems have generated the need for a new set of bigger LP test problems. For reporting results on algorithmic and implementational improvements of LP software, researchers have considered a variety a large real world LP's and many of these have been made publicly available for down-loading. For our tests we have used a collection of 26 of these problems (including some of the bigger Netlib problems) used in a recent LP solver benchmark by Mittelmann [14]. These problems have varying characteristics with respect to density, aspect ratio, application area, etc. The problems are available form different sources as described in [14], but all of them can also be obtained from `http://www.imm.dtu.dk/~thh/big26`. Some of the problems contain explicitly fixed variables: mod2 (1652), pilot87 (220), sgpf5y6 (61), watson-1 (1537) and world (848). The sizes of these problems which we shall refer to as the Big26 LP Testset are shown in Table C.2.

To get an impression of the reductions possible using only a very limited LP reduction procedure, the problems were submitted to the following *trivial* LP reduction algorithm. Only primal reductions were considered (that is, c was not used). The reductions included were elimination of empty rows, elimination of explicitly fixed variables, elimination of variables fixed by singleton rows and elimination of variables fixed by forcing rows. No bound strengthening was used, except when a variable was fixed by a forcing row. In this case the bound at which the variable was fixed was used both as a a lower and upper bound in the detection of further forcing rows. The number of rows and columns which were removed from each problem by using only these trivial reductions are shown in the last two columns of Table C.2. We see that some problems were considerably reduced by this extremely fast reduction scheme. But as expected, the actual reductions obtained are highly problem dependent.

Since our principal goal is to examine ways of reducing LP problems further than possible using only the standard techniques, we considered reduced problems obtained with the CPLEX version 6.5's problem reduction, for the remaining computational tests. Each problem from the Big26 LP Testset was reduced by CPLEX version 6.5 with default options. The size of the resulting problems is shown in Table C.4.

We see that only a few problems were reduced substantially more by

Problem	m	n	nz	slacks	**rows**	**cols**
baxter	27441	30733	111576	15605	3210	176
dano3mip	3202	15851	81633	1978	0	0
dbir2	18906	45877	1158159	18522	4704	7039
dfl001	6071	12230	35632	0	0	0
fome12	24284	48920	142528	0	0	0
gen4	1537	4298	107103	1	0	0
ken-18	105127	154720	358192	21	26247	26247
l30	2701	16281	52070	901	0	0
lp22	2958	16392	68518	2958	0	0
mod2	34774	66409	254787	34681	829	2660
nsct2	23003	37563	697738	22582	7374	11051
nug15	6330	22275	94950	0	0	0
nw04	36	87482	636666	0	0	0
nw14	73	123409	904910	0	0	0
osa-60	10280	243246	1408073	10280	0	0
pds-40	66844	217531	466800	4672	2568	3146
pilot87	2030	6680	74949	1797	20	258
qap12	3192	8856	38304	0	0	0
rlfprim	58866	62716	320591	54664	0	0
route	20894	43019	206782	19096	0	0
self	960	8324	1149805	960	0	0
seymour	4944	6316	38493	4944	117	234
sgpf5y6	246077	312252	831688	3618	102433	102494
stormG2	66185	172431	433256	14935	625	375
watson-1	201155	386992	1055093	3065	19933	92126
world	34506	67147	255161	34413	812	1840

Table C.2: Trivial reductions in problems from the Big26 LP Testset. The figure shows the number of rows, m, columns, n, and non zero elements of the coefficient matrix, nz, for each of the problems. The number of slack variables, slacks, is included in both n and nz. The columns **rows** and **cols** show the number of rows and columns eliminated by trivial LP reduction.

Problem	m	n	nz	slacks
baxter	22849	29053	103790	14329
dano3mip	3187	15836	81588	1963
dbir2	7230	31898	1085755	7033
dfl001	3866	9650	32232	587
fome12	15464	38600	128928	2348
gen4	1475	4174	104237	1
ken-18	78846	128418	293658	114
l30	2698	16264	52000	901
lp22	2872	11565	63053	2872
mod2	27192	54270	149013	27141
nsct2	7797	18947	619756	7650
nug15	6330	22275	94950	0
nw04	36	82977	601078	0
nw14	73	113014	829173	0
osa-60	10209	234334	594462	10209
pds-40	64115	214224	456665	4085
pilot87	1966	6362	72145	1775
qap12	3192	8856	38304	0
rlfprim	36294	40240	204167	36194
route	20894	43019	206782	19096
self	960	8324	1149805	960
seymour	4808	5961	37855	4808
sgpf5y6	143546	209948	504816	3915
stormG2-125	56286	162405	400159	23786
watson-1	119782	233426	688054	4601
world	27067	55876	149179	27016

Table C.3: The Big26 Testset, reduced by CPLEX 6.5. default reduction. The table contain the size of the reduced problems: m is the number of rows, n is the number of columns and nz is the number of non-zero elements in the coefficient matrix A. The number of slack variables needed to convert the problem into standard form is shown in the fourth column. The slack columns are included in both n and nz.

CPLEX 6.5 than by the trivial reductions mentioned before.

In Table C.4 we present the results of applying further reduction to the problems reduced by CPLEX. All 26 problems were submitted to the reduction algorithm, but the problems not mentioned in Table C.4 were not reduced further.

Problem	f	s	d	br	time	f*	s*	d*
baxter	0	0	12	5405	0.74	0	0	12
dano3mip	0	0	23	1168	0.44	0	0	23
dfl001	1	8	188	247	0.18	1	8	60
fome12	4	32	752	988	0.79	4	32	240
ken-18	0	1	5298	56282	2.61	0	1	779
mod2	3068	366	1682	8850	1.00	924	233	671
nsct2	1	1	0	82	2.76	1	1	0
pds-40	0	85	5843	48883	3.85	0	85	19
pilot87	0	0	213	918	0.35	0	0	122
seymour	9	1	0	192	0.19	0	0	0
sgpf5y6	717	0	23467	43583	3.98	0	0	0
stormG2-125	0	0	8500	9188	2.70	0	0	125
world	2888	423	1715	8642	0.99	730	261	587

Table C.4: Results of full primal dual myopic bound strengthening for reduced Big26 LP test problems. (Problems not mentioned were not reduced further.) f = number of fixed variables, s = number of implied free singleton columns, d = number of implied free doubleton columns, br = number of implied free columns with 3 or more entries. With and without (in the columns marked with an *) removal of weakly redundant bounds.

The beneficial effect of the removal of weakly redundant bounds shows clearly in Table C.4. The removal of weakly redundant bounds has two desirable effects, 1) the number of detected implied free variable is increased, 2) the corresponding strengthening of the dual bounds sometimes results in an increased number of fixed primal variables.

It should be mentioned that some of the problems can be reduced much further by CPLEX if non-default parameters are used. In CPLEX a parameter (with default value 10) is used to limit the application of Gaussian elimination. If an application of Gaussian elimination would result in more new nonzeros than specified by the value of this parameter the elimination is rejected. This restriction also applies to implied free column doubletons although the actual number of non zeros in the reduced problem can never increase in this case. Elimination of an implied free column doubleton involves the removal of one of the two involved rows, however the removed row must be retained such that the value of the eliminated variable can be calculated after the reduced problem is solved. Apparently the non zeros 'saved' by the removal of a row is not discounted in the non zero count used by CPLEX to determine the eligibility of an elimination. By raising the fill in threshold parameter from 10 to 1000000 it was possible to reduce the number of rows and columns of some problems (most notably ken-18, pds-40, sgpf5y6 and stormG2-125) considerably without increasing the number of nonzeros. In Table C.4 it is seen that these problems are the ones which have the highest number of implied free doubletons. (On other problems, setting the fill in threshold parameter to high, can result in an explosion of the number of non zeros, this is the case with the problem l30 which is 'reduced' to 901 rows and 14446 variables but no less than 1722998 non zeros.) The two problems which were reduced most by the our primal dual reduction algorithm (mod2 and world) almost did not benefit from the more aggressive fill in threshold parameter setting. In the case of mod2, 1 extra variable and constraint were removed and in the case of world 10 extra variables and constraints were removed.

The further reductions of the problems mod2 and world as compared to CPLEX clearly originates from the use of primal information to fix dual variables. With this option disabled no variables of the two mentioned problems are fixed by the reduction algorithm.

C.9.1 Implementation issues

In this subsection we investigate the effect on the execution time for bound strengthening of including the preprocessing step to separate negative and positive entries described in section C.6. This technique was only implemented for the primal bound strengthening so the execution times reported in Table C.5 only includes an initial round of primal bound strengthening. In this experiment a pass visits all rows and the passes continues as long as new fixed variables are discovered. The actual number of passes is included in the table.

The strategy of separating positive and negative matrix entries is meant for situations where the matrix must be examined repeatedly. For the problems

Problem	nosep	sep 1	sep 2	p	nzpr
baxter	86	89	8	4	4.1
dano3mip	9	8	5	1	25.5
dbir2	319	284	43	2	61.3
dfl001	6	7	5	1	5.9
fome12	24	26	11	1	5.9
gen4	11	10	4	1	69.7
ken-18	415	436	31	6	3.4
l30	5	4	3	1	19.3
lp22	9	8	3	1	23.2
mod2	111	112	15	3	7.3
nsct2	193	172	29	2	30.3
nug15	10	13	6	1	15.0
nw04	184	171	21	1	17685.2
nw14	258	244	30	1	12396.0
osa-60	269	271	47	1	137.0
pds-40	213	202	30	2	7.0
pilot87	22	19	4	2	36.9
qap12	8	5	2	1	12.0
rlfprim	44	42	15	1	5.4
route	39	40	11	1	9.9
self	157	139	38	1	1197.7
seymour	13	12	2	2	7.8
sgpf5y6	254	258	68	2	3.4
stormG2-125	147	134	28	2	6.5
watson-1	342	335	70	2	5.2
world	106	102	14	3	7.4

Table C.5: Comparison of primal bound strengthening time (in milliseconds) with and without prior separation of negative and positive entries. **nosep:** bound strengthening time without separation. **sep 1:** time of separation. **sep 2:** bound strengthening time with separation. **p:** number of passes. **nzpr:** average number of non zeros per row.

which only require only a single pass (to determine that no reductions are possible) it is not surprising that the total time with separation (sep1 + sep2) exceeds the time without separation. However it can be observed in Table C.5 that even when disregarding the preprocessing time (sep 1) the bound strengthening time increases for many problems when using separation. This is mostly the case for problems with a small average number of non zeros per row and is likely caused by some overhead in setting up an extra loop and/or the extra memory reference to get the position in the array where the sign changes.

Overall the results of applying separation are very disappointing. The only problems where some benefit can be expected are problems with a relatively high number of non zeros per rows and which requires many passes through the matrix.

C.9.2 Fourier-Motzkin reduction

Table C.6 show the results of applying Fourier-Motzkin reduction to the reduced problems from the Big26 Testset. The table does not contain execution times. For the few problems which are reduced considerably the execution time was very high due to an inefficient implementation with repeated allocation of small chunks of memory. On the other hand it is rather fast to determine if Fourier-Motzkin reduction can be applied so for the problems that can not be reduced we do not spend much time to determine this.

As noted in the previous section, the problem sgpf5y6 can be reduced further by CPLEX by using a non default parameter setting. The size of the resulting reduced problem is m=19499 n=42935 nz=113162. This problem appear in the table as sgf5y6a. We see that even this very reduced problem can be further reduced by Fourier-Motzkin reduction.

C.9.3 Egocentric bound strengthening

The application of egocentric bound strengthening requires the solution of a huge number of small LP's. As an example, consider the problem 80bau3b from the Netlib Testset. After reduction by CPLEX 6.5 the problem has 1965 rows, 10645 variables and 20946 non zeros. One round of egocentric

Problem	n	m	m'	nz	r. m'	r. nz	elim
dbir2	24866	39165	7428	1215876	7424	1215854	2
dfl001	9064	16228	7146	63550	7135	63506	12
fome12	36253	64909	28581	254197	28537	254021	48
mod2	27130	66366	27244	142467	27104	143573	513
pilot87	4588	8325	2158	74125	2142	74157	39
sgpf5y6	206034	489211	283178	1025206	251353	888498	31825
sgpf5y6a	39021	74104	35084	218431	24951	167794	10133
stormG	138620	227525	88787	713618	83412	688493	4750
world	28861	69013	27119	143583	26974	144901	630

Table C.6: Results of Fourier-Motzkin reduction for reduced Big26 test problems. (Only the problems for which Fourier-Motzkin reduction could be applied at least once appears in the table.) The table shows the size of the inequality formulation of the problem (n: number of variables, m: number of inequalities, m': number of inequalities excluding simple bounds, nz: number of non zeros) and the size of the reduced problem (r. m' and r. nz) as well as the number of eliminated variables (elim).

bound strengthening of the primal variables of this problem thus requires the solution of 21290 LP's with approximately 2 rows on average (around 40% of the problems has only one row and the maximum number of rows is 11). An implementation of egocentric bound strengthening using the CPLEX callable library for solving these small LP's required 220 seconds in total, while the original problem can be solved in 1.6 seconds. It seems that the invocation of the CPLEX solver induces an overhead, which although it can be neglected for bigger problems, becomes a serious liability when thousands of tiny problems must be solved.

In Table C.7 the effect of different primal bound strengthening strategies are compared. In order to be able to conduct the experiments in a reasonable amount of time a number of very small Netlib problems were selected. As a measurement of the effect of the bound strengthening we use the number of fixed variables and the number of finite variable bounds which have been strengthened. f_1 and b_1 are the number of fixed variables and finite bounds which have been strengthened using myopic bound strengthening. f_2 and b_2 are the number of fixed variables and finite bounds which have been strengthened using egocentric bound strengthening. f_3 and b_3 are the number of fixed variables and finite bounds which have been strengthened using full primal bound strengthening.

It is seen that egocentric bound strengthening is almost as effective as full bound strengthening (i.e. the best possible using only primal information). On many problems the myopic bound strengthening comes close as well, however the results indicate that a certain room for obtaining improvements using a more aggressive bound strengthening strategy is present. Whether such more aggressive strategies can be implemented sufficiently efficient to become worthwhile is still an open question which requires further investigation.

To use egocentric bound strengthening in practice it will be necessary to develop an LP solver specialized for tiny problems and working directly on the data-structure of the original problem.

To benefit from a stronger bound strengthening it is crucial to get rid of as many weakly redundant bounds as possible. I conducted some experiments with full bound strengthening on some of the smaller Netlib problems. Surprisingly often no, or only slight, improvements in the number of eliminated columns and rows were obtained compared to myopic bound strengthening. However, if our heuristic for removing weakly redundant bounds (described in section 5) was applied first, most of the problems were eliminated by LP reduction based on full bound strengthening.

C.10 Conclusion

The ideas of LP reduction have been exposed in a unified manner. Modelling formulation practice leading to easily detectable reductions were identified. The importance of passing information from the primal to the dual problem and the ability to detect weakly redundant simple bounds of the formulation was stressed.

Two of the twenty six big test-problems were reduced substantially more than otherwise possible by using these new techniques. While two out of twenty six might not be impressive, it is still very beneficial to use full primal-dual interaction provided that no serious execution penalty is paid for the problems which does not benefit from using this scheme. In fact this is the case, since bound strengthening is only re-applied if some bound has been changed. The extra time used for the problems which do benefit from

problem	f_1	b_1	f_2	b_2	f_3	b_3
adlittle	1	0	1	0	1	0
afiro	0	0	0	2	1	3
agg	51	0	137	284	137	360
bandm	55	63	72	106	72	127
blend	0	0	0	1	0	1
boeing2	0	8	26	9	40	47
bore3d	116	9	148	93	148	104
brandy	37	32	64	40	65	56
capri	32	23	32	139	32	165
e226	8	6	33	11	34	19
etamacro	140	46	146	66	146	77
finnis	62	24	82	57	82	71
grow7	0	18	0	21	0	22
israel	0	0	0	7	0	13
kb2	0	0	0	1	0	2
lotfi	8	9	8	9	8	9
recipe	37	44	43	58	43	65
sc105	0	0	1	0	1	0
sc205	0	0	2	0	2	0
scagr25	1	1	1	39	1	102
scagr7	1	1	1	21	1	30
scfxm1	17	19	22	25	22	101
scorpion	13	0	67	100	67	106
share1b	5	9	5	23	5	34
share2b	0	1	0	3	0	28
stocfor1	11	26	11	26	11	26
vtpbase	53	44	157	99	157	123

Table C.7: Comparison of primal bound strengthening strategies. 1. miopic, 2. egocentric, 3. full

full primal-dual interaction is extremely low due to the implementation of the bound strengthening (described in section 7) which only visit active rows.

The goal (dream) of finding a computationally feasible (fast) LP reduction algorithm capable of eliminating much more rows and columns than possible using only myopic bound strengthening was not fulfilled. But further research should be conducted before the goal is abandoned. To achieve this goal a very efficient implementation of some bound strengthening capable of looking at more than one row at a time is needed. Whatever strategy is selected, it is possible that time is wasted for some problems where this strategy does not bring any further reductions. But if a reasonable number of practical problems are substantially reduced the goal has been fulfilled.

Bibliography

[1] E. D. Andersen and K. D. Andersen. Presolving in linear programming. *Mathematical Programming*, 71:221–245, 1995.

[2] R. E. Bixby. Progress in linear programming. *ORSA Journal on Computing*, 6(1):15–22, 1994.

[3] A. L. Brearly, G. Mitra, and H. P. Williams. Analysis of mathematical programming problems prior to applying the simplex algorithm. *Mathematical Programming*, 15:54–83, 1975.

[4] R. Fletcher and J. A. J. Hall. Ordering algorithms for irreducible sparse linear systems. *Annals of O.R.*, 43:15–32, 1993.

[5] R. Fourer and D. M. Gay. Experience with a primal presolve algorithm. Technical report, AT& T Bell Laboratories, 1993.

[6] T. Gal. Weakly redundant constraints and their impact on postoptimal analyses in LP. *European Journal of Operational Research*, 1992.

[7] David M. Gay. Electronic mail distribution of linear programming test problems. *COAL Newsletter*, 13:10–12, December 1985.

[8] J. Gondzio. Presolve analysis of linear programs prior to applying an interior point method. *INFORMS Journal on Computing*, 9(1):73–91, 1997.

[9] H. J. Greenberg. How to analyze the results of linear programs - part4: Forcing substructures. *Interfaces*, 24:121–130, 1994.

[10] H. J. Greenberg. Consistency, redundancy, and implied equalities in linear systems. *Annals of Mathematics and Artificial Intelligence*, 17:37–83, 1996.

[11] E. Hellerman and D. C. Rarick. The partitioned preassigned pivot procedure (P^4). In D. J. Rose and R. A. Willoughby, editors, *Sparse Matrices and Their Applications*, pages 67–76. Plenum Press, 1972.

[12] I. J. Lustig, R. E. Marsten, and D. F. Shanno. Interior point methods for linear programming: Computational state of the art. *ORSA Journal on Computing*, 6(1):1–14, Winter 1994.

[13] B. A. McCarl and T. H. Spreen. Applied mathematical programming using algebraic systems. `http://agrinet.tamu.edu/mccarl/regbook.htm`.

[14] H. Mittelmann. Benchmark of LP solvers on a Linux-PC. `ftp://plato.la.asu.edu/pub/lplinux.txt`.

[15] M. A. Stadtherr and S. E. Wood. Sparse matrix methods for equation based chemical process flowsheeting - I. Reordering phase. *Computers and Chemical Engineering*, 8:9–18, 1984.

[16] J. A. Tomlin and J. S. Welch. A pathological case in the reduction of linear programs. *Operations Research Letters*, 2:53–57, 1983.

Appendix D

Formulating Linear Optimization Problems in C++

Paper D:

Formulating Linear Optimization Problems in C++

Abstract

A prototype implementation of a C++ class library, FLOPC++, for formulating linear optimization problems is presented. Using FLOPC++, linear optimization models can be specified in a declarative style, similar to algebraic modelling languages such as GAMS and AMPL. While preserving the traditional strengths of algebraic modelling languages, FLOPC++ eases the integration of linear optimization models with other software components. The class library implements a full-fledged algebraic modelling language with indexed variables and constraints, repeated sums, index arithmetic and conditional exceptions. Extensive use of operator overloading provides a natural syntax for specifying model constraints.

D.1 Introduction

The appearance of algebraic modelling languages such as GAMS[2] and AMPL[6], with associated translators, has been a major factor in the practical success of applied linear optimization. Prior to the advent of the first algebraic modelling language, GAMS, the translation from an algebraic description of a model to a format recognizable by the software used to solve it was often difficult and error-prone and presented a major obstacle for the practical success of optimization models. Algebraic modelling languages provide a machine readable notation for formulating optimization models which closely resembles the algebraic notation which most mathematically trained modellers find natural. Using this notation, a text file specifying a model must be prepared. Data needed to generate an instance of this model must be provided in the same or, preferably, a separate text file using the notation of the modelling language. When the translator encounters a solve statement in the input file, a model instance is generated and passed to the underlying optimization software in the required format. After the problem has been solved, the primal and dual solutions are passed back to the modelling system where it can be used for report generation.

In realistic applications, the formulation and solution of the optimization problem is often a minor part. The model data must typically be extracted from one or more tools such as spreadsheets, databases and graphical user interfaces. And often the data must be processed by other software components both before and after the optimization step. Therefore most modelling languages include algorithmic constructs and links to other software tools. The communication between the modelling system and external software components is often file based, which limits the efficiency and flexibility. By extending the C++ language with classes for model formulation an efficient and versatile modelling environment is achieved.

D.2 Overview of existing systems

The idea of embedding algebraic modelling facilities within a programming language is not new. Nielsen [12] developed a C++ class library similar to the one described in this paper. However the practical use of this library is limited, since no facilities for indexing variables and constraints are included.

The LP-toolkit[5] from Euro-decision and the ILOG Planner [9] from ILOG are two commercial C++ class libraries for linear optimization modelling. They both use operator overloading to improve the syntax for specifying constraints. However, as we shall see, these libraries have some important omissions compared to the possibilities found in pure declarative algebraic modelling languages.

