Analysis of standard and new algorithms for the integer ans linear constraint satisfaction problem

Jean-Claude Sogno

Rapports de Recherche

N°1828

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Programme 2

Calcul symbolique, Programmation et Génie logiciel

ANALYSIS OF STANDARD AND NEW ALGORITHMS FOR THE INTEGER AND LINEAR CONSTRAINT SATISFACTION PROBLEM

Jean-Claude SOGNO

Decembre 1992

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél.:(1)39 63 55 11

Analysis of standard and new algorithms for the integer and linear constraint satisfaction problem *

Analyse d'algorithmes classiques et nouveaux pour le problème de satisfaction de contraintes linéaires à variables entières

Jean-Claude Sogno

INRIA - Rocquencourt
BP 105 - 78153 Le Chesnay cedex
e-mail: Jean-Claude.Sogno@inria.fr

Décembre 1992

Abstract

The integer and linear constraint satisfaction problem, which consists in proving the emptiness of the set of integer points satisfying a set of linear constraints or the existence of a solution, is very frequent in the field of computer science (vectorization, code scheduling, etc.). Most methods proposed in the literature deal with various specific instances of this problem. In this paper, the problem is considered in its general form. Some standard methods are analyzed. Some new algorithms are proposed, either to simplify the problem, or to solve exactly (by cutting plane methods) the reduced form of the problem.
A sequence of such algorithms is implemented in the automatic parallelizer available under PIAF, an Interactive Programming environment for FORTRAN. However, these algorithms could be applied to more "difficult" problems than the usual ones appearing in data dependence analysis.

Résumé

Le problème de satisfaction de contraintes linéaires à variables entières est très fréquent dans nombre de domaines comme la vectorisation automatique ou l'ordonnancement de taches. Les méthodes de résolution proposées dans la littérature sont généralement spécifiques de cas particuliers du problème traité. Dans cet article, le problème est considéré dans sa forme générale. Plusieurs méthodes classiques sont analysées. Sont proposés plusieurs algorithmes de réduction (permettant de se ramener à un problème de taille plus réduite) et de résolution générale (par des méthodes de coupe) du problème simplifié.
Une telle combinaison d'algorithmes est implémentée dans le paralléliseur automatique disponible sous PIAF, environnement de Programmation Interactif pour FORTRAN. Cependant, ces algorithmes pourraient s'appliquer à des problèmes plus "difficiles" que ceux issus de l'analyse des dépendances de données.

*This work was partially supported by ESPRIT Project COMPARE

1 Introduction

The integer and linear constraint satisfaction problem consists in proving either the emptiness of a domain defined by linear diophantine constraints or the existence of any one solution. The set of constraints may include equations or inequalities:

$$\begin{array}{ll} \sum_{j=1}^{n} a_{ij}x_j \leq b_i, & i \in \{1, \ldots m_1\} \\ \sum_{j=1}^{n} a_{ij}x_j = b_i, & i \in \{m_1+1, \ldots m_1+m_2\} \\ x_j \in \mathbb{Z} & \end{array}$$

We assume that all input data are integer.

This kind of problem occurs frequently in the computer science field, and specially in the case of dependence analysis required for automatic parallelization.

1.1 Usual problems

In data dependence analysis, the question is whether two references to a same m-dimensional array within n nested loops may be references to the same element of this array. This leads to **basic "$n-m$" problems**, where systems include m equations, $4*n$ inequalities, and $2*n$ variables. For instance:

problem 1-1:

```
DO I=L,U
    A(α1*I+γ1) ...
    ...A(α2*I+γ2)
ENDDO
```

The resulting system is:

$$\begin{array}{c} \alpha_1 i_1 + \gamma_1 = \alpha_2 i_2 + \gamma_2 \\ L \leq i_1 \leq U \\ L \leq i_2 \leq U \\ i_1, i_2 \in \mathbb{Z} \end{array}$$

problem 2-1:

```
DO I=Li,Ui
    DO J=Lj,Uj
        A(α1*I+β1*J+γ1) ...
        ...A(α2*I+β2*J+γ2)
    ENDDO
ENDDO
```

The system is:

$$\begin{array}{c} \alpha_1 i_1 + \beta_1 j_1 + \gamma_1 = \alpha_2 i_2 + \beta_2 j_2 + \gamma_2 \\ L_i \leq i_1 \leq U_i \\ L_i \leq i_2 \leq U_i \\ L_j \leq j_1 \leq U_j \\ L_j \leq j_2 \leq U_j \\ i_1, i_2, j_1, j_2 \in \mathbb{Z} \end{array}$$

The **"directional" problem** differs from the basic one by added constraints created by dependence directions, for instance, with the previous example:

$$\begin{array}{c} i_1 = i_2 \\ j_1 < j_2 \end{array}$$

An **"extended" problem** is defined when linear constraints (equations or inequalities) of any one form, resulting from data-flow analysis are added to the basic problem.

1.2 Usual methods

We may note these two conflicting points:

- An important quality required for methods used in data dependence analysis is rapidity, because such tests are very frequently applied.
- The general integer linear programming problem is NP-complete.

Then, many methods proposed in the literature are not "exact" (they are approximated tests), or are specific methods (they can be applied only in particular cases).

A method is inexact when three answers to the problem are possible:

1. No, there is no solution.
2. Yes, a solution exists.
3. Maybe, but we are unable to conclude.

When such methods are applied to data dependence analysis, parallelization is possible only if the answer is no. If the third case happens, it means that a dependence possibility exists, so the analyser works as if the answer were yes.

Inexact methods are mainly based on relaxation of constraints. We have two kinds:
First, the **integrity relaxation**. The problem is treated as if the variables are not restrained to integer values. In place of the original problem, we consider the following one:

$$\begin{array}{ll} \sum_{j=1}^{n} a_{ij}x_j \leq b_i, & i \in \{1, \ldots m_1\} \\ \sum_{j=1}^{n} a_{ij}x_j = b_i, & i \in \{m_1+1, \ldots m_1+m_2\} \\ x_j \in \mathbb{R} & \end{array}$$

Second, the **loop bounds relaxation**. We consider the problem as if variables were unbounded. This technique concerns problems where inequalities come

only from variables bounds (the basic "$n-m$" problems). The 1-1 problem is replaced by

$$\alpha_1 i_1 + \gamma_1 = \alpha_2 i_2 + \gamma_2$$
$$i_1, i_2 \in \mathbb{Z}$$

The 2-1 problem is replaced by

$$\alpha_1 i_1 + \beta_1 j_1 + \gamma_1 = \alpha_2 i_2 + \beta_2 j_2 + \gamma_2$$
$$i_1, i_2, j_1, j_2 \in \mathbb{Z}$$

Specific methods profit by some particular forms of the problem (for instance the number m of equations of the basic "$n-m$" problem).
Note that some methods are both inexact and specific.

More recently, some exact methods have been developped for data dependence analysis. Some are based on mathematical programming ([Wal88], [BP89]), others on the successive application of exact and cheap tests ([MHL91], [GKT91], [Pug91]).

1.3 Purpose of this paper

With the development of data flow analysis, general accurate algorithms will become increasingly necessary in order to solve the integer linear constraint satisfaction problem. As was noted above, the general problem is NP-complete. Nevertheless, we may think that problems resulting from data dependence analysis are not really difficult and can be solved exactly at a low computation cost. We may note some common features of these problems:

1. The value of a solution has no importance. Only its existence has one.
2. The sizes of problems (number of variables, number of constraints) are "small".
3. Variables have lower and upper bounds, so the domain of the system is finite.
4. The data matrix is sparse (constraints have many zero coefficients).
5. Non-zero coefficients of constraints are small.
6. Non-zero coefficients of constraints are often equal to ± 1.

We could turn these points to account. Firstly (point 1), we don't really need to solve the problem. Hence, we should try to *reduce* the problem: replacing it by an equivalent and smaller one (with less variables or constraints). Some reduction techniques are proposed in this paper. They should be efficient and fast (particularly due to point 6). At the same time, reduction techniques may solve our original problem. In some cases, they reduce the problem in such a way that its conclusion is immediate. In other cases, they prove that the system is infeasible. Then, on account, on the one hand, of points 2 and 3, and on the other hand, of reduction techniques, the resulting problem should be solved exactly and quickly by some integer programming method. With regard to this last point, many methods have been developed over the years. Cutting plane algorithms seem particularly suitable to "small" problems. "Rudimentary" algorithms are generally very fast with small problems, however we are not sure of their finiteness, which is guaranteed with more expensive finite algorithms. In data dependence analysis, reduced problems are so small that all "rudimentary" algorithms work. However, some rudimentary algorithms are proposed in this paper, in order to offer a "good chance" of finiteness in case of bigger problems.

In the computer science field, many problems have some features analoguous to the previous ones. They may be not exactly the same, for instance the bounds of the variables may be unknown, etc., hence, the problems could be considered as being more "difficult" than those of data dependence analysis. However, it is the author's opinion is that some similar reduction techniques and integer programming algorithms could be applied.

This paper is organized as follows.

Note: The examples included in this paper are not specific to data dependence analysis. They have been chosen in order to make the explanations relative to algorithms easier.

2 General remarks

This chapter should just specify some vocabulary and general points.

2.1 Generality of a method

Some methods can apply only to particular problems (for instance, some of basic "$n-m$" problems). Let us define a method as being *general* if it can be applied to any system after a possible "standardization" (see section 3).

2.2 Accuracy

A method is specified as being inexact when it may end without proving or disproving the existence of an integer solution.

Methods based on integrity relaxation are inconclusive when *the existence of a solution is proved* and one of these cases happens:

- *The value of the solution is unknown and we cannot prove its integrity.*

  Example, the classical Fourier-Motzkin elimination.

- *The value of the solution is fractional and we cannot prove its singleness.*

  Example, the usual simplex method.

Methods based on constraint relaxation are inconclusive when *the existence of a solution is proved* (for instance, the GCD test).

We should distinguish these cases from the methods ending without giving an answer to the question of the existence of an integer solution, but leading to a reduced system to which another method can be applied.

2.3 System equivalence

Consider the problem: prove or disprove the existence of an integer solution satisfying the system:

$$\begin{array}{rrcr} -x_1 & & \leq & -1 \\ x_1 & & \leq & 8 \\ & -x_2 & \leq & -2 \\ & x_2 & \leq & 8 \\ 2x_1 & -5x_2 & = & 12 \end{array}$$

We could obviously substitute the problem: prove or disprove the existence of an integer solution satisfying the system:

$$\begin{array}{rrcr} -x_1 & & \leq & -1 \\ x_1 & & \leq & 8 \\ & -x_2 & \leq & -2 \\ & x_2 & \leq & 8 \\ 2x_1 & -5x_2 & \leq & 12 \\ 2x_1 & -5x_2 & \geq & 12 \end{array}$$

More generally in this paper, let us define:

- *If we can solve a satisfaction problem over a system S_1 by means of a satisfaction problem over a system S_2, the systems S_1 and S_2 are said to be equivalent.*

Note that if we can replace S_1 by S_2, conversely we can replace S_2 by S_1.
Equivalent systems should be built in order to simplify the problem (2.4) or to standardize the system (3).

2.4 System reduction

In some methods, some variables or constraints may be eliminated. Every time an elimination is performed, we have in place of the original problem an equivalent reduced problem (see 5, 6.2 ...).

Other methods try to partially solve the system (constraints relaxation). If the subsystem has no solution, hence the whole system is infeasible. If the subsystem has a solution, we cannot conclude upon the whole system: the isolated method is inexact. Nevertheless, we may decide to build an equivalent system from the solution, in order to solve the problem with another method (see 4.3).

2.5 Arithmetic

Original problems are described with constraints including integer coefficients. Most methods described in this paper use integer arithmetic, they are "all integer". However, some methods need fractional arithmetic since they may handle non integer numbers (integrity relaxation methods, fractional cutting plane algorithms,...).

Consider the number resulting from the division 15/9. If we intend to handle it, we could use two different ways:

1. **Real arithmetic**.

   The result is represented by a "floating" number with a necessarily inexact value (such as 1.666666666 or 1.66666667). Then, if we multiply this number by 6, in place of the expected

integer number 10, we should obtain a number which could be 10.0000001. If we need to know whether this number has to be considerated as an integer, we must be able to estimate that, firstly the roundness error is greater than 0.0000001, secondly no fractional number resulting from any operation performed in the current method can be represented by a floating value so close to an integer value.

2. **Rational arithmetic**.

   Each number is represented by two integer numbers. In order to represent 15/9, we could use both numbers 15 and 9. However, since $15/9 = 5/3$, it would be better to use 5 and 3 in some algorithms.

Real arithmetic can work with methods requiring only the knowledge of the integrity of numbers (simplex methods must detect the cases where very small numbers have to be replaced by their correct value equal to *zero*). Rational arithmetic is indispensable for methods requiring the knowledge of the fractional representation of each number (fractional cutting plane algorithms). However, both arithmetics must manage the knowledge of numbers integrity. Roughly speaking, we can estimate that operations using real or rational arithmetic are at least twice as expensive as similar operations using integer arithmetic (for instance pivoting operations used for changes of variables).

3 "Standardization" of the problem

Imagine we have at our disposal some efficient algorithm able to solve the following problem:

prove or disprove the existence of a solution satisfying the system:

$$\begin{array}{l} \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i \in \{1, \ldots m_1\} \\ x_j \in \mathbb{Z} \end{array}$$

Now, suppose we intend to apply this algorithm to a system with a somewhat different form:

$$\begin{array}{l} \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i \in \{1, \ldots m_1\} \\ \sum_{j=1}^{n} a_{ij} x_j = b_i, \quad i \in \{m_1 + 1, \ldots m_1 + m_2\} \\ x_j \in \mathbb{Z} \end{array}$$

Before applying the method, we should modify the formulation of the problem so as to obtain an equivalent system without any equation. We must "standardize" the problem in order for it to conform with the "standard" form of the method.

The "standard" form of various integer programming methods could include:

- free variables
- binary variables (0 or 1)
- constrained variables (≥ 0)
- equations
- inequalities

The way we obtain the standard form of a problem may lead to very different systems, and that may considerably affect the behaviour of the chosen method. The author's opinion is that obtaining a "good" standard form is often as important as the choice of the method itself.

Let us have a look at some specific points.

3.1 Systems with only inequalities

Many methods require inequalities systems. In the first place, we don't deal with variables. Consider an original system including equations and inequalities with free variables:

$$\begin{array}{l} \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i \in \{1, \ldots m_1\} \\ \sum_{j=1}^{n} a_{ij} x_j = b_i, \quad i \in \{m_1 + 1, \ldots m_1 + m_2\} \\ x_j \in \mathbb{Z} \end{array}$$

Our goal is to handle an equivalent system, but without equations. Let us recall some classical techniques (which are not specific to integer programming):

1. **Replacing each equation by two inequalities**

   In place of:

   $$\sum_{j=1}^{n} a_{ij} x_j = b_i$$

   we can substitute the set:

   $$\begin{array}{l} \sum_{j=1}^{n} a_{ij} x_j \leq b_i \\ \sum_{j=1}^{n} a_{ij} x_j \geq b_i \end{array}$$

   Then, the previous system could be replaced by the equivalent one:

   $$\begin{array}{l} \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i \in \{1, \ldots m_1\} \\ \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i \in \{m_1 + 1, \ldots m_1 + m_2\} \\ \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \quad i \in \{m_1 + 1, \ldots m_1 + m_2\} \\ x_j \in \mathbb{Z} \end{array}$$

2. **Replacing n equations by $n+1$ inequalities.**

For instance, this set of 3 equations:

$$\begin{array}{l}\sum_{j=1}^{n} a_{1j}x_j = b_1\\ \sum_{j=1}^{n} a_{2j}x_j = b_2\\ \sum_{j=1}^{n} a_{3j}x_j = b_3\end{array}$$

can be replaced by these 4 inequalities:

$$\begin{array}{l}\sum_{j=1}^{n} a_{1j}x_j \leq b_1\\ \sum_{j=1}^{n} a_{2j}x_j \leq b_2\\ \sum_{j=1}^{n} a_{3j}x_j \leq b_3\\ \sum_{j=1}^{n} (a_{1j}+a_{2j}+a_{3j})x_j \geq b_1+b_2+b_3\end{array}$$

Now, consider the system:

$$\begin{array}{rrrrcr} -4x_1 & +6x_2 & & +x_4 & \leq & -9\\ 4x_1 & & -x_3 & & \leq & 9\\ -3x_1 & & +x_3 & & \leq & -20\\ 2x_1 & -x_2 & +2x_3 & & = & 12\\ x_1 & +2x_2 & -5x_3 & +x_4 & = & -5 \end{array}$$

The first technique should lead to the system:

$$\begin{array}{rrrrcr} -4x_1 & +6x_2 & & +x_4 & \leq & -9\\ 4x_1 & & -x_3 & & \leq & 9\\ -3x_1 & & +x_3 & & \leq & -20\\ 2x_1 & -x_2 & +2x_3 & & \leq & 12\\ x_1 & +2x_2 & -5x_3 & +x_4 & \leq & -5\\ 2x_1 & -x_2 & +2x_3 & & \geq & 12\\ x_1 & +2x_2 & -5x_3 & +x_4 & \geq & -5 \end{array}$$

the second one to:

$$\begin{array}{rrrrcr} -4x_1 & +6x_2 & & +x_4 & \leq & -9\\ 4x_1 & & -x_3 & & \leq & 9\\ -3x_1 & & +x_3 & & \leq & -20\\ 2x_1 & -x_2 & +2x_3 & & \leq & 12\\ x_1 & +2x_2 & -5x_3 & +x_4 & \leq & -5\\ 3x_1 & +x_2 & -3x_3 & +x_4 & \geq & 7 \end{array}$$

Obviously, these techniques lead to more voluminous systems. However, by applying some reduction methods (5.1), we can easily state that the original system is *equivalent* to this one:

$$\begin{array}{rrcr} 3x_1 & +13x_3 & \leq & 44\\ 4x_1 & -x_3 & \leq & 9\\ -3x_1 & +x_3 & \leq & -20 \end{array}$$

Clearly, each time we need to "standardize" a problem in order to apply some final integer programming method, we should choose techniques which are conjointly "reduction" techniques.

Conversely, we should choose a final integer programming method for which standardization allows a strong reduction of the system.

3.2 Free variables

In most problems issuing from data dependence analysis, task scheduling, etc., variables are "free", meaning that they can be given any integer value (positive or negative). Then, if a certain method requires free variables, no change is needed. We may imagine that a method requires free variables while the problem is described with constrained or binary variables. Replacing constrained or binary variables by free ones presents no difficulties (inequalities such as: $y_j \geq 0$, are explicitly added to the system).

3.3 Constrained variables

Most algorithms (simplex and others) used to solve the general integer programming problem require systems with constrained (non negative) variables. Generally, original systems include free variables. If we intend to replace free variables by constrained ones, we should apply some techniques described in section 7 allowing a reduced standard form.

However, if the original system includes one or more equations and the standard form of the intended method is a system of inequalities (with constrained variables), we should firstly remove the equations (see 3.1 and section 5.1), and secondly change the variables. Removing equations including constrained variables would be somewhat incongruous. We would have to previously replace one or more constrained variables by free variables!

3.4 Binary variables

This paper does not deal with methods requiring binary variables. However let us have a look at the way binary variables could be introduced.

When a variable has lower and upper bounds, it can be replaced by a linear combination of binary variables. Let us imagine a system including these two inequalities:

$$x_{min} \leq x \leq x_{max} \qquad (1)$$

Introducing n_b binary variables, we may remove inequalities (1) and change x:

$$x = x_{min} + \sum_{j=1}^{n_b} 2^{j-1} y_j$$

the number n_b is the smallest integer such that:

$$x_{max} - x_{min} + 1 \leq 2^{n_b} \qquad (2)$$

In the case of strict inequality ($<$) of (2), we must add:

$$\sum_{j=1}^{n_b} 2^{j-1} y_j \leq x_{max} - x_{min}$$

For instance, if we have:

$$5 \leq x \leq 17$$

We may change the variable x thus:

$$x = 5 + y_1 + 2y_2 + 4y_3 + 8y_4$$

and both inequalities (1) are replaced by:

$$y_1 + 2y_2 + 4y_3 + 8y_4 \leq 12$$

The knowledge of the bounds of variables is required (which is generally the case in data dependence analysis). However, if each free variable is changed thus and the values of the bounds are large, the technique may be expensive (a great number of variables!) and methods based on binary variables should be avoided in data dependence analysis.

3.5 Equations

Imagine a method requiring that the system includes (partially or totally) equalities. Any inequality can be replaced by an equality if we introduce a constrained variable ("slack" variable). Consider the inequality (variables x_j may be free or constrained):

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i$$

We can substitute:

$$\begin{aligned} y_e &\geq 0 \\ \sum_{j=1}^{n} a_{ij} x_j + y_e &= b_i \end{aligned} \qquad (3)$$

This mode of changing is used implicitly with simplex methods, where the slack ("dummy") variable is included in a single constraint. Some authors note it as a "solved" variable, and the inequality is handled as a "solved" equation: the dummy variable can be replaced by another variable which becomes a dummy variable, and at any time, the equation can be replaced by an inequality. In such a case, we can consider that this technique does not really change the inequality and the size of the system does not increase.

Obviously, if the method works in such a manner that the slack variable (or some replacing slack variable) may lose its status of "solved" variable (it may be included in several constraints), the inequality must be explicitly replaced by an equation.

3.6 Objective function

Consider the general problem P_1: prove or disprove the existence of a solution to:

$$\begin{aligned} Ax \ \Re \ b \\ x \in \mathbb{Z}^n \end{aligned}$$

where the symbol $\Re$ denotes a column of relations $=$, $\geq$ or $\leq$.

Now, imagine we can solve the optimization problem P_2 (including some objective function $z = cx$):

$$\begin{aligned} \min cx = z \\ Ax \ \Re \ b \\ x \in \mathbb{Z}^n \end{aligned}$$

If a (finite or infinite) solution to such a problem exists, it means that the problem P_1 has a solution. Then, by solving the problem P_2, we can prove or disprove the feasibility of P_1.

The problem P_2 requires any linear objective function. We could choose the simplest one, with coefficients equal to *zero*.

4 Diophantine equations: classical methods

In this section, we recall some classical tools and methods used when systems include one or more equations.

4.1 Classical tools

Most methods solving diophantine equations use the Greater Common Divisor (GCD) or the Extended Euclid algorithm.

4.1.1 GCD of 2 numbers

1. **Euclid Algorithm** [Knu81] [Ban88]
   Published 300 years B.C., and probably known well before!
   We present the modern and simplified version (using the zero) of this algorithm computing $p = GCD(a, b)$:

   1. $p \leftarrow |a|$, $v \leftarrow |b|$

   2. while $v \neq 0$: $\{r \leftarrow p \bmod v,\ p \leftarrow v,\ v \leftarrow r\}$

2. **Binary Algorithm of J. Stein** [Knu81]
   Published in 1961, this algorithm does not require division operation. It uses substraction, parity test, left binary shift on numbers binary form.

D. Knuth declares that it is 20 % faster than the Euclid algorithm on his mythic computer MIX.

The GCD of two numbers is frequently used (see 4.1.2, 4.2.1, section 5, section 7), so a fast GCD is desirable. The Binary Algorithm is faster than the classic Euclid Algorithm, however, the estimated benefit is slight and the handling of any binary algorithm is not very pleasant. So, using a table of precomputed GCD values seems preferable (see 5.6)

4.1.2 GCD of more than 2 numbers

The computation of the GCD of more than two numbers uses the property:

$$gcd(a_1, a_2 \cdots, a_n) = gcd(a_1, gcd(a_2 \cdots, a_n))$$

The order of numbers has no importance.
If we remark that:

$$gcd(a_1 \cdots, 1) = 1$$

useless computation may be avoided.

4.1.3 Extended Euclid algorithm

Given two non-negative integer numbers a_1 and a_2, the algorithm computes an integer vector (u_1, u_2, u_3) such that:

$$a_1 u_1 + a_2 u_2 = u_3 = gcd(a_1, a_2) \tag{4}$$

Let us describe the principle ([Knu81]):

- Classical version:

  1. $(u_1, u_2, u_3) \leftarrow (1, 0, a_1)$
     $(v_1, v_2, v_3) \leftarrow (0, 1, a_2)$
  2. while $v_3 \neq 0$:
     $q \leftarrow [u_3/v_3]$
     $(t_1, t_2, t_3) \leftarrow (u_1, u_2, u_3) - (v_1, v_2, v_3)q$
     $(u_1, u_2, u_3) \leftarrow (v_1, v_2, v_3)$
     $(v_1, v_2, v_3) \leftarrow (t_1, t_2, t_3)$

- Fast Version (Gordon H. Bradley):
  u_2, v_2, t_2 are not computed during the algorithm.
  u_2 is computed at the end by (4)

4.2 GCD-Test (Loop bounds constraint relaxation)

4.2.1 Principle

Let us recall a well-known result about the existence of a solution to a linear diophantine equation:

Theorem 4.2.1 *A necessary and sufficient condition for the existence of an integer solution to the equation* $\sum_{j=1}^{n} a_{ij} x_j = b_i$ *is that* $gcd(a_{i1}, a_{i2}, \cdots a_{in})$ *divides* b_i

Now, consider a system including this equation. Obviously, if no (integer) solution to the equation exists, no solution to the system exists. Nevertheless, if a solution to the equation exists, we cannot conclude. This is the classical inexact GCD-test [Ban79]:

Lemma 4.2.1 *A necessary condition for the existence of an integer solution to a system including the equation* $\sum_{j=1}^{n} a_{ij} x_j = b_i$ *is that* $gcd(a_{i1}, a_{i2}, \cdots a_{in})$ *divides* b_i

First example

$$\begin{aligned} 1 \leq x_1 \leq 10 \\ 1 \leq x_2 \leq 12 \\ 9x_1 - 6x_2 = 4 \end{aligned}$$

Since gcd(9,-6), equal to 3, does not divide 4, the system has no solution.

