Ask & Tell Linear Constraints in Constraint Logic Programming

Philippe Refalo
LIM, URA CNRS 1787
Faculte des sciences de Luminy
13009 Marseille, France
refalo@lim.univ-mrs.fr

Abstract

This article adresses the problem of linear constraint entailment in constraint logic programming. After a brief overview of linear constraint solving, two forms for ask constraints are introduced that simplifies the entailment test. The incremental algorithms are also described and has been integrated to a CLP language. Practical experiments with operation research problems show that substantial reductions of the search tree and of the computation time are obtained.

Keywords : constraint entailment, linear constraints, constraint logic programming, concurrent constraint language.

1. Introduction

Constraint entailement is the basis of new generation constraint logic programming languages (CLP) such as `cc(FD)` [vHSD99] or `CLP(BNR)` [BO96]. It is also the fundamental aspect of concurrent constraint languages [Sar93] which consider (1) a *constraint store* that is monotonicaly increasing by the addition of *tell* constraints, and (2) a set of *ask* constraints. Goals are associated to ask constraints and are delayed until every solution of the constraint store is a solution of the ask constraint (i.e., the ask constraint is *entailed*).

Ask constraints arise naturaly form the use of high level operators such as the *implication operator* $c \rightarrow B$, where the goal B is delayed until the ask constraint c becomes entailed. Other high level operator has been integrated in CLP languages such as the *cardinality operator* [vHSD99], the `if_then_else` operator of `CHIP` [DvHS88] and the possibility to associate a boolean value to the satisfiability of the constraint in `CLP(BNR)`. They are all operationaly equivalent to a conjunction of implication operators. Many applications benefit from these operators; their use simplifies the problem statement and achieve also an important pruning of the search tree for some combinatorial problems.

Essentialy, constraint entailment has been used over finite domains and interval constraints. In these domains, the approximate but sound algorithms provide an efficient but incomplete entailment test. Linear constraints, althought integrated in first generation constraint programming languages such as `Prolog III` [Col90], CLP($\mathcal{R}$) [JMSY92] and `CHIP`, has not really been considered for entailment. The existing theoretical works on this subject [Las90a, Las90b] did not lead to algorithms usable in a CLP language. An application of linear constraint entailment in `CHIP` have been presented in [TGZ90] but the algorithms used are incomplete and were not described.

The purpose of this article is to adress the problem of an efficient algorithm for linear constraint entailment in order to define high-level operators over linear constraints. The associated problem of deciding if no solution of the constraint store is solution of an ask constraint (*disentailment*) is fully symmetric to the entailment problem in our case and it will not be discussed here.

The starting point of this study is the canonical form introduced by [LM89] and the solved forms for linear constraints used in the languages mentioned above. We introduce a *dereferenced form* and a *partial precise form* for linear ask constraints and we show how to maintain incrementally ask constraints in these forms. We also state several important properties of these forms for entailment and we describe the entailment algorithm for each type of constraints. These algorithms have been implemented in the CLP language `Athena` [RH96] and we present practical results showing that our algorithm improves the naive entailment test (i.e., S entails c iff $S \cup \{\neg c\}$ is unsolvable) and that a substantial reduction of the search space and of the computation time is achieved in comparison with standard backtracking.

The rest of this paper is organised as follows: Section 2 introduces definitions and notations, our normal form for linear constraints and the constraint addition agorithm. This section helps to introduce the properties and algorithms for constraint entailment of Section 3. Section 4 presents practical results and Section 5 concludes the paper.

2. Linear Constraint Solving

2.1. Definitions and Notations

Let V be an infinite and countable set of variables, a *linear term* is an expression of the form α, x, αx, $t_1 + t_2$, $t_1 - t_2$ where $\alpha \in \mathcal{R}$, $x \in V$ and the expressions t_1 and t_2 are linear terms. A *linear constraint* is an expression of the form $t_1 = t_2$, $t_1 \leq t_2$, $t_1 < t_2$ or $t_1 \neq t_2$, where t_1 and t_2 are linear terms. These constraints are called respectively *equation*, *inequation*, *strict inequation* and *disequation*. A *constraint system* is a finite set of constraints. A variable has an *occurrence* in a constraint if it appears with a non-zero coefficient in that constraint.

An *assignment* is a mapping $\sigma : V \mapsto \mathcal{R}$ which associates a real number with each variable of the set V. A mapping is extended to linear terms in the natural way. An assignment σ is a *solution* of a linear constraint $t_1 \diamond t_2$ if $\sigma(t_1) \diamond \sigma(t_2)$. An assignment is a solution of a constraint system if it is a solution of every constraint of the system. A constraint system having at least one solution is *solvable* otherwise it is *unsolvable*.

A constraint is *trivial* if it is of the form $\alpha \diamond \beta$, where α and β are real numbers. An inequation $t_1 \leq t_2$ of a constraint system S is an *implicit equality* if every solution of S is a solution of the equation $t_1 = t_2$. A variable x is *fixed* in a constraint system S if every solution of S assigns x to the same value.

To simplify the presentation, matrix notations are used; let A be a matrix of real numbers, the expression A_i denotes the i^{th} row of A, the expression $A_{.j}$ denotes the j^{th} column of A and the expression A_{ij} denotes the i^{th} entry of the j^{th} column of A. If $N = \{n_1, \ldots, n_k\}$ is an index set the matrix A_N is the matrix composed of the columns $A_{n_1}, \ldots, A_{n_k}$. A *vector* is a matrix having one column. If A is an $m \times n$ matrix, a a m dimensional vector of real numbers and x a m dimensional vector of variables and β a real number, $ax \leq \beta$ is an inequality and $Ax = a$ is a set of equations. We implicitly assume compatibility of the size of matrices and vectors when they are multiplied. In the following, matrices are denoted by upper case letters, vectors by lower case letters and real numbers by Greek symbols.

¿From a geometric standpoint, if Z is a subspace of the d-dimensional euclidean space $\mathcal{R}^d$ and if p is a vector in $\mathcal{R}^d$, the translation $p + Z$ is called an *affine space*. The intersection of all affine spaces that contain a set of points X is also an affine space called the *affine hull* of X and is denoted by *Aff(X)*. If $ax = \beta$ is an equality, the set of points $H = \{x \mid ax = \beta\}$ is called an *hyperplane*. An hyperplane is also an affine space. The *supporting hyperplane* of a constraint $ax \diamond \beta$ where $\diamond \in \{=, \leq, <, \neq\}$ is the hyperplane $H' = \{x \mid ax = \beta\}$ and the set of points $P = \{x \mid Ax \leq b\}$ is called a *polyhedron*.

2.2. Normal Form

The normal form for generalized linear constraints is a system of equations and disequations over possibily non-negative variables which is defined as follows:

Definition 1 (Normal Form) *A linear constraint system S is in* normal form *if it can be partitioned into 4 sets E_A, E_S, D and P*

$$S = \left\{ \begin{array}{ll} x_J = Mx_K + M'x_N + m & (E_A) \\ x_B = Ax_N + b & (E_S) \\ d \neq Cx_K + C'x_N & (D) \\ x_N \geq 0,\ x_B \geq 0 & (P) \end{array} \right. \tag{1}$$

such that

- *K, J, B and N are distinct sets of index variables,*
- *$E_S \cup P$ is solvable and does not contain implicit equalities,*
- *D does not contain the constraint $0 \neq 0$.*

Variables x_b, $b \in J \cup B$ are called *basic* variables, variables x_i, $i \in K \cup N$ are called *non-basic* variables, variables x_j, $j \in J \cup K$ are called *arbitrary* variables while variables x_k, $k \in B \cup N$, that can be assigned to positive values only, are called *slack* variables. This normal form has two fundamental properties:

- A system of linear constraints is solvable if and only if it can be mapped to normal form.
- Every fixed variable is explicit, i.e. it appears in an equation of the form $x_i = \alpha$.

The proof of these result can be found in [LM89].

3. Constraint Addition to Normal Form

Adding a constraint to normal form requires the decomposition of every constraint to a set of equations and disequations over arbitrary and slack variables. An inequation $ex + \gamma \leq 0$ is introduced by creating a new slack variable x_s (such that $x_s \geq 0$) and by adding the equation $ex + x_s + \gamma = 0$. A strict inequation is similarly decomposed in $ex + x_s + \gamma = 0$ and $x_s \neq 0$, where x_s is a new slack variable.

The method for incrementaly adding an equation c of the form $ex + \gamma = 0$ to a system in normal form is as follows :

1. [**Dereference:**] Replace every basic variable of S in the equation by its right member to obtain
$$ax_K + a'x_N + \beta = 0$$
with $a = e_K + e_J \times M$, $a' = e_N + e_J \times M' + e_B \times A$ and $\beta = \gamma + e_J \times m + e_B \times b$. If this equation is trivial the system $S \cup \{c\}$ is solvable if and only if $\beta = 0$.

2. [**Addition:**] Let $V_K = \{i \mid a_i \neq 0\}$ and $V_N = \{i \mid a'_i \neq 0\}$, if $V_K \neq \emptyset$, choose an index i in V_K otherwise choose i in V_N and rewrite the equation above in c' of the form
$$x_i = ux_K + u'x_N + \alpha$$
by isolating x_i. Replace x_i by the right member of c' in S. If a disequation becomes $0 \neq 0$ then $S \cup \{c\}$ is not solvable, otherwise add c' and update the sets J,K,B and N accordingly.

3. [**Simplex & Implicit Equalities:**] If $V_K = \emptyset$ then determine the solvability of $E_S \cup \{c'\}$ and the set I of equalities that are implicit in this system. If $E_S \cup \{c'\}$ is not solvable then $S \cup \{c\}$ is not solvable, otherwise repeat steps 1 and 2 for every equation of I.

Adding a disequation is essentialy similar, only the first step need to be applied. If the disequation is not trivial it is simply added to the system and the extended system is solvable.

For determining the solvability of the system $E_S \cup \{c'\}$ and for detecting implicit equalities, the system E_S is maintained in *simplex solved form* without implicit equalities. Constraints over slack variables only are incrementaly added to this system with the simplex algorithm which is also used to incrementaly detect new implicit equalities.

3.1. Simplex Solved Form

This section describes the simplex solved form, the simplex algorithm, and the incremental addition of a constraint to this form.

Definition 2 (Simplex Solved Form) *A system of linear equations over slack variables is in* simplex solved form *if it has the form*
$$x_B = Ax_N + b \tag{2}$$
and if, for every i, $b_i \geq 0$.

A system of linear equation over slack variables is solvable if and only if it can be mapped to this form (see [Chv83]). A solution of such a system can easily be constructed. Consider an assignment σ such that

$$\sigma(x_i) = \begin{cases} b_k & \text{if } i \in B \text{ and } x_i \text{ is the basic variable of constraint } k \\ 0 & \text{otherwhise} \end{cases}$$

This assignment is clearly a solution of 2 and is called the *basic solution* of this system. Note that the value of this assignment for a linear term of the form $ax_N + \beta$ is simply β.

3.1.1. The simplex algorithm

The simplex algorithm minimizes an objective function $ax_N + \beta$ subject to a set of constraints over slack variables $x_B = Ax_N + b$. The algorithm moves from a solved form to another until the objective function cannot be decreased or until it proves that the objective function is not bounded. It can be decomposed in three steps.

The first step consists of choosing a variable whose value can be increased to decrease the value of the objective function. This variable is chosen from the set

$$X_E = \{x_i \mid a_i < 0\}$$

If the set X_E is empty then the basic feasible solution is *optimal.*

The second step consists of choosing a basic variable to leave the basis in order to make room for the variable x_e entering the basis. This leaving variable is chosen to maintain the solved form. The idea is to determine how much x_e can vary while satisfying the equation and the non-negativity constraints. The variation allowed by constraint i is given by

$$v_i = \begin{cases} -\frac{b_i}{A_{ie}} & \text{if } A_{ie} < 0 \\ \infty & \text{otherwise} \end{cases}$$

in addition the variation allowed on x_e is

$$\alpha = \min(v_1, \ldots, v_n)$$

If $\alpha = \infty$ the algorithm terminates since the objective function is not lower bounded. Otherwise choose the leaving variable form the set

$$X_L = \{x_i \mid \alpha = v_i\}$$

The last step of the algorithm consist of constructing a new solved form $x_{B'} = A'x_{N'} + b'$ and a new objective function $a'x_{N'} + \beta'$ such that $B' = (B - \{r\}) \cup \{e\}$ and $N' = (N - \{e\}) \cup \{r\}$. This operation can be performed with gaussian elimination. Note that there are various heuristics to select the entering and the leaving variable from the sets X_E and X_L (see [Chv83]).

3.1.2. Constraint Addition to Simplex Solved Form

Let S be a system in simplex solved form $x_B = Ax_N + b$ whose basic solution is σ and let c be an equation of the form $ex + \gamma = 0$. The incremental addition of c to S consists of three steps: (1)

eliminating the basis variables from $ex+\gamma = 0$ to obtain the equation $ax_N+\beta = 0$; (2) checking the solvability of the system $S \cup \{c\}$ and (3) constructing an extended system in solved form equivalent to $S \cup \{c\}$.

The first step is easy to perform. Note that when to equation is dereferenced with respect to the simplex solved form, we have $\sigma(ax_N + \beta) = \beta$. For the second step, consider the general case where the basic solution of S is not a solution of c, we have $\beta > 0$ (resp. $\beta < 0$). For the second step, the simplex algorithm can determine wether the extended system has a solution or not. Minimizing the objective function $ax_N + \beta$ (resp. $-ax_N - \beta$) subject to the constraints of S leads to one of the three following results:

- the objective function is not bounded and the extended system is solvable;
- the objective function is bounded and the optimal solution satisfies $ax_N + \beta \leq 0$ (resp. $-ax_N - \beta \leq 0$), the extended system is clearly solvable.
- the objective function is bounded and the optimal solution satisfies $ax_N + \beta > 0$ (resp. $-ax_N - \beta > 0$), the extended system is clearly unsolvable.

However, it is possible to do better, the simplex algorithm can be stopped as soon as the entering variable x_e can be assigned to a value that gives the zero value to the objective function. More precisely, when an entering variable x_e is chosen in the actual objective function $ux_N + \lambda$ such that the possible variation α is infinite or satisfies

$$\lambda + \alpha u_e \leq 0$$

The extended system is then constructed by introducing the variable x_e into the basis of the new constraint. This new system is

$$x_{B'} = M'x_{N'} + b'$$

such that $B' = B \cup \{e\}$ and $N' = N - \{e\}$. Here again, this operation can be performed by gaussian elimination.

A modification of the simplex algorithm is also used for finding implicit equalities. Our method is based on the quasi-dual formulation of the set $E_S \cup P$ [Las90a]. A detailed description of this method is beyond the scope of this paper and is not necessary to understand the following. See [RH96] for a complete description of the algorithms and for the proofs of the results presented in this section.

4. Linear Constraint Entailment

Recall, first the basic defintion of constraint entailment. A system S entails a constraint c, denoted by $S \models c$, if every solution of S is a solution of c. The general problem of constraint entailment in constraint logic programming can be defined as follows. Let S be a system in normal form and $Q = \{c_1, \ldots, c_n\}$ be a set of *ask* constraints such that for every constraint c_i, we have $S \not\models c_i$. When a constraint t is added to the system S, leading to a new system in normal form S', the problem is to determine the subset Q_E of Q satisfying $c_i \in Q_E$ if and only if $S' \models c_i$.

The relation $S \models c$ is equivalent to the unsolvability of $S \cup \{\neg c\}$ so we consider a first method, called the *naive method*. This method determines the ask constraints that are entailed by determining the solvability of the systems $S' \cup \{\neg c_1\}$, $S' \cup \{\neg c_2\}, \ldots, S' \cup \{\neg c_n\}$ each time a constraint t is added to the initial system S. In the case of linear constraints, the mapping to normal form is the easiest way to do this.

Another approach, called the *incremental method*, is to maintain n systems T_i, one for each constraint c_i of Q, such that T_i is the system $S \cup \{\neg c_i\}$ in normal form. The new constraint t is then added to the system S and if $S \cup \{t\}$ can be mapped to normal form, t is also added to each of the n systems T_i with the algorithm described above. The set of those systems T_i that are unsolvable determine immediately the set of constraints that are entailed.

Practicaly, the naive method is less efficient than the incremental method since the latter reuse the previous computations. On the other hand, the incremental method, is unacceptable in terms of space complexity since, having n ask constraints, we have to maintain not only $n+1$ systems in normal form but also additional information on each system to support backtracking. So we have to consider an intermediate method. Note that the constraint addition algorithm in Section 2.2. always start by dereferencing the basic variables of the constraint to be added before performing additional operations to determine solvability. Our method is to maintain incrementally the constraints of the set Q without any occurence of basic variables of the system S. This implies several important properties for deciding entailment that are described in the following sections for each type of constraint.

4.1. Dereferenced Form

Definition 3 (Dereferenced Form) *A linear constraint is in* dereferenced form *for a system in normal form 1 if this constraint is without any occurrence of a basic variable, i.e. if it has the form* $ax_K + a'x_N + \beta \diamond 0$ *with* $\diamond \in \{=, \leq, <, \neq\}$.

A constraint in dereferenced form involving only slack variables is called a *slack constraint*, otherwise it is called an *arbitrary constraint*

Ask constraints can be incrementaly mapped to dereferenced form. Consider a system S in normal form and a set of ask constraints Q such that every constraint of Q is in dereferenced form for S. When a new equation is added to S it is sufficient to dereference the basic variable of that equation in each constraint of Q to obtain the new system of ask constraints in dereferenced form.

The following property determines the non-entailment of a constraint with respect to the occurence of an arbitrary variable in that constraint.

Property 1 *Let S be a constraint system in normal form, and c be an equation, an inequation or a strict inequation in dereferenced form for S. If c contains an occurrence of an arbitrary variable then S does not entail c.*

Proof: Let S be a constraint system in normal form 1 and let σ_1 be a solution of S over variables x_B and x_N, this can be the basic solution of the system $E_S \cup P$ in simplex solved form. Let c be of the form $ax_K + a'x_N + \beta \diamond 0$ with $\diamond \in \{=, \leq, <\}$, if c contains an occurrence of an arbitrary variable then it is always possible to extend the assignment σ_1 to an assignment σ_2 over variables x_K, x_B and x_N that is a solution of $ax_K + a'x_N + \beta < 0$ and, hence, not a solution of c. Consider an index $k \in K$ such that $a_k \neq 0$, a real number ε such that $\varepsilon > 0$ and the real number $\alpha = \sigma_1(a'x_N + \beta)$, the assignment σ_2 can be constructed as follows:

$$\sigma_2(x_i) = \begin{cases} \sigma_1(x_i) & \text{if } i \in B \cup N \\ 0 & \text{if } i \in K \text{ and } i \neq k \\ -\frac{\alpha}{a_k} + \varepsilon & \text{if } i = k \text{ and } a_k > 0 \\ \frac{\alpha}{a_k} + \varepsilon & \text{if } i = k \text{ and } a_k < 0 \end{cases}$$

This assignment can be extended naturaly over variables of x to a solution of $S \cup \{\neg c\}$. Consequently, if c contains an occurrence of an arbitrary variable then we have $S \not\models c$. □

The next property hold for entailment with systems of linear equations in normal form.

Property 2 *Let S be a system of linear equations in normal form and c be a constraint in dereferenced form for the system S, we have $S \models c$ if and only if the constraint c is trivial and solvable.*

Proof: Let S be a system of equations in normal form and c be a constraint in dereferenced form for S. Suppose that $S \models c$ and that c is not trivial. This means that every solution of S is a solution of c and that c has the form $ax + \beta \diamond 0$, where $\diamond \in \{=, \leq, <, \neq\}$ with a containing at least one non-zero element. It is always possible to construct an assignment σ_1 of the non-basic variables of S such that σ_1 does not satisfy c (if c has the form $ax = \beta$ then σ_1 is such that $\sigma_1(ax) \neq -\beta$, if c has the form $ax + \beta \leq 0$, then σ_1 is such that $\sigma_1(ax) > -\beta$ and so on ...). The assignment σ_1 can be extended in a unique way over variables of x to a solution σ_2 of S and σ_2 is a solution of the system $S \cup \{\neg c\}$. Thus we have $S \not\models c$, which falsifies the initial hypothesis. As a consequence, $S \models c$ implies c to be trivial and, obviously, solvable. The other direction is straightforward.□

In the following, those two properties are used to state additional properties for entailment of each type of constraint. The entailment method in an incremental context is also described for each case.

4.2. Equation

Consider an equation in dereferenced form for a constraint system S. From property 1 this equation is not entailed if it contains an occurrence of an arbitrary variable. However, the following property simplifies the entailment condition for an equation.

Property 3 *Let S be a constraint system in normal form and c be an equation in dereferenced form for S. We have $S \models c$ if and only if c is trivial and solