Extension of the Revised Simplex Method for Detecting Implicit Equalities in Constraint Logic Programming

Philippe Refalo*
ILOG, S.A.
9 rue de Verdun - BP 85,
94253 Gentilly, France
refalo@ilog.fr

Abstract

This paper deals with the incremental detection of implicit equalities using the revised simplex method. This algorithm is more efficient and more suitable to practical problems than the tableau method that is usually applied in constraint logic programming. We describe and discuss the adaptation to the revised simplex of three approaches: the `CLP(`$\mathcal{R}$`)` approach, the `Prolog III` approach and the quasi-dual one. All of them have been integrated into the constraint logic programming language **`Athena`** based on a revised simplex method over exact-precision rationals. This system is used to compare these methods on a set of typical CLP problems over linear constraints.

keywords: constraint logic programming, incremental linear constraint solving, implicit equalities.

1 Introduction

Detecting implicit equalities is closely related to solving linear constraints in constraint logic programming (CLP). An implicit equality is an inequation $a_1x_1 + \cdots + a_nx_n \leq a_0$ from a set of constraints S that can be converted into the equation $a_1x_1 + \cdots + a_nx_n = a_0$ without changing the set of solutions of S. For instance in the set $\{x_1 + x_2 \leq 2, 1 \leq x_1, 1 \leq x_2\}$ all inequations are implicit equalities. Indeed this set can be rewritten to $\{x_1 + x_2 = 2, x_1 = 1, x_2 = 1\}$ without changing its set of solutions.

Detecting implicit equalities is one of the fundamental operations for solving generalized linear constraints [13] that include disequations ($\neq$) and strict inequations ($<$). This method for solving constraints is integrated into languages like `Prolog III` [5] and `Prolog IV` [7].

This detection enables the discovering of all variables fixed to a single value. This is applied to the delayed solving of linear constraints [6]. In languages like `CLP(`$\mathcal{R}$`)` [10] or `Prolog III` [5], solving the non linear constraint $2xy + 3z + t = 0$ is delayed until one of the variables x or y is fixed by the linear constraint solver. For instance, if x becomes fixed to 3, the constraint $6y + 3z + t = 0$ is then added to the linear constraint solver.

*A part of this work was done when the author was in Laboratoire d'informatique de Marseille, Faculte des sciences de Luminy, 13009 Marseille - France

Detecting implicit equalities is also usefull for cooperating constraint solvers that associate a constraint propagation based solver (on intervals of finite domains) with a linear constraint solver. The former gives bounds on variables to the latter while the latter maintains a rational solution of linear constraints and gives the former fixed variables [2, 7]. For instance in the set $\{x - y + z \leq 0,\ y - z \leq 0,\ 0 \leq x, x \leq 2, 0 \leq y, y \leq 2x \in \{0,1,2\}, y \in \{0,1,2\}\}$, the inequation $x \leq 0$ is an implicit equality. Consequently x is fixed to 0 and the new domain of y is $\{0,1\}$. This inference is not possible with standard finite domain constraint solvers.

Linear constraint solving mainly uses a Gaussian elimination method (for equations and disequations) and on the simplex method (for inequations[1]). Implicit equalities are incrementally detected by extensions of the simplex method. Among the first methods developed for this purpose, there are the `Prolog III` method [8, 5], the `CLP(`$\mathcal{R}$`)` method [16, 10] and the `CHIP` method [18], which is now also used in `Prolog IV` [1]. These methods are implemented with a tableau simplex algorithm over non-negative variables.

This work deals with the incremental detection of implicit equalities with the revised simplex method. This method is widely used for solving practical linear optimization problems [4, 3]. In comparison with the tableau simplex method, the revised simplex method has the advantages of being much more efficient (many optimizations are possible), more numerically stable (accuracy of computations can be controlled), and of using less memory (the rate of non-zero coefficients increases rapidly with the tableau simplex and not with the revised simplex).

The revised simplex method is applied to solving linear constraints in new generation CLP languages like `CLAIRE` [2] or `Athena` [15, 14]. Detection of implicit equalities is then based on the quasi-dual formulation of a system of inequations introduced by Lassez [11].

In this article we describe how to incrementally detect the presence of implicit equalities. We also detail adaptations to the revised simplex of the `Prolog III` method, the `CLP(`$\mathcal{R}$`)` method and the quasi-dual based method. The method using the lexicographic form [18] is not detailed here since it requires the simplex tableau to be applied. For each method, we provide the main theoretical results without providing proofs. They can be found in references which are given instead. We also provide the algorithm that identifies all implicit equalities.

These algorithms have been integrated to the CLP language `Athena`. This make quantitative comparison possible. The linear constraint solver of `Athena` is based on a revised simplex method over exact-precision rationals [15]. It includes advanced linear programming techniques such as LU factorization and smart pivoting techniques. This system is used to compare these approaches on a set of typical CLP problems over linear constraints.

The rest of this paper is organized as follows. Definitions and notations are given in Section 2. The solved form for linear constraints and the revised simplex method are presented in Section 3. Section 4 describes incremental linear constraint solving and detection of implicit equalities. Section 5 details the approach of `CLP(`$\mathcal{R}$`)`. The approach of `Prolog III` is described in Section 6, and the quasi-dual method is described in Section 7. Details about implementation and results from the practical comparison are given in Section 8.

2 Definitions and Notations

Let $\mathcal{R}$ be the set of real numbers and $\overline{\mathcal{R}} = \mathcal{R} \cup \{-\infty, +\infty\}$. Let V be an infinite and countable set of variables. A *linear term* is an expression α, αx or $t_1 + t_2$ where α is a real number, x is a variable from V and t_1 and t_2 are linear terms. A *linear constraint* is an expression $t_1 = t_2$ or $t_1 \leq t_2$ where

[1]A strict inequation is usually decomposed to an equation and a disequation

t_1 and t_2 are linear terms. These constraints are called respectively *equation* and *inequation*. A *constraint system* is a finite set of constraints. A variable has an *occurrence* in a term or constraint if it appears with a non zero coefficient in the term or constraint. Let S be a system of inequations. The system noted $S_=$ is composed of constraints of S where the symbol $\leq$ is replaced by $=$.

An *assignment* is a mapping from set V to set $\mathcal{R}$ that assigns a value to each variable of V. An assignment can be extended naturally to a mapping from the set of terms to $\mathcal{R}$. A *solution* of a constraint is an assignment σ such that σ is a solution of the constraint $t_1 \diamond t_2$ if $\sigma(t_1) \diamond \sigma(t_2)$ where $\diamond \in \{=, \leq\}$. An assignment is a solution of a constraint system if it is a solution of each constraint of the system. A system having at least one solution is *solvable*; otherwise it is *unsolvable*. A variable is *fixed* for a constraint system S if every solution of the system assigns the same value to the variable. An inequation $t_1 \leq t_2$ from a constraint system S is an *implicit equality* if every solution of S is a solution of $t_1 = t_2$.

In the following, matrix notations are used. Let A be an $m \times n$ matrix. The expression A_{i*} represents the i^{th} row of A, the expression A_{*j} represents the j^{th} column of A. Let $P \subset \{1, \ldots, m\}$, the expression A_P represents a submatrix of A whose columns are columns of A whose indexes are in P. The inverse of a square matrix M is denoted by M^{-1}.

If e is a row vector of n real numbers, x a column vector of n variables, b a column vector of m real numbers and β a real number, the expression $ex \leq \beta$ represents an inequality and the expression $Ax = b$ represents a system of equations. We implicitly assume compatibility of the sizes of matrices and vectors when they are multiplied. Matrices and sets are denoted by upper case letters, vectors by lower case letters and real numbers by Greek symbols.

3 Solved Forms and Revised Simplex

3.1 Tableau Solved Form

The tableau solved form is composed of a set of equations and a set of inequations that bound the value of variables. A system of linear constraints in tableau solved form is at first in *basic form*:

$$S = \left\{ \begin{array}{l} x_B = Mx_L + M'x_U + b \\ l \leq x \leq u \end{array} \right. \tag{1}$$

where x is a vector of variables, b a column-vector of reals, M and M' are matrices of reals, l and u are vectors of elements of $\overline{\mathcal{R}}$ and where B, L and U are disjoint sets of indexes such that $l_i \neq -\infty$, for $i \in L$ and $u_i \neq +\infty$, for $i \in U$. The set of indexes B is called the *basis* of the system. Variables from vector x_B are called *basic variables*. Variables from vectors x_L and x_U are called *non-basic variables*. The values l_i and u_i are respectively the *lower bound* and the *upper bound* of the variable x_i. For each basic form there exists a unique assignment σ_S such that

$$\sigma_S(x_i) = \left\{ \begin{array}{ll} l_i & \text{if } i \in L \\ u_i & \text{if } i \in U \\ \sigma_S(t) & \text{if } i \in B \text{ and } x_i \text{ is basic in } x_i = t \end{array} \right.$$

When this assignment is also a solution of S the system is in *tableau solved form* and σ_S is called the *basic solution* of S. It is well known that any solvable system of linear equations and inequations can be mapped to tableau solved form [4].

3.2 Revised Solved Form

The revised solved form is the basis for the revised simplex method. The main difference from the tableau solved form is the basic variables that are not eliminated in the equations. However eliminating basic variables gives a system in tableau solved form. A system in *revised solved form* is a system

$$S' = \begin{cases} A_B x_B + A_L x_L + A_U x_U = d \\ l \leq x \leq u \end{cases} \tag{2}$$

where A_B is a non-singular square matrix and such that the system

$$\mathcal{T}(S') = \begin{cases} x_B = -A_B^{-1} A_L x_L - A_B^{-1} A_U x_U + A_B^{-1} d) \\ l \leq x \leq u \end{cases}$$

where basic variables are eliminated, is in solved form. The matrix A_B of basic variables coefficients is called the *basic matrix* of system S'.