Second example

$$\begin{aligned} 3 \leq x_1 \leq 10 \\ 3 \leq x_2 \leq 10 \\ 4x_1 + 6x_2 = 20 \end{aligned}$$

$gcd(4, 6)$, equal to 2, divides 20. Hence, we cannot conclude about the system.

4.2.2 Probability of success

Since the GCD-test is inexact, it would be interesting to know the probability of the answer being a successful result, i.e. the equation has no solution. In order to estimate the probability $p_g(n)$ that the GCD-test applied to an equation with n variables is conclusive, we could proceed as follows:

1. Calculate the probability $p_a(n, k)$ that $gcd(a_1, \cdots a_n) = k$, for any positive integer k.
2. Calculate the probability $p_b(k)$ that k does not divide the right-hand side, for any positive integer k.
3. Calculate the probability:

$$p_g(n) = \sum_{k=1}^{\infty} p_a(n, k) * p_b(k) \tag{5}$$

This method supposes the existence and the knowledge of a probability law about the value of every constituent of the equation (coefficients and right-hand side). In the first place, we assume that the absolute values of these terms are independently and uniformly distributed in an interval $[1 \cdots N]$, with $N \rightarrow \infty$. We say that the numbers are "chosen at random".

The classical notations will be used:
$H_M^r = \sum_{k=1}^{M} 1/k^r$
$\zeta(r) = H_\infty^r$ ("zeta function" of Riemann)

with:
$$\zeta(2) = \pi^2/6$$
$$\zeta(3) \approx 1.202057$$
$$\zeta(4) = \pi^4/90)$$

First, let us recall a well-known result of number theory (G. Lejeune Dirichlet, 1849):

Theorem 4.2.2 *If u and v are integers chosen at random, the probability that $gcd(u, v) = 1$ is:* $p_a(2, 1) = 1/\zeta(2) = 6/\pi^2 \approx 0.60793$

Let us state it with a heuristic argument (see [Knu81], section 4.5.2):

1. We assume without proof the existence of a well-defined probability $p_a(2, 1)$.
2. Consequently, the probability that $gcd(u, v) = k$ (k divides u, k divides v, $gcd(u/k, v/k) = 1$) is:
$$p_a(2, k) = p_a(2, 1)/k^2 \qquad (6)$$
3. Let us sum all probabilities over all possible values of k:
$$1 = \sum_{k=1}^{\infty} p_a(2, k)$$
$$1 = p_a(2, 1) \sum_{k=1}^{\infty} 1/k^2$$
Hence, the theorem is established.

This result can be generalized (two or more variables):

Theorem 4.2.3 *If $a_1, a_2, \cdots a_n$ are integers chosen at random, the probability that $gcd(a_1, a_2, \cdots a_n) = 1$ is:* $p_a(n, 1) = 1/\zeta(n)$

Let us state this result as previously:

1. We assume without proof the existence of a well-defined probability $p_a(n, 1)$.
2. Therefore, the probability that $gcd(a_1, a_2, \cdots a_n) = k$ is:
$$p_a(n, k) = p_a(n, 1)/k^n \qquad (7)$$
3. Let us sum:
$$1 = \sum_{k=1}^{\infty} p_a(n, k)$$
$$1 = p_a(n, 1) \sum_{k=1}^{\infty} 1/k^n$$
Hence, the theorem is established.

Using (7), we can immediately generalize the previous theorem:

Theorem 4.2.4 *If $a_1, a_2, \cdots a_n$ are integers chosen at random, the probability that $gcd(a_1, a_2, \cdots a_n) = k$ is:* $p_a(n, k) = 1/(\zeta(n)\, k^n)$

Now, we can state:

Theorem 4.2.5 *If $a_{i1}, a_{i2}, \cdots a_{in}$ and b_i are integers chosen at random, the probability that the GCD-test is conclusive with the equation $\sum_{j=1}^{n} a_{ij} x_j = b_i$ is:* $p_g(n) = 1 - \zeta(n+1)/\zeta(n)$

Proof

Consider the statement (5):
$$p_g(n) = \sum_{k=1}^{\infty} p_a(n, k) * p_b(k)$$
The probability that k divides the right-hand side (chosen at random) is equal to $1/k$, hence:
$$p_b(k) = 1 - 1/k$$
Using theorem(4.2.4), the statement (5) becomes:
$$p_g(n) = \sum_{k=1}^{\infty} (1/(\zeta(n)\, k^n))(1 - 1/k)$$
$$p_g(n) = (1/(\zeta(n)))(\sum_{k=1}^{\infty} 1/k^n - \sum_{k=1}^{\infty} 1/k^{n+1})$$
$$p_g(n) = (1/\zeta(n))(\zeta(n) - \zeta(n+1))$$
and finally:
$$p_g(n) = 1 - \zeta(n+1)/\zeta(n)$$

Theorems 4.2.3 and 4.2.5 indicate:

Probability that $gcd(a_1, a_2, \cdots a_n) = 1$:
$$p_a(2, 1) = 1/\zeta(2) = \approx 0.60793$$
$$p_a(3, 1) = 1/\zeta(3) = \approx 0.84$$
$$p_a(4, 1) = 1/\zeta(4) = \approx 0.937$$

Probability that the GCD-test is conclusive:
$$p_g(2) = 1 - \zeta(3)/\zeta(2) \approx 0.26294$$
$$p_g(3) = 1 - \zeta(4)/\zeta(3) \approx 0.09960$$

From three variables, the probability is very low.

The case of data dependence analysis

The previous results were based on the hypothesis that the involved data were chosen at random. However, in data dependence analysis, data are not actually chosen at random. They are generally "small".

So, we could suppose that the absolute values of the coefficients are independently and uniformly distributed in an interval $[1 \cdots N]$, with some small value of N.

Note that most problems include unitary coefficients (in which case, the GCD-test is inconclusive), then, we may imagine that the GCD-test could be applied only if methods based on unitary coefficients do not work, i.e. some equation does not include any unitary coefficient. Consequently, we could consider the case where the absolute values of the coefficients are independently and uniformly distributed in an interval $[2 \cdots N]$, with some small value of N.

Hence, in place of theorem 4.2.2 for instance, we could fix the value of N and estimate simply by enumeration the probability that the GCD of two numbers is equal to 1. We would obtain:

N	$[1 \cdots N]$		$[2 \cdots N]$	
4	11/16	= 0.687	4/9	= 0.444
5	19/25	= 0.76	10/16	= 0.625
6	23/36	= 0.639	12/25	= 0.48
7	35/49	= 0.714	22/36	= 0.611
8	43/64	= 0.672	28/49	= 0.571
9	55/81	= 0.679	38/64	= 0.594
10	63/100	= 0.63	44/81	= 0.543
11	83/121	= 0.686	62/100	= 0.62
12	91/144	= 0.632	68/121	= 0.562
13	115/169	= 0.680	90/144	= 0.625
14	127/196	= 0.648	100/169	=0.592
15	143/255	= 0.636	114/196	=0.581

The values obtained from a small interval $[1 \cdots N]$ are slighty superior to the value (≈ 0.60793) given by the theorem 4.2.2, while they are comparable with an interval $[2 \cdots N]$.

We could operate in a similar way to estimate the probability that the GCD-test is conclusive. However, obtaining precise values is not very important. The results would not be significanly different from the theoretical values. The GCD-test is strongly inexact, and from three variables, it becomes very weak. It should be applied only as a "help" method before some exact or reduction method.

4.3 Integer solution to equations

With the GCD-test, we saw that we can decide if a single integer equation has a solution. With a set of equations, this test does not work, as we can see with this simple example:

$$2x_1 - 3x_2 = 4$$
$$3x_1 - 2x_2 = 2$$

Each of these equations has a solution. However, any system including both of them is incompatible; we may remark that by summing the original equations, we build a new equation which has no solution:

$$5x_1 - 5x_2 = 6$$

Moreover, as for a single equation, there is a possibility that a set of equations is compatible while the whole system is not.

However, a linear diophantine equations system can be solved. Then, if a system includes m_1 inequalities and m_2 equations, we can consider the subsystem composed by the equations and try to solve it. If the sub-system has no solution, then the full system is incompatible. If the subsystem has a solution, we may decide to stop (in this case the method is inexact, like the GCD-test), but we may decide to change in the set of inequalities the variables solved by the sub-system. If the rank of the matrix relative to the sub-system is r, we obtain a system with r variables less.

4.3.1 Single equation in two variables

Let the equation be:

$$a_1 x_1 + a_2 x_2 = b \tag{8}$$

If there is a solution to this equation (this supposes that b is a multiple of $d = gcd(a_1, a_2)$, see 4.2.1), it can be proved ([Ban88]) that the general solution $x = (x_1, x_2)$ can be obtained as follows:

1. Two auxiliary equations are defined:

$$a_1 y_1 + a_2 y_2 = 0 \tag{9}$$

$$a_1 z_1 + a_2 z_2 = d \tag{10}$$

2. A particular solution to (10) is given by the *Extended Euclid algorithm* (4.1.3):

$$z^0 = (z_1^0, z_2^0)$$

3. If b is not a multiple of d, then there is no solution to the original equation, else we define:

$$\lambda = b/d$$
$$\lambda_1 = -a_2/d$$
$$\lambda_2 = a_1/d$$

4. The general solution to (9) is given by:

$$y = (y_1, y_2) = (\lambda_1 t, \lambda_2 t) \tag{11}$$

where t is an arbitrary integer.

5. The general solution to (8) is:

$$x = y + \lambda z^0$$

which we write:

$$(x_1, x_2) = (\lambda_1 t + \lambda z_1^0, \lambda_2 t + \lambda z_2^0) \tag{12}$$

Comments

- If the equation includes a unitary coefficient, a solution can be directly obtained by an elimination (see 5.1).
- The value of the GCD, which is necessary for the computation (11) of the associated equation (9), is given by the Extended Euclid algorithm performed with the equation (10)

Example

$$\begin{array}{rrcl} x_1 & & \geq & 1 \\ x_1 & & \leq & 20 \\ & x_2 & \geq & 1 \\ & x_2 & \leq & 20 \\ 2x_1 & -3x_2 & = & 36 \end{array}$$

The particular auxiliary solution is:

$$\begin{array}{rcl} z_1^0 & = & 2 \\ z_2^0 & = & 1 \end{array}$$

We obtain:

$$\begin{array}{l} d = 1 \\ \lambda = 36 \\ \lambda_1 = 3 \\ \lambda_2 = 2 \end{array}$$

The general associated solution (11) is:

$$\begin{array}{rcl} y_1 & = & 3t \\ y_2 & = & 2t \end{array}$$

The general solution (12) is:

$$\begin{array}{rcll} x_1 & = & 72 & +3t \\ x_2 & = & 36 & +2t \end{array}$$

Then, we can change the inequalities:

$$\begin{array}{rcl} 3t & \geq & -71 \\ 3t & \leq & -52 \\ 2t & \geq & -35 \\ 2t & \leq & -16 \end{array}$$

We can immediatly conclude (see 6.1.3) that no solution exists.

4.3.2 Equation with more than two variables

$$a_1 x_1 + a_2 x_2 \cdots + a_n x_n = b \tag{13}$$

Let be

$$gcd(a_1, a_2 \cdots, a_n) = d$$

A solution exists if d divides b. The previous technique is generalized. Two auxiliary equations are introduced:

$$a_1 y_1 + a_2 y_2 \cdots + a_n y_n = 0 \tag{14}$$

$$a_1 z_1 + a_2 z_2 \cdots + a_n z_n = d \tag{15}$$

The general solution to (13) should be obtained by combining a general solution to (14) with a particular solution to (15):

$$x = y + (b/d)z$$

Several notations are possible for the following explanations. We will use the same ones as in [Ban88], where details and proofs are available. Let

$$A = (a_1, a_2 \cdots, a_n)^t$$

A $n \times n$ unimodular matrix U is built as follows. Let

$$U^{'}.A = D^{'}$$

We start with the values $U^{'} = I$ (identity matrix) and $D^{'} = A$. Applying elementary row operations, we can obtain:

$$U.A = D$$

where

$$D = (d, 0, \ldots, 0)^t$$

Then, the particular solution to (15) is the first row of U:

$$(z_1, z_2 \cdots, z_n) = (u_1, u_2 \cdots, u_n)$$

while the general solution to (14) is:

$$(y_1, y_2 \cdots, y_n) = (0, t_2 \cdots, t_n).U$$

4.3.3 System with more than one equation

This method [Ban88] is the extension of previous one to multi-dimensional systems, and is known as the Banerjee Extended GCD-Test when used in Loop bounds constraints relaxation.

It is based on the fact that an integer $m \times n$ matrix A can be factored in $A = DU$ where U is an integer unimodular matrix and D is a $m \times n$ echelon matrix (known as the reduced Hermite form of a matrix [Min83, Sch86]).

We will not describe this classical method, which is detailed in the quoted literature.

4.4 Square Rows Submatrix(Integrity relaxation)

We will now consider a special case, where the number of equations is greater than or equal to the number of variables. Let the following system be ($m_2 \geq n$):

$$\begin{array}{l} \sum_{j=1}^{n} a_{ij}x_j \leq b_i, \quad i \in \{1, \ldots m_1\} \\ \sum_{j=1}^{n} a_{ij}x_j = b_i, \quad i \in \{m_1+1, \ldots m_1+m_2\} \\ x_j \in \mathbb{Z} \end{array}$$

Using classical Gauss elimination or another one, we try to solve the equations sub-system (integrity relaxation):

$$\begin{array}{l} \sum_{j=1}^{n} a_{ij}x_j = b_i, \quad i \in \{m_1+1, \ldots m_1+m_2\} \\ x_j \in \mathbb{R} \end{array}$$

Then:

1. If no solution exists, the original system is impossible.
2. If more than one solution exists (the row rank of the matrix is smaller than the number n), we cannot conclude.
3. If a single solution exists:
   (a) If this solution is not integer, the original problem has no solution.
   (b) If one inequality is incompatible with this solution, the original system has no solution.
   (c) Else the system is feasible.

There is a possibility that the equations are not linearly independent (case 2), then strictly speaking, the method is inexact. In any case, rational arithmetic is required (see 2.5). An equations reduction (section 5) would relieve us from these problematical aspects.

First example

$$\begin{array}{rrcr} x_1 & & \geq & 0 \\ x_1 & & \leq & 20 \\ & x_2 & \geq & 0 \\ & x_2 & \leq & 10 \\ 8x_1 & -3x_2 & = & 1 \\ 4x_1 & +x_2 & = & 3 \end{array}$$

A single solution is obtained:

$$x_1 = 1/2 \qquad x_2 = 1$$

The solution is not integer, hence the systeme is infeasible.

Second example

$$\begin{array}{rrcr} x_1 & & \geq & 0 \\ x_1 & & \leq & 20 \\ & x_2 & \geq & 0 \\ & x_2 & \leq & 10 \\ 2x_1 & -3x_2 & = & 10 \\ 4x_1 & +3x_2 & = & 2 \end{array}$$

A single and integer solution is obtained:

$$x_1 = 2 \qquad x_2 = -2$$

However, the third inequality is not satisfied by this solution, then the system is infeasible.

5 Equations reduction

In this section, we suppose that the system includes equations and inequalities. Some techniques for reducing the original problem, as long as diophantine equations are present, are described. We do not try to explicitly solve equations, as in section 4.

If the original system has the following form:

$$\begin{array}{l} \sum_{j=1}^{n} a_{ij}x_j \leq b_i, \quad i \in \{1, \ldots m_1\} \\ \sum_{j=1}^{n} a_{ij}x_j = b_i, \quad i \in \{m_1+1, \ldots m_1+m_2\} \\ x_j \in \mathbb{Z} \end{array}$$

we may obtain a succession of equivalent systems:

$$\begin{array}{l} \sum_{j=1}^{n'} a'_{ij}x'_j \leq b'_i, \quad i \in \{1, \ldots m_1\} \\ \sum_{j=1}^{n'} a'_{ij}x'_j = b'_i, \quad i \in \{m_1+1, \ldots m_1+m'_2\} \\ x'_j \in \mathbb{Z} \end{array}$$

and unless an impossibility is proved during a reduction step, we come easily to the equivalent form:

$$\begin{array}{l} \sum_{j=1}^{n''} a''_{ij}x''_j \leq b''_i, \quad i \in \{1, \ldots m_1\} \\ x''_j \in \mathbb{Z} \end{array}$$

If we note by r_2 (generally equal to m_2) the rank of the matrix defined by original equations, the final number n'' of variables is:

$$n'' = n - r_2$$

We deliberately denote the variables of the final system: x''_j because they are not necessarily the same ones (see 5.4) as the original variables, which we can now forget about.

5.1 Integer Gauss elimination

This technique eliminates one equation and one variable: if one of the coefficients of an equation has an unitary value (i.e. ± 1), a Gaussian elimination of the variable associated with that coefficient will build an equivalent and smaller integer system. The expression of the eliminated variable includes only integer coefficients, so we are guaranteed that the transformed system remains equivalent to the previous one. Let us consider the following system:

$$\begin{array}{rrrcr} x_1 & +x_2 & & \leq & 10 \\ & x_2 & -4x_3 & \leq & 12 \\ -6x_1 & +x_2 & -7x_3 & = & -4 \\ -4x_1 & +3x_2 & -2x_3 & = & -1 \end{array}$$

The first equation gives an integer expression of x_2:

$$x_2 = 6x_1 + 7x_3 - 4$$

Therefore, by replacing x_2, we derive the equivalent system:

$$\begin{array}{rrcr} 7x_1 & +7x_3 & \leq & 14 \\ 6x_1 & +3x_3 & \leq & 16 \\ 14x_1 & +19x_3 & = & 11 \end{array}$$

and we can now forget about x_2.

Since this can be applied when one coefficient is ± 1, many other techniques described in this section are aimed at generating unitary coefficients.

5.1.1 Choice of the pivot

If several coefficients ("pivots") compete for a integer Gauss elimination, a good strategy consists in choosing the sparsest column (or the sparsest row). This minimizes the amount of computation, keeps the system matrix sparse and keeps the coefficients unitary. This also makes the next steps easier (chapters 6 and 7).

Let's have a look at this system:

$$\begin{array}{rrrrcr} -4x_1 & +6x_2 & & +x_4 & \leq & -9 \\ 4x_1 & & -x_3 & & \leq & 9 \\ -3x_1 & & +x_3 & & \leq & -20 \\ 2x_1 & -x_2 & +2x_3 & & = & 12 \\ x_1 & +2x_2 & -5x_3 & +x_4 & = & -5 \end{array}$$

We can eliminate the first equation (and x_2) or the second one (and x_1 or x_4). Let us choose the second equation and x_4:

$$\begin{array}{rrrcr} -5x_1 & +4x_2 & +5x_3 & \leq & -4 \\ 4x_1 & & -x_3 & \leq & 9 \\ -3x_1 & & +x_3 & \leq & -20 \\ 2x_1 & -x_2 & +2x_3 & = & 12 \end{array}$$

Three rows (the second and the third inequalities and, which currently interests us, the first equation) are unchanged since their coefficients in column x_4 were equal to *zero*. Then, we can eliminate the first equation (and x_2):

$$\begin{array}{rrcr} 3x_1 & +13x_3 & \leq & 44 \\ 4x_1 & -x_3 & \leq & 9 \\ -3x_1 & +x_3 & \leq & -20 \end{array}$$

For the first elimination, we could also choose x_2 from the first equation since it would have kept two rows (the second and third inequalities) and the column x_4 unchanged.

The worst choice would have been x_1 from the second equation:

$$\begin{array}{rrrcr} 14x_2 & -20x_3 & +5x_4 & \leq & -29 \\ -8x_2 & +19x_3 & -4x_4 & \leq & 29 \\ 6x_2 & -14x_3 & +3x_4 & \leq & -35 \\ -5x_2 & +12x_3 & -2x_4 & = & 22 \end{array}$$

5.1.2 Data overflow risk

Consider the system:

$$\begin{array}{l} \sum_{j=1}^{n} a_{ij}x_j \leq b_i, \quad i \in \{1, 2, \ldots m_1\} \\ \sum_{j=1}^{n} a_{ij}x_j = b_i, \quad i \in \{m_1 + 1, \ldots m_1 + m_2\} \end{array}$$

Suppose we perform a gaussian elimination of x_k from the equation with index r ($m_1 + 1 \leq r \leq m_1 + m_2$). The new system is:

$$\begin{array}{r} \sum_{j \neq k}(a_{ij} - a_{rj}a_{ik}/a_{rk})x_j \leq b_i - b_r a_{ik}/a_{rk}, \\ i \in \{1, 2, \ldots m_1\} \\ \sum_{j \neq k}(a_{ij} - a_{rj}a_{ik}/a_{rk})x_j = b_i - b_r a_{ik}/a_{rk}, \\ i \neq r \in \{m_1 + 1, \ldots m_1 + m_2\} \end{array}$$

Since $a_{rk} = \pm 1$, the values of the new coefficients and right-hand sides are:

$$\begin{array}{c} a'_{ij} = a_{ij} \pm a_{rj}a_{ik} \\ b'_i = b_i \pm b_r a_{ik} \end{array}$$

These values may increase exponentially at each iteration. Let us see this system:

$$\begin{array}{rrrrcr} -4x_1 & +6x_2 & & +x_4 & \leq & -9 \\ 4x_1 & & -x_3 & & \leq & 9 \\ -3x_1 & & +x_3 & & \leq & -20 \\ 200x_1 & -10x_2 & +3x_3 & & = & 12 \\ x_1 & +2x_2 & -5000x_3 & +10x_4 & = & -5 \end{array}$$

After the gaussian elimination of x_1, it becomes:

$$\begin{array}{rrrcr} 14x_2 & -2000x_3 & +41x_4 & \leq & -29 \\ -8x_2 & +19999x_3 & -40x_4 & \leq & 29 \\ 6x_2 & -14999x_3 & +30x_4 & \leq & -35 \\ -410x_2 & +1000003x_3 & -2x_4 & = & 1012 \end{array}$$

This is the well-known problem of expression swell (see [DKT87]), which must not be underestimated. However, in most systems coming from data dependence analysis as well as others, the size of coefficients a_{ij} is very small (less than 10) and should not make a data overflow appear. As for the right-hand sides b_i which may have greater values than those of coefficients a_{ij}, they are not involved quadratically in the upper formulas.

Therefore, in this paper, this problem will not be discussed.

5.2 Division by GCD

When the GCD test (4.2.1) fails and the GCD is not 1, then performing the (integer) division of all the coefficients by the GCD diminishes the value of coefficients. This will make the other stages less complex. Furthermore, this is likely to make a ± 1 coefficient appear. For instance, the following equation:

$$4x_1 - 2x_2 + 4x_3 = 110$$

becomes (after division by 2):

$$2x_1 - 1x_2 + 2x_3 = 55$$

Now x_2 elimination can be performed.

The same technique can be applied to inequalities (see 6.1.4).

5.3 Mixing two equations

Systems can also be transformed by combining equations: it is straightforward to show that the two following equations:

$$\begin{array}{rcl} \sum_{k=1}^{n} a_{ik}x_k & = & b_i \\ \sum_{k=1}^{n} a_{jk}x_k & = & b_j \end{array}$$

can be replaced by the equivalent system:

$$\begin{array}{rcl} \sum_{k=1}^{n} a_{ik}x_k & = & b_i \\ \sum_{k=1}^{n} (\lambda_i a_{ik} + \lambda_j a_{jk})x_k & = & \lambda_i b_i + \lambda_j b_j \end{array}$$

provided that λ_j is non-zero. If we find λ_i and λ_j such that

$$(\lambda_i a_{ik} + \lambda_j a_{jk}) = \pm 1 \qquad (16)$$

for one k, then x_k elimination can again be performed.

This can be achieved as soon as a_{ik} and a_{jk} are relatively prime, in which case the extended Euclid algorithm (4.1.3) finds two such integers λ_i and λ_j.

Consider for instance the system:

$$\begin{array}{rrrcr} x_1 & +x_2 & & \leq & 10 \\ & x_2 & -4x_3 & \leq & 12 \\ -2x_1 & +5x_2 & +3x_3 & = & 2 \\ -4x_1 & +3x_2 & -2x_3 & = & -1 \end{array}$$

The coefficients of x_1 are -2 and -4. Since their GCD is 2, this technique does not work. However, the coefficients of x_2 are 5 and 3. Their GCD is 1 and we find that:

$$(-1)5 + (2)3 = 1$$

Thus, by multiplying the first equation by $\lambda_1 = -1$ and second one by $\lambda_2 = 2$ and summing them, we obtain:

$$-6x_1 \quad +x_2 \quad -7x_3 \quad = \quad -4$$

Hence the new system is:

$$\begin{array}{rrrcr} x_1 & +x_2 & & \leq & 10 \\ & x_2 & -4x_3 & \leq & 12 \\ -6x_1 & +x_2 & -7x_3 & = & -4 \\ -4x_1 & +3x_2 & -2x_3 & = & -1 \end{array}$$

where now x_2 elimination can be performed.

5.4 Changing two variables

Instead of combining equations, it is also possible to "combine" two variables in some sense, by a judicious change of variables. Imagine that in the following equation

$$\sum_{j=1}^{n} a_j x_j \quad = \quad d \qquad (17)$$

a_1 and a_2 are relatively prime. Then any integer t_1 can be written in the form $t_1 = a_1 x_1 + a_2 x_2$, where x_1 and x_2 are integers. This suggests the use of t_1 as a new variable. In order to maintain an equivalent system, a new variable t_2 must also be introduced. Variable changing is thus described by a transformation such as:

$$\left\{ \begin{array}{rcrr} t_1 & = & a_1 x_1 & +a_2 x_2 \\ t_2 & = & b_1 x_1 & +b_2 x_2 \end{array} \right.$$

For this transformation to be unimodular, we must have

$$a_1 b_2 - a_2 b_1 = \pm 1$$

Again the extended Euclid algorithm provides two such integer coefficients b_1 and b_2 (if $\lambda_1 a_1 + \lambda_2 a_2 = 1$, we take $b_1 = -\lambda_2$ and $b_2 = \lambda_1$).

Now it is easy to find the inverse transformation:

$$\left\{ \begin{array}{rcrr} x_1 & = & \lambda_1 t_1 & -a_2 t_2 \\ x_2 & = & \lambda_2 t_1 & +a_1 t_2 \end{array} \right.$$

Therefore, x_1 and x_2 are replaced by t_1 and t_2 everywhere in the system and equation (17) becomes

$$t_1 + \sum_{j=3}^{n} a_j x_j \quad = \quad d$$

where t_1 elimination can be performed.

Let us consider the previous example:

$$\begin{array}{rrrcr} x_1 & +x_2 & & \leq & 10 \\ & x_2 & -4x_3 & \leq & 12 \\ -2x_1 & +5x_2 & +3x_3 & = & 2 \\ -4x_1 & +3x_2 & -2x_3 & = & -1 \end{array}$$

In the last equation, we observe that we can apply this transformation to (x_1, x_2) or (x_2, x_3) (but not to (x_1, x_3) since the value of $GCD(-4, -2)$ is 2). We choose (x_1, x_2) for instance and obtain:

$$\begin{array}{rcrr} x_1 & = & -t_1 & -3t_2 \\ x_2 & = & -t_1 & -4t_2 \end{array}$$

and the system becomes:

$$\begin{array}{rrrcr} -2t_1 & -7t_2 & & \leq & 10 \\ -t_1 & -4t_2 & -4x_3 & \leq & 12 \\ -3t_1 & -14t_2 & +3x_3 & = & 2 \\ t_1 & & -2x_3 & = & -1 \end{array}$$

A Gauss elimination is now possible with variable t_1.

5.5 Shortening the number of variables of an equation

Even when there is no pair of relatively prime coefficients, the technique of changing variables can be used to eliminate a variable. By applying this technique iteratively, we obtain the case when two coefficients are relatively prime, provided that the GCD of the coefficients of the original equation was 1. If $GCD(a_1, a_2) = p$, then we can create the new variable t_1 such that $pt_1 = a_1x_1 + a_2x_2$ i.e. $t_1 = \frac{a_1}{p}x_1 + \frac{a_2}{p}x_2$, which yields to the previous case. Then we compute λ_1 and λ_2 such that $\lambda_1\frac{a_1}{p} + \lambda_2\frac{a_2}{p} = 1$ and perform the following variable change:

$$\left\{ \begin{array}{rcrr} t_1 & = & \frac{a_1}{p}x_1 & +\frac{a_2}{p}x_2 \\ t_2 & = & -\lambda_2 x_1 & +\lambda_1 x_2 \end{array} \right.$$

with inverse:

$$\left\{ \begin{array}{rcrr} x_1 & = & \lambda_1 t_1 & -\frac{a_2}{p}t_2 \\ x_2 & = & \lambda_2 t_1 & +\frac{a_1}{p}t_2 \end{array} \right.$$

Now equation (17) becomes:

$$pt_1 + \sum_{j=3}^{n} a_j x_j \quad = \quad d$$

As an example, consider the following system:

$$\begin{array}{rrrcr} x_1 & +x_2 & & \leq & 10 \\ & x_2 & -4x_3 & \leq & 12 \\ -6x_1 & -10x_2 & +15x_3 & = & 2 \end{array}$$

There is no equation which contains a pair of coefficients which are relatively prime, so we consider (x_1, x_2) and compute the formula:

$$\begin{array}{rcrr} x_1 & = & -2t_1 & +5t_2 \\ x_2 & = & t_1 & -3t_2 \end{array}$$

and the system becomes:

$$\begin{array}{rrrcr} -t_1 & +2t_2 & & \leq & 10 \\ t_1 & -3t_2 & -4x_3 & \leq & 12 \\ 2t_1 & & +15x_3 & = & 2 \end{array}$$

Now the two coefficients of the equation are relatively prime and the system can be reduced with the previous method (section 5.4).

5.6 Fast GCD

The methods discussed above make intensive use of GCD calculations. Therefore, a fast algorithm for determining the GCD of two numbers is desirable. The principle of our method is to have in memory a table of precomputed GCD's, say a table of GCD of every pair of integers less than a given value m. Then, if it happens that the numbers whose GCD we want to compute are less that m, it is straightforward and does not cost anything to consult the table. If not, then the table can still be of use for speeding up the GCD computation. To explain this, let us recall the Euclid algorithm for computing the GCD of two positive numbers a and b such that a is greater than b:

```
gcd (a,b) =
        { if  b = 0 then a
                    else gcd (b, a mod b) }
```

Now, if we have a table *tgcd* such that $gcd(a, b) = tgcd(a, b)$ for any a and b less than m, then the algorithm becomes:

```
gcd (a,b) =
        {if (b = 0) then a
                    else
                    { if  a ≤ m then tgcd (a,b)
                    else gcd (b, a mod b) } }
```

In systems issuing from data dependence analysis, coefficients are often small. In this case, one memory reference suffices to compute the GCD of two numbers. As a result, the methods described above run fast.

Let us see how we could compute $gcd(16, 6)$ if the following table $tgcd$ ($m = 10$) were at our disposal:

	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1
2	1	2	1	2	1	2	1	2	1	2
3	1	1	3	1	1	3	1	1	3	1
4	1	2	1	4	1	2	1	4	1	2
5	1	1	1	1	5	1	1	1	1	5
6	1	2	3	2	1	6	1	2	3	2
7	1	1	1	1	1	1	7	1	1	1
8	1	2	1	4	1	2	1	8	1	2
9	1	1	3	1	1	3	1	1	9	1
10	1	2	1	2	5	2	1	2	1	10

Euclid algorithm:

$$\begin{array}{lll} gcd(16,6) & = gcd(6, 16 \bmod 6) & = gcd(6,4) \\ gcd(6,4) & = gcd(4, 6 \bmod 4) & = gcd(4,2) \\ gcd(4,2) & = gcd(2, 4 \bmod 2) & = gcd(2,0) \\ gcd(2,0) & = 2 & \end{array}$$

"Precomputed" method:

$$\begin{array}{lll} gcd(16,6) & = gcd(6, 16 \bmod 6) & = gcd(6,4) \\ gcd(6,4) & = tgcd(6,4) & = 2 \end{array}$$

5.7 Fast extended Euclid algorithm

The EEA (extended Euclid algorithm, 4.1.3) for determining coefficients of equation 16 can be adapted in the same manner and make use of a table of precomputed Bezout's coefficients.

However, having a fast EEA is less important than having a fast GCD. In order to change two variables for instance (5.4), the selection of the set of variables may need several computations of GCD, while the change of the chosen set needs a single computation of EEA.

5.8 Equations reduction: conclusion

Various strategies are possible, for instance the following one:

1. Test the following case, and stop if it works:

   No equation remains.

2. Search a unitary coefficient. If found:
   (a) Apply the Gauss elimination (5.1).
   (b) Go to 1.

3. Search an equation where every coefficient = 0. If found:
   (a) If the RHS (right-hand side) $\neq 0$, stop (infeasibility).
   (b) Remove the equation.
   (c) Go to 1.

   Note: Steps 2 and 3 can easily be mixed.

4. Choose an equation. Then:
   (a) Apply the GCD-test. If it is conclusive, stop (infeasibility).
   (b) If $GCD \neq 1$:
      i. Divide the equation by GCD.
      ii. If a unitary coefficient appears, go to 2a.

5. Search two relatively prime coefficients (5.4 or 5.3). If found:
   (a) Build a unitary coefficient.
   (b) Go to 2a.

6. (a) Shorten the number of non-zero coefficients (5.5).
   (b) Go to 5.

6 Inequalities: variables eliminations

In this chapter, we suppose that the system has the following form:

$$\begin{array}{l} \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i \in \{1, \ldots m\} \\ x_j \in \mathbb{Z} \end{array}$$

6.1 Particular techniques

Some simple criteria may avoid calculation. The following cases may seem quite trivial and without interest, however, with the specific problems of data dependence analysis, they should frequently appear after some system reduction (gauss (5.1), partial Fourier-Motzkin (6.2)).

6.1.1 Non negative right-hand sides

If all right-hand sides (b_i) are non negative, the systeme is compatible since an obvious integer solution exists (all x_j equal to zero). For instance:

$$\begin{array}{rrrcr} 2x_1 & -3x_2 & +2x_3 & \leq & 2 \\ -x_1 & +4x_2 & -2x_3 & \leq & 0 \end{array}$$

6.1.2 Single inequality

A system including a single inequality is compatible this case excepted: all the coefficients are equal to *zero* and the right-hand side is negative.

6.1.3 Single variable

If a system includes a single variable:

$$a_{ik}x_k \leq b_i, \quad i \in \{1, \ldots m\}$$

inequalities can be replaced by [1]:

$$x_k \leq \lfloor b_i/a_{ik} \rfloor \quad \forall a_{ik} > 0 \tag{18}$$

$$x_k \geq \lfloor b_i/a_{ik} \rfloor \quad \forall a_{ik} < 0 \tag{19}$$

This is a particular case of the inequality division by GCD (6.1.4). The solution is trivial.

Let us consider the following system:

$$\begin{array}{rcr} -18x_2 & \leq & -183 \\ x_2 & \leq & 24 \\ 18x_2 & \leq & 207 \end{array}$$

We get a compatible system:

$$\begin{array}{rcl} x_2 & \geq & 11 \\ x_2 & \leq & 24 \\ x_2 & \leq & 11 \end{array}$$

6.1.4 Inequality division by GCD

Since coefficients ± 1 are desirable for reduction steps (6.2 and 7), we can apply to inequalities the technique used with equations. Let the inequality be:

$$\textstyle\sum_{j=1}^{n} a_j x_j \leq b$$

We note $d = gcd(a_1, \cdots a_n)$. If its value is not equal to 1, the inequality can be replaced by:

$$\sum_{j=1}^{n} (a_j/d) x_j \leq \lfloor b/d \rfloor \tag{20}$$

[1] $\lfloor x \rfloor$ stands for the greatest integer less than or equal to x

With analoguous hypothesis and results as in 4.2.2, we could estimate the probability that an inequality with n variables can be simplified ($GCD \neq 1$):

$$\begin{array}{ll} n = 2 & \approx 0.40 \\ n = 3 & \approx 0.16 \\ n = 4 & \approx 0.063 \end{array}$$

Let us consider the system,:

$$\begin{array}{rrrcr} 2x_1 & -3x_2 & & \leq & 2 \\ -2x_1 & +8x_2 & -4x_3 & \leq & -11 \\ & x_2 & +2x_3 & \leq & 8 \\ & & -x_3 & \leq & -5 \end{array}$$

We can divide the second inequality ($GCD = 2$):

$$\begin{array}{rrrcr} 2x_1 & -3x_2 & & \leq & 2 \\ -x_1 & +4x_2 & -2x_3 & \leq & -6 \\ & x_2 & +2x_3 & \leq & 8 \\ & & -x_3 & \leq & -5 \end{array}$$

6.2 Selective Fourier-Motzkin

The classical Fourier-Motzkin elimination method ([Dan63, Sch86]) applies to systems including only inequalities. At every step, a variable is eliminated while the number of inequalities is modified (it may increase or decrease). The process runs until none remains or until a contradiction is found. The method is known to be rather computationally expensive, especially for large systems because it may generate an *exponential* number of additional inequalities. Furthermore, the result it gives is exact for real variables but *non exact* in general when variables are constrained to be integers.

However, in this section, we show that the Fourier-Motzkin method can serve as a preprocessing step for reducing the system by partial variables elimination, while still working in equivalent systems, hence keeping the method exact.

First, let us describe the general method.

The classical algorithm

We suppose that the current system S has m inequalities with n variables:

$$a_{i_1}x_1 + a_{i_2}x_2 \cdots + a_{i_n}x_n \leq b_i$$

A variable, x_k, is chosen, and we split inequalities into three classes:

$$\begin{array}{l} I_1 = \{i \ / a_{ik} < 0\} \\ I_2 = \{i \ / a_{ik} > 0\} \\ I_3 = \{i \ / a_{ik} = 0\} \end{array}$$

Let us set:

$$m_1 = |I_1|$$
$$m_2 = |I_2|$$
$$m_3 = |I_3|$$

The notation $x^{'}$ will designate now the $n^{'}$-vector ($n^{'} = n - 1$) of variables $x_j, j \neq k$
Inequalities where the variable x_k is present (classes 1 and 2) can be written thus:

$$\forall i / a_{ik} \neq 0, \quad a_{ik} x_k + g_{ik}(x^{'}) \leq 0$$

(note that $g_{ik}(x^{'})$ is a linear fonction with integer coefficients)
Let us formulate thus:

$$-a_{ik} x_k \geq g_{ik}(x^{'}), i \in I_1 \tag{21}$$

$$a_{ik} x_k \leq -g_{ik}(x^{'}), i \in I_2 \tag{22}$$

If we define:

$$f_{ik}(x^{'}) = -g_{ik}(x^{'})/a_{ik} \tag{23}$$

we obtain:

$$x_k \geq f_{ik}(x^{'}), i \in I_1 \tag{24}$$

$$x_k \leq f_{ik}(x^{'}), i \in I_2 \tag{25}$$

A new system, $S^{'}$ (with $n^{'}$ variables), is defined:

- Inequalities including variable x_k (class 1 and class 2) are replaced by $m_1 m_2$ new inequalities. From every inequality of the first class (24) and every inequality of the second class (25), an inequality is built:

  $$\forall i_1 \in I_1, \forall i_2 \in I_2, \quad f_{i_1 k}(x^{'}) \leq f_{i_2 k}(x^{'}) \tag{26}$$

  In order to maintain integer coefficients, we can write:

  $$\forall i_1 \in I_1, \forall i_2 \in I_2, \quad a_{i_2 k} g_{i_1 k}(x^{'}) - a_{i_1 k} g_{i_2 k}(x^{'}) \leq 0 \tag{27}$$

- Inequalities where variable x_k is missing (class 3) are unchanged.

Comments

- *The general method is not exact*

  1. Obviously, if an integer solution x to system S exists, an integer solution $x^{'}$ to system $S^{'}$ exists.
  2. We cannot assert the converse. If an (integer) solution $x^{'}$ to system $S^{'}$ exists, we have necessarily :

     $$\max_{i_1 \in I_1} f_{i_1 k}(x^{'}) \leq \min_{i_2 \in I_2} f_{i_2 k}(x^{'})$$

     then, from this solution $x^{'}$ and any integer value of x_k such that:

     $$\max_{i_1 \in I_1} f_{i_1 k}(x^{'}) \leq x_k \leq \min_{i_2 \in I_2} f_{i_2 k}(x^{'}) \tag{28}$$

     we can obtain an integer solution x to system S. However, the interval (28) may be not empty while containing no integer.

- *The number of inequalities may increase exponentially.*

  At every step, $m_1 + m_2$ inequalities are replaced by $m_1 m_2$ inequalities.

A selective algorithm

We may conceive an application of this method not to the whole set of variables, but only to some of them, in the case where we are certain that the method is exact and the number of inequalities will not increase.

- We noted that the method is exact if it can be proved that at every step the useful interval (28) contains an integer. This occurs if the interval is infinite or if one of the bounds has an integer value.

  The interval is infinite if one of the sets I_1 and I_2 is empty (all non-zero coefficients a_{ik} have the same sign).

  The lower bound ($\max_{i_1 \in I_1} f_{i_1 k}(x^{'})$) has an integer value if every function $f_{i_1 k}(x^{'})$ has an integer value. This happens if every function has integer coefficients, which is guaranteed if every coefficient $a_{i_1 k}$ has a value equal to -1 (see the expression (23)).

  Likewise, we can prove that the upper bound ($\min_{i_2 \in I_2} f_{i_2 k}(x^{'})$) has an integer value if every coefficient $a_{i_2 k}$ has a value equal to 1.

  Then, we can state:

  Theorem 6.2.1 *The transformed system is equivalent to the previous one when one of the following criteria is verified:*

  Criterion 1 *The value of every negative coefficient of x_k is -1*

  Criterion 2 *The value of every positive coefficient of x_k is 1*

- There still remains the problem of the number of generated inequalities. It is clear that it will not increase in any of these cases:

  Case 1 $m_1 \leq 1$

  Case 2 $m_2 \leq 1$

  Case 3 $m_1 = 2$ and $m_2 = 2$

Therefore it is worth applying this method as long as the system includes more than one variable and we can find a variable satisfying both conditions of exactitude and limitation. If a single variable remains, the problem is trivial (6.1.3) Moreover, every time a variable is eliminated, it should be seen whether particular techniques can be applied.

Example

Let us consider the following system:

$$\begin{array}{rrrcr} 2x_1 & -3x_2 & & \leq & 2 \\ -x_1 & +4x_2 & -2x_3 & \leq & -6 \\ & x_2 & +2x_3 & \leq & 8 \\ & & -x_3 & \leq & -5 \end{array}$$

x_1 is the only variable we can eliminate (Criterion 1, Case 1). The two first inequalities are replaced by an inequality of type (27):

$$\begin{array}{rrcr} 5x_2 & -4x_3 & \leq & -10 \\ x_2 & +2x_3 & \leq & 8 \\ & -x_3 & \leq & -5 \end{array}$$

In this new system, the variable x_2 (Criterion 1, Case 1) can be chosen. The two first inequalities are removed. A single inequality remains:

$$-x_3 \leq -5$$

This last inequality has an obvious integer solution, therefore we can conclude that the initial system has also an integer solution.

7 Introducing non-negative variables

Most algorithms used to solve the general linear integer programming problem (section 11) require that variables be "constrained", i.e. of non-negative value. Generally the problems as originally posed include "free" (unconstrained) variables. Directly replacing free variables by constrained variables is naturally possible (see 7.1) , but these classical techniques increase the number of variables. However, building an equivalent system without increasing the number of variables is always possible. Moreover, the techniques which will be detailed in this chapter lead to reduced systems (see lemma 7.3.3).

In this section, we suppose that the system has the following form:

$$\begin{array}{l} \sum_{j=1}^{n_1} a_{ij}x_j + \sum_{j=n_1+1}^{n_1+n_2} a_{ij}y_j \leq b_i, \quad i \in \{1, \ldots m\} \\ x_j \in \mathbb{Z}, \ \mathsf{y_j} \in \mathbb{Z}, \ \mathsf{y_j} \geq 0 \end{array}$$

although most of described methods could apply to systems including equations.

First, let us recall classical methods.

7.1 Added variables techniques

7.1.1 Doubling the number of variables

A very simple technique consists in replacing each free variable by two non-negative variables, as with this example:

$$\begin{array}{rrrcr} -4x_1 & -2x_2 & +x_3 & \leq & -9 \\ -3x_1 & +x_2 & & \leq & -4 \\ x_1 & -x_2 & +2x_3 & \leq & -3 \\ 2x_1 & +2x_2 & -5x_3 & \leq & 5 \end{array}$$

Changing the (free) variables as follows:

$$x_i = y_i - z_i \quad (y_i \geq 0, z_i \geq 0)$$

we obtain the equivalent (but bulky!) system:

$$\begin{array}{rrrrrrcr} -4y_1 & +4z_1 & -2y_2 & +2z_2 & +y_3 & -z_3 & \leq & -9 \\ -3y_1 & +3z_1 & +y_2 & -z_2 & & & \leq & -4 \\ y_1 & -z_1 & -y_2 & +z_2 & +2y_3 & -2z_3 & \leq & -3 \\ 2y_1 & -2z_1 & +2y_2 & -2z_2 & -5y_3 & +5z_3 & \leq & 5 \end{array}$$

7.1.2 "Single added variable"

An often used technique replaces n free variables x_i by $n+1$ constrained variables:

$$x_i = y_i - y_0$$

Then, changing the variables thus:

$$\begin{array}{l} x_1 = y_1 - y_0 \\ x_2 = y_2 - y_0 \\ x_3 = y_3 - y_0 \end{array}$$

the system of the previous example becomes:

$$\begin{array}{rrrrcr} -4y_1 & -2y_2 & +y_3 & +5y_0 & \leq & -9 \\ -3y_1 & +y_2 & & +2y_0 & \leq & -4 \\ y_1 & -y_2 & +2y_3 & -2y_0 & \leq & -3 \\ 2y_1 & +2y_2 & -5y_3 & +y_0 & \leq & 5 \end{array}$$

Note that this technique can be applied to systems including both free and constrained variables, for instance the one obtained in 7.2:

$$\begin{array}{rrrcr} 18x_1 & -3y_5 & -2y_4 & \leq & 27 \\ -42x_1 & +8y_5 & +5y_4 & \leq & -72 \end{array}$$

The resulting system would be:

$$\begin{array}{rrrrcr} 18y_6 & -18y_7 & -3y_5 & -2y_4 & \leq & 27 \\ -42y_6 & +42y_7 & +8y_5 & +5y_4 & \leq & -72 \end{array}$$

7.2 "Dummy" eliminations

Systems of inequalities stemming from data dependence analysis involve generally *unitary* coefficients. Then, by introducing a slack (dummy) variable ≥ 0, every such inequality can be replaced by an equality, yielding a possibility of Gaussian elimination. For instance:

$$a_{i1}x_1 + a_{i2}x_2 \cdots + a_{ij}x_j \cdots + a_{in}x_n \leq b_i$$

The equivalent equation ($y_i \geq 0$ added) is:

$$y_i + a_{i1}x_1 + a_{i2}x_2 \cdots + a_{ij}x_j \cdots + a_{in}x_n = b_i$$

y_i is necessarily an integer variable. Now suppose that the chosen free variable (with a unitary coefficient) is x_1. We can write:

$$x_1 = [b_i - y_i - a_{i2}x_2 \cdots - a_{ij}x_j \cdots - a_{in}x_n]/a_{i1}$$

All the coefficients of this expression are integer since the value of a_{i1} is ± 1. Then, if we change the variable x_1 in all other equations or inequalities of the system, we obtain a new system where all coefficients are integer. At the same time, an inequality has disappeared from the system. Note that there is no assumption that the other variables ($x_2 \cdots x_n$) are free (they could be constrained).

Consider the previous example (7.1):

$$\begin{array}{rrrcr} -4x_1 & -2x_2 & +x_3 & \leq & -9 \\ -3x_1 & +x_2 & & \leq & -4 \\ x_1 & -x_2 & +2x_3 & \leq & -3 \\ 2x_1 & +2x_2 & -5x_3 & \leq & 5 \end{array}$$

It involves four unitary coefficients. Let us select x_3 and the first constraint (introducing y_4):

$$\begin{array}{rrrrcr} y_4 & -4x_1 & -2x_2 & +x_3 & = & -9 \\ & -3x_1 & +x_2 & & \leq & -4 \\ & x_1 & -x_2 & +2x_3 & \leq & -3 \\ & 2x_1 & +2x_2 & -5x_3 & \leq & 5 \end{array}$$

After the elimination of x_3, the new equivalent system is:

$$\begin{array}{rrrcr} -3x_1 & +x_2 & & \leq & -4 \\ 9x_1 & +3x_2 & -2y_4 & \leq & 15 \\ -18x_1 & -8x_2 & +5y_4 & \leq & -40 \end{array}$$

A similar operation is possible with the variable x_2 and the upper inequality:

$$y_5 - 3x_1 + x_2 = -4$$

The system is now:

$$\begin{array}{rrrcr} 18x_1 & -3y_5 & -2y_4 & \leq & 27 \\ -42x_1 & +8y_5 & +5y_4 & \leq & -72 \end{array}$$

Such an operation is not possible with the last variable x_1. Then, we could use an added variables technique (7.1). However, a "redundant" technique (7.4) would avoid increasing the size of the system.

A "dummy" elimination works technically like a gaussian elimination (equations, 5.1). Consequently:

- **Choice of the pivot**

  We can adopt the same "sparse" strategy (see 5.1.1): choose the pivot from the sparsest column (or the sparsest row).
  In the previous example, the first pivot was chosen from the sparsest column (x_3). If it had been chosen from the full column x_1 (and the full third row), the resulting system would not have included any unitary coefficient:

$$\begin{array}{rrrcr} 4y_6 & -6x_2 & +9x_3 & \leq & -21 \\ 3y_6 & -2x_2 & +6x_3 & \leq & -13 \\ -2y_6 & +4x_2 & -9x_3 & \leq & 11 \end{array}$$

- **Expression swell**

  For the same reasons as in 5.1.2, the problem of data overflow will not be discussed here (or in the rest of this paper).

The following result should avoid ulterior computation (see 7.6):

Theorem 7.2.1 *The GCD of the set of coefficients (Γ_i) of each remaining inequality is unchanged by a "dummy" elimination.*

Proof:
Consider the system including the free variable x_k:

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i, \quad i \in \{1, 2, \ldots m\}$$

Let us note the "GCD" of each inequality as follows:

$$\Gamma_i = gcd(a_{i1}, \cdots, a_{in}) \qquad (29)$$

Suppose we perform a "dummy" elimination of x_k from the inequality r:

$$y_r + \sum_{j=1}^{n} a_{rj} x_j = b_r$$

The new system is:

$$\begin{aligned}(-a_{ik}/a_{rk})y_r + \textstyle\sum_{j\neq k}(a_{ij} - a_{rj}a_{ik}/a_{rk})x_j \\ \leq b_i - b_r a_{ik}/a_{rk}, \quad i \neq r \ \in \{1, \dots m\}\end{aligned}$$

Since $a_{rk} = \pm 1$, the values of the GCD are now:

$$\Gamma_i^{'} = gcd(a_{ik}, \cdots, (a_{ij} \pm a_{rj}a_{ik})_{j\neq k \ \in \{1,\dots n\}} \cdots)$$

Using the classical properties of the GCD, and particularly:

$$gcd(a - qb, b) = gcd(a, b)$$

we can easily state:

$$\Gamma_i^{'} = gcd(a_{ik}, \cdots, (a_{ij})_{j\neq k \ \in \{1,\dots n\}} \cdots)$$

and finally:

$$\Gamma_i^{'} = gcd(a_{i1}, \cdots, a_{ij} \cdots, a_{in}) = \Gamma_i$$

Theorem 7.2.2 *The GCD of the set of coefficients relative to free variables (Δ_i) of each remaining inequality does not decrease or becomes equal to zero after a "dummy" elimination.*

Proof:
Consider a system including the free variable x_k (the other variables may be free or constrained):

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i \in \{1, 2, \dots m\}$$

Let us define [2] the set $F \subseteq \{1, \dots n\}$ of the indices of the free variables and let us note:

$$\Delta_i = gcd(\cdots, (a_{ij})_{j\in F,\dots}) \tag{30}$$

Suppose $a_{rk} = \pm 1$. Let us perform a "dummy" elimination of x_k from the inequality r. The new system is:

$$\begin{aligned}(-a_{ik}/a_{rk})y_r + \textstyle\sum_{j\neq k}(a_{ij} - a_{rj}a_{ik}/a_{rk})x_j \\ \leq b_i - b_r a_{ik}/a_{rk}, \quad i \neq r \ \in \{1, \dots m\}\end{aligned}$$

$$\Delta_i^{'} = gcd(\cdots, (a_{ij} - a_{rj}a_{ik}/a_{rk})_{j\neq k \ \in F,\dots})$$

Now, suppose $\Delta_i^{'} \neq 0$ (else the theorem is obviously proved). Let us define:

$$\Delta_i^{''} = gcd(a_{ik}, \Delta_i^{'})$$

[2] in this paper, $A \subseteq B$ stands for the inclusion of A in B (possibly: $A = B$)

Obviously:

$$\Delta_i^{''} \leq \Delta_i^{'} \tag{31}$$

We can write:

$$\begin{aligned}\Delta_i^{''} &= gcd(a_{ik}, \cdots, (a_{ij} - a_{rj}a_{ik}/a_{rk})_{j\neq k \ \in F,\dots}) \\ \Delta_i^{''} &= gcd(a_{ik}, \cdots, (a_{ij} \pm a_{rj}a_{ik})_{j\neq k \ \in F,\dots})\end{aligned}$$

As previously, we can easily state:

$$\Delta_i^{''} = gcd(\cdots, (a_{ij})_{j\in F,\dots}) = \Delta_i$$

Hence from (31):

$$\Delta_i \leq \Delta_i^{'}$$

7.3 Unimodular changes of free variables

Suppose that the system includes one or more free variables and that no unitary coefficient relative to one of these variables exists. First, we should be assured that inequalities have been divided by their GCD (see 7.6). Then, we can try to change the system by unimodular transformations of free variables in order that

1. if possible:

   some inequality includes a free variable with a unitary coefficient.

   We should then perform the "dummy" elimination of the convenient new free variable.

2. otherwise:

   some inequality includes a single free variable with a non-zero coefficient.

   Then, a "single" cutting technique (7.4.3) can replace this variable by a non negative one.

The same notations Γ_i (29) and Δ_i (30) as in 7.2 will be used here and in other parts of the chapter.

7.3.1 "Unitary coefficient" technique (from two free variables)

Imagine that an inequality includes two free variables x_{j_1} and x_{j_2} whose coefficients $a_{i_r j_1}$ and $a_{i_r j_2}$ satisfy the condition:

$$gcd(a_{i_r j_1}, a_{i_r j_2}) = 1$$

Using a technique described in 5.4, we can change the two variables in such a way that a unitary coefficient appears. Then, we may apply a "dummy" elimination.

Consider the system with free variables x_1 and x_2:

$$\begin{array}{rrrcr} -3x_1 & +4x_2 & & \leq & -4 \\ 9x_1 & -3x_2 & -2y_4 & \leq & 15 \\ -18x_1 & & +5y_4 & \leq & -40 \end{array}$$

For the two inequalities including both of these variables, we have:

$$\begin{array}{rcl} gcd(-3,4) & = & 1 \\ gcd(9,-3) & = & 3 \end{array}$$

Let us perform this unimodular transformation (from the first inequality):

$$\begin{array}{rcrr} x_1 & = & t_1 & +4t_2 \\ x_2 & = & t_1 & +3t_2 \end{array}$$

We obtain the equivalent system (with free variables: t_1, t_2):

$$\begin{array}{rrrcr} t_1 & & & \leq & -4 \\ 6t_1 & +27t_2 & -2y_4 & \leq & 15 \\ -18t_1 & -72t_2 & +5y_4 & \leq & -40 \end{array}$$

Let us remark that the unimodular transformation keeps the right-hand sides unchanged. This could influence the choice of the inequality (see 7.7).

Now, we can perform the "dummy" elimination of t_1:

$$\begin{array}{rrrcr} -6y_5 & +27t_2 & -2y_4 & \leq & 39 \\ 18y_5 & -72t_2 & +5y_4 & \leq & -112 \end{array}$$

In order to eliminate the last free variable t_2, a cutting technique (7.4.3) is possible.

7.3.2 Shortening the number of free variables (of an inequality)

If there is no pair of relatively prime coefficients, we can use a technique analoguous to that described in 5.5 in order to shorten the number of free variables of an inequality until, either a unitary coefficient appears, or a single free variable remains. In this last case, if the coefficient is not unitary, the cutting technique (7.4.3) should apply.

Example with three free variables

Consider the system (free variables: x_2, x_3, x_4, constrained variable: y_5):

$$\begin{array}{rrrrcr} 5y_5 & -2x_2 & & -4x_4 & \leq & -4 \\ -2y_5 & +9x_2 & -3x_3 & & \leq & 30 \\ 5y_5 & & +2x_3 & -6x_4 & \leq & -1 \\ -4y_5 & -6x_2 & -10x_3 & +15x_4 & \leq & -7 \end{array}$$

In order to simplify the last inequality, we can change x_2 and x_3:

$$\begin{array}{rcrr} x_2 & = & -2t_2 & +5t_3 \\ x_3 & = & t_2 & -3t_3 \end{array}$$

$$\begin{array}{rrrrcr} 5y_5 & +4t_2 & -10t_3 & -4x_4 & \leq & -4 \\ -2y_5 & -21t_2 & +54t_3 & & \leq & 30 \\ 5y_5 & +2t_2 & -6t_3 & -6x_4 & \leq & -1 \\ -4y_5 & +2t_2 & & +15x_4 & \leq & -7 \end{array}$$

Now, since $GCD(2,15) = 1$, we can apply the "unitary coefficient" technique (see 7.3.1) by changing t_2 and x_4:

$$\begin{array}{rcrr} t_2 & = & -7t_4 & -15t_5 \\ x_4 & = & t_4 & +2t_5 \end{array}$$

$$\begin{array}{rrrrcr} 5y_5 & -32t_4 & -10t_3 & -68t_5 & \leq & -4 \\ -2y_5 & +147t_4 & +54t_3 & +315t_5 & \leq & 30 \\ 5y_5 & -20t_4 & -6t_3 & -42t_5 & \leq & -1 \\ -4y_5 & +t_4 & & & \leq & -7 \end{array}$$

and consequently the reduced system (with two constrained variables):

$$\begin{array}{rrrrcr} -123y_5 & +32y_7 & -10t_3 & -68t_5 & \leq & -228 \\ 586y_5 & -147y_7 & +54t_3 & +315t_5 & \leq & 1059 \\ -75y_5 & +20y_7 & -6t_3 & -42t_5 & \leq & -141 \end{array}$$

Example with two free variables

Consider the system (free: x_1, x_2, constrained: y_4):

$$\begin{array}{rrrcr} -2x_1 & +4x_2 & +5y_4 & \leq & -4 \\ 9x_1 & +3x_2 & -2y_4 & \leq & 15 \\ -18x_1 & -8x_2 & +5y_4 & \leq & -40 \end{array}$$

No couple of coefficients relative to x_1, x_2 can lead to a unitary coefficient since:

$$\begin{array}{rcl} gcd(-2,4) & = & 2 \\ gcd(9,3) & = & 3 \\ gcd(-18,-8) & = & 2 \end{array}$$

In order to simplify the third inequality, we can apply the unimodular transformation:

$$\begin{array}{rcrr} x_1 & = & -t_1 & +4t_2 \\ x_2 & = & 2t_1 & -9t_2 \end{array}$$

and obtain the system:

$$\begin{array}{rrrcr} 10t_1 & -44t_2 & +5y_4 & \leq & -4 \\ -3t_1 & +9t_2 & -2y_4 & \leq & 15 \\ 2t_1 & & +5y_4 & \leq & -40 \end{array}$$

Now, we can apply a cutting technique (7.4.3).

7.3.3 Which unimodular changes?

The way of choosing unimodular changes may be difficult. First, let us state the following results.

Theorem 7.3.1 *The GCD of the set of coefficients (Γ_i) of each inequality is unchanged by a unimodular transformation.*

Theorem 7.3.2 *The GCD of the set of coefficients relative to free variables (Δ_i) of each inequality is unchanged by a unimodular transformation.*

Proof:
Let the transformation be:

$$\begin{cases} t_1 = a_1 x_1 + a_2 x_2 \\ t_2 = b_1 x_1 + b_2 x_2 \end{cases}$$

with the unimodularity condition:

$$a_1 b_2 - a_2 b_1 = 1$$

and the inverse transformation:

$$\begin{cases} x_1 = b_2 t_1 - a_2 t_2 \\ x_2 = -b_1 t_1 + a_1 t_2 \end{cases}$$

Now, consider any expression with both of the original variables (c_1 or c_2 may be equal to *zero*):

$$z = c_1 x_1 + c_2 x_2 \tag{32}$$

Changing the variables, we get:

$$z = (c_1 b_2 - c_2 b_1) t_1 + (-c_1 a_2 + c_2 a_1) t_2$$

which we may express symbolically as:

$$z = \alpha_1 t_1 + \alpha_2 t_2 \tag{33}$$

If we define:

$$\gamma_x = gcd(c_1, c_2) \tag{34}$$

the formula (32) can be written:

$$z = \gamma_x (c_1^{'} x_1 + c_2^{'} x_2) \tag{35}$$

Since $c_1^{'}$ and $c_2^{'}$ are relatively prime, the parenthetical expression of (35) may be given any integer value (from appropriate values of x_1 and x_2). Hence:

z may be given any integer value multiple of γ_x and only such a value.

On the other hand the formula (33) can be modified similarly:

$$z = \gamma_t (\alpha_1^{'} t_1 + \alpha_2^{'} t_2)$$

with

$$\gamma_t = gcd(\alpha_1, \alpha_2) \tag{36}$$

For the same reasons:

z may be given any integer value multiple of γ_t and only such a value.

Then, necessarily:

$$\gamma_t = \gamma_x$$

$$gcd(\alpha_1, \alpha_2) = gcd(c_1, c_2) \tag{37}$$

Consider any inequality including n_1 free variables x_j and n_2 constrained variables y_j:

$$c_1 x_1 + c_2 x_2 + \sum_{j=3}^{n_1} c_j x_j + \sum_{j=n_1+1}^{n_1+n_2} c_j y_j \leq d$$

As in 7.2, let us define the GCD of the set of all coefficients and the GCD of the set of coefficients relative to free variables:

$$\Gamma = gcd(c_1, c_2, c_3, \cdots, c_{n_1+n_2})$$
$$\Delta = gcd(c_1, c_2, c_3, \cdots, c_{n_1})$$

Let us change x_1 and x_2 as previously:

$$\alpha_1 t_1 + \alpha_2 t_2 + \sum_{j=3}^{n_1} c_j x_j + \sum_{j=n_1+1}^{n_1+n_2} c_j y_j \leq d$$

The values of both GCD are now:

$$\Gamma^{'} = gcd(\alpha_1, \alpha_2, c_3, \cdots, c_{n_1+n_2})$$
$$\Delta^{'} = gcd(\alpha_1, \alpha_2, c_3, \cdots, c_{n_1})$$

Since we have (37), these values are unchanged:

$$\Gamma^{'} = \Gamma$$
$$\Delta^{'} = \Delta$$

This proves the theorem (7.3.1) and the theorem (7.3.2). Therefore, these points can be deduced and should avoid needless computation:

- *dividing (again) an inequality by its GCD is unnecessary* (see 7.6).

Lemma 7.3.1 *It is possible to build a unitary coefficient of a free variable on some inequality (i) by unimodular changes if and only if $\Delta_i = 1$.*

Proof:

Consider an inequality with at least one *non-zero* coefficient ($\Delta_i \neq 0$). Imagine we apply unimodular changes. The value of Δ_i is unchanged (theorem 7.3.2). If a unitary coefficient is built, this means that $\Delta_i = 1$. Conversely, if $\Delta_i = 1$, we can apply changes until we obtain a single coefficient necessarily equal to $\Delta_i = 1$.

Lemma 7.3.2 *It is not possible to build a unitary coefficient of a free variable on some inequality (i) by "dummy" eliminations or by unimodular changes if $\Delta_i \neq 1$.*

Proof:

Consider an inequality with $\Delta_i \neq 1$. Imagine we apply "dummy" eliminations or unimodular changes. The value of Δ_i cannot become equal to 1 (theorem 7.2.2). Hence, it is not possible to build a unitary coefficient of a free variable (lemma 7.3.1).

Lemma 7.3.3 *At least one "dummy" elimination is possible with any system of inequalities with free variables.*

Proof:

Dividing any inequality (i) by its GCD, we have: $\Gamma_i = 1$
Since all variables are free: $\Delta_i = \Gamma_i = 1$
Consequently, building a unitary coefficient is possible (lemma 7.3.1).

Some applications

- Consider the first example of 7.3.2. We were guaranteed to obtain a unitary coefficient from the original third inequality since (lemma 7.3.1):

$$\Delta_3 = gcd(-6, -10, 15) = 1$$

- Look at the following system (constrained variable: y_5, free variables: x_2, x_3, x_4):

$$\begin{array}{rrrrcr} 5y_5 & -2x_2 & & -4x_4 & \leq & -4 \\ -2y_5 & 9x_2 & -3x_3 & & \leq & 30 \\ 5y_5 & & +4x_3 & -6x_4 & \leq & -1 \\ -4y_5 & -6x_2 & -9x_3 & +9x_4 & \leq & -7 \end{array}$$

We note that all Γ_i are equal to 1. Obtaining a unitary coefficient is not possible and will never be possible since all $\Delta_i \neq 1$ (lemma 7.3.2):

$$\begin{array}{ll} \Delta_1 = gcd(-2, 0, -4) & = 2 \\ \Delta_2 = gcd(9, -3, 0) & = 3 \\ \Delta_3 = gcd(0, 4, -6) & = 2 \\ \Delta_4 = gcd(-6, -9, 9) & = 3 \end{array}$$

In such a case, we should try a "redundant inequality" method (7.4). If we intend to apply the "single" cutting technique (which requires an inequality including a single free variable, see 7.4.3), it is preferable to simplify the inequality including the smallest number of free variables (in order to minimize the number of changes). However, no unimodular change is necessary with the previous example: we can apply directly a "GCD" cutting technique (7.4.4).

7.4 Additional (redundant) inequalities

Imagine that we cannot find or build a unitary coefficient relative to some free variable. However, if we find a new valid inequality including a free variable with a unitary coefficient, we can add it to the system without changing it, and therefore use this inequality for a "dummy" elimination (7.2). So we obtain an equivalent system without increasing the number of variables. Of course the number of inequalities does not decrease, but we can remember that the new slack variable was obtained from a redundant inequality. This point is important: if we intend to apply to the resulting system some cutting-plane method (section 11), the slack variable will be handled as all classical "cut" variables: as soon as it comes back to the basis, it can be removed from the system (see 7.8.3).

We may imagine various methods to build the new inequality.

7.4.1 Arbitrary bound

First, we can simply give a great value bound to the variable. We may admit that if a solution exists, a particular finite solution exists such that we can write for any free variable one of the following inequalities:

$$\begin{array}{rcr} x_i & \leq & M_i \\ -x_i & \leq & -M_i \end{array}$$

where M_i is a sufficiently large positive number (this number could be theoretically computed from the whole data of the system). Then, the problem is not modified by this new constraint. The system previously obtained (7.2) could become:

$$\begin{array}{rrrcr} 18x_1 & -3y_5 & -2y_4 & \leq & 27 \\ -42x_1 & +8y_5 & +5y_4 & \leq & -72 \\ x_1 & & & \leq & 1000000 \end{array}$$

After the "dummy" elimination of x_1:

$$\begin{array}{rrrcr} -18w_6 & -3y_5 & -2y_4 & \leq & -17999973 \\ 42w_6 & +8y_5 & +5y_4 & \leq & 41999928 \end{array}$$

We remark that the right-hand sides have taken values strongly greater than the bound M_i. A risk of data overflow is possible if the bound is chosen with a too important value.

Another disavantage is that this technique gives for final cutting plane algorithms a starting point far from the domain where we can prove or disprove the feasibility of the system. Applying some final cutting plane algorithms to such systems, we observed

in some cases a strong sensitivity (number of cuts) to the value of the bound (the greater the bound, the greater the number of cuts).

7.4.2 Mixing two inequalities

Second, we may multiply two inequalities by positive parameters and add them in such a way that the coefficient of the free variable is unitary (a technique analogous to that of 5.3). For instance:

$$\begin{array}{rrrcr} 5x_1 & -3y_5 & -2y_4 & \leq & 12 \\ -2x_1 & +8y_5 & +5y_4 & \leq & -15 \end{array}$$

Multiplying the first inequality by 1 and the second one by 2, we get:

$$\begin{array}{rrrcr} 5x_1 & -3y_5 & -2y_4 & \leq & 12 \\ -2x_1 & +8y_5 & +5y_4 & \leq & -15 \\ x_1 & +13y_5 & +8y_4 & \leq & -18 \end{array}$$

This technique works only if these two conditions are satisfied:

1. the original coefficients have different signs.

   However, in an inequalities system, we are guaranteed to find two coefficients of different sign (otherwise we can immediatly remove the free variable by a trivial selective Fourier-Motzkin elimination, see the second example from 7.5.2)

2. The GCD of the two coefficients is equal to 1.

7.4.3 "Single free variable" cutting technique

Let us consider the following case: an inequality includes a single free variable and its coefficient is not unitary. Suppose, for the sake of argument, that this variable is x_1:

$$a_{r1}x_1 + \sum_{j=2}^{n} a_{rj}y_j \leq b_r$$

Then, the following inequality (with a unitary coefficient relative to x_1) is satisfied:

$$(a_{r1}/|a_{r1}|)x_1 + \sum_{j=2}^{n} \lfloor a_{rj}/|a_{r1}| \rfloor y_j \leq \lfloor b_r/|a_{r1}| \rfloor$$

Proof:
Let us temporarily change x_1 as follows:

$$x_1 = y_1 - z_1 \quad (y_1 \geq 0, z_1 \geq 0)$$

The original inequality becomes:

$$a_{r1}y_1 - a_{r1}z_1 + \sum_{j=2}^{n} a_{rj}y_j \leq b_r$$

Now (all variables are constrained), we can apply the classical "cut" (47) (from 11.1.1)

$$\lfloor a_{r1}/|a_{r1}| \rfloor y_j + \lfloor -a_{r1}/|a_{r1}| \rfloor z_j + \sum_{j=2}^{n} \lfloor a_{rj}/|a_{r1}| \rfloor y_j \leq \lfloor b_r/|a_{r1}| \rfloor$$

Since:

$$\begin{array}{rcl} \lfloor a_{r1}/|a_{r1}| \rfloor & = & a_{r1}/|a_{r1}| \\ \lfloor -a_{r1}/|a_{r1}| \rfloor & = & -a_{r1}/|a_{r1}| \end{array}$$

the cut inequality becomes:

$$(a_{r1}/|a_{r1}|)(y_1 - z_1) + \sum_{j=2}^{n} \lfloor a_{rj}/|a_{r1}| \rfloor y_j \leq \lfloor b_r/|a_{r1}| \rfloor$$

and finally:

$$(a_{r1}/|a_{r1}|)x_1 + \sum_{j=2}^{n} \lfloor a_{rj}/|a_{r1}| \rfloor y_j \leq \lfloor b_r/|a_{r1}| \rfloor$$

Example

Consider the system obtained in (7.2):

$$\begin{array}{rrrcr} 18x_1 & -3y_5 & -2y_4 & \leq & 27 \\ -42x_1 & +8y_5 & +5y_4 & \leq & -72 \end{array}$$

From the first inequality, we get:

$$\begin{array}{rrrcr} (18/18)x_1 + \lfloor -3/18 \rfloor y_5 + \lfloor -2/18 \rfloor y_4 & & & \leq & \lfloor 27/18 \rfloor \\ x_1 & - y_5 & - y_4 & \leq & 1 \end{array}$$

The new inequality is added to the system:

$$\begin{array}{rrrcr} 18x_1 & -3y_5 & -2y_4 & \leq & 27 \\ -42x_1 & +8y_5 & +5y_4 & \leq & -72 \\ x_1 & -y_5 & -y_4 & \leq & 1 \end{array}$$

Now, we can perform a "dummy" elimination:

$$\begin{array}{rrrcr} -18w_6 & +15y_5 & +16y_4 & \leq & 9 \\ 42w_6 & -34y_5 & -37y_4 & \leq & -30 \end{array}$$

Note: This technique is a particular case of the following one (7.4.4).

7.4.4 "GCD" cutting technique: more than one free variable

Consider the inequality:

$$\begin{array}{l} \sum_{j=1}^{n_1} a_{rj}x_j + \sum_{j=n_1+1}^{n_1+n_2} a_{rj}y_j \leq b_r \\ x_j \in \mathbb{Z},\ \ y_j \in \mathbb{Z},\ \ y_j \geq 0 \end{array}$$

Suppose that the coefficient a_{rk} of some free variable x_k satisfies:

$$\Delta_r = gcd(a_{r1}, \cdots a_{rj}, \cdots a_{rn_1}) \quad = |a_{rk}| \geq 1$$

All the quantities a_{rj}/Δ_r relative to free variables ($j \leq n_1$) have integer values. Hence, we can easily prove (as in 7.4.3):

$$\sum_{j=1}^{n_1}(a_{rj}/\Delta_r)x_j + \sum_{j=n_1+1}^{n_1+n_2}\lfloor a_{rj}/\Delta_r \rfloor y_j \leq \lfloor b_r/\Delta_r \rfloor$$

In this new inequality, the coefficient a_{rj}/Δ_r of the variable x_k is unitary.