The ActiveX Mathematical Modelling Objects (AMMO) [1] from Drayton Analytics and EZMod [11] from Modellium Inc. provide modelling capabilities similar to LP-toolkit and ILOG Planner, but since they are both based on the ActiveX/Com technology from Microsoft their use is limited to Microsoft Windows platforms. Another drawback of this approach is that operator overloading is not available.

The EMOSL library [13] from Dash Associates provides facilities for querying and modifying model instance entities and solution values in terms of the names of variables, constraints and index set elements of the model. However the model must still be read from a model file, which uses the XPRESS algebraic modelling language, since no modelling facilities are provided by the library. The OptiMax2000 [10] component library from Maximal Software is similar but is based on ActiveX/Com and uses the MPL algebraic modelling language.

In the conclusion of the paper by Tebboth and Daniel [13], the authors mention the possibility of including modelling facilities in the EMOSL library: "There are two major areas in which the library could be improved. One would be to allow structural alterations to the problem, so that model entities could be added or removed within the application program, potentially allowing a model to be built up from scratch and dispensing with the model file altogether if the user desired." This has recently been achieved with the release of the XPRESS-MP Builder Subroutine Library (XBSL) [4]. XBSL is a C subroutine library and can therefore not use operator overloading, but is otherwise similar to ILOG Planner and LP Toolkit.

The recent appearance of several commercial modelling libraries, as described above, illustrate the growing awareness among mathematical programming software developers of the usefulness of such libraries. However, FLOPC++, is the first modelling library which provides a truly declarative model formulation with index based algebraic notation.

D.3 Overview of FLOPC++

The FLOPC++ class library enables algebraic modelling facilities within a C++ program. The use of indexing for repetition is an essential feature of algebraic modelling languages. Therefore FLOPC++ provides classes for data, variables and constraints with overloaded constructors for each possible dimension (currently maximum 5). Data and index sets are model independent, but variables and constraints must belong to a specific model. This is achieved by supplying a pointer to the model to which a variable or constraint belongs in the corresponding declaration.

To illustrate the use of FLOPC++ we show how a small transportation problem can be formulated and solved. The problem appears in Dantzig's classical book [3] and has also been used as an example in the GAMS tutorial [2]. The FLOPC++ formulation is shown in Figure 1.

The model is stated in a declarative manner similar to algebraic modelling languages. This is achieved by using classes for variables (`MP_variable`), data (called parameters in GAMS) (`MP_data`), objective function (`MP_objective`) and constraints (`MP_contraint`).

The use of operator overloading, which is possible in C++, improves the syntax of FLOPC++ by enabling the notation

```
sum(j,D,x(i,j)) <= capacity(i)
```

instead of something like

```
lesseq(sum(j,D,x(i,j)), capacity(i))
```

which enhances the readability of the models. Consequently the FLOPC++ formulation only differs from the corresponding GAMS formulation [2] by small syntactical differences.

In GAMS the same symbol is used for an index set and a dummy index of the set. For this reason it is necessary to create so called aliases for index sets when two or more independent indices of the same set are needed. This is avoided in FLOPC++ by distinguishing between dummy indices (objects of the class `MP_index`) and index sets. At the same time the formulation get closer to standard algebraic notation (one normally writes $\sum_{i \in S} x(i)$


```
#include "flopc++.h"

main() {
  enum  {seattle,sandiego,Number_of_suppliers};
  enum  {newyork,chicago,topeka,Number_of_consumers};

  MP_set S(Number_of_suppliers), D(Number_of_consumers);
  MP_data capacity(S), dem(D), d(S,D);

  capacity(seattle)=350;  capacity(sandiego)=600;
  dem(newyork)=325;  dem(chicago)=300;  dem(topeka)=275;

  d(seattle,newyork)= 2.5;  d(sandiego,newyork)= 2.5;
  d(seattle,chicago)= 1.7;  d(sandiego,chicago)= 1.8;
  d(seattle,topeka) = 1.8;  d(sandiego,topeka) = 1.4;

  MP_data c(S,D);
  const double f=90;

  MP_index i,j;
  forall(i,S)
    forall(j,D)
      c(i,j) = f * d(i,j) / 1000;

  MP_model m;
  MP_variable  x(&m,S,D);
  MP_constraint supply(&m,S), demand(&m,D);
  MP_objective total_cost;

  forall(i,S)
    supply(i) = sum(j,D,x(i,j)) <= capacity(i);

  forall(j,D)
    demand(j) = sum(i,S,x(i,j)) >= dem(j);

  total_cost = sum(i,S, sum(j,D, c(i,j)*x(i,j)) );

  m.minimize(total_cost);
  x.display();
}
```

Figure D.1: A FLOPC++ formulation of a simple transportation problem not $\sum_S x(S)$). (Other languages, like AMPL, also distinguishes between dummy indices and index sets.)

The class `MP_set` is used for representing index sets. The number of elements is specified when an index set is declared. If an index set with n elements is declared the underlying set consists of the elements $\{0, 1, ..., n-1\}$. When an indexed entity (data, variable or constraint) is declared we simply specify the desired index set of each dimension of indexation. Note the use of `enum` to create symbolic names for the set elements. For example, the enumeration, `enum {seattle,sandiego,Number_of_suppliers};`, associates `seattle` with 0, `sandiego` with 1 and `Number_of_suppliers` with 2, the number of suppliers.

The model formulation is independent on the solver which is used. FLOPC++ currently offers an interface to CPLEX 6.5 [8] only, but interfaces to other solvers can easily be implemented if required.

The optimal value of the decision variables can be displayed using the `MP_variable` member function, display. Alternatively the optimal value of the individual variables can be accessed with the member function, level, as in `x.level(i,j)`.

D.4 Advantages of using C++ class library

Essentially FLOPC++ provides the same functionality as existing algebraic modelling languages (or rather their associated translators), so one might well ask why there is a need for FLOPC++. By using a C++ class library instead of a traditional modelling language we provide improved support to the modeller in at least three important areas:

Integration As mentioned in the first section, the integration of a linear optimization model with other software components is often indispensable in practical applications. By working in a popular programming language we gain immediate access to a huge number of class libraries for different tasks such as creation of graphical user interfaces, links to databases, visualization tools, etc.

Model tailored solution algorithms The algebraic description of a model convey structural information that is hidden in the coefficient matrix representation required by the solver. Sometimes this information can be used to develop efficient specialized solution algorithms while still depending on a general linear optimization solver. Examples of such algorithms include decomposition, column generation and model specific cutting plane algorithms. The development of such algorithms is greatly facilitated by working in an environment where the algebraic description of the model is available, but efficiency considerations make traditional algebraic modelling languages a bad choice. These issues are further discussed in [13].

Software engineering Program structuring facilities in existing algebraic modelling languages are virtually non existing. In GAMS, for example, there is no concept of a scope. In practice this means that, although multiple models are possible, the same identifier cannot be used in two different ways in different models contained in the same GAMS input file.

The support for object oriented programming found in C++, in contrast to existing algebraic modelling language, provides powerful means for code/model reuse and development of application area specific extensions.

Of course algebraic modelling languages are not static. Indeed, developers of algebraic modelling languages, realizing the need for better procedural support, have started to adapt their languages to meet this need by incorporating more procedural constructs, but it does not seem realistic that the capabilities of C++ will be matched.

D.5 Comparison with LP-toolkit and ILOG Planner

To compare FLOPC++ with LP-toolkit and ILOG Planner we have included two small example model formulations using these products. The model formulations were taken from [5] and [9] respectively. Apparently there is no way to obtain detailed descriptions of these two class libraries unless you by the products. Therefore our conclusions concerning the capabilities of these class libraries are based on what we deduce from these two examples of their use.

D.5.1 LP-toolkit

Consider a transportation problem LP-toolkit formulation shown below.

```
#include "lptk4.h"// include the lp-toolkit header file
// define maximum sizes
#define nMaxClient      100 // Max number of clients
#define nMaxPlant       20  // Max number of plants
// sizes of problem
int nClients = 4, nPlants = 3;   // number of clients and plants
// Results storage
double vol[nMaxPlant][nMaxClient];  // volumes delivered
double activ[nMaxPlant];            // Activities of plants
// Create class of linear problem, derived from CpxlpProb
// which is a LP to be solved with CPLEX
class MyLPClass : public CpxlpProb {
public:
   MyLPClass(int nMaxVar, int nMaxConst, int nMaxCoef) :
            CpxlpProb(nMaxVar, nMaxConst, nMaxCoef) {}
   int CreateModel(); // this method will construct the model
};

MyLPClass::CreateModel() {
   int i,j;
 // input data
   lpFloatArray COST(nPlants, nClients),// distrib. costs
                DEMAND(nClients),       // demands
                CAPA(nPlants);          // plants capacities
 // Variables array
   lpVarArray X(nPlants, nClients);
 // Constraints
   lpConstArray DCONST(nClients),       // demand constraints
                CCONST(nPlants);        // capac. constraints
 // Read input data
   COST.input("cost.dat");
   DEMAND.input("demand.dat");
   CAPA.input("capa.dat");

 // for each variable, give cost, bounds, and address of
```



```
 // place where result is to be stored
   for (i=0;i<nPlants;i++)
     for (j=0;j<nClients;j++)
       X(i,j) = newVar(COST(i,j), 0.0, 10000.0, &vol[i][j]);

   for (j=0;j<nClients;j++) {   // Create demand constraints
     DCONST(j) = newConst(DEMAND(j), DEMAND(j));
     for (i=0;i<nPlants;i++)  // create non-zero elements
       newElt(DCONST(j),X(i,j),1.0);
   }
   lpVarArray XX(nClients);    // use temporary array
   for (i=0;i<nPlants;i++) {   // Create capacity constraints
     for (j=0;j<nClients;j++)
       XX(j) = X(i,j);
     lpSend (lpSum(XX) < CAPA(i)); // use symbolic form
   }
}

int main (int argc, char **argv) {
   // Declare a problem of type MyLPClass
   MyLPClass *transProb = new MyLPClass(1000, 1000, 1000);
   transProb->CreateModel();          // generate model
   int status = transProb->optimise();// solve problem
}
```

This formulation requires the modeller to write a C++ class to represent the model. It is therefore necessary for the modeller to have a certain knowledge of C++. In FLOPC++ it is possible to write simple models without knowing what a class is. On the hand, to take full advantage of the integration of algebraic modelling with the C++ programming language, knowledge of C++ is required by a modeller using FLOPC++ as well.

We note that a sum construction `lpSum` is provided by LP-toolkit but judging from its rather awkward use in the example, it appears that only sums over one-dimensional variables are allowed.

The specification of the maximum number of variables, constraints and non zero coefficients as well as the maximum number of clients and plants seem inelegant and unnecessary. This is related to fact that it is the responsibility of the modeller to allocate memory for the solution values of the variables. This burden should have been handled internally by the class library, not only to simplify the model description but also to ensure consistency between the model description and the report generation. It would increase readability if the solution value of a variable `X(i,j)` was referred to by something like `X.level(i,j)` (or maybe even `X(i,j)`) instead of `vol[i][j]` or some other name chosen by the user.

The use of

```
newElt(DCONST(j),X(i,j),1.0);
```

to directly specify a non zero element of the constraint matrix is certainly closer to the format required by the solver than to the format desired in an algebraic modelling language.

D.5.2 ILOG Planner

Figure 3 shows a simple production planning problem formulated in ILOG Planner

Here the sum construction is replaced by the `IlcScalProd` inner product function which takes two arrays as arguments. There also exists a `IlcSum` function, but it is simply used as a short hand notation for an inner product with an array of ones (like in LP-toolkit).

It might seem irrelevant whether a matrix notation

```
IlcScalProd(consumption[r],inside)
```

is used in favour of an algebraic notation

```
sum(j,Prods, consumption(r,j)*inside(j) )
```

but in fact the distinction is important. The latter notation comes closest to the way a modeller normally conceives the expression. It is also far more general and flexible. Suppose we wanted to know the total resource consumption for each product. With the algebraic notation this poses no problems


```
IlcManager m(IlcNoEdit);
const IlcInt nbProds = 3;
const IlcInt nbResources = 2;
IlcFloatArray consumption[nbResources];
IlcFloatArray demand;
IlcFloatArray capacity;
IlcFloatArray insideCost, outsideCost;

consumption[0] = IlcFloatArray(m, nbProds, .5, .4, .3);
consumption[1] = IlcFloatArray(m, nbProds, .2, .4, .6);
capacity = IlcFloatArray(m, nbResources, 20., 40.);
demand = IlcFloatArray(m, nbProds, 100., 200. , 300.);
insideCost = IlcFloatArray(m, nbProds, .6, .8, .3);
outsideCost = IlcFloatArray(m, nbProds, .8, .9, .4);

IlcFloatVarArray inside(m, nbProds, 0, IlcInfinity);
IlcFloatVarArray outside(m, nbProds, 0, IlcInfinity);
IlcSimplex simplex(m);

for (IlcInt r=0; r<nbResources; r++)
  simplex.add(
    IlcScalProd(consumption[r],inside) <= capacity[r] );

for (IlcInt p=0; p<nbProds; p++)
  simplex.add(inside[p]+outside[p] == demand[p]);

IlcFloatVar costVar =
  simplex.setObjMin(IlcScalProd(insideCost, inside) +
                    IlcScalProd(outsideCost, outside));
```

Figure D.2: An ILOG Planner formulation of a simple production planning problem

```
sum(i, Resources, consumption(i,p)*inside(p));
```

whereas there does not seem to be any elegant solution using ILOG Planner. It would probably involve the creation of a temporary array (like in the LP-toolkit example) to formulate this expression in ILOG Planner, thereby destroying the illusion of a declarative model formulation completely. (The situation is equally hopeless if the formulation of a restricted sum or a nested sum is desired.)

Apparently constraints are not named in ILOG Planner, so there seem to be no way to retrieve the optimal Lagrangian multipliers in terms of model entities.

D.5.3 Conclusion of comparison.

Both LP-toolkit and ILOG planner have a distinct procedural flavour, which stems from the lack of an index controlled repeated sum construct. This omission facilitates the implementation of the libraries enormously since the creation of expression trees as described in the next section becomes superfluous, but it also constitutes the discriminating factor which makes them mere modelling tools as opposed to a full-fledged algebraic modelling language like FLOPC++.

D.6 Implementation

A linear program is specified by $A \in R^{m*n}, b^m, c^n, l^n, u^n$ for some m and n. The rim data, b, c, l, u, are typically passed to the solver as arrays of doubles, whereas the coefficient matrix A typically is required in some variant of column compressed sparse storage format by the solver. This format consists of four arrays: Elm (nz doubles), Rnr (nz integers), Cst (n integers) and Clg (n integers), where nz is the total number of non zero elements in A. Elm and Rnr contains the value and row-number respectively of each non zero element. The non zero elements of each column are stored consecutively in Elm and Rnr. Cst[j] stores the position in Elm and Rnr of the first non zero element of the j'th column of A and Clg[j] stores the number of non zero elements in the j'th column.

Each constraint and variable of the algebraic model description corresponds to a unique row or column in the coefficient matrix. The main work of the translation from algebraic form to matrix form consists in keeping track of this correspondence.

As the constraints and variables of a model are defined, their corresponding row or column numbers are assigned. To do this the MP_model class must include counters for the number of rows and columns defined in the model so far.

An example of an indexed constraint definition in FLOPC++ is

```
forall(i,S)
  supply(i) = sum(j,D,x(i,j)) <= capacity(i);
```

The `forall` construct is implemented by a macro which expands to a for loop

```
for(int i=0; i<S.cardinality(); i++)
  supply(i) = sum(j,D,x(i,j)) <= capacity(i);
```

Since all indices on the left hand side must be controlled by a for loop, each time a constraint definition is reached, the left hand side refers to a specific single constraint of the model to which it belongs and the corresponding row number can be looked up. The corresponding row of A is given by the expression

```
sum(j,D,x(i,j)) <= capacity(i);
```

which is infix notation for the function call

```
operator<=( sum(j,D,x(i,j)), capacity(i));
```

It is therefore not possible to implement the repeated sum construct `sum(j,D,x(i,j))` by a macro which expands to a for loop.

The function `sum` takes 3 arguments, a dummy index, an index set and an expression (`x(i,j)` in this case). Since the index `j` is not instantiated in the expression `x(i,j)` when the function is called, expressions must be represented by a pointer to an object with a method for evaluating the expression. To evaluate expressions which contain dummy indices, a context specifying the bindings from dummy indices to set elements must be supplied. The object returned by the `sum` function must itself be an object representing an expression. The particular object pointed to by the pointer of the returned object must include a reference to the running index, the controlling set and the underlying expression. The evaluation function of the object augments the context by instantiating the running index by each of the possible elements of the controlling set in turn. This is implemented by a for loop where, in each iteration, the underlying expression is evaluated in the current context and the results are accumulated to give the result of the evaluation of the sum expression.

The arithmetic operators are overloaded to build up an expression tree similarly to the sum function. In C++ it is forbidden for overloaded operators to take pointers as arguments, this is why the pointers to expression have been embedded in a class as described above. In this way a complete expression tree can be assembled for both the right and left hand sides of a constraint, which are only evaluated by request when the entire constraint is known (and all indices are controlled by some sum expression).

When requested, an expression in FLOPC++ evaluates to a sparse vector corresponding to a (partial) row of the LP being generated, the index -1 is used for the right hand side element. So for example the expression `capacity(i)` with `i` instantiated evaluates to a one element vector with its non-zero element in position -1. And if both `i` and `j` are instantiated the expression `x(i,j)` evaluates to a vector with the value 1 in the position corresponding to the referenced variable.

Besides the expressions described above which evaluates to a row, FLOPC++ also uses integer valued expressions (for index expressions), double valued and boolean valued expressions (for conditions).

The interpretation of expressions represented by abstract syntax trees is a frequently occurring pattern in object oriented software construction. A discussion of this and other patterns can be found in [7].

Note that the algebraic model description is row oriented, such that the coefficient matrix naturally will be build up row wise. However it is a simple matter to convert the matrix representation from row compressed storage format to column compressed storage format as required by the solver.

D.7 More details of FLOPC++

D.7.1 Index arithmetic

In many models, typically involving a sequence of time periods, the use of ordered sets and associated index arithmetic (leads and lags in GAMS) are essential. In FLOPC++ it is no problem to specify a constraint such as

```
forall(p,P)
   forall(t,T)
      Balance(p,t) =
         stock(p,t+1) == stock(p,t) + purchase(p,t)
                       - consumed(p,t);
```

A question arises about how to handle the situation when an index expression takes on a value outside the index set for which the variable was defined. We have chosen to treat this situation similarly to GAMS, that is, a variable reference with an index out of bound is treated as zero. This convention usually allows the formulation of the desired behaviour in a natural and elegant manner. However we envisage that FLOPC++ should be revised to issue a warning each time an out of bound index is treated as zero.

D.7.2 Conditions

Conditional constraints can be handled directly with the C++ if-statement since all relevant indices are instantiated at the point where the individual constraint definition is reached. For example

```
forall(m,M)
  forall(i,I)
    if (mpos[m][i]==true)
      myconstraint(m,i) = sum(p, P, b(m,p)*z(p,i)) <= k(m,i);
```

where `mpos` is some two-dimensional boolean array.

Conditional repeated sums are handled by restricting the controlling set with the `such_that` member function like this

```
myconstraint(m,i) =
  sum(p, P.such_that(ppos(p,i) == 1), b(m,p)*z(p,i) ) <= k(m,i);
```

It is also possible to use relations between indices, as in

```
cons(j,h) =
  gamma(j,h) <= sum(k, K.such_that(k >= h+1), lambda(j,k));
```

The `operator()` of the class `MP_set` has been overloaded to allow the simplified syntax `K(k>=h+1)` for `K.such_that(k >= h+1)`.

D.7.3 Changing attributes of variables

By default, variables in FLOPC++ are continuous with lower bound equal to zero and upper bound equal to infinity. Two specializations `MP_integer_variable` and `MP_binary_variable` are provided for variables which are required to take on integer values. The default upper bound on `MP_integer_variable` and `MP_binary_variable` is infinity and 1 respectively.

To change the bounds of variables the member functions `set_lb` and `set_ub` are provided. Bounds can be changed simultaneously for an indexed collection of variables or for an individual variable.

```
x.set_ub(100);
x(seattle,chicago).set_ub(500);
```

It is also possible to change the type of individual variables from continuous to integer valued.

```
x(seattle,newyork).integer();
```

In this way it is possible to use an indexed variable including both continuous and integer variables; a useful feature which, as far as I know, is not included in any traditional modelling language.

D.7.4 The default model

For the occasional user which does not need multiple models it might be cumbersome to specify the model when variables and constraints are declared. Therefore FLOPC++ supports the notion of a default model to which all declared variables and constraints belong, unless their model is explicitly included in the declaration.

D.8 Object oriented modelling

In an objected oriented programming language, such as C++, code reuse is facilitated by the class inheritance mechanism. By using a C++ class library for modelling it is possible to encapsulate a model in a class. This way instances of the model can be generated as objects of the model class. An extension of a model can conveniently be represented as a class derived from the base model class.

D.9 Possible improvements and extensions

D.9.1 Non-linear models

As the title suggests this library is intended for linear models only. This is a major limitation compared with modelling languages such as GAMS and AMPL. There is nothing in the principles behind FLOPC+ which prevent the use non-linear expressions in the constraints and objective function. However the implementation would require to much work to be considered in the context of this project.

D.9.2 Support for standard modelling tricks

Linearization of the object function

Often the formulation of mathematical programming models involve the use of idioms (standard modelling tricks). For example it is well known (see for example [14]) how certain apparently non-linear objective functions can be reformulated as linear models. This is the case for example for so called minimax objectives, ratio objectives and piecewise linear objectives. It would be useful for the modeller to rely on the underlying software for carrying out these model reformulations. ILOG Planner, for example, has a construct which allows piecewise linear functions to be easily expressed. A similar construct is expected in FLOPC++ soon.

Binary modelling variables

Binary variables are often used as modelling artifacts for expressing certain kinds of constraints. For example we might express a constraint requiring the number of plants which produces 100 or more entities to be at least five, in the following manner.

```
MP_variable x(P);
MP_binary_variable d(P);
MP_constraint art(P), atleastfive;

forall (p,P)
   art(p) = x(p) >= 100 * d(p);

atleastfive = sum(p, P, d(p)) >= 5;
```

The binary variables `d(P)` are needed to express the desired constraint, but are otherwise not part of the conception of the model. It might be preferable to express the idea above in a direct way, for example

```
MP_variable x(P);
MP_constraint atleastfive;

atleastfive = sum(p, P, x(p) >= 100) >= 5;
```

This constraint would then be automatically transformed to the form above (hidden from the modeller).

This and other similar constructs will possibly become part of the set of modelling abstractions offered by FLOPC++. Furthermore it is possible

for a modeller to design and implement his own extensions of FLOPC++. To my knowledge, this is impossible in traditional modelling languages but can be done in FLOPC++ due to the procedural and object oriented support offered by C++.

D.9.3 Implicit quantification

In GAMS an uncontrolled index is implicitly considered to be universally quantified over the index set corresponding to the index (in GAMS set names are used as indices). This gives a more succinct model formulation.

It will be no problem to implement this in FLOPC++ too, but since indices are not a priori connected to index sets, the controlling domain must be determined from the declaration of the left hand entity. Hence, in

```
MP_constraint c(S);
c(i) = sum(j, D, x(i,j)) <= h(i);
```

the index `i` is taken to be quantified over `S`. Conditions can be specified directly in the declaration, for example

```
MP_constraint c(S(h(i)>0));
```

The inclusion of implicit quantification in FLOPC++ is expected soon.

D.9.4 Modifying a model instance

When a solve request for a model is encountered in FLOPC++ the resulting model instance is generated from scratch. It is possible to solve a model, inspect the results and use them for modifying the model, for example by adding a new constraint. This could be used for example in a model specific cutting plane algorithm. However, this approach would be hopelessly inefficient in FLOPC++ (and in traditional modelling languages as well) since the solver would have no way of knowing that the second problem is just a small modification of the first. To circumvent this problem we need a two step approach. A model object should be used to produce a model instance object. Model instance objects would then be used for solution requests and the model instance class would include modification member functions. Off course this is only useful when a solver, such as CPLEX 6.5, with a callable library which supports modification of problems is used.

D.10 Conclusion

Instances of linear optimization models can be generated and passed to a solver via its callable library in a conventional programming language to ensure a seamless integration with other software components. Such code, however, can be difficult to write, to understand and to modify. The FLOPC++ class library has been developed to automate this step by letting the modeller express his model directly in terms of indexed variables and constraints. This is also achieved by using a traditional algebraic modelling language, but the use of a C++ class library gives several advantages as described in this paper.

Language extensions of FLOPC++ are facilitated by its organization as a C++ class library and can be implemented by the modeller to meet specific demands. Such language extensions might for example be constructs to support the formulation of stochastic optimization problems.

FLOPC++ is intended to provide an easy to use, modern algebraic modelling environment. The modelling abstractions provided are created to allow models to be formulated as simple and natural as possible.

Bibliography

[1] Drayton Analytics. Using AMMO. `http://www.draytonanalytics.com/ammo.htm`.

[2] A. Brooke, D. Kendrick, A. Meeraus, R. Raman, and R. Rosenthal. *GAMS - A user's guide*. GAMS Development Corporation, December 1998.

[3] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 1963.

[4] Dash Associates. *XPRESS-MP Builder Subroutine Library XBSL. User Guide and Reference Manual*, June 2000.

[5] Euro-decision. LP-TOOLKIT : a development and modelling tool for solving mathematical programming problems. `http://www.eurodecision.fr/English/html/lp-toolkit.html`.

[6] R. Fourer, D. M. Gay, and B. W. Kernighan. A modeling language for mathematical programming. *Management Science*, 36:519–554, 1990.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patters. Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[8] ILOG. *ILOG Cplex 6.5 User's Manual.*

[9] ILOG. ILOG optimization suite white paper. Discovering a competitive advantage. Technical report, 1998.

[10] Maximal Software Inc. Optimax 2000. `http://www.maximal-usa.com/optimax/`.

[11] Modellium Inc. Ezmod. `http://www.modellium.com/products.htm`.

[12] S. Nielsen. A C++ class library for mathematical programming. In *The impact of emerging technologies on computer science and operations research*. Kluwer, 1995.

[13] J. Tebboth and R. Daniel. A tightly integrated modelling and optimisation library: A new framework for rapid algorithm development. *Annals of Operations Research*, 1998.

[14] H. P. Williams. *Model building in mathematical programming. Third edition.* Wiley, 1993.

Ph. D. theses from IMM

1. **Larsen, Rasmus.** (1994). *Estimation of visual motion in image sequences. xiv* + 143 pp.
2. **Rygard, Jens Moberg.** (1994). *Design and optimization of flexible manufacturing systems. xiii* + 232 pp.
3. **Lassen, Niels Christian Krieger.** (1994). *Automated determination of crystal orientations from electron backscattering patterns. xv* + 136 pp.
4. **Melgaard, Henrik.** (1994). *Identification of physical models. xvii* + 246 pp.
5. **Wang, Chunyan.** (1994). *Stochastic differential equations and a biological system. xxii* + 153 pp.
6. **Nielsen, Allan Aasbjerg.** (1994). *Analysis of regularly and irregularly sampled spatial, multivariate, and multi-temporal data. xxiv* + 213 pp.
7. **Ersbøll, Annette Kjær.** (1994). *On the spatial and temporal correlations in experimentation with agricultural applications. xviii* + 345 pp.
8. **Møller, Dorte.** (1994). *Methods for analysis and design of heterogeneous telecommunication networks.* Volume 1-2, *xxxviii* + 282 pp., 283-569 pp.
9. **Jensen, Jens Christian.** (1995). *Teoretiske og eksperimentelle dynamiske undersøgelser af jernbanekøretøjer. viii* + 174 pp.
10. **Kuhlmann, Lionel.** (1995). *On automatic visual inspection of reflective surfaces.* Volume 1, *xviii* + 220 pp., (Volume 2, *vi* + 54 pp., fortrolig).
11. **Lazarides, Nikolaos.** (1995). *Nonlinearity in superconductivity and Josephson Junctions. iv* + 154 pp.
12. **Rostgaard, Morten.** (1995). *Modelling, estimation and control of fast sampled dynamical systems. xiv* + 348 pp.
13. **Schultz, Nette.** (1995). *Segmentation and classification of biological objects. xiv* + 194 pp.
14. **Jørgensen, Michael Finn.** (1995). *Nonlinear Hamiltonian systems. xiv* + 120 pp.
15. **Balle, Susanne M.** (1995). *Distributed-memory matrix computations. iii* + 101 pp.
16. **Kohl, Niklas.** (1995). *Exact methods for time constrained routing and related scheduling problems. xviii* + 234 pp.
17. **Rogon, Thomas.** (1995). *Porous media: Analysis, reconstruction and percolation. xiv* + 165 pp.
18. **Andersen, Allan Theodor.** (1995). *Modelling of packet traffic with matrix analytic methods. xvi* + 242 pp.
19. **Hesthaven, Jan.** (1995). *Numerical studies of unsteady coherent structures and transport in two-dimensional flows.* Risø-R-835(EN) 203 pp.
20. **Slivsgaard, Eva Charlotte.** (l995). *On the interaction between wheels and rails in railway dynamics. viii* + 196 pp.
21. **Hartelius, Karsten.** (1996). *Analysis of irregularly distributed points. xvi* + 260 pp.
22. **Hansen, Anca Daniela.** (1996). *Predictive control and identification - Applications to steering dynamics. xviii* + 307 pp.
23. **Sadegh, Payman.** (1996). *Experiment design and optimization in complex systems. xiv* + 162 pp.
24. **Skands, Ulrik.** (1996). *Quantitative methods for the analysis of electron microscope images. xvi* + 198 pp.
25. **Bro-Nielsen, Morten.** (1996). *Medical image registration and surgery simulation. xxvii* + 274 pp.

26. **Bendtsen, Claus.** (1996). *Parallel numerical algorithms for the solution of systems of ordinary differential equations.* viii + 79 pp.

27. **Lauritsen, Morten Bach.** (1997). *Delta-domain predictive control and identification for control.* xxii + 292 pp.

28. **Bischoff, Svend.** (1997). *Modelling colliding-pulse mode-locked semiconductor lasers.* xxii + 217 pp.

29. **Arnbjerg-Nielsen, Karsten.** (1997). *Statistical analysis of urban hydrology with special emphasis on rainfall modelling.* Institut for Miljøteknik, DTU. xiv + 161 pp.

30. **Jacobsen, Judith L.** (1997). *Dynamic modelling of processes in rivers affected by precipitation runoff.* xix + 213 pp.

31. **Sommer, Helle Mølgaard.** (1997). *Variability in microbiological degradation experiments - Analysis and case study.* xiv + 211 pp.

32. **Ma, Xin.** (1997). *Adaptive extremum control and wind turbine control.* xix + 293 pp.

33. **Rasmussen, Kim Ørskov.** (1997). *Nonlinear and stochastic dynamics of coherent structures.* x + 215 pp.

34. **Hansen, Lars Henrik.** (1997). *Stochastic modelling of central heating systems.* xxii + 301 pp.

35. **Jørgensen, Claus.** (1997). *Driftsoptimering på kraftvarmesystemer.* 290 pp.

36. **Stauning, Ole.** (1997). *Automatic validation of numerical solutions.* viii + 116 pp.

37. **Pedersen, Morten With.** (1997). *Optimization of recurrent neural networks for time series modeling.* x + 322 pp.

38. **Thorsen, Rune.** (1997). *Restoration of hand function in tetraplegics using myoelectrically controlled functional electrical stimulation of the controlling muscle.* x + 154 pp. + Appendix.

39. **Rosholm, Anders.** (1997). *Statistical methods for segmentation and classification of images.* xvi + 183 pp.

40. **Petersen, Kim Tilgaard.** (1997). *Estimation of speech quality in telecommunication systems.* x + 259 pp.

41. **Jensen, Carsten Nordstrøm.** (1997). *Nonlinear systems with discrete and continuous elements.* 195 pp.

42. **Hansen, Peter S.K.** (1997). *Signal subspace methods for speech enhancement.* x + 226 pp.

43. **Nielsen, Ole Møller.** (1998). *Wavelets in scientific computing.* xiv + 232 pp.

44. **Kjems, Ulrik.** (1998). *Bayesian signal processing and interpretation of brain scans.* iv + 129 pp.

45. **Hansen, Michael Pilegaard.** (1998). *Metaheuristics for multiple objective combinatorial optimization.* x + 163 pp.

46. **Riis, Søren Kamaric.** (1998). *Hidden markov models and neural networks for speech recognition.* x + 223 pp.

47. **Mørch, Niels Jacob Sand.** (1998). *A multivariate approach to functional neuro modeling.* xvi + 147 pp.

48. **Frydendal, Ib.** (1998.) *Quality inspection of sugar beets using vision.* iv + 97 pp. + app.

49. **Lundin, Lars Kristian.** (1998). *Parallel computation of rotating flows.* viii + 106 pp.

50. **Borges, Pedro.** (1998). *Multicriteria planning and optimization. - Heuristic approaches.* xiv + 219 pp.

51. **Nielsen, Jakob Birkedal.** (1998). *New developments in the theory of wheel/rail contact mechanics.* xviii + 223 pp.

52. **Fog, Torben.** (1998). *Condition monitoring and fault diagnosis in marine diesel engines.* xii + 178 pp.

53. **Knudsen, Ole.** (1998). *Industrial vision. xii* + 129 pp.

54. **Andersen, Jens Strodl.** (1998). *Statistical analysis of biotests. - Applied to complex polluted samples. xx* + 207 pp.

55. **Philipsen, Peter Alshede.** (1998). *Reconstruction and restoration of PET images. vi* + 132 pp.

56. **Thygesen, Uffe Høgsbro.** (1998). *Robust performance and dissipation of stochastic control systems.* 185 pp.

57. **Hintz-Madsen, Mads.** (1998). *A probabilistic framework for classification of dermatoscopic images. xi* + 153 pp.

58. **Schramm-Nielsen, Karina.** (1998). *Environmental reference materials methods and case studies. xxvi* + 261 pp.

59. **Skyggebjerg, Ole.** (1999). *Acquisition and analysis of complex dynamic intra- and intercellular signaling events.* 83 pp.

60. **Jensen, Kåre Jean.** (1999). *Signal processing for distribution network monitoring. xv* + 199 pp.

61. **Folm-Hansen, Jørgen.** (1999). *On chromatic and geometrical calibration. xiv* + 238 pp.

62. **Larsen, Jesper.** (1999). *Parallelization of the vehicle routing problem with time windows.xx* + 266 pp.

63. **Clausen, Carl Balslev.** (1999). *Spatial solitons in quasi-phase matched struktures. vi* + (flere pag.)

64. **Kvist, Trine.** (1999). *Statistical modelling of fish stocks. xiv* + 173 pp.

65. **Andresen, Per Rønsholt.** (1999). *Surface-bounded growth modeling applied to human mandibles. xxii* + 125 pp.

66. **Sørensen, Per Settergren.** (1999). *Spatial distribution maps for benthic communities.*

67. **Andersen, Helle.** (1999). *Statistical models for standardized toxicity studies. viii* + (flere pag.)

68. **Andersen, Lars Nonboe.** (1999). *Signal processing in the dolphin sonar system. xii* + 214 pp.

69. **Bechmann, Henrik.** (1999). *Modelling of wastewater systems. xviii* + 161 pp.

70. **Nielsen, Henrik Aalborg.** (1999). *Parametric and non-parametric system modelling. xviii* + 209 pp.

71. **Gramkow, Claus.** (1999). *2D and 3D object measurement for control and quality assurance in the industry. xxvi* + 236 pp.

72. **Nielsen, Jan Nygaard.** (1999). *Stochastic modelling of dynamic systems. xvi* + 225 pp.

73. **Larsen, Allan.** (2000). *The dynamic vehicle routing problem. xvi* + 185 pp.

74. **Halkjær, Søren.** (2000). *Elastic wave propagation in anisotropic inhomogeneous materials. xiv* + 133 pp.

75. **Larsen, Theis Leth.** (2000). *Phosphorus diffusion in float zone silicon crystal growth. viii* + 119 pp.

76. **Dirscherl, Kai.** (2000). *Online correction of scanning probe microscopes with pixel accuracy.* 146 pp.

77. **Fisker, Rune.** (2000). *Making deformable template models operational. xix* 210 pp.

78. **Hultberg, Tim Helge.** (2000). *Topics in computational linear optimization. xiv* + 180 pp.