A *regular* system is a system in revised solved form that does not contain implicit equalities. The purpose of this article is to solve and maintain incrementally regular systems.

3.3 The Revised Simplex Method

The revised simplex method considers a linear program of the form

$$\min\{a x_L + a' x_U \mid S\}$$

The system S is in revised solved form and the term $a x_L + a' x_U$ is an objective function where basic variables have been eliminated. It computes a *solved* linear program (i.e. such that $a_i > 0$ and $a'_i < 0$). The basic solution of S is then an optimal solution of the linear program (see [4]).

The revised simplex method first chooses an *entering variable* x_k such that $a_i < 0$ or $a'_i > 0$. A variation of the value $\sigma_S(x_k)$ decreases the value of the objective function.

The method next chooses a *leaving variable* from basic variables in order to make room for the variable x_k entering the basis and to change the value of x_k. To choose the leaving variable, the vector d of coefficients of x_k in the system $\mathcal{T}(S)$ is computed:

$$d = -A_B^{-1} A_{*k} \tag{3}$$

This vector is used to obtain the variation α_i of x_k (in absolute value) allowed by the bounds of the basic variable x_i and by the equation where x_i is basic. The greatest possible variation for x_k is $\alpha_r = \min \alpha_i$. When $\alpha_r = -\infty$ the algorithm stops. The variation of x_k is unbounded and thus the linear program is also unbounded.

In other cases, the system and the objective function are pivoted over the entering variable x_k and the leaving variable x_r. The variable x_r reaches one of its bounds and its index is added to one of the sets L or U. The index of variable x_k leaves the set where it appeared (L or U) and is introduced in the set B. Let B', L' and U' be the new set of indexes after pivoting. The new basis matrix is $A_{B'}$ and the new objective function $f x_{L'} + f' x_{U'}$ where the basic variables are eliminated, is computed as follows:

$$\begin{aligned} f &= e_{L'} - e_{B'} A_{B'}^{-1} A_{L'} \\ f' &= e_{U'} - e_B A_{B'}^{-1} A_{U'} \end{aligned} \tag{4}$$

It may happen after a pivot that the value of the objective function for the basic solution has not changed. This is the case if $\alpha_r = 0$. The pivot is then *degenerated*.

The efficiency of the revised simplex method hinges around the representation of the matrix A_B^{-1} and the update of this matrix during a pivot. Some of the numerous approaches for this are summarized in [4]. The representation used in `Athena` is an LU factorization of the basis matrix and the update is based on matrix accumulation [15, 14].

4 Linear Constraint Solving

This section briefly presents some fundamental results for incremental solving of equation and inequations and the detection of implicit equalities. References containing more details are provided.

Linear constraint solving in constraint programming must be incremental. From a regular system S, and a constraint c, the basic operation is to compute a regular system S' equivalent to $S \cup \{c\}$.

4.1 Determining Solvability and the Presence of Implicit Equalities

Assume that the constraint c is an inequation $ex \leq \beta$. Determining the solvability of the system $S \cup \{c\}$ and the presence of implicit equalities in this system amounts to solving the linear program

$$\gamma = \min\{ex \mid S\}$$

with, for instance, the revised simplex method. The following three cases have to be considered:

- if $\gamma > \beta$ then $S \cup \{c\}$ is unsolvable ;
- if $\gamma < \beta$ then $S \cup \{c\}$ is solvable and does not contain implicit equalities;
- if $\gamma = \beta$ then $S \cup \{c\}$ is solvable and contains implicit equalities.

The absence of implicit equalities when $\gamma < \beta$ is proven in [13]. This fundamental result leads to efficient incremental methods for detecting implicit equalities. In other words, solving a system that does not contain any implicit equalities does not require anything more than searching for a solution of this system.

Observe that when $\gamma = \beta$, the inequation c is an implicit equality itself. Moreover, it is not the only one in the system $S \cup \{c\}$. Note also that iterations of the simplex method can terminate as soon as a solution is reached that gives the objective function a value strictly inferior to β.

If the constraint c is an equation $ex = \beta$, adding c is equivalent to adding the two inequations $ex \leq \beta$ and $\beta \leq ex$. The basic solution σ_S can often discard one of these two inequations. Indeed, if $\sigma_S(ex) < \beta$, it is sufficient to search for a solution of the inequation $\beta \leq ex$. Conversely, if $\sigma_S(ex) > \beta$, it is sufficient to search for a solution of the inequation $ex \leq \beta$. When $\sigma_S(ex) = \beta$, both inequations must be considered since each of them can be an implicit equality. However, since S does not contain any implicit equalities, if $ex \leq \beta$ is an implicit equality then the inequation $\beta \leq ex$ is not an implicit equality and vice versa.

The constraint is added to the system only if $\gamma < \beta$ (we will see later that it is not necessary to add it when $\gamma = \beta$). This problem have been studied for some time with the tableau solved form [10, 18, 9] and more recently with the revised solved form [2, 15, 14]. We will not go into further details herein since the purpose of this work is to describe the incremental computation of a regular system.

4.2 Determining All Implicit Equalities

Once the presence of implicit equalities is proven, the next step is to determine all implicit equalities of $S' = S \cup \{c\}$ and to replace them by equations to obtain the new system.

A naive approach is to solve a linear program for each inequation $\delta x_i \leq \alpha$ of the system S' to decide if it is an implicit equality or not. In [17] Telgen proposes some improvements to this method. He gives some syntactic criteria to identify some implicit equalities in a tableau solved form. For each system obtained by the application of the tableau simplex, the criteria identify additional implicit equalities. Moreover, the basic solution σ_S that is associated to each system is used to discard any inequation verifying $\sigma_S(\delta x_i) < \alpha$ (that is obviously not an implicit equality). It remains that in the worst case, this approach can solve n linear programs for n inequations in the system.

An ideal method would solve a single linear program for each implicit equality found. Most of the incremental approaches that are described herein are very close to the ideal method and sometimes better.

Each of the incremental method considers first the solved linear program proving the presence of implicit equalities in the system $S \cup \{ex \ \diamond \ \beta\}$ where $\diamond \in \{=, \leq\}$:

$$\min\{ax_L + a'x_U \mid S\} \tag{5}$$

This linear program contains informations for determining with certainty a first non empty set of implicit equalities. Let $l \leq x \leq u$ be the set of inequations of system S. The *restrictive inequations* of this linear program are the inequations of S that directly restrict the decrease of the objective function. They are characterized by the set

$$R = \{l_i \leq x_i \mid a_i \neq 0\} \cup \{x_i \leq u_i \mid a'_i \neq 0\}$$

As a fundamental result, each restrictive inequation of the solved linear program is an implicit equality in the system $S \cup \{ex \leq \beta\}$ (see [14]). This result is applied in most of the implementations of CLP languages over linear constraints.

It can be proved that the systems $S \cup \{ex \leq \beta\}$, $S \cup \{ex = \beta\}$ and $S \cup R_=$ are all equivalent (see [14]). Consequently, adding the constraint $ex \leq \beta$ or $ex = \beta$ amounts to fixing the variables appearing in $R_=$ to one of their bounds.

Each algorithm presented in the following considers a regular system S and a constraint $ax_L + a'x_U \leq \beta$ such that the linear program $\min\{ax_L + a'x_U \mid S\}$ is solved and $\sigma_S(ax_L + a'x_U) = \beta$. It returns a regular system S' equivalent to $S \cup \{ax_L + a'x_U \leq \beta\}$. It always detects implicit equalities as restrictive inequations of linear programs that are successively solved. The final system is thus built step by step by replacing each variable fixed to a value by this value.

5 Adaptation of the CLP($\mathcal{R}$) Method

The method described here is an adaptation of the incremental method developed in [16]. It has been integrated to the `CLP`($\mathcal{R}$) language [10] with the tableau solved form over non-negative variables.

5.1 Method Principles

The basic principle of this method is to consider each implicit equality found as an implicit equality in the solved linear program (5) and to add the corresponding equation in the system. If this

input : S a regular system and $ax_L + a'x_U \leq \beta$ an inequation
output : S' a regular system equivalent to $S \cup \{ax_L + a'x_U \leq \beta\}$
begin
1. let E be the set of restrictive inequations of $\min\{ax_L + a'x_U \mid S\}$
2. **repeat**
   a. remove a constraint $\delta x_i \leq \alpha$ from E and rewrite S to
   $$S \leftarrow \begin{cases} (S - \{l_i \leq x_i \leq u_i\}) \cup \{x_i \leq l_i\} & \text{if } \frac{\alpha}{\delta} = l_i \\ (S - \{l_i \leq x_i \leq u_i\}) \cup \{u_i \leq x_i\} & \text{if } \frac{\alpha}{\delta} = u_i \end{cases}$$
   b. solve the linear program $\gamma = \min\{\delta x_i \mid S\}$,
   c. let R be the new set of restrictive inequations
   d. **if** $\gamma = \alpha$ **then** $E \leftarrow E \cup R$
   e. $S \leftarrow S \cup \{x_i = \frac{\alpha}{\delta}\}$
   until $E = \emptyset$
3. **return** S

end

Figure 1: `CLP(`$\mathcal{R}$`)` Method

addition involves the apparition of implicit equalities, the new solved linear program that proves this is used to identify a part of them as restrictive inequations. The corresponding equations are added the same way. When no more inequations need to be reintroduced, the final system is regular.

More precisely, for each restrictive inequation $l_i \leq x_i$ found, the equation $x_i = l_i$ is added to the system to obtain a system equivalent to

$$S \cup \{x_i = l_i\}$$

This system is obviously solvable since $\sigma_S(x_i) = l_i$. It is only necessary to decide if it contains implicit equalities. These are searched for in one of the systems $S \cup \{x_i \leq l_i\}$ or $S \cup \{l_i \leq x_i\}$. Since S is regular the constraint $l_i \leq x_i$ is not an implicit equality in S and we have to decide if $x_i \leq l_i$ is an implicit equality in the system

$$S' = (S - \{l_i \leq x_i \leq u_i\}) \cup \{x_i \leq l_i\}$$

For this purpose the linear program $\min\{x_i \mid S'\}$ is solved (the case where the restrictive inequation is $x_i \leq u_i$ is similar).

If one of the inequations $x_i \leq l_i$ or $u_i \leq x_i$ is an implicit equality, the minimization computes a linear program in revised solved form that gives new implicit equalities as restrictive inequations. The whole process is then repeated for each of them.

5.2 Algorithm

The algorithm described in figure 1 begins by computing the set E of restrictive inequations of the linear program given as input (step 1). An inequation $\delta x_i \leq \alpha$ from E is chosen and the bounds on x_i are replaced by the inequation $\alpha \leq \delta x_i$ (step 2a). The expression γx_i is then minimized (step 2b). If $\min\{\delta x_i \mid S\} = \alpha$, the inequation $\delta x_i \leq \alpha$ is an implicit equality and the restrictive inequations of the solved linear program are added to E (step 2d). The variable x_i is then explicitly fixed in the system S (step 2e).

This algorithm has two main advantages for incrementality. Stuckey emphasizes in [16] that the number of steps of the algorithm, and thus the number of linear programs solved, is equal to the number of implicit equalities found. So this method is equivalent to the ideal method. Moreover, the minimization at step 2b for deciding the presence of implicit equalities in the system $S \cup \{\delta x_i \leq \alpha\}$ is initialized with a system whose basic solution satisfies the constraint $\delta x_i = \alpha$. Consequently, few simplex iterations are necessary to prove that $\min\{\delta x_i \mid S\} = \alpha$ or $\min\{\delta x_i \mid S\} < \alpha$. Assuming that there is no degeneracy, a single pivot is sufficient to decide. This is often the case in practice.

6 Adaptation of the `Prolog III` Method

The method presented herein was developed for the linear constraint solver of `Prolog III` [5], with the tableau solved form over non-negative variables. The method is briefly described in [8] ; it has not been detailed nor proven.

When the first implicit equalities found as restrictive inequations are replaced by equations, the new system found may contain more implicit equalities. The `Prolog III` methods identifies some constraints that, once removed from this system, lead assuredly to a regular system. Those constraints are then reintroduced one by one in this system. If during the introduction the presence of new implicit equalities is detected, other constraints are removed and reintroduced the same way later on.

6.1 Definitions and Properties

In the `Prolog III` linear constraint solver, the constraints removed are the equations that contain an occurrence of a variable that also appears in the objective function of the solved linear program proving the presence of implicit equalities.

A more efficient approach is to remove some saturated inequations called bounding inequations. A *saturated inequation* of a system S in revised solved form is an inequation $l_i \leq x_i$ such that $\sigma_S(x_i) = l_i$ or an inequation $x_i \leq u_i$ such that $\sigma_S(x_i) = u_i$.

Definition 1 (Bounding Inequation) *The* bounding inequations *of a solved linear program* $\min\{a x_L + a' x_U \mid S\}$ *are the saturated inequations over variables that are basic in an equation of* S *and whose right member shares an occurrence of variables with the objective function.*

When restrictive inequations are replaced by equations, the resulting system is not necessarily regular. At this point, the removal of bounding inequations leads with certainty to a different but regular system [14].

Theorem 1 *Let* P *be a linear program in solved form whose system* S *is regular. Let* R *be set of restrictive inequations of* P *and* S_B *be its set of bounding inequations. The system* $(S - S_B) \cup R_=$ *is regular.*

Computing the system S_B requires computing in $\mathcal{T}(S)$ the coefficients of variables that have an occurrence in the objective function. These are obtained by solving the system (3) for each variable x_k that has a non zero coefficient in the objective function.

input : S a regular system and $ax_L + a'x_U \leq \beta$ an inequation
output : S' a regular system equivalent to $S \cup \{ax_L + a'x_U \leq \beta\}$
begin
1. let R be the set of restrictive inequations of $\min\{ax_L + a'x_U \mid S\}$
2. let E be the set of bounding inequations of $\min\{ax_L + a'x_U \mid S\}$
3. $S \leftarrow S \cup R_=$
4. **repeat**
   a. remove a constraint $\delta x_i \leq \alpha$ from E and solve $\gamma = \min\{\delta x_i \mid S\}$
   b. **if** $\gamma = \alpha$
      then $S \leftarrow S \cup \{x_i = \frac{\alpha}{\delta}\}$
      else $S \leftarrow S \cup \{\delta x_i \leq \alpha\}$
   c. let R be the set of restrictive inequations of $\min\{\delta x_i \mid S\}$ solved
   d. let S_B be the set of bounding inequations of $\min\{\delta x_i \mid S\}$ solved
   e. $S \leftarrow (S - S_B) \cup R_=$
   f. $E \leftarrow E \cup S_B$.
   until $E = \emptyset$
4. **return** S
end

Figure 2: `Prolog III` Method

6.2 Algorithm

The algorithm described in figure 2 begins by initializing to empty the set E of restrictive inequations to reintroduce (step 1) and the set S_B of bounding inequations to remove(step 2). It then computes the system $(S - S_B) \cup R_=$ which is regular (step 3). An inequation $\delta x_i \leq \alpha$ is chosen in the set E and is reintroduced in S by minimizing the term δx_i to decide if it is an implicit equality (step 4a). If so, the variable x_i is fixed to the value $\frac{\alpha}{\delta}$, otherwise the inequation is reintroduced in the system (step 4b). Then the algorithm continues until there are no more inequations to reintroduce.

Despite the improvement proposed, this algorithm can remove at each step as many inequations as there are equations. Reintroducing these constraints requires solving the same number of linear programs. This method is thus weakly incremental. However, the basic solution of the system S at step 2e is a solution of the equation $\delta x_j = \alpha$ and, as in the `CLP(`$\mathcal{R}$`)` method, the minimization requires few iterations in practice. Here again, a single non-degenerate pivot is sufficient to prove that the inequation is not an implicit equality.

In conclusion this method seems less efficient than the `CLP(`$\mathcal{R}$`)` method that solves only a linear program for each implicit equality found. Both methods are unable to prove in one step the absence of implicit equalities in the system $S \cup R_=$. This is possible with the quasi-dual method.

7 Adaptation of the Quasi-dual Formulation

The quasi-dual formulation of a system of linear constraints was introduced by Lassez [12]. The use of this formulation for detecting implicit equalities is suggested in [11] among other possible applications. This approach has been integrated to the `CLAIRE` language [2] and to the `Athena` language [15].

Let $S = \{Ax \leq b\}$ be a system of m inequations. The quasi-dual formulation of this system is the linear program

$$\eta = \min\{yb \mid yA = 0, y \geq 0, \sum y_i = 1\}$$

where y is a row-vector of m variables. If $\eta = 0$ then the system S contains implicit equalities otherwise it does not. With this formulation, the absence of implicit equalities is decided by solving a single linear program. We consider here the dual of the quasi-dual, i.e. the linear program

$$\mu = \max\{x_0 \mid Ax + u.x_0 \leq b, x_0 \geq 0\}$$

where u is a vector of m coefficients, all of them set to 1. If these two linear programs are bounded, the well-known relation $\eta = \mu$ holds. As a consequence, the dual of the quasi-dual can also be solved to detect implicit equalities. This last formulation is more interesting since it involve fewer modifications of the original system S.

In the following, we introduce the (tableau or revised) auxiliary form that is an extension of the dual of the quasi-dual formulation to the (tableau or revised) solved form. The algorithm presented here computes a system in revised auxiliary form from a system in revised solved form by introducing a new variable in some equations. That variable is then minimized. If the optimal value is zero, then the system contains implicit equalities that are identified as restrictive inequations of the solved linear program. Those inequations are replaced by the corresponding equations and the whole process is repeated for the new system to find more implicit equalities or to prove that there are no more.

7.1 Definitions and Properties

Computing the auxiliary form of a tableau solved form S consists of introducing a new variable x_0 in each equation whose basic variable x_b is assigned to one of its bounds in the basic solution. The variable x_0 is not introduced in every constraint as in the dual of the quasi-dual formulation because the basic solution of S can discard a priori some inequations that are not implicit equalities (inequations $l_i \leq x_i$ or $x_i \leq u_i$ such that $l_i < \sigma_S(x_i)$ or $\sigma_S(x_i) < u_i$).

To clarify this presentation, the adaptation is first presented with the tableau solved form and then with the revised one.

Definition 2 (Tableau Auxiliary Form) *Let S be a system in tableau solved form (1). The* tableau auxiliary form *of S is the system*

$$S_A = \begin{cases} x_B = w.x_0 + Mx_L + M'x_U + b \\ l \leq x \leq u \\ 0 \leq x_0 \leq +\infty \end{cases}$$

where $\sigma_{S_A}(x_0) = 0$ and w is a vector such that

$$w_i = \begin{cases} 0 & \text{if } l_i < \sigma_S(x_i) < u_i \\ 1 & \text{if } \sigma_S(x_i) = l_i \\ -1 & \text{if } \sigma_S(x_i) = u_i \end{cases}$$

Note that S_A is in revised solved form since its basic solution corresponds to the basic solution of S on all variables except on x_0 which is assigned to zero.

input : S a regular system and $ax_L + a'x_U \leq \beta$ an inequation
output : S' a regular system equivalent to $S \cup \{ax_L + a'x_U \leq \beta\}$
begin
  1. let E be the set of restrictive inequations of $\min\{ax_L + a'x_U \mid S\}$
  2. **repeat**
    a. $S \leftarrow S \cup E_=$
    b. let S_A be the revised auxiliary system of S and x_0 be the var. introduced.
    c. **if** $\min\{-x_0 \mid S_A\} < 0$
      then return S.
      else let E be the set of restrictive inequations of $\min\{-x_0 \mid S_A\}$ solved.
  until false
end

Figure 3: Quasi Dual Method

Definition 3 (Revised Auxiliary Form) *Let S' be a system in revised solved form (2). The* revised auxiliary form *of S' is*

$$S'_A = \begin{cases} Ax_B + w'.x_0 + A'x_L + A''x_U = 0 \\ l \leq x \leq u \\ 0 \leq x_0 \leq +\infty \end{cases}$$

where x_0 is a variable assigned to zero in the basic solution of S'_A and w' is a vector such that $w' = -A_B w$ where w is the vector from definition 2.

Assume that $S = \mathcal{T}(S')$ then $S_A = T(S'_A)$. The vector w' is thus the appropriate vector to introduce in S'_A to verify this relation. The solving of a single linear program can determine the presence or the absence of implicit equalities in the system S'.

Theorem 2 *With notations of definition 2, the system S' contains implicit equalities if and only if* $\min\{-x_0 \mid S'_A\} = 0$.

This result leads to the incremental algorithm of figure 3.

7.2 Algorithm

This algorithm is rather simple. It identifies first the restrictive inequations of the solved linear program (step 1). It fixes the variables appearing in those inequations (step 2a). The auxiliary revised form is computed (step 2b) and the linear program of theorem 2 is solved with the simplex method (step 2c). If the optimal value is strictly negative, the system S is regular. Otherwise new restrictive inequalities are found and the algorithm continues. Note that the iterations of the simplex can be stopped as soon as the term $-x_0$ can be assigned to a strictly negative value for the basic solution.

In comparison with the previous methods, the quasi-dual method has many advantages. The computation of the auxiliary form is simple and proving the absence of implicit equalities requires solving a single linear program. In practice this proof needs few simplex iterations since the term $-x_0$ is already assigned to zero in the basic solution of the auxiliary forms.

		Quasi-dual meth.				Prolog III meth.				CLP($\mathcal{R}$) meth.			
Problems	Nb	Time	Piv	Step	IEPiv	Time	Piv	Step	IEPiv	Time	Piv	Step	IEPiv
Square 9	933	4360	2826	607	2826	4850	2756	174	2756	4680	2618	410	2618
Square 14	4293	42930	18485	2939	18485	47120	17631	861	17631	45610	16886	1435	16886
Transport	80	190	125	1	0	260	125	0	125	280	125	80	0
Donald	46	780	179	52	171	800	145	28	137	770	127	46	119
Periodic	39	110	96	46	96	130	72	17	72	120	67	23	67
Cut-stock	245	5000	909	207	887	5020	693	124	671	4650	587	245	565

Figure 4: Comparison of all Methods

8 Practical Results

The three approaches above have been integrated to the CLP language `Athena`. The linear constraint solver of `Athena` is based on a revised simplex method on exact-precision rationals. It includes advanced linear programming techniques and an efficient procedure for dynamic backtracking [15].

Various examples have been tested with the three approaches. Most of these examples are typical CLP problems over linear constraints that need to detect implicit equalities for handling disequations.

The `Donald` problem is the well-known cryptarithm problem [5]. `Periodic` solves the problem of proving that a mathematical sequence is periodic [5]. It contains a single disequation, however finding implicit equalities reduces the search space. The `Square` problem [5] is a two dimensional placement problem that is very combinatorial. It consists of filling a rectangle with squares of different sizes such that no squares overlaps. All pivoting operations done to solve this problem are for detecting implicit equalities. Two instances of this problem are solved: one with 9 squares, the other with 14 squares. `Transport` [19] is a deterministic problem of assigning providers to customers. At the optimum value all of the 80 variables are fixed showing that there is no other optimal solution. `Cut-stock` is a cutting stock problem solved to optimality with a branch and bound approach [19]. All these problems are non deterministic except for `Transport`.

Figure 4 shows the number of implicit equalities (**Nb**) to be found or each problem. For each method, it shows the computation time[2] to solve the problem (**Time**), the total number of pivotings required (**Piv**), the number of steps of the algorithm (**Step**) which equals the number of linear programs solved to detect implicit equalities, and the number of pivoting needed to solve those linear programs (**IEPiv**).

Results in this table show that no method is clearly superior to the others. The quasi-dual method seemed to be the best one and to be well suited to the revised simplex, but it is not far better that the others. On average it is 12% faster than the `Prolog III` method and 10% faster than the `CLP(`$\mathcal{R}$`)` method. However, it can do many more iterations than the others, since fewer implicit equalities are discovered at each step. The exception is the `Transport` problem where the implicit equalities are all found at one time as restrictive inequations and the quasi-dual method proves that there are no more of them in only one step.

The `Prolog III` method performs much less iteration than the other methods. The advantage is negated by the need to compute columns of the tableau solved form to determine bounding inequalities. This result is surprising since it is the worst method in terms of complexity.

The `CLP(`$\mathcal{R}$`)` method is slightly faster than the `Prolog III` methods. Note that it does not

[2] In milliseconds on a Sun Sparc 10

perform as many iterations as there are implicit equalities because transforming a restrictive inequation into an equation sometimes leads to a failure and consequently no linear program is solved for the remaining restrictive inequations. The performance of this method lies between the `Prolog III` and the quasi-dual methods.

9 Conclusion

This paper has described how to detect implicit equalities with the revised simplex algorithm. This algorithm is more efficient and more suitable to practical problems than the tableau method that is usually applied in CLP. We have described the adaptation to this simplex of three main approaches for the incremental detection of implicit equalities: the CLP($\mathcal{R}$) approach, the `Prolog III` approach and the quasi-dual one.

A practical comparison with the CLP language `Athena` was done on some CLP problems. It shows that all these methods are almost equivalent from the efficiency standpoint. The quasi-dual method, which was considered the best method for the revised simplex, is not much faster that the others while the `Prolog III` method, which was considered computationally too costly, is the one that performs the least steps.

As a future study it would be interesting to combine the `Prolog III` and the quasi-dual methods in order to use the quick detection of the absence of implicit equalities of the quasi-dual method and the few number of iterations of the `Prolog III` method.

References

[1] Frédéric Benhamou and Touraïvane. Prolog IV: langage et algorithmes. In *JFPL'95: IVèmes Journées Francophones de Programmation en Logique*, pages 51–65, Dijon, France, 1995. Teknea.

[2] H. Beringer and B. de Backer. Combinatorial problem solving in constraint logic programming with cooperating solvers. In *Logic Programming : Formal Methods and Practical Applications.* Elsevier Science Publishers, 1994.

[3] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization.* Athena Scientific, Belmont, Massachusetts, 1997.

[4] V. Chvatal. *Linear Programming.* W.H. Freeman and Company, New York, 1983.

[5] A. Colmerauer. An introduction to Prolog III. *Communications of the ACM*, 33(7):69–91, July 1990.

[6] A. Colmerauer. Naive resolution of non-linear constraints. In Frederic Benhamou and Alain Colmerauer, editors, *Constraint Logic Programming: Selected Research*, pages 89–112. MIT Press, 1993.

[7] A. Colmerauer. Spécifications de Prolog IV. Technical report, Laboratoire d'Informatique de Marseille, 1996.

[8] Michel Henrion. Les algorithmes numériques de Prolog III. Technical report, Prologia, 1989.

[9] J.-L. Imbert and P. van Hentenryck. On the handling of disequations in CLP over linear rational arithmetic. In Frederic Benhamou and Alain Colmerauer, editors, *Constraint Logic Programming: Selected Research*, pages 49–72. MIT Press, 1993.

[10] J. Jaffar, S. Michaylov, P. Stuckey, and R.H.C. Yap. The CLP($\mathcal{R}$) language and system. *Transactions on Programming Languages and Systems*, 14(3), July 1992.

[11] J-L Lassez. Parametric queries, linear constraints and variable elimination. In *DISCO 90, LNCS 429*, pages 164–173. Springer-Verlag Lecture Notes in Computer Science, 1990.

[12] J-L Lassez. Querying constraints. In *Proceedings of the ACM Conference on Principles of Database Systems*, Nashville, 1990.

[13] J-L Lassez and K. McAloon. A canonical form for generalised linear constraints. *Journal of Symbolic Computation*, (1):1–24, 1992.

[14] P. Refalo. *Resolution et implication de contraintes lineaires en programmation logique par constraintes.* Ph.D Thesis. Laboratoire d'informatique de Marseille, Marseille, 1997.

[15] P. Refalo and P. van Hentenryck. `CLP(`$\mathcal{R}_{lin}$`)` revised. In *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 1–14, Bonn, Germany, 1996.

[16] P. J. Stuckey. Incremental linear constraint solving and implicit equalities. *ORSA Journal of Computing*, 3(4):269–274, 1991.

[17] J. Telgen. *Redundancy and Linear Programs.* Mathematical Centre Tracts, 1981. number 137.

[18] P. van Hentenryck and T. Graf. Standard forms for rational linear arithmetics in constraint logic programming. In *The International Symposium on Artificial Intelligence and Mathematics*, Fort Lauderdale, Florida, January 1990.

[19] P. van Hentenryck and V. Ramachandran. Backtracking without trailling in `CLP(`$\mathcal{R}_{lin}$`)`. *ACM Transaction on Programming Langages and Systems*, 1(1):0–0, 1995.