Let us return to the example of 7.3.3:

$$\begin{array}{rrrrcr} 5y_5 & -2x_2 & & -4x_4 & \leq & -4 \\ -2y_5 & 9x_2 & -3x_3 & & \leq & 30 \\ 5y_5 & & +4x_3 & -6x_4 & \leq & -1 \\ -4y_5 & -6x_2 & -9x_3 & +9x_4 & \leq & -7 \end{array}$$

Since:

$$\begin{array}{rl} \Delta_1 = gcd(-2,0,-4) & = 2 \\ \Delta_2 = gcd(9,-3,0) & = 3 \end{array}$$

we can cut the first or the second inequality. Let us choose the first one:

$$\begin{array}{rrrcr} \lfloor 5/2 \rfloor y_5 + (-2/2)x_2 + (-4/2)x_4 & & & \leq & \lfloor -4/2 \rfloor \\ 2y_5 & -x_2 & -2x_4 & \leq & -2 \end{array}$$

Let us add the inequality to the system:

$$\begin{array}{rrrrcr} 5y_5 & -2x_2 & & -4x_4 & \leq & -4 \\ -2y_5 & 9x_2 & -3x_3 & & \leq & 30 \\ 5y_5 & & +2x_3 & -6x_4 & \leq & -1 \\ -4y_5 & -6x_2 & -9x_3 & +9x_4 & \leq & -7 \\ 2y_5 & -x_2 & & -2x_4 & \leq & -2 \end{array}$$

Now, we can perform the "dummy" elimination of x_2.

7.5 Fourier-Motzkin again

The set of techniques described in this chapter should be applied when no possibility of any previous elimination (selective Fourier-Motzkin, see 6.2) exists (or no longer exists). However, after some of these techniques have been performed, such a possibility may appear. Consequently, "non-negative variables" techniques and Fourier-Motzkin eliminations can be mixed in some problems.

The appearance of Fourier-Motzkin opportunities is "helped" by "non-negative variables" techniques. Let's have a look at the following cases.

7.5.1 After a "dummy" elimination

The number of inequalities, and consequently, the number of coefficients on the column of each variable, decrease. Hence, for each remaining free variable a Fourier-Motzkin possibility may appear.

Consider the system (all free variables):

$$\begin{array}{rrrcr} -4x_1 & -2x_2 & +x_3 & \leq & -9 \\ -3x_1 & +x_2 & & \leq & -4 \\ x_1 & -6x_2 & +2x_3 & \leq & -3 \\ 2x_1 & +2x_2 & -5x_3 & \leq & 5 \end{array}$$

First, we can perform a "dummy" elimination (x_1,x_2 or x_3). Let us choose x_3 (first inequality):

$$\begin{array}{rrrcr} -3x_1 & +x_2 & & \leq & -4 \\ 9x_1 & -2x_2 & -2y_4 & \leq & 15 \\ -18x_1 & -8x_2 & +5y_4 & \leq & -40 \end{array}$$

Then, we could similarly eliminate x_2. Nevertheless, a selective Fourier-Motzkin elimination of this variable is now possible (criterion 2, case 2):

$$\begin{array}{rrcr} 3x_1 & -2y_4 & \leq & 7 \\ -42x_1 & +5y_4 & \leq & -72 \end{array}$$

As concerns x_1, the only way is a "cut" variable (7.4.3). If we choose the last inequality, we first modify the system as follows:

$$\begin{array}{rrcr} 3x_1 & -2y_4 & \leq & 7 \\ -42x_1 & +5y_4 & \leq & -72 \\ -x_1 & & \leq & -2 \end{array}$$

and we get the final system of constrained variables (including the "redundant" variable w_8):

$$\begin{array}{rrcr} 3w_8 & -2y_4 & \leq & 1 \\ -42w_8 & +5y_4 & \leq & 12 \end{array}$$

The system is feasible since all right-hand sides have non-negative values.

7.5.2 After a unimodular transformation

This technique modifies two columns of the system. Then, Fourier-Motzkin could be possible with each of the new variables. Note that no new Fourier-Motzkin possibility may appear in the columns of the other free variables (they are unchanged by the transformation).

First example

Consider the system (all free variables):

$$\begin{array}{rrrcr} 6x_1 & +11x_2 & +6x_3 & \leq & -4 \\ 5x_1 & +2x_2 & -2x_3 & \leq & 30 \\ & -2x_2 & -13x_3 & \leq & -1 \\ -3x_1 & -5x_2 & +3x_3 & \leq & -7 \end{array}$$

In order to apply the "unitary coefficient" technique (7.3.1) to the last inequality, let us change x_1 and x_2:

$$\begin{array}{rcrr} x_1 & = & -2t_1 & +5t_2 \\ x_2 & = & t_1 & -3t_2 \end{array}$$

$$\begin{array}{rrrcr} -t_1 & -3t_2 & +6x_3 & \leq & -4 \\ -8t_1 & +19t_2 & -2x_3 & \leq & 30 \\ -2t_1 & +6t_2 & -13x_3 & \leq & -1 \\ t_1 & & +3x_3 & \leq & -7 \end{array}$$

Now, we can perform a "dummy" elimination of t_1, but we can also eliminate it by Fourier-Motzkin:

$$\begin{array}{rrcr} -3t_2 & +9x_3 & \leq & -11 \\ +19t_2 & +22x_3 & \leq & -26 \\ +6t_2 & -7x_3 & \leq & -15 \end{array}$$

Note that the new first inequality can be divided by its $GCD = 3$ (see 7.6), allowing a new Fourier-Motzkin elimination.

Second example

Consider the first example of (7.3.2). The following system (with two constrained variables) was obtained:

$$\begin{array}{rrrrcr} -123y_5 & +32y_7 & -10t_3 & -68t_5 & \leq & -228 \\ 586y_5 & -147y_7 & +54t_3 & +315t_5 & \leq & 1059 \\ -75y_5 & +20y_7 & -6t_3 & -42t_5 & \leq & -141 \end{array}$$

If we choose to simplify the first inequality, we can apply this unimodular transformation:

$$\begin{array}{rcrr} t_3 & = & -7t_6 & +34t_7 \\ t_5 & = & t_6 & -5t_7 \end{array}$$

$$\begin{array}{rrrrcr} -123y_5 & +32y_7 & +2t_6 & & \leq & -228 \\ 586y_5 & -147y_7 & -63t_6 & +261t_7 & \leq & 1059 \\ -75y_5 & +20y_7 & & +6t_7 & \leq & -141 \end{array}$$

Now, we could cut (7.4.3) this inequality (with the single free variable t_6). However, the variable t_7 can be removed by a trivial Fourier-Motzkin elimination:

$$\begin{array}{rrrcr} -123y_5 & +32y_7 & +2t_6 & \leq & -228 \end{array}$$

This final system is obviously feasible.

7.6 Division by GCD again?

Before applying the techniques described in this chapter, we should divide each inequality by its GCD (see 6.1.4). Moreover, we know that the "non-negative variables" techniques keep the GCD unchanged (theorems 7.2.1 and 7.3.1). Then, trying to divide inequalities again after performing any of these techniques is needless: we are guaranteed that the value of the GCD of each remaining inequality is still equal to 1. Nevertheless, each time Fourier-Motzkin is executed, the GCD of each new inequality may have any value, and dividing any such inequality could work (see the first example from 7.5.2).

7.7 "Simplex strategy"

Replacing free variables in a system by constrained variables has two consequences:

1. The size of the system is modified.

2. A final cutting-plane method (section 11) can be applied.

Then, we could imagine a strategy working in such a way that, first the system size decreases, second, the formulation of the resulting system makes the intended cutting-plane method more efficient.

As regards the first point, "dummy" eliminations (and Selective Fourier-Motzkin) are obviously attractive.

As concerns the second point, we can recall that a system of inequalities with constrained variables owns an obvious solution when all the right-hand sides have non-negative values (this criterion works also if some of the variables are free), and that no solution exists when all the coefficients of an inequality with a negative right-hand side are non-negative. The principle underlying many typical linear programming methods is to make such cases appear by judicious changes of variables. Our present aim is to change free variables, but at the same time, we can try to perform such changes in a way analogous to that used with linear programming methods. Then, if several eliminations compete (7.2, 7.3.1, 7.3.2), a "simplex strategy" could be applied (for instance, choosing, as with dual algorithms, the inequality with the most negative right-hand side).

"Dummy" eliminations and "simplex strategy" may be conflicting. As an example, consider the very simple system (with the free variable x_1 and the constrained variable y_4):

$$\begin{array}{rrcr} 3x_1 & -2y_4 & \leq & 41 \\ -4x_1 & +5y_4 & \leq & -13 \\ -x_1 & & \leq & 3 \end{array}$$

Performing a "dummy" elimination on the last inequality, we can obtain the reduced equivalent system:

$$\begin{array}{rrcr} 3y_7 & -2y_4 & \leq & 50 \\ -4y_7 & +5y_4 & \leq & -25 \end{array}$$

But if we choose a "simplex strategy", i.e. cutting (7.4.3) the second inequality (with the most negative right-hand side):

$$-x_1 \quad +y_4 \quad \leq \quad -4$$

fortunately, we get a feasible system:

$$\begin{array}{rrcr} 3w_6 & +y_4 & \leq & 29 \\ -4w_6 & +y_4 & \leq & 3 \\ -w_6 & -y_4 & \leq & 7 \end{array}$$

Nevertheless, we could easily find a similar example where the "simplex strategy" does not give such a result. The author's opinion is that in the first place, the choice of the strategy should be based on criteria leading quickly to reduced systems (selective Fourier-Motzkin, unitary coefficient, sparse row,...). Then, if "equivalent" eliminations compete, a "simplex strategy" could be applied.

7.8 Constrained variables: conclusion

7.8.1 General strategy

In any case, the following sequence should guarantee a strongly reduced system or the proof of the system feasibility:

1. Divide all inequalities by their GCD.
2. Test the following cases, and stop if one works:
   (a) *No free variable remains.*
   (b) *Every right-hand side is non-negative.*
3. Search a selective Fourier-Motzkin opportunity. If found:
   (a) Apply the Fourier-Motzkin elimination.
   (b) Divide new inequalities by their GCD.
   (c) Go to 2.
4. Search a unitary coefficient. If found:
   (a) Apply the "dummy" elimination (7.2).
   (b) Go to 2.
5. Search an inequality with $\Delta_i = 1$. If found:
   (a) Build a unitary coefficient (see lemma 7.3.1).
   (b) Search a selective Fourier-Motzkin opportunity (7.5.2). If found, go to 3a.
   (c) Apply the "dummy" elimination.
   (d) Go to 2.
6. Search a free variable v_k. If found:
   (a) Search a coefficient $a_{ik} \neq 0$. If not found:
      i. Remove variable v_k from the system.
      ii. Go to 6.
   (b) If $|a_{ik}| \neq \Delta_i$ (see 7.4.4):
      i. Build a single (free) coefficient (7.3.2)
      ii. Search a selective Fourier-Motzkin opportunity (7.5.2). If found, go to 3a.
   (c) Cut inequality i.
   (d) Apply a "dummy" elimination to the cut.
   (e) If the feasibility criterion works, stop:
   Every right-hand side is non-negative.
   (f) Go to 6.
7. Stop.

7.8.2 Simplified strategy

In the case of data dependence analysis, experiments [ES92] with the INRIA PIAF parallelizer [GLL+90] have shown that this sequence is quite sufficient:

1. Divide all inequalities by their GCD.
2. Test the following cases, and stop if one works:
   (a) *No free variable remains.*
   (b) *Every right-hand side is non-negative.*
3. Search a selective Fourier-Motzkin opportunity. If found:
   (a) Apply the Fourier-Motzkin elimination.
   (b) Go to 2.
4. Search a unitary coefficient. If found:
   (a) Apply the "dummy" elimination.
   (b) Go to 2.
5. Apply the "Single added variable" technique (7.1.2) to remaining free variables.

The step 5 was included in order to define a complete sequence, however it was never needed.

7.8.3 Interest for next steps

In this section, it was shown that reducing the system is always possible, therefore the classical added variables techniques (7.1) should be avoided. However, the reducing techniques present other advantages. Consider the following system with free variables:

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i, \quad i \in \{1, \dots m\}$$
$$x_j \in \mathbb{Z}$$

Firstly, let us recall the features of systems we could obtain with different techniques:

1. *"Single added variable" technique* (7.1.2)
   - m inequalities
   - $(n+1)$ "usual" constrained variables.

   Note: (Constrained) variables should be referred to as "usual" just to distinguish them from "cut" variables.

2. *Reducing techniques* (for the sake of argument, without Fourier-Motzkin).
   - $m - n_d$ inequalities
   - n constrained variables:
     - n_d "usual" variables
     - $(n - n_d)$ "cut" variables.

   In the best case, $n_d = n$. In the worst case, $n_d = 1$ (lemma 7.3.3).

Now, let us anticipate the section 11 and imagine we apply a final cutting-plane method. Such a method is characterized by the introduction of "cuts", which may increase the number of inequalities. However, the number of necessarily maintained cuts cannot exceed the number n of variables. Consequently, the size of the system could not exceed the following values:

1. *"Single added variable" technique*
   - $m + n$ inequalities
   - $n + 1$ constrained variables.
2. *Reducing techniques*
   - m inequalities
   - n constrained variables.

8 Simplex methods (Integrity relaxation)

The methods which will be examined here essentially use techniques and algorithms deriving directly from the famous method of simplex, by G.B. Dantzig (for details see [Dan63, Sim62] or other books). These methods are inexact since they work with integrity relaxation. However, the exact algorithms of section 11 are modified versions of the algorithms which will be detailed here. We first recall some terminology that will be used throughout this section and section 11.

8.1 Vocabulary et notations

The aim of simplex method is to solve the linear optimization problem:

$$Ax = b$$
$$x \geq 0$$
$$\min z = cx$$

The function z is referred to as *objective function*. The previous form, including equations, is the general *standard form* of the simplex, which should not be confused with the standard form (as defined in section 3) required by practical simplex algorithms and referred to in the literature as the *canonical* form. This canonical form includes "solved" equations defined from inequalities. Let us consider the system (with constrained variables):

$$\begin{array}{rrcr} & x_2 & \leq & 3 \\ -3x_1 & -2x_2 & \leq & -2 \\ -x_1 & -4x_2 & \leq & -1 \end{array}$$

Introducing slack variables ≥ 0, we can build the equivalent system:

$$\begin{array}{rrrcr} x_3 & & +x_2 & = & 3 \\ x_4 & -3x_1 & -2x_2 & = & -2 \\ x_5 & -x_1 & -4x_2 & = & -1 \end{array}$$

This is a canonical equations system, which we may write:

$$\begin{array}{rrrcr} x_3: & & x_2 & \leq & 3 \\ x_4: & -3x_1 & -2x_2 & \leq & -2 \\ x_5: & -x_1 & -4x_2 & \leq & -1 \end{array}$$

Variables appearing in the inequalities (x_1, x_2) are referred to as *non-basic variables*. Variables naming inequalities (x_3, x_4, x_5) are referred to as *basic variables*. The basic transformation (*pivoting*) used in simplex algorithms is a Gauss elimination that takes

a basic variable out of the basis and inserts an non-basic variable into the basis. Consider a system including a set of inequalities:

$$x_{n+i}: \quad a_{i1}x_1 \quad +a_{i2}x_2 \quad +\ldots \quad \leq \quad b_i$$

Pivoting on (x_k, x_{n+r}) (x_k out of the basis and x_{n+r} in the basis) means replacing x_k by

$$(b_r - x_{n+r} - \sum_{j=1, j\neq k}^{n} a_{rj}x_j)/a_{rk}$$

in the whole system. Inequality x_{n+r} is renamed x_k. The coefficient a_{rk} is called the *pivot*. A pivoting operation keeps a canonical system.
A canonical system is *feasible* when every $b_i \geq 0$: an obvious solution is obtained by giving the value 0 to every non-basic variable and the value b_i to basic variable x_{n+i}.
When one or more right-hand sides of inequalities of a feasible system are equal to *zero*, the system is said to be *degenerate*.

8.1.1 Duality

Given any linear optimization (minimization) problem referred as *primal*, it is proven ([Dan63] [Sim62]) that it can be replaced by an equivalent maximization problem referred as *dual*. Then, exactly one of the following relationships between both problems must hold:

1. Both the primal and the dual are infeasible.
2. One problem is unbounded and the other problem is infeasible.
3. The primal and the dual have finite optimal solutions and these solutions are equal:

$$\min z = \max w$$

When the primal includes only inequalities with variables ≥ 0, the duality is very simple:

$$\begin{array}{cc} Ax \geq b & uA \leq c \\ x \geq 0 & u \geq 0 \\ \min z = cx & \max w = ub \end{array}$$

Now, let us define this notation: $\bar{v} = -v$. From the previous result, we can immediately state the following duality formulation:

$$\begin{array}{cc} Ax \leq b & u\bar{A} \leq c \\ x \geq 0 & u \geq 0 \\ \min z = cx & \min \bar{w} = ub \end{array}$$

with

$$\min z = \min \bar{w}$$

These problems are strictly symmetric. We obtain dual from primal or conversely by transposing all data and changing the sign of the matrix A, as in this example:

$$\begin{array}{rrcr} & \min z & & \\ 2x_1 & +7x_2 & = & z \\ -x_1 & -3x_2 & \leq & -5 \\ 2x_1 & +9x_2 & \leq & 12 \\ & x_i \geq 0 & & \end{array} \qquad \begin{array}{rrcr} & \min \bar{w} & & \\ -5u_1 & +12u_2 & = & \bar{w} \\ u_1 & -2u_2 & \leq & 2 \\ 3u_1 & -9u_2 & \leq & 7 \\ & u_i \geq 0 & & \end{array}$$

8.2 Basic algorithms

These algorithms are tools used in simplex and other methods.

8.2.1 The Primal Simplex algorithm

Given the following optimization problem:

$$\begin{array}{l} \min z = \sum_{j=1}^{n} c_j x_j \\ \sum_{j=1}^{n} a_{ij}x_j \leq b_i, \quad i \in \{1, 2, \ldots m\} \\ x_j \in \mathsf{R}^+, \quad \mathsf{j} \in \{1, 2, \ldots \mathsf{n}\} \end{array}$$

The primal simplex algorithm can be applied when b_i are non negative.

Let us write the "canonical" formulation:

$$\begin{array}{l} \min z = \sum_{j=1}^{n} c_j x_j \\ x_{n+i}: \ \sum_{j=1}^{n} a_{ij}x_j \leq b_i, \quad i \in \{1, 2, \ldots m\} \\ x_j \in \mathsf{R}^+ \end{array}$$

The algorithm executes pivoting operations (a basic variable becomes non-basic and an non-basic variable become basic) in such a way that a feasible system (all $b_i \geq 0$) is constantly maintained, and finally, either the coefficients of the function z are all non negative, or the solution is infinite. We will briefly describe it (using the notations of the initial system):

1. (Test for optimality) If $c_j \geq 0$ for every j, the current solution is optimal and we stop.
2. (Test for failure) If the number of iterations exceeds an arbitrary number $tmax$ (usually equal to $2m + n$), the method fails and we stop.
3. (Choosing the entering variable) Select a variable x_{j_p} with $c_{j_p} < 0$
   A common rule is to choose the variable with the most negative c_j. Another rule (for small systems) is to choose the variable with greatest $b_{i_p}/a_{i_p j_p}$ (see further step 5).
4. (Test for infinity) If no coefficient $a_{ij_p} > 0$ exists, the solution is unbounded and we stop.

5. (Choosing a departing variable) Select a basic variable x_{n+i_p} such that
   $b_{i_p}/a_{i_p j_p} = \min(b_i/a_{ij_p})\ \ \forall a_{ij_p} > 0$

6. (pivoting) Execute a pivoting operation between x_{j_p} and x_{n+i_p} and return.

There are many possibilities for the choice of the pivot.
Entering variable (step 3): The cost of each iteration is more expensive with algorithms based on steepest edge rules [FG91]. They are efficient with large size problems.
Departing variable (step 5): there is no common rule when several variables compete (which mainly occurs in the case of degeneracy). Thus, choosing a pivot with a value -1 (it is often possible with data dependence problems) may avoid that fractional coefficients appear. If the computer does not use fractional arithmetic, the choice of an pivot with an absolute value not inferior to 1 may diminish rounding errors.

Example

$$\begin{array}{rrrcl} \min z = & -x_1 & -x_2 & & \\ x_3: & 2x_1 & +x_2 & \leq & 8 \\ x_4: & x_1 & +2x_2 & \leq & 7 \\ x_5: & & x_2 & \leq & 3 \end{array}$$

We choose arbitrarily x_1 as entering variable (x_2 can also be chosen), and then inequality x_3.
Pivoting (x_3, x_1) is proceeded:

$$\begin{array}{rrrcll} & 1/2x_3 & -1/2x_2 & = & 4 & +z \\ x_1: & 1/2x_3 & +1/2x_2 & \leq & 4 & \\ x_4: & -1/2x_3 & +3/2x_2 & \leq & 3 & \\ x_5: & & x_2 & \leq & 3 & \end{array}$$

Now, the single choice is x_2, and we select x_4.
We pivot with (x_4, x_2):

$$\begin{array}{rrrcll} & 1/3x_3 & 1/3x_4 & = & 5 & +z \\ x_1: & 2/3x_3 & -1/3x_4 & \leq & 3 & \\ x_2: & -1/3x_3 & +2/3x_4 & \leq & 2 & \\ x_5: & 1/3x_3 & -2/3x_4 & \leq & 1 & \end{array}$$

Every coefficient c_j is > 0. The solution is:

$$\begin{array}{l} z = -5 \\ x_1 = 3 \\ x_2 = 2 \\ x_3 = 0 \\ x_4 = 0 \\ x_5 = 1 \end{array}$$

The simplex algorithm is particular on two points:

- **Cycling possibility**
  In linear programming, degeneracy is not exceptional (some b_i become equal to *zero*). Then, it may be that during some iterations the quantity $b_{i_p}/a_{i_p j_p}$ (step 5 of the algorithm: choosing a departing variable) is equal to *zero*. There is no change in the solution. Theoretically, it might be that periodically we have the same set of variables in the basis: a cycling occurs and the algorithm does not terminate. In order to avoid this, we have three main techniques: First, the Infinitesimal change of data. Second, the lexicographic rule. Third, Bland's rule. This last rule is easy: we suppose that an order is given to every variable. Step 3 becomes: choose the possible variable with the lowest indice. Step 5 becomes: choose the possible variable with the lowest indice.
  Nevertheless, these techniques are expensive, they increase the average number of iterations. On the other hand, simplex cycling is quite exceptional (it may occur on artificially built problems). Then, most algorithms do not take cycling into account. Simply, the number of iterations is limited (step 2)

- **The Simplex algorithm is not polynomial**
  In the worst case, the behavior of the algorithm is exponential, as it was proved by Klee and Minty in 1972 (see [Sch86] p.139). However, like cycling, this is quite exceptional. In practical terms, the behavior is polynomial and makes the simplex the fastest of possible algorithms. Statistically, the number of iterations is running to the number m of constraints (between m and $3m$ according to G. Dantzig). Note that since 1982, theoretical works on the simplex average behavior have been published ([Sch86] chapter 11.5).

8.2.2 The Dual Simplex algorithm

Given the same optimization problem as in 8.2.1:

$$\begin{array}{l} \min z = \sum_{j=1}^{n} c_j x_j \\ \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i \in \{1, 2, \ldots m\} \\ x_j \in \mathsf{R}^+, \quad \mathsf{i} \in \{1, 2, \ldots \mathsf{n}\} \end{array}$$

The dual simplex algorithm can be applied when c_j are non negative.
As we noted in section 8.1.1, we may build very easily an equivalent dual formulation and apply the primal algorithm. However, transposing is not necessary. Let us describe the algorithm, using the same tableau and notations (basis) as with the primal (8.2.1):

1. (Test for optimality) If $b_i \geq 0$ for every i, the current solution is optimal and we stop.
2. (Test for failure) If the number of iterations exceeds an arbitrary number $tmax$ (usually equal to $2m+n$), the method fails and we stop.
3. (Choosing the departing variable) Select a basic variable x_{n+i_p} with $b_{i_p} < 0$
   A common rule is to choose the variable with the most negative b_i. Another rule (for small systems) is to choose the variable with the smallest $c_{i_p}/a_{i_p j_p}$ (see further step 5).
4. (Test for vacuity) If no coefficient $a_{i_p j} > 0$ exists, the system is infeasible and we stop.
5. (Choosing the entering variable) Select an non-basic variable x_{j_p} such that
   $c_{j_p}/a_{i_p j_p} = \max(c_j/a_{i_p j})\ \ \forall a_{i j_p} < 0$
6. (pivoting) Execute a pivoting operation between x_{j_p} and x_{n+i_p} and return.

All of the remarks (performances, complexity, degeneracy) we made about the primal simplex apply exactly to the dual simplex.

8.3 The methods

Two methods will be detailed (some variants are possible).

8.3.1 Single artificial variable

This method supposes that the system is composed as follows:

- *Inequalities*
- *Variables* ≥ 0

Let the system S be:

$$\begin{array}{l} \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i \in \{1, 2, \ldots m\} \\ x_j \geq 0 \end{array}$$

If every $b_i \geq 0$, the system is feasible. Else:

1. Introducing a variable $x_0 \geq 0$, referred to as *artificial variable*, we define the new system:

   $$\begin{array}{l} x_{n+i} : \ a_{i0} x_0 + \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i \in \{1, \ldots m\} \\ x_j \geq 0 \end{array}$$

   with

   $$\begin{array}{l} a_{i0} = 0 \quad \forall b_i \geq 0 \\ a_{i0} = -1 \quad \forall b_i < 0 \end{array}$$

   Note that if we give a *zero* value to the artificial variable, the system is equivalent to the original one.
2. We select an inequality x_{n+i_p} such that b_{i_p} is the most negative right-hand side. Performing a pivoting operation with x_{n+i_p} and x_0, we obtain a feasible system S' (every right-hand side is non-negative).

   $$\begin{array}{l} x_0 : \ -x_{n+i_p} - \sum_{j=1}^{n} a_{i_p j} x_j \leq -b_{i_p} \\ x_{n+i} : \ a_{i0} x_{n+i_p} + \sum_{j=1}^{n} a'_{ij} x_j \leq b'_i, \quad i \neq i_p \\ x_j \geq 0 \end{array}$$

3. Defining an objective function

   $$z = x_0 = x_{n+i_p} + \sum_{j=1}^{n} a_{i_p j} x_j - b_{i_p}$$

   we consider the minimization problem:

   $$\begin{array}{l} \min x_0 \\ x_0 : \ -x_{n+i_p} - \sum_{j=1}^{n} a_{i_p j} x_j \leq -b_{i_p} \\ x_{n+i} : \ a_{i0} x_{n+i_p} + \sum_{j=1}^{n} a'_{ij} x_j \leq b'_i, \quad i \neq i_p \\ x_j \geq 0 \end{array}$$

   This problem can be solved by the primal simplex (8.2.1). If the solution is strictly positive, the original system is infeasible. If the solution x_0 is equal to *zero*, we cannot conclude (on account of integrity relaxation), unless every x_j has an integer value.
   The method is *inexact*.

Example
Let us consider the problem:

$$\begin{array}{rrcr} 2x_1 & +x_2 & \leq & 3 \\ -3x_1 & -2x_2 & \leq & -2 \\ -x_1 & -4x_2 & \leq & -1 \end{array}$$

The artificial variable is added:

$$\begin{array}{rrrrcr} x_3 : & & 2x_1 & +x_2 & \leq & 3 \\ x_4 : & -x_0 & -3x_1 & -2x_2 & \leq & -2 \\ x_5 : & -x_0 & -x_1 & -4x_2 & \leq & -1 \end{array}$$

We select basic variable x_4 and from:

$$x_4 - x_0 - 3x_1 - 2x_2 = -2$$

we perform the pivoting (x_4, x_0):

$$\begin{array}{rrrrcr} x_3 : & & 2x_1 & +x_2 & \leq & 3 \\ x_0 : & -x_4 & +3x_1 & +2x_2 & \leq & 2 \\ x_5 : & -x_4 & +2x_1 & -2x_2 & \leq & 1 \end{array}$$

we apply the primal simplex (the coefficients of the objective function are the opposite of those of inequality x_0). The solution is obtained after a single pivoting (x_0, x_2):

$$\begin{array}{rrrrcr} x_3 : & 1/2x_4 & +1/2x_1 & -1/2x_0 & \leq & 2 \\ x_2 : & -1/2x_4 & +3/2x_1 & +1/2x_0 & \leq & 1 \\ x_5 : & -2x_4 & +5x_1 & +x_0 & \leq & 3 \end{array}$$

x_0 is now out of the basis, and we have a solution:

$$x_1 = 0 \quad x_2 = 1 \quad x_3 = 2 \quad x_4 = 0 \quad x_5 = 3$$

As every x_j is integer, we may conclude that the original system has integer solutions.

8.3.2 Degenerate Dual Algorithm

"Standard" form:

- *Inequalities*
- *Variables* ≥ 0

Consider the problem P_1: prove or disprove the existence of a solution to:

$$\begin{aligned} Ax &\leq b \\ x &\geq 0 \end{aligned}$$

Introducing any linear objective function $z = cx$ with coefficients $c_j \geq 0$, we may replace the problem P_1 by an optimization problem P_2 (see 3.6):

$$\begin{aligned} Ax &\leq b \\ x &\geq 0 \\ \min z &= cx \end{aligned}$$

The problem P_2 can be solved by the Dual Simplex algorithm (8.2.2). Then, we may prove or disprove the feasibility of P_1.

Since any linear objective function $z = cx$ with coefficients $c_j \geq 0$ is possible, let us choose a function with coefficients equal to *zero*. We then have a totally degenerate dual system, nevertheless we may remark:

- In linear programming, degeneracy is not really a disadvantage (see 8.2.1).
- At each iteration, computational volume is less great: No objective function handling is necessary and the choice of the pivot is simpler, hence faster.

Then, we have a simplified algorithm from the Simplex Dual (8.2.2).

Algorithm

1. (Test for optimality) If $b_i \geq 0$ for every i, the current solution is optimal and we stop.
2. (Test for failure) If the number of iterations exceeds an arbitrary number $tmax$ (usually equal to $2m + n$), the method fails and we stop.
3. (Choosing the departing variable) Select a basic variable x_{n+i_p} with $b_{i_p} < 0$.
   Common rule: choose the variable with the most negative b_i.
4. (Test for vacuity) If no coefficient $a_{i_pj} > 0$ exists, the system is infeasible and we stop.

   Note: The vacuity test can be performed with each inequality with $b_i < 0$.
5. (Choosing the entering variable) Select an non-basic variable x_{j_p} such that $a_{i_pj_p} < 0$
6. (pivoting) Execute a pivoting operation between x_{j_p} and x_{n+i_p} and return.

Example
Let us consider the previous problem (8.3.1):

$$\begin{array}{lrcrcr} x_3: & 2x_1 & +x_2 & \leq & 3 \\ x_4: & -3x_1 & -2x_2 & \leq & -2 \\ x_5: & -x_1 & -4x_2 & \leq & -1 \end{array}$$

Pivoting (x_4, x_1):

$$\begin{array}{lrcrcr} x_3: & 2/3x_4 & -1/3x_2 & \leq & 5/3 \\ x_1: & -1/3x_4 & +2/3x_2 & \leq & 2/3 \\ x_5: & -1/3x_4 & -10/3x_2 & \leq & -1/3 \end{array}$$

Pivoting (x_5, x_2):

$$\begin{array}{lrcrcr} x_3: & 7/10x_4 & -1/10x_5 & \leq & 17/10 \\ x_1: & -4/10x_4 & +2/10x_5 & \leq & 6/10 \\ x_2: & 1/10x_4 & -3/10x_5 & \leq & 1/10 \end{array}$$

Right-hand sides are positive. We have a solution:

$$x_1 = 0.6 \quad x_2 = 0.1$$

Comments

- A fractional solution to the previous problem was obtained. Thus, we cannot conclude that an integer solution exists. This method is *inexact*. Nevertheless, an integer solution to the same problem was obtained by the Single artificial variable method (8.3.1). This is hazard.
- This method should be preferred to the Single artificial variable method (8.3.1). It can be applied directly without building a system including one more variable (this point is important for small systems) and iterations are faster.

9 Interior Point Methods (Integrity relaxation)

From the work of L.G. Khachiyan [Kha79], [Sch86], the linear-programming problem is known to be polynomial.

9.1 Some algorithms:

9.1.1 Khachiyan Ellipsoid Algorithm

Standard Form:

- *Inequalities*
- *Free Variables*

Let us describe very briefly the algorithm in a simplified case; we make the following assumptions:

- *the polyhedron* $\in \mathsf{R}^n$ *is bounded.*
  Then, it can be enclosed in a ball of radius R (hence a particular case of ellipsoid).
- *the polyhedron is full-dimensional.*
  Then, if it is not empty, there exists a ball of radius $r > 0$ inside it.

Values of R and r can be computed from the size of data.
At every step t, an ellipsoid E_t including the polytope is known.
If every constraint is strictly satisfied at the center y of this ellipsoid, the problem is terminated.
Else, from an unsatisfied constraint $a_i x \leq b_i$, a smaller ellipsoid E_{t+1} including the intersection of the ellipsoid E_t and the half-space $a_i x \leq a_i y$ is built.
It is proven that the decrease of volume is each time greater than $1/(e^{-1/2(n+1)})$. If the number of iterations exceeds a fixed number $tmax$,the volume of the ellipsoid is smaller than the volume of the ball r. It cannot include a non empty polyhedron, hence we may conclude that the polyhedron is empty.
Note that the algorithm requires calculations with infinite precision.

9.1.2 Karmarkar Algorithm

This algorithm ([Kar84], [Sch86]) based on a projective method, is faster than the previous one. We will not describe it in this paper.

9.2 Comments

- These methods search for a point with real coordinates inside a polytope. The co-ordinates are generally not rational, and rational arithmetic cannot be used, unlike with simplex methods. There is no probability that an integer point is obtained.
  Thus, these methods, based on integrity relaxation, are *inexact.*
- **Complexity**
  These algorithms are polynomial in relation to the number n of variables and the number m of constraints, but generally with a factor depending on the data size T ($O(n^5 logT)$ arithmetic operations for the ellipsoid method). They are not really ("strongly", "genuinely") polynomial.

  The behavior of these algorithms seems in connection with the worst-case, the opposite of that of the simplex. A great number of algorithms have been published in recent years. The performances of interior-point methods are good for problems including a great number of variables (hundreds of thousands!), however recent works indicated major improvements for interior point methods. An up-to-date comparison of the simplex and interior point methods has been published ([LMS91]). It seems that for some problems with about one thousand variables and one thousand constraints, interior point methods could compete with the simplex.

With a view to solving problems resulting from data dependence, these methods should be avoided.

10 Enumeration Methods

10.1 Rounding Method

Consider the system:

$$\begin{array}{l} \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i \in \{1, \ldots m\} \\ x_j \in \mathbb{Z} \end{array}$$

(we make no assumptions about the sign of variables).
When an integer relaxation method fails and gives a continue (fractional) solution:

$$x^0 = (\xi_1, \xi_2 \cdots \xi_n)$$

a simple technique is the following one:
select in the immediate neighbourhood of the fractional solution all the integer points:

$$x = (x_1, x_2 \cdots x_n) \qquad / x_j \in \mathbb{Z}, \quad |x_j - \xi_j| < 1$$

If the set of constraints is satisfied on a point, we may conclude that the system is feasible.
Else, we cannot conclude, because it may be that the system is feasible, while no integer solution exists near to the fractional solution.
The method is **inexact**.
Note: the method applies to systems including only inequalities. There is no chance that the method works with equations.

10.1.1 Examples

Three cases are possible.

1. Consider the problem of 8.3.2. A fractional solution was obtained

   $$x_1 = 0.6 \quad x_2 = 0.1$$

   We select the four points

   $$\begin{array}{ll} x_1 = 0 & x_2 = 0 \\ x_1 = 0 & x_2 = 1 \\ x_1 = 1 & x_2 = 0 \\ x_1 = 1 & x_2 = 1 \end{array}$$

   The first point $(0, 0)$ is not in the polyhedron. The second point $(0, 1)$ is in the polyhedron. Then, the system is feasible.

2. Let the system be:

   $$\begin{array}{rrcr} 9x_1 & +10x_2 & \leq & 90 \\ -4x_1 & -5x_2 & \leq & -41 \end{array}$$

   A fractional solution can be obtained (for instance by a simplex method):

   $$x_1 = 8 \quad x_2 = 1.8$$

   No integer solution exists in its neighbourhood. We cannot conclude.
   However, the system is feasible, but the nearest integer solution is:

   $$x_1 = 4 \quad x_2 = 5$$

3. Consider the system:

   $$\begin{array}{rrcr} 9x_1 & +10x_2 & \leq & 90 \\ -4x_1 & -5x_2 & \leq & -41 \\ -x_1 & +x_2 & \leq & 0 \end{array}$$

   A fractional solution can be obtained:

   $$x_1 = 8 \quad x_2 = 1.8$$

   No integer solution exists in its neighbourhood. We cannot conclude.
   However the system is infeasible (no integer point exists in the whole polyhedron).

10.1.2 Comments

This method may prove that the system is feasible. But *it will never prove that the system is infeasible* (the suitable case for parallelization in data dependence analysis).

Then, this method should be used only after the failure of a cheap and inexact (integrity relaxation) method M_1 and before applying an exact and (probably) more expensive method M_2 (Branch and Bound, fractional cutting...). If no ulterior exact method is intended, the rounding method is totally useless. Then, the strategy should be:

1. Apply M_1.
2. If M_1 fails, apply the Rounding Method.
3. If the Rounding Method fails, apply M_2.

If the system has n variables, it may be that we examine 2^n points, which can be expensive (and we have no guarantee of concluding). If the method M_2 is not based on the results of the method M_1, the whole strategy appears undesirable.
We think that the single case where the Rounding Method could apply is the end of the first phase of a fractional cutting plane method (see 11.4.3), whose strategy is equivalent to: $M_1 + M_2$. However, a common opinion is that the direct cutting method would be quicker.

10.2 Branch and Bound Methods

This paper does not deal with classical Branch and Bound methods, which are known to be efficient for large size problems.

11 Cutting Plane Methods

The methods which will be examined here are based on techniques and algorithms of the simplex method (section 8), modified in order to obtain integer solutions. Most integer programming specific techniques were essentially developed by R.E. Gomory and are described in classic books such as [GN72], [SM89], etc. This chapter is organized as follows.

1. **Overview**
   The main principles of Cutting Plane programming are recalled.
2. **Basic algorithms**
   Classical cutting-plane algorithms are recalled. They should be used as tools by basic methods and "Splitting" methods.
3. **Surrogate constraints techniques**
   We present the way surrogate constraints can be used, either to modify basic methods, or to define "splitting" methods. Note that these techniques are not specific to Cutting Plane programming.

4. **Basic methods**
   In order to solve the satisfaction problem, we propose some methods very close to basic algorithms and a new algorithm based on surrogate constraints: the Surrogate Dual All-Integer Test (11.4.5).

5. **"Splitting" methods**
   These methods split the satisfaction problem into a succession of elementary problems, each one being solved by a basic algorithm. We review some methods and propose modified methods: "FAS3T+Dual" (11.5.3) and "NNS" (11.5.4).

11.1 Overview

Classical simplex algorithms search for "real" solutions. In order to obtain integer solutions, additional constraints (cuts) are added. There are two principal families of algorithms: the "fractional" algorithms (requiring fractional arithmetic) and the "all-integer" algorithms (requiring only integer arithmetic).

11.1.1 Cutting techniques

Given some integer feasibility or minimization problem over a polytope, the goal of a cut is to diminish the number of real points (or solutions) without changing the number of integer points (or solutions). In subsection 6.1, we saw some simple cuts: when an inequality includes a single variable (see 6.1.3) or when an inequality is divided by GCD (see 6.1.4).
For instance, the inequality:

$$4x_1 - 2x_2 \leq 13 \tag{38}$$

can be replaced by:

$$2x_1 - x_2 \leq 6 \tag{39}$$

(38) and (39) are *equivalent.*
Now, consider the inequality with variables ≥ 0:

$$3x_1 + x_2 \leq 8 \tag{40}$$

We can easily prove that it involves:

$$x_1 \leq 2 \tag{41}$$

Nevertheless, (41) does not involve (40). Constraints (40) and (41) are not equivalent.
Then, if a system includes constraint (40), the cut (41) can be added, but constraint (40) cannot be removed.
Most cuts are not equivalent.

A general cut

Consider the general diophantine equation:

$$x_e + \sum_{j=1}^{n} \alpha_j x_j = \beta \tag{42}$$
$$x_e \geq 0, \quad x_j \geq 0$$

We make no assumptions about data α_j and β, which may be fractional or integer.
Let any number $\lambda \neq 0$. A general cut can be obtained (see [GN72]):

$$\sum_{j=1}^{n} (\lfloor \lambda \rfloor \alpha_j - \lfloor \lambda \alpha_j \rfloor) x_j \geq \lfloor \lambda \rfloor \beta - \lfloor \lambda \beta \rfloor \tag{43}$$

By giving different values to λ, many cuts can be built.

1. **fractional cuts**
   If λ has an integer value, the previous cut (43) becomes:

   $$\sum_{j=1}^{n} (\lambda \alpha_j - \lfloor \lambda \alpha_j \rfloor) x_j \geq \lambda \beta - \lfloor \lambda \beta \rfloor \tag{44}$$

   Particularly, if $\lambda = 1$:

   $$\sum_{j=1}^{n} (\alpha_j - \lfloor \alpha_j \rfloor) x_j \geq \beta - \lfloor \beta \rfloor \tag{45}$$

   We can easily verify that cuts (44) and (45) provide no profit if data α_j and β are integer. Then, such cuts are suitable only if the data are fractional. Consider the equation equivalent to (42) with integer data (and $d > 1$):

   $$d\ x_e + \sum_{j=1}^{n} a_j x_j = c$$

   Then, we have the cut [3] equivalent to (44):

   $$\sum_{j=1}^{n} |\lambda a_j|_d\, x_j \geq |\lambda c|_d \tag{46}$$

   Note: Many values of λ can be chosen. For variants of Gomory Dual Algorithm (11.2.1, 11.2.2), more efficient cuts can be built. λ is given a value equal to an integer prime with c. A practical rule is: choose λ in order that $|\lambda c|_d$ is maximum. A possible technique is (see [Min83]): If $\delta = gcd(d, c)$, Euclid algorithm gives λ and μ such that:
   $gcd(\lambda, d) = 1$ and $-\lambda c + \mu d = \delta$.

[3] in this paper, $|u|_d$ denotes the integer (u mod d)

2. **all-integer cuts**
   Let the inequality be (data and variables are integer):
   $$\sum_{j=1}^{n} a_j x_j \leq c$$
   $$x_j \geq 0$$
   Introducing an integer slack variable $x_e \geq 0$, the inequality is changed into an equation:
   $$x_e + \sum_{j=1}^{n} a_j x_j = c$$
   Then, we may apply the fundamental cut (43). With any number λ such that $0 < \lambda < 1$, we obtain:
   $$\sum_{j=1}^{n} \lfloor \lambda a_j \rfloor x_j \leq \lfloor \lambda c \rfloor$$
   We can also write:
   For each number $k > 1$:
   $$\sum_{j=1}^{n} \lfloor a_j/k \rfloor x_j \leq \lfloor c/k \rfloor \qquad (47)$$
   Note that generally the value of k is chosen in such a way that the new constraint (cut) includes an unitary coefficient (± 1). A common value is some $|a_j|$.
   Note that the previous result can be applied to the equation (data and variables are integer):
   $$\sum_{j=1}^{n} a_j x_j = c$$
   $$x_j \geq 0$$
   As the equation verifies either of these both inequalities:
   $$\sum_{j=1}^{n} a_j x_j \leq c$$
   $$\sum_{j=1}^{n} a_j x_j \geq c$$
   we obtain from (47):
   For each number $k > 1$:
   $$\sum_{j=1}^{n} \lfloor a_j/k \rfloor x_j \leq \lfloor c/k \rfloor \qquad (48)$$
   $$\sum_{j=1}^{n} \lfloor -a_j/k \rfloor x_j \leq \lfloor -c/k \rfloor \qquad (49)$$

11.1.2 Cutting strategies

Basic cutting plane algorithms (11.2) are basic simplex algorithms modified in two ways:

1. *Fractional Cutting Plane algorithms*
   The problem is solved firstly with integrity relaxation. If the solution is not integer, the following sequence is performed as many times as necessary until an integer solution is obtained:
   (a) One (or more) *fractional cut* is introduced in order to make the solution incompatible with the new system.
   (b) The modified problem is solved with integrity relaxation.
2. *All-Integer algorithms*
   An algorithm analoguous to the simplex algorithm is performed. Each time a non unitary pivot is obtained, an *All-Integer cut* is introduced in order to allow a pivoting operation from an unitary coefficient chosen on the "cut". If a solution is obtained, it is necessarily integer.

Generally, an infinity of possible cuts exists, and the choice of a cut is a hard problem.

11.1.3 Standard Form

Most methods require a system of inequalities with constrained variables.

11.1.4 Strategies of various methods

In this paper, we should distinguish:

- The basic methods, close to the basic algorithms.
- The "splitting" methods, where we split the problem into a succession of elementary problems, each one being solved by a basic algorithm.

11.2 Basic algorithms

The aim of all these algorithms is to solve the problem (P^i):

$$\min z = \sum_{j=1}^{n} c_j x_j$$
$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i \in \{1, 2, \ldots m\}$$
$$x_j \in \mathbb{Z}, \mathsf{x_j} \geq 0, \quad \mathsf{j} \in \{1, 2, \ldots \mathsf{n}\}$$

We assume that all data are integer.
This problem is analogous to the "real" simplex algorithms (8.2.1 and 8.2.2). Here, the solution must be integer.

11.2.1 Gomory Dual Algorithm: classical method

This algorithm is also referred to in literature as:

Method of Integer Forms (because it uses integer values of λ, see 11.1.1).

Gomory Dual Algorithm

Fractional Cutting Plane Algorithm (because it handles fractional coefficients).

It should not be confused with the Dual All-Integer Algorithm (11.2.5).
This algorithm can be applied to the problem (P^i) if we assume that *all c_j are non negative.*
The following sequence is performed as many times as necessary:

1. Solve the linear programming problem (with integrity relaxation) by the Dual Simplex algorithm (8.2.2).
2. If the system is infeasible, we stop.
3. If an integer solution is obtained, we stop.
4. (Test for failure) If the number of iterations exceeds an arbitrary number *tmax*, the method fails and we stop.
5. Select a constraint with a fractional right-hand side
6. Build a fractional cut (11.1.1) relative to the selected basis variable (the right-hand side of this new constraint will be necessarily negative) and return.

Note:

If during step 1 an non-basic "cut" variable becomes basic, the constraint is removed from the system.

Comments

- We suppose that in the initial system c_j are non negative. If this is not the case, the first dual simplex is replaced by any suitable simplex method.
- This algorithm requires fractional arithmetic.

11.2.2 Gomory Dual Algorithm: "decreasing congruences"

This algorithm (see [Gon73], [Min83] tome 2, pp 22-28) is a variant of the previous one:
Step 1 is replaced by;

1. If the sequence is performed for the first time, solve the linear programming problem, if not, perform a single iteration by the dual simplex (even if the optimization criterion is not reached).

Comments

- It has been proven that this algorithm is finite.
- Both in theory and in practice, this algorithm is much more efficient than the classical Gomory Dual (see [Min83] tome 2): experiments stated that from an non-integer solution, the average number of iterations in order to obtain an integer solution is about $\log_2(d)$ with the best choice of λ (the meaning of d is involved in the cut (46), see 11.1.1).
- However, it requires fractional arithmetic.

11.2.3 The Finite Primal All-Integer Algorithm

This convergent algorithm (see [GN72]) can be applied to the problem (P^i) if we assume that *all right-hand sides b_i are non negative.* It is also referred to in literature ([NW88]) as:

Simplified Primal Algorithm (SPA)

We will not describe this algorithm of which a rudimentary form is detailed in 11.2.4.

11.2.4 The Rudimentary Primal All-Integer Algorithm

This algorithm can be applied to problem (P^i) if we assume that *all right-hand sides b_i are non negative.* It is a tool which is used extensively in methods described in sections 11.5.1 and 11.5.2.

The principle of the algorithm is also very close to that of the "real" primal simplex algorithm described above (8.2.1). The criteria for pivot selection, optimality, and infinitude are the same. The integer algorithm differs on the following points:

- It is clear that when the chosen pivot a_{rk} is ± 1 (more exactly 1 since the pivot is positive), then a Gaussian elimination can be performed and the coefficients of the resulting system are still integer. Now, when the pivot choice criterion designates a pivot with a value other than 1, the choice of the pivot is canceled. From the inequality including it:

  $$\sum_{j=1}^{n} a_{rj}\, x_j \leq b_r$$

  a new (integer) inequality is constructed, referred to as a "cut" (see all-integer cuts (47) in 11.1.1):

  $$\sum_{j=1}^{n} \lfloor a_{rj} / |a_{rk}| \rfloor\, x_j \leq \lfloor b_r / |a_{rk}| \rfloor$$

  This new inequality is added to the system, and the element located in column k, whose value is now 1 (and is also a possible pivot according to simplex criteria) is chosen as the pivot.

- Every time a "cut" variable returns to the basis after a pivoting operation, the inequality relative to this cut variable is removed from the system (this point is not strictly obligatory, see the comments below).

Let us consider the problem

$$\begin{array}{lrrrcr} z = & 2x_1 & +2x_2 & -5x_3 & - & 0 \\ x_4: & -3x_1 & -2x_2 & +x_3 & \leq & 3 \\ x_5: & -2x_1 & +3x_2 & -3x_3 & \leq & 4 \\ x_6: & 4x_1 & -x_2 & +2x_3 & \leq & 3 \end{array}$$

The primal criterion designates the variable x_3 and the inequality x_6. The value of the coefficient a_{rk} is 2, so a cut is added:

$$x_7: \quad 2x_1 \quad -x_2 \quad +x_3 \quad \leq \quad 1$$

and we can execute the pivoting operation (x_3, x_7):

$$\begin{array}{lrrrcr} z = & 12x_1 & -3x_2 & +5x_7 & - & 5 \\ x_4: & -5x_1 & -x_2 & -x_7 & \leq & 2 \\ x_5: & 4x_1 & & +3x_7 & \leq & 7 \\ x_6: & & +x_2 & -2x_7 & \leq & 1 \\ x_3: & 2x_1 & -x_2 & +x_7 & \leq & 1 \end{array}$$

In the next step the set (x_2, x_6) is selected. Since the coefficient a_{rk} is 1, we can pivot directly:

$$\begin{array}{lrrrcr} z = & 12x_1 & +3x_6 & -x_7 & - & 8 \\ x_4: & -5x_1 & +x_6 & -3x_7 & \leq & 3 \\ x_5: & 4x_1 & & +3x_7 & \leq & 7 \\ x_2: & & +x_6 & -2x_7 & \leq & 1 \\ x_3: & 2x_1 & +x_6 & -x_7 & \leq & 2 \end{array}$$

The same criterion indicates now the set (x_7, x_5). A cut of inequality x_5 is needed (since $a_{rk} = 3$):

$$x_8: \quad x_1 \quad +x_7 \quad \leq \quad 2$$

a third pivoting is performed with the set (x_7, x_8):

$$\begin{array}{lrrrcr} z = & 13x_1 & +3x_6 & +x_8 & - & 10 \\ x_4: & -2x_1 & +x_6 & +3x_8 & \leq & 9 \\ x_5: & x_1 & & -3x_8 & \leq & 1 \\ x_2: & 2x_1 & +x_6 & +2x_8 & \leq & 5 \\ x_3: & 3x_1 & +x_6 & +x_8 & \leq & 4 \\ x_7: & x_1 & & +x_8 & \leq & 2 \end{array}$$

The cut inequality x_7 can be removed, but this is unnecessary: since the coefficients of the function z are ≥ 0, the problem is finished. Minimum is $z^0 = -10$ with the variables;

$$x_1 = 0 \quad x_2 = 5 \quad x_3 = 4$$

Comments

This algorithm is "all-integer" since the coefficients of the transformed systems all remain integer. The fact that only integer arithmetic is required is an attractive feature of this algorithm, unlike real simplex methods or fractional-cutting planes algorithms (11.2.1) where fractional arithmetic has to be used to avoid rouding errors and to build cuts.

Nevertheless, it is a rudimentary algorithm. Its completion is not guaranteed. As opposed to "real" simplex, cycling is not exceptional. An example of cycling is exhibited in [SM89] (pp 240-244) for a problem with only three variables and two inequalities.

Note that the algorithm specifies that if a slack variable from a cut becomes basic, its row is deleted. One reason is that, first, this avoids an excessive increment of the size of the data matrix (the pivoting operations become very expensive), second, many "old" cuts have an obsolete information (new cuts are often strictly "deeper" than some old cuts). Nevertheless, some problems we could not solve (see the comments of 11.4.5) by the R.P.A.I. could terminate after the decision of maintaining old cuts (inversely, some previously solved problems could not terminate!).

Some rules may be added in such a way that the algorithm is finite (11.2.3), but "unfortunately, it is not a practical algorithm because it tends to require an exorbitant number of cuts", as mentioned in [NW88]). However, since most problems which occur in data dependence analysis are small, this algorithm may offer some advantages.

11.2.5 The Dual All-Integer Algorithm

This algorithm, described in [GN72][SM89], can be applied to problem (P^i) if we assume that *all c_j are non negative.*

We will not describe this algorithm of which a rudimentary form is used in 11.4.4

11.3 Surrogate constraints techniques

Consider any system of inequalities:

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i, \quad i \in \{1, 2, \ldots m\}$$

In the first place, we make no assumptions about data a_{ij} and b_i (which may be fractional or integer), the integrity of variables, and the type of variables (free, constrained, or binary).

Consider any set of non-negative coefficients $(\lambda_1, \lambda_2, \ldots, \lambda_m)$, with at least one $\lambda_i > 0$. Obviously,

we can build a new valid inequality:

$$\sum_{j=1}^{n}(\sum_{i=1}^{m}\lambda_i a_{ij})x_j \leq \sum_{i=1}^{m}\lambda_i b_i \qquad (50)$$

The surrogate constraint (50) has the advantage of generally containing information different from that of any individual constraint. The concept of surrogate constraints is not specific to cutting-plane methods, or more generally to integer linear programming (ILP). It was developped for binary ILP methods (see [GN72]), but we could imagine use it for classical linear programming (LP).

We could use these surrogate inequalities in two ways.

11.3.1 "Surrogate choice" algorithms

Firstly, we can modify basic methods thus:
The principle of many integer programming algorithms is based on the choice, at every step of the method, of an inequality. If several inequalities compete, we can imagine the choice of a surrogate inequality. For instance, if we must choose an inequality with a negative right-hand side, the surrogate inequality could be constructed according to one of these rules:

1. if $b_i < 0$, $\lambda_i = 1$ else $\lambda_i = 0$
2. if $b_i < 0$, $\lambda_i = -b_i$ else $\lambda_i = 0$
3. if $b_i \leq b_{i_{min}}/q$, $\lambda_i = 1$ else $\lambda_i = 0$
   where $b_{i_{min}}$ is the most negative b_i and q is some positive integer, for instance 2.

11.3.2 Surrogate methods

Secondly, we can split the original problem into a succession of elementary problems as follows:
Consider the system S (no assumption about data and variables):

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i, \quad i \in \{1, 2, \ldots m\} \qquad (51)$$

The method starts with a point X^1 (vector of components x_j). At every step k of the method, we have a current point X^k and two sets of constraints, constraints that are verified by X^k and constraints that are not:

$$I_1^k = \{i \ / A_i X^k \leq b_i\}$$
$$I_2^k = \{i \ / A_i X^k > b_i\}$$

Obviously:

If the set I_2^k is empty, X^k is a solution to S.

Now, consider the surrogate inequality:

$$\sum_{i\in I_2^k} \lambda_i^k A_i X \leq \sum_{i\in I_2^k} \lambda_i^k b_i$$

and the "surrogate" system S^k:

$$\begin{array}{rcll} \sum_{i\in I_2^k} \lambda_i^k A_i X & \leq & \sum_{i\in I_2^k} \lambda_i^k b_i & \\ A_i X & \leq & b_i, & i \in I_1^k \end{array}$$

We can state:

Theorem 11.3.1 *Let S^k be a system defined from a system S and a point X^k. Then:*

1. *If the system S^k is infeasible, the system S is infeasible.*
2. *If the system S^k owns a solution X^{k+1} and the set I_2^k is non empty, we can define a system S^{k+1} with a set $I_2^{k+1} \subset I_2^k$* [4]

Proof:

1. Since the condition upon the system S^k is weaker than the condition upon the system S, the first assertion is trivial.
2. Suppose that the method gives such a solution X^{k+1} that $I_2^{k+1} = I_2^k$. Consequently:

   $$A_i X^{k+1} > b_i, \quad i \in I_2^k$$

   Since at least one $\lambda_i^k > 0$, this would contradict the satisfaction of S^k:

   $$\sum_{i\in I_2^k} \lambda_i^k A_i X^{k+1} \leq \sum_{i\in I_2^k} \lambda_i^k b_i$$

The number of such steps is obviously finite.

We may imagine various methods with different choices of the surrogate inequality and of the algorithm used at each step (see 11.5).

11.4 Basic methods

Combining various strategies of "real" methods and basic integer algorithms, we can imagine a lot of methods.
The methods which will be described here suppose that the system is composed of inequalities with constrained variables:

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i, \quad i \in \{1, 2, \ldots m\}$$
$$x_j \geq 0$$

[4] in this paper, $A \subset B$ stands for the strict inclusion of A in B ($A \neq B$)

11.4.1 An equivalent dual problem

Consider the problem P_1: prove or disprove the existence of a solution to:

$$\begin{array}{c} Ax \leq b \\ x \geq 0 \\ x \in \mathbb{Z}^n \end{array}$$

We assume that all data are integer.
As in 8.3.2, by introducing any linear objective function $z = cx$ with integer coefficients $c_j \geq 0$ (hence bounded), we can replace the problem P_1 by an optimization problem P_2:

$$\begin{array}{c} \min z = cx \\ Ax \leq b \\ x \geq 0 \\ x \in \mathbb{Z}^n \end{array}$$

If a solution to such a problem exists, it is necessarily finite. Then, by solving the problem P_2, we may prove or disprove the feasibility of P_1.

Since any linear objective function $z = cx$ with coefficients $c_j \geq 0$ is possible, let us choose a function with coefficients equal to *zero*.

This technique is the principle of the various dual methods described in this subsection.

11.4.2 Single integer artificial variable

As with the Single artificial variable method (8.3.1), we can define a primal optimization problem

$$\begin{array}{l} \min x_0 \\ x_0 : \ -x_{n+i_p} - \sum_{j=1}^{n} a_{i_p j} x_j \leq -b_{i_p} \\ x_{n+i} : \ a_{i0} x_{n+i_p} + \sum_{j=1}^{n} a'_{ij} x_j \leq b'_i, \quad i \neq i_p \\ x_j \geq 0 \end{array}$$

This problem can be solved exactly by the (Rudimentary or other) Primal All-Integer Algorithm (11.2.4), or some Gomory Dual Algorithm (11.2.2) (in this case, the first problem of the method is solved by the classical primal simplex (8.2.1)).
If the solution is strictly positive, the original system is infeasible.

11.4.3 Gomory "Degenerate" Dual Algorithm

As we noted in 11.4.1 we can replace the feasibility problem by an optimization problem where the objective function has coefficients $c_j = 0$: Then, we can solve by a Gomory Dual Algorithm, the classical one (11.2.1), or "decreasing congruences" (11.2.2). Obviously the latter should be chosen.

Example

Consider the problem of (8.3.2). As with the Degenerate Dual Algorithm, we firstly obtain a fractional solution:

$$\begin{array}{lrrcr} x_3 : & 7/10x_4 & -1/10x_5 & \leq & 17/10 \\ x_1 : & -4/10x_4 & +2/10x_5 & \leq & 6/10 \\ x_2 : & 1/10x_4 & -3/10x_5 & \leq & 1/10 \end{array}$$

A cut can be built from any basic variable. Let us choose x_1:

$$x_1 - 4/10x_4 + 2/10x_5 = 6/10$$

We can write:

$$10x_1 - 4x_4 + 2x_5 = 6$$

A cut (46) (for instance with $\lambda = 1$), gives:

$$6x_4 + 2x_5 \geq 6$$

It is added to other inequalities:

$$\begin{array}{lrrcr} x_3 : & 7/10x_4 & -1/10x_5 & \leq & 17/10 \\ x_1 : & -4/10x_4 & +2/10x_5 & \leq & 6/10 \\ x_2 : & 1/10x_4 & -3/10x_5 & \leq & 1/10 \\ x_6 : & -6x_4 & -2x_5 & \leq & -6 \end{array}$$

A pivoting operation (variables x_6 and x_4) is performed:

$$\begin{array}{lrrcr} x_3 : & 7/60x_6 & -1/3x_5 & \leq & 1 \\ x_1 : & -2/30x_6 & +1/3x_5 & \leq & 1 \\ x_2 : & 1/60x_6 & -1/3x_5 & \leq & 0 \\ x_4 : & -1/6x_6 & +1/3x_5 & \leq & 1 \end{array}$$

An integer solution is obtained:

$$x_1 = 1 \quad x_2 = 0$$

Comments

The features of this method (behavior, arithmetic) are similar to algorithm 11.2.2. Though the problem is "degenerate", this does not affect the general behavior of the algorithm.

11.4.4 Simple Dual All-Integer Test

We can substitute (see 11.4.1) the optimization problem:

$$\begin{array}{l} \min z = \sum_{j=1}^{n} c_j x_j \\ \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i \in \{1, 2, \ldots m\} \\ x_j \in Z, x_j \geq 0, \quad j \in \{1, 2, \ldots n\} \end{array}$$

(The coefficients c_j are equal to *zero*)
Then, we can apply the classical finite Dual All-Integer algorithm (11.2.5).

Nevertheless, the proposed method does not use the somewhat computationally expensive lexicographic rule mentioned in [GN72] for ensuring finiteness of algorithm. Naturally,the choice of the pivot is different from the classical algorithm. Consequently, as with the Rudimentary Primal All-Integer Algorithm (11.2.4), we are not guaranteed that the algorithm is finite.

The steps of the method are very simple.

We start with the initial system and we execute the following sequence as many times as necessary:

1. (a) If in the current system all b_i are ≥ 0, an obvious solution exists.

   (b) If the current system includes an inequality i such that $b_i < 0$ and all coefficients a_{ij} are ≥ 0, the system has no solution.

   (c) If a system reduction is possible (see 11.4.6 perform the reduction, then return to (1a).

   (d) If the number of iterations exceeds a fixed number $tmax$, for instance $3(m+n)$, we stop. We cannot conclude anything about the solution.

2. We select the inequality (r) with the most negative b_i (this rule is highly recommended):

   $$\sum_{j=1}^{n} a_{rj} x_j \leq b_r$$

   and the variable (x_k) relative to the most negative coefficient a_{rj} of the chosen inequality (this rule is not strict).

3. - If the coefficient a_{rk} is equal to -1, it is chosen as pivot element.
   - Otherwise, an all-integer cut is added to the current system of inequalities:

     $$\sum_{j=1}^{n} \lfloor a_{rj} / |a_{rk}| \rfloor x_j \leq \lfloor b_r / |a_{rk}| \rfloor$$

     and the coefficient a_{sk} (equal to -1) of the new inequality (s) is chosen as pivot element (note that the cutting rule is the same than that used in the Rudimentary Primal All-Integer Algorithm (11.2.4)).

4. A pivoting operation is performed.

5. If the variable x_k is a "cut variable" (i.e. if x_k appeared in connection with a previous cutting operation), the corresponding inequality is removed from the current system.

For example, let us consider the system with constrained variables:

$$\begin{array}{rrrcr} -3x_1 & -2x_2 & +x_3 & \leq & 3 \\ -2x_1 & +3x_2 & -3x_3 & \leq & -4 \\ x_1 & -x_2 & +2x_3 & \leq & -3 \\ 2x_1 & +2x_2 & -5x_3 & \leq & -1 \end{array}$$

Using slack variables ≥ 0 this system is written:

$$\begin{array}{rrrrcr} x_4: & -3x_1 & -2x_2 & +x_3 & \leq & 3 \\ x_5: & -2x_1 & +3x_2 & -3x_3 & \leq & -4 \\ x_6: & x_1 & -x_2 & +2x_3 & \leq & -3 \\ x_7: & 2x_1 & +2x_2 & -5x_3 & \leq & -1 \end{array}$$

We select the inequality x_5 and the variable x_3. Since the value (-3) of the corresponding coefficient is not -1, we add a cut, introducing a cut variable x_8 constructed from x_5:

$$\begin{array}{rrrrcr} x_4: & -3x_1 & -2x_2 & +x_3 & \leq & 3 \\ x_5: & -2x_1 & +3x_2 & -3x_3 & \leq & -4 \\ x_6: & x_1 & -x_2 & +2x_3 & \leq & -3 \\ x_7: & 2x_1 & +2x_2 & -5x_3 & \leq & -1 \\ x_8: & -x_1 & +x_2 & -x_3 & \leq & -2 \end{array}$$

A pivoting operation is executed (variables x_8, x_3):

$$\begin{array}{rrrrcr} x_4: & -4x_1 & -x_2 & +x_8 & \leq & 1 \\ x_5: & x_1 & & -3x_8 & \leq & 2 \\ x_6: & -x_1 & +x_2 & +2x_8 & \leq & -7 \\ x_7: & 7x_1 & -3x_2 & -5x_8 & \leq & 9 \\ x_3: & x_1 & -x_2 & -x_8 & \leq & 2 \end{array}$$

Next, we select the inequality x_6 and the variable x_1. This time, no cut is necessary and we may execute the pivoting operation (x_6, x_1):

$$\begin{array}{rrrrcr} x_4: & -4x_6 & -5x_2 & -7x_8 & \leq & 29 \\ x_5: & x_6 & +x_2 & -x_8 & \leq & -5 \\ x_1: & -x_6 & -x_2 & -2x_8 & \leq & 7 \\ x_7: & 7x_6 & +4x_2 & +9x_8 & \leq & -40 \\ x_3: & x_6 & & +x_8 & \leq & -5 \end{array}$$

The inequalities x_7 and x_3 have no solution. Hence the system has no solution.

A particular solution to the dual problem

In the previous algorithm, our goal was to prove or disprove the feasibility of the system. Consequently, the value of a hypothetical solution had no importance, and we could reduce the problem (11.4.6).

However, in some cases, we cannot reduce the problem since we need the value of a solution (see 11.5.3).

Consider a system where all the coefficients of a variable x_q are ≤ 0, including at least one inequality i with $a_{iq} < 0$ and $b_i < 0$:

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i \in \{1, 2, \ldots m\}$$
$$x_j \geq 0$$

Let us choose the inequality with $a_{iq} < 0$ and the greatest b_i / a_{iq}

$$\sum_{j=1}^{n} a_{rj} x_j \leq b_r$$

If the coefficient a_{rq} is equal to -1, it is chosen as pivot element. Otherwise, an all-integer cut is added to the current system of inequalities:

$$\sum_{j=1}^{n} \lfloor a_{rj} / |a_{rq}| \rfloor x_j \leq \lfloor b_r / |a_{rq}| \rfloor$$

and the coefficient a_{sq} (equal to -1) of the new inequality (s) is chosen as pivot element. We can easily prove that if we perform such a pivoting operation, all the right-hand sides of the inequalities with a *non-zero* coefficient become non negative. The right-hand sides of the other inequalities are unchanged. Hence:

- *If the right-hand sides of the inequalities including a zero coefficient are non negative, we get a solution.*

For instance:

$$\begin{array}{lrrrcr} x_4: & -4x_1 & -x_2 & +x_7 & \leq & 1 \\ x_5: & x_1 & & -3x_7 & \leq & 2 \\ x_6: & -x_1 & -4x_2 & +2x_7 & \leq & -7 \\ x_3: & x_1 & -3x_2 & -x_7 & \leq & -5 \end{array}$$

In the column x_2, we select the coefficient of the row x_3. Let us add a cut x_8 from x_3:

$$\begin{array}{lrrrcr} x_4: & -4x_1 & -x_2 & +x_7 & \leq & 1 \\ x_5: & x_1 & & -3x_7 & \leq & 2 \\ x_6: & -x_1 & -4x_2 & +2x_7 & \leq & -7 \\ x_3: & x_1 & -3x_2 & -x_7 & \leq & -5 \\ x_8: & & -x_2 & -x_7 & \leq & -2 \end{array}$$

After a pivoting operation (x_8, x_2), we obtain:

$$\begin{array}{lrrrcr} x_4: & -4x_1 & -x_8 & +2x_7 & \leq & 3 \\ x_5: & x_1 & & -3x_7 & \leq & 2 \\ x_6: & -x_1 & -4x_8 & +6x_7 & \leq & 1 \\ x_3: & x_1 & -3x_8 & +2x_7 & \leq & 1 \\ x_2: & & -x_8 & +x_7 & \leq & 2 \end{array}$$

Such systems which may appear in methods 11.5.3 and 11.5.4 include a single negative right-hand side, for instance the one from the first example of 11.5.3:

$$\begin{array}{lrrcr} x_6: & 2x_1 & -5x_2 & \leq & -8 \\ x_5: & 2x_1 & -x_2 & \leq & 6 \end{array}$$

In this case, the way the pivoting operation is specified is quite simple:

- *Choose the column whose the variable proved the infinitude (all coefficients ≤ 0) and the row with a negative right-hand side. If needed, introduce a cut.*

Comments

Like the Rudimentary Primal All-Integer Algorithm (11.2.4)), this algorithm is attractive since it is all integer. Like the R.P.A.I., it is not a finite algorithm. However, general behavior of non finite algorithms is often as good or better than finite algorithms (essentially because in practice the finite number of expected iterations is very large, this is the case, for instance, the classical simplex method). Its behavior is probably comparable to that of finite Dual All-Integer algorithm, but precise comments upon the finite algorithm behavior are lacking in literature.

Experiments seemed interesting. Some problems with 9 variables, for instance, could be solved easily while the same problems were very hard to solve by "splitting" methods using the R.P.A.I. (we must specify that these systems do not stem from data dependence analysis, but from more complex task scheduling problems). However, the algorithm seems sensitive to the choice of the pivot. Some problems could be solved with very few iterations, but required many more iterations or could not terminate with a different choice of pivot. But this inconvenience was not noticed with problems coming from data dependence analysis (they are very "easy").

11.4.5 Surrogate Dual All-Integer Test

This algorithm is a variant of the previous one. The principle is similar to the Simple Dual All-Integer Test. We start with the initial system and we execute an analogous sequence of operations as many times as necessary. The sequence differs on these points:

- If the number of inequalities with a negative right-hand side is greater than 1, a surrogate constraint is built (11.3.1), which is the sum of all such inequalities. A similar rule for selecting a pivot is applied to this new inequality. If no negative pivot is found, the system has no solution. If the value of the pivot obtained is -1, the surrogate inequality is added to the system (introducing a "surrogate" slack variable), otherwise a cut is built from the surrogate inequality and added to the system. Then, the usual pivoting operation is executed,

- If a "surrogate" variable comes back to the basis, it is removed from the current system (like a

"cut" variable).

Let us return to the previous problem (11.4.4):

$$\begin{array}{lrrrrr} x_4: & -3x_1 & -2x_2 & +x_3 & \leq & 3 \\ x_5: & -2x_1 & +3x_2 & -3x_3 & \leq & -4 \\ x_6: & x_1 & -x_2 & +2x_3 & \leq & -3 \\ x_7: & 2x_1 & +2x_2 & -5x_3 & \leq & -1 \end{array}$$

3 inequalities have a negative right-hand side. Their sum gives another inequality (y denotes a temporary variable):

$$y: \quad x_1 \quad +4x_2 \quad -6x_3 \quad \leq \quad -8$$

The coefficient of x_3 is selected, but its value is -6. A cut x_8 is built from y and added to the system:

$$x_8: \qquad -x_3 \quad \leq \quad -2$$

A pivoting operation is executed (x_8, x_3):

$$\begin{array}{lrrrrr} x_4: & -3x_1 & -2x_2 & +x_8 & \leq & 1 \\ x_5: & -2x_1 & +3x_2 & -3x_8 & \leq & 2 \\ x_6: & x_1 & -x_2 & +2x_8 & \leq & -7 \\ x_7: & 2x_1 & 2x_2 & -5x_8 & \leq & 9 \\ x_3: & & & -x_8 & \leq & 2 \end{array}$$

The set (x_6, x_2) is selected. No cut is necessary and we can pivot:

$$\begin{array}{lrrrrr} x_4: & -5x_1 & -2x_6 & -3x_8 & \leq & 15 \\ x_5: & x_1 & +3x_6 & +3x_8 & \leq & -19 \\ x_1: & -x_1 & -x_6 & -2x_8 & \leq & 7 \\ x_7: & 4x_1 & +2x_6 & -x_8 & \leq & -5 \\ x_3: & & & -x_8 & \leq & -2 \end{array}$$

There is no solution to the inequality x_5. Hence the system has no solution.

Choice of the surrogate constraint

With most problems which were experimented, the behavior of this algorithm was generally analogous to that of the Simple Dual All-Integer Test. However long sequences of iterations were not observed with the Surrogate Test, which seems less sensitive to the choice of the pivot.

Other choices of the surrogate constraint seemed efficient such as the following one and should be more intensively experimented:

- if $b_i \leq b_{i_{min}}/q$, $\lambda_i = 1$ else $\lambda_i = 0$
  where $b_{i_{min}}$ is the most negative b_i and $q = 2$.

11.4.6 Reduction of the dual problem

We should examine here the possibilities of reducing the system while some All-Integer algorithms (11.4.4, 11.4.5 ...) are performed.

Let the system with constrained variables be (all integer data):

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i, \quad i \in \{1, 2, \ldots m\}$$
$$x_j \geq 0$$

Suppose we (temporarily) replace a variable x_k (for instance x_1) by a free one:

$$-t_1 \leq 0$$
$$a_{i1}t_1 + \sum_{j=2}^{n} a_{ij}x_j \leq b_i, \quad i \in \{1, 2, \ldots m\}$$
$$x_j \geq 0$$

Then, we could search for an opportunity for a Selective Fourier-Motzkin elimination (6.2) of this free variable, which may appear after a pivoting operation. Remark that the criteria relative to constrained variables are the same as those relative to free ones (theorem 6.2.1):

Criterion 1 The value of every negative coefficient of x_k is -1

Criterion 2 The value of every positive coefficient of x_k is 1

In the following cases, the number of inequalities does not increase (however we could adopt another strategy):

1. *All the coefficients of a column are* ≤ 0:

$$\begin{array}{lrrrrr} x_4: & -4x_1 & -x_2 & +x_8 & \leq & 1 \\ x_5: & x_1 & & -3x_8 & \leq & 2 \\ x_6: & -x_1 & & +2x_8 & \leq & -7 \\ x_7: & 7x_1 & & -5x_8 & \leq & 9 \\ x_3: & x_1 & -3x_2 & -x_8 & \leq & -2 \end{array}$$

   The inequalities including a *non-zero* coefficient are removed:

$$\begin{array}{lrrrr} x_5: & x_1 & -3x_8 & \leq & 2 \\ x_6: & -x_1 & +2x_8 & \leq & -7 \\ x_7: & 7x_1 & -5x_8 & \leq & 9 \end{array}$$

2. *All the coefficients of a column are* ≥ 0:

$$\begin{array}{lrrrrr} x_4: & -4x_1 & +x_2 & +x_8 & \leq & 1 \\ x_5: & x_1 & & -3x_8 & \leq & 2 \\ x_6: & -x_1 & +4x_2 & +2x_8 & \leq & -7 \\ x_7: & 7x_1 & & -5x_8 & \leq & 9 \\ x_3: & x_1 & +3x_2 & -x_8 & \leq & -2 \end{array}$$

The column is removed from the system:

$$\begin{array}{rrrcr} w_4: & -4x_1 & +x_8 & \leq & 1 \\ x_5: & x_1 & -3x_8 & \leq & 2 \\ w_6: & -x_1 & +2x_8 & \leq & -7 \\ x_7: & 7x_1 & -5x_8 & \leq & 9 \\ w_3: & x_1 & -x_8 & \leq & -2 \end{array}$$

3. *A single coefficient of a column is* ≥ 0 *(and one of both criteria is satisfied)*:

$$\begin{array}{rrrrcr} x_4: & -4x_1 & +x_2 & +x_8 & \leq & 1 \\ x_5: & x_1 & & -3x_8 & \leq & 2 \\ x_6: & -x_1 & -4x_2 & +2x_8 & \leq & -7 \\ x_7: & 7x_1 & & -5x_8 & \leq & 9 \\ x_3: & x_1 & -3x_2 & -x_8 & \leq & -2 \end{array}$$

The column is removed, however the inequalities with a negative coefficient are modified:

$$\begin{array}{rrrcr} w_4: & -4x_1 & +x_8 & \leq & 1 \\ x_5: & x_1 & -3x_8 & \leq & 2 \\ w_6: & -17x_1 & +6x_8 & \leq & -3 \\ x_7: & 7x_1 & -5x_8 & \leq & 9 \\ w_3: & -11x_1 & +2x_8 & \leq & 1 \end{array}$$

11.5 "Splitting" methods

11.5.1 Constraint-Matrix Test

This method, described in [Wal88], can be applied if the system is composed of m_1 inequalities and m_2 equations such that variables are constrained, and the right-hand sides of inequalities are non negative:

$$\begin{array}{l} \sum_{j=1}^{n} a_{ij}x_j \leq b_i, \quad i \in \{1, \ldots m_1\}, \quad b_i \geq 0 \\ \sum_{j=1}^{n} a_{ij}x_j = b_i, \quad i \in \{m_1+1, \ldots m_1+m_2\} \\ x_j \geq 0 \end{array}$$

The main principle of the method is the following: at every step, an equivalent and analogous system is built, but with one equation less (and one variable less). The number of inequalities may increase.

If a system without equations or with only equations with right-hand sides equal to zero is obtained, then the system has an obvious solution (all x_j equal to zero). Let us describe the process.

1. An equation is chosen, for instance the first one ($l = m_1 + 1$). We assume that $b_l \geq 0$. If not, we change the sign of every coefficient in the equation.

2. If $b_l > 0$ we consider the subsystem:

$$\begin{array}{l} \sum_{j=1}^{n} a_{ij}x_j \leq b_i, \quad i \in \{1, \ldots m_1\} \\ \sum_{j=1}^{n} a_{lj}x_j = b_l \end{array}$$

We define the economic linear function z:

$$z = b_l - \sum_{j=1}^{n} a_{lj}x_j$$

and the corresponding optimization problem:

$$\begin{array}{l} \min z = z^0 \\ \sum_{j=1}^{n} a_{ij}x_j \leq b_i, \quad i \in \{1, \ldots m_1\} \\ \sum_{j=1}^{n} a_{lj}x_j \leq b_l \end{array}$$

(note that the function and the last constraint are analogous)

We solve this problem by the Rudimentary Primal All-Integer Algorithm (11.2.4). The solution z^0 is necessarily finite and non negative.

- If the value of z^0 is positive, the subsystem has no solution, hence the original system has no solution also.
- Otherwise z^0 is zero, indicating that the subsystem has a solution. If there are no other equations, the full system has one also.

3. The full system is now of the form (for the sake of argument, we use the notations of the initial system, but note that introducing cuts may have increased the number of inequalities):

$$\begin{array}{l} \sum_{j=1}^{n} a_{ij}x_j \leq b_i, \quad i \in \{1, \ldots m_1\} \quad (b_i \geq 0) \\ \sum_{j=1}^{n} a_{ij}x_j = 0, \quad i = m_1 + 1 \\ \sum_{j=1}^{n} a_{ij}x_j = b_i, \quad i \in \{m_1+2, \ldots m_1+m_2\} \\ x_j \geq 0 \end{array}$$

Our purpose is now to eliminate the first equality (while keeping non-negative right-hand sides of the inequalities). It is easy in any of these cases:

all the coefficients of the equation have the same sign (variables with non zero coefficients are necessarily equal to zero and disappear like the equation from the system)

the equation has an unitary coefficient (a Gaussian elimination is performed and the equation is replaced by an inequality, see the note below).

Otherwise, a cut (analogous to those used in the Rudimentary Primal All-Integer Algorithm) followed by a pivoting operation is performed repeatedly, until an opportunity for elimination appears in the equation.

Note: When a Gaussian elimination step removes a constrained variable from a system, the equation must be replaced by an inequality indicating that the

eliminated variable was constrained. For instance, we can eliminate x_2 if the system includes an equation such as:

$$-2x_{10} \quad +x_2 \quad +x_9 \quad = \quad 0$$

but since $x_2 \geq 0$, we must introduce the inequality:

$$-2x_{10} \quad +x_9 \quad \leq \quad 0$$

11.5.2 Method "FAS3T"

This algorithm [BP89] assumes that the system is composed of inequalities:

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i \in \{1, 2, \ldots m\}$$

The principle of the method does not depend on whether or not the variables are constrained, and can be described as a "surrogate" method (11.3.2):

- The surrogate constraint is defined thus:

$$\begin{array}{l} \forall i \in I_2^k, \quad \lambda_i^k = 1 \\ \forall i \in I_1^k, \quad \lambda_i^k = 0 \end{array}$$

$$\sum_{i \in I_2^k} A_i X \leq \sum_{i \in I_2^k} b_i$$

- The feasibility (or the infeasibility) of each system S^k is proved by solving a minimization problem of a function z_k, which we denote P^k:

$$\begin{array}{l} \min z_k = z_k^0 \\ \sum_{i \in I_2^k} A_i X - \sum_{i \in I_2^k} b_i = z_k \\ A_i X \leq b_i, \qquad\qquad i \in I_1^k \end{array}$$

  If $z_k^0 > 0$, the system S^k is infeasible.
  If $z_k^0 \leq 0$, the system S^k is feasible and X^{k+1} is defined as the point where z_k is minimized.

A Particular case

Consider the case (at step k) where $z_k^0 = 0$. Let the set of all possible solutions be:

$$E^k = \{X / z_k(X) = z_k^0\}$$

It may be proven that the original system S has no solution if:

$$\forall X \in E^k : (\exists i \in I_2^k \ / A_i X \neq b_i)$$

Then, we could replace the original feasibility problem of S by the following easier one:

$$\begin{array}{l} A_i X \leq b_i, \quad i \in I_1^k \\ A_i X = b_i, \quad i \in I_2^k \end{array}$$

Practical aspects

This method requires a finite solution for each current problem. Generally, variables of systems resulting from dependence analysis are bounded, but it may occur that domains relative to the generic problems used by the method are not. In these cases, we are not guaranteed that finite solutions will be obtained. If necessary, constraints limiting the domain will be added in such a way that they do not modify the result of the algorithm (for example giving bounds with great values to variables). The authors of the method propose the following sequence at each step:

1. (If necessary) change the problem with constrained variables by the "Single added variable" technique (7.1.2).
2. Apply the Rudimentary Primal All-Integer Algorithm (11.2.4) to P^k.
3. If the domain relative to P^k is unbounded and an infinite solution occurs, add a constraint of the type $cX \leq M$, with M sufficiently large, in order to obtain a finite solution.

The initialization of the method could be performed according to one of these ways:

- split the inequalities into two sets, then introduce constrained variables
- introduce constrained variables, then split the inequalities.

Clearly, the second one should be preferred. We would be entirely free to reduce the system, and the next steps would be made easier.

11.5.3 Method "FAS3T+Dual"

The most delicate point of "FAS3T" method is the adjunction of an auxiliary constraint in the case of infinitude. Adding a constraint with a big arbitrary bound has got disadvantages (see 7.4.1). However, the obtainment of the minimum value of $z_k(X)$ is not obligatory. Any point X of the polytope satisfying $z_k(X) \leq 0$ satisfies also S^k and can be chosen as a point X^{k+1} in order to define a system S^{k+1} (theorem 11.3.1).

Then, as soon as the infinitude of z_k^0 is detected, we should look at the value of z_k^* at the current point X^* of the primal algorithm. If $z_k^* \leq 0$, X^* can be chosen immediately as a point X^{k+1}. If $z_k^* > 0$, a constraint could be built with a small value of M, just enough to ensure a negative value of z_k^0. Obviously, since such a constraint is not strictly valid,

it would not be maintained in the following steps of the method, which consequently would be more complex. A simpler technique could be to search for a finite solution (which necessarily exists in this case) to the current system S^k by applying a dual algorithm. Such a solution should be obtained with a *single* iteration (see: "A particular solution to the dual problem" in 11.4.4). Then, the following strategy could be applied:
Firstly, introduce (if needed) constrained variables by reducing techniques, then repeat this sequence as many times as necessary:

1. Apply the Rudimentary Primal All-Integer Algorithm to P^k.
2. - If a finite solution z_k^0 is obtained at a point X_k^0:
     (a) If $z_k^0 > 0$, the system S is infeasible.
     (b) If the set I_2^k is a singleton (it includes a single inequality index), the system S (in this case identical to S^k) is feasible.
     (c) Otherwise choose $X^{k+1} = X_k^0$
   - If an infinite solution occurs (necessarily after a finite point X^*):
     (a) If the set I_2^k is a singleton, the system S is feasible.
     (b) If $z_k^* \leq 0$, choose $X^{k+1} = X^*$.
     (c) Otherwise, search a finite solution X^{k+1} to the current system S^k by applying a single iteration of the dual all-integer algorithm.

Example 1

Consider the system S with constrained variables:

$$\begin{array}{lrrcr} x_3: & 5x_1 & -7x_2 & \leq & -3 \\ x_4: & -3x_1 & +2x_2 & \leq & -5 \\ x_5: & 2x_1 & -x_2 & \leq & 6 \end{array}$$

Step 1

The method starts with the point X^1 ($x_1 = 0$, $x_2 = 0$). Let us split the inequalities:

$$I_1^1 = \{5\}$$
$$I_2^1 = \{3, 4\}$$

Let us define the surrogate inequality $(x_3 + x_4)$:

$$x_6: \quad 2x_1 \quad -5x_2 \quad \leq \quad -8$$

the function:

$$z_1 = -x_6$$

and the problem P^1:

$$\begin{array}{lrrcr} & & \min z_1 & = & z_1^0 \\ -z_1 & +2x_1 & -5x_2 & = & -8 \\ x_5: & 2x_1 & -x_2 & \leq & 6 \end{array}$$

The solution z_1^0 is infinite (x_2 can be given any positive value). Consequently, consider the system S^1:

$$\begin{array}{lrrcr} x_6: & 2x_1 & -5x_2 & \leq & -8 \\ x_5: & 2x_1 & -x_2 & \leq & 6 \end{array}$$

Let us build a cut x_7 from x_6:

$$\begin{array}{lrrcr} x_6: & 2x_1 & -5x_2 & \leq & -8 \\ x_5: & 2x_1 & -x_2 & \leq & 6 \\ x_7: & & -x_2 & \leq & -2 \end{array}$$

After a single pivoting operation (x_7, x_2), the system S^1 is primal feasible (all right-hand sides ≥ 0):

$$\begin{array}{lrrcr} x_6: & 2x_1 & -5x_7 & \leq & 2 \\ x_5: & 2x_1 & -x_7 & \leq & 8 \\ x_2: & & -x_7 & \leq & 2 \end{array}$$

Removing the surrogate constraint x_6 and introducing the constraints x_3 and x_4 in their new form, we obtain a new formulation of the system S :

$$\begin{array}{lrrcr} x_3: & 5x_1 & -7x_7 & \leq & 11 \\ x_4: & -3x_1 & +2x_7 & \leq & -9 \\ x_5: & 2x_1 & -x_7 & \leq & 8 \\ x_2: & & -x_7 & \leq & 2 \end{array}$$

Step 2

The method continues with the point X^2 ($x_1 = 0$, $x_2 = 2$). Let the sets of inequalities be:

$$I_1^2 = \{3, 5\}$$
$$I_2^2 = \{4\}$$

the function:

$$z_2 = -x_4$$

and the problem P^2:

$$\begin{array}{lrrcr} & & \min z_2 & = & z_2^0 \\ -z_2 & -3x_1 & +2x_7 & = & -9 \\ x_3: & 5x_1 & -7x_7 & \leq & 11 \\ x_5: & 2x_1 & -x_7 & \leq & 8 \\ x_2: & & -x_7 & \leq & 2 \end{array}$$

We obtain (we don't give the computational details):
$z_2^0 = -2$ (with $x_1 = 5$, $x_2 = 4$).
Since the set $I_2^2 = \{4\}$ is a singleton, the system S is feasible.

Example 2

Consider the system S with constrained variables:

$$\begin{array}{lrrcr} x_3: & 5x_1 & -7x_2 & \leq & -3 \\ x_4: & -3x_1 & +2x_2 & \leq & -5 \\ x_5: & 2x_1 & -x_2 & \leq & 6 \\ x_6: & -4x_1 & +x_2 & \leq & 2 \end{array}$$

Step 1

We start as follows:
the point X^1 $(x_1 = 0, x_2 = 0)$.

$$I_1^1 = \{5, 6\}$$
$$I_2^1 = \{3, 4\}$$

Surrogate inequality $(x_3 + x_4)$:

$$x_7: \quad 2x_1 \quad -5x_2 \quad \leq \quad -8$$

the function:

$$z_1 = -x_7$$

and the problem P^1:

$$\begin{array}{lrrcr} & & \min z_1 & = & z_1^0 \\ -z_1 & +2x_1 & -5x_2 & = & -8 \\ x_5: & 2x_1 & -x_2 & \leq & 6 \\ x_6: & -4x_1 & +x_2 & \leq & 2 \end{array}$$

The first iteration (x_6, x_2) of P^1 gives (no cut is necessary):

$$\begin{array}{lrrcr} -z_1 & -18x_1 & +5x_6 & = & 2 \\ x_5: & -2x_1 & +x_6 & \leq & 8 \\ x_2: & -4x_1 & +x_6 & \leq & 2 \end{array}$$

A second iteration of P^1 would give an infinite value of z_1^0 (x_1 can be given any positive value). Since $z_1^* = -2$, we adopt the point

$$X^2 = X^*(x_1 = 0, x_2 = 2).$$

Removing the surrogate constraint x_7 and introducing the constraints x_3 and x_4 in their new form, we obtain a new formulation of the system S :

$$\begin{array}{lrrcr} x_3: & -23x_1 & +7x_6 & \leq & 11 \\ x_4: & 5x_1 & -2x_6 & \leq & -9 \\ x_5: & -2x_1 & +x_6 & \leq & 8 \\ x_2: & -4x_1 & +x_6 & \leq & 2 \end{array}$$

Step 2

The method continues with the point X^2 $(x_1 = 0,$ $x_2 = 2)$. Let the sets of inequalities be:

$$I_1^2 = \{3, 5, 6\}$$
$$I_2^2 = \{4\}$$

the function:

$$z_2 = -x_4$$

and the problem P^2:

$$\begin{array}{lrrcr} & & \min z_2 & = & z_2^0 \\ -z_2 & +5x_1 & -2x_6 & = & -9 \\ x_3: & -23x_1 & +7x_6 & \leq & 11 \\ x_5: & -2x_1 & +x_6 & \leq & 8 \\ x_2: & -4x_1 & +x_6 & \leq & 2 \end{array}$$

We obtain (we don't give the computational details):
$z_2^0 = -2$ (with $x_1 = 5$, $x_2 = 4$).
Since the set $I_2^2 = \{4\}$ is a singleton, the system S is feasible.

11.5.4 Method "NNS" (Non Negative Surrogate variable)

This method should be applied in the same conditions as in 11.5.3 (inequalities with constrained variables). Here, we don't really search the minimum of a function $z_k(X)$. We simply look for *any point* $z_k(X)$ *satisfying:* $z_k(X) \leq 0$. In order to obtain such a point, we could apply a minimization algorithm to $z_k(X)$ and interrupt it as soon as the function becomes non positive. Let us use the same notations as previously. The strategy applied at each step k should be the following one:

1. Apply the Rudimentary Primal All-Integer Algorithm to P^k. Interrupt it if a $z_k^* \leq 0$ is observed.

2. (a) If $z_k^* \leq 0$:
   i. If the set I_2^k is a singleton, the system S is feasible.
   ii. otherwise choose $X^{k+1} = X_k^*$
   (b) If a finite solution z_k^0 (> 0) is obtained: the system S is infeasible.
   (c) If an infinite solution occurs (after a finite point X^*):
   i. If the set I_2^k is a singleton, the system S is feasible.
   ii. otherwise, search a finite solution X^{k+1} to the current system S^k by applying a single iteration of the dual algorithm.

11.5.5 Other surrogate methods

We can imagine various surrogate methods, with different choices of the surrogate constraint and the algorithm leading from a system S^k to a system S^{k+1}

- *Surrogate constraint:*

  In place of identical λ_i equal to 1, we could give different values to the λ_i (weights) according to the values of the right-hand sides for instance (see 11.3.1).

- *Algorithm:*

  In place of the Rudimentary Primal All-Integer Algorithm or other minimization algorithm, we could apply a feasibility algorithm such as a dual one.

12 Which strategy?

Since an efficient reduction phase is desirable before applying some exact programming method, clearly, we should be concentrating on final methods whose standard form allows the greatest system reduction, that is composed of:

inequalities

constrained variables

The preprocessing phase does not present any difficulty. The reduction sequence could be the following one:

1. Eliminate all equalities (see 5.8).
2. Eliminate as many remaining variables as possible (selective Fourier-Motzkin)
3. Introduce constrained variables.

However, steps 2 and 3 could be mixed with profit (see 7.5, 7.8.1).

The choice of the final method is less simple. It depends on the size of the final system. As we noted, the Fractional Cutting Plane Algorithms seem suitable for "middle-size" problems. And the "All-Integer" algorithms are very attractive in the case of "small" problems. Then, a *Dual-All-Integer algorithm* (for instance a surrogate one, 11.4.5), the "FAS3T+Dual" method (11.5.3) or the method "NNS" (11.5.4) could be applied to "small" problems, while the *Fractional Cutting Plane Algorithm* "decreasing congruences" (11.2.2) could be applied to larger ones. But how to decide whether or not a problem is "small" is no simple matter. Some problems appear in an unpredictable way much more difficult than other ones with the same size.

However, in the case of data dependence analysis, the All-Integer algorithms are quite sufficient. A sequence composed of a reduction phase and the surrogate Dual All-Integer Test is implemented in the INRIA PIAF parallelizer [GLL+90]. The preliminary measurements performed on PIAF were presented in [ES92]. They show that the percentage of time spent in the data dependence analyzer is insignificant, and frequently, the preprocessing phase gives the solution!

13 Conclusion

In this paper we have reviewed some standard algorithms and presented some new algorithms which could apply to any integer linear programming satisfiability problem. The current sequence of algorithms implemented in PIAF parallelizer could handle non standard data information resulting from sophisticated data flow analysis methods. Moreover, it can be applied to other phases in a compiler. For now, it is used also for variable compatibility verifications, but we plan to use this sequence (or a more complex one) to solve also a scheduling problem for exploiting fine grain parallellism.

Acknowledgments

The author wish to thank Christine EISENBEIS, William JALBY and François THOMASSET for their valuable and friendly comments.

References

[Ban79] Uptal Banerjee. *Speedup of ordinary programs.* PhD thesis, University of Illinois, 1979.

[Ban88] U. Banerjee. *Dependence Analysis for Supercomputing.* Kluwer Academic Publishers, Norwell, Mass, 1988.

[BP89] H. Bennaceur and G. Plateau. Sur le problème de satisfaction de contraintes. Rapport LIPN 89- 6, Universite Paris Nord, Centre Scientifique et Polytechnique, Département de mathématiques et informatique, Juin 1989.

[Dan63] G.B. Dantzig. *Linear Programming and Extensions.* Princeton University Press, Princeton, New-Jersey, 1963.

[DKT87] P. D. Domich, R. Kannan, and L. E. Trotter.JR. Hermite normal form computation using modulo determinant arithmetic. *Mathematics of Operations Research*, 12(1), February 1987.

[ES92] Christine Eisenbeis and Jean-Claude Sogno. A general algorithm for data dependence analysis. In *Proceedings of 1992 International Conference on Supercomputing*, pages 291–299, Washington, July 19-23 1992. appears also as: Rapport de Recherche INRIA no 1699.

[FG91] John J. Forrest and Donald Goldfarb. Steepest edge simplex algorithms for linear programming. Research Report No 17147, IBM Research Division, June 1991.

[GKT91] Gina Goff, Ken Kennedy, and Chau-Wen Tseng. Practical dependence testing. In SIGPLAN Notices, editor, *Proc. of the 1991 ACM SIGPLAN Conference on Programming Language Design and Implementation*, volume 26, pages 15–29, Toronto, Ontario, Canada, June 26-28 1991. ACM.

[GLL+90] Marie Christine Giboulot, Eve Rose Lebon, Michel Loyer, Gregory Popovitch, Husein Shafie, and François Thomasset. Parallel execution of fortran programs on the ews workstation. In Roland Glowinski and Alain Lichnewsky, editors, *Computing Methods in Applied Sciences and Engineering*, pages 413–423, Philadelphia, January 1990. SIAM.

[GN72] R. S. Garfinkel and G. N. Nemhauser. *Integer Programming.* Wiley-Interscience, 1972.

[Gon73] M. Gondran. Un outil pour la programmation en nombres entiers : La méthode des congruences decroissantes. *Rev. Fr. Automatique, Informatique, Rech. Opérationnelle 3,pp. 35-54*, 1973.

[Kar84] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth Annual ACM Symposium of computing*, Washington, 1984.

[Kha79] L. G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademii Nauk SSSR*, (244):1093–1096, 1979. (in Russian) English translation: Soviet Mathematics Doklady 20, 1, pp191-194, 1979.

[Knu81] D. E. Knuth. *The Art of Computer Programming, Vol 2, Seminumerical Algorithms.* Second Edition, Addison-Wesley, Reading, Massachusetts, 1981.

[LMS91] I. J. Lustig, R. E. Marsten, and D. F. Shanno. Interior method vs simplex method: Beyond netlib. *Mathematical Programming Society, Committee on Algorithms Newsletter*, (19):41–44, August 1991.

[MHL91] Dror E. Maydan, John L. Hennessy, and Monica S. Lam. Efficient and exact data dependence analysis. In SIGPLAN Notices, editor, *Proc. of the 1991 ACM SIGPLAN Conference on Programming Language Design and Implementation*, volume 26, pages 1–14, Toronto, Ontario, Canada, June 26-28 1991. ACM.

[Min83] M. Minoux. *programmation mathématique, théorie et algorithmes, tomes 1 et 2.* Dunod, 1983.

[NW88] G. N. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* Wiley-Interscience, 1988.

[Pug91] William Pugh. The omega test: a fast and practical integer programming algorithm for dependence analysis. In *Proceedings of 1991 International Conference on Supercomputing*, November 1991. To appear in Communications of the ACM.

[Sch86] Alexander Schrijver. *Theory of Linear and Integer Programming.* Wiley-Interscience, 1986.

[Sim62] B. Simonnard. *Programmation linéaire.* Dunod, Paris, 1962.

[SM89] Harvey M. Salkin and Kamlesh Mathur. *Foundations of Integer Programming.* North-Holland, 1989.

[Wal88] D. R. Wallace. Dependence of multi-dimensional array references. In *Proceedings of 1988 International Conference on Supercomputing*, Saint-Malo, France, July 1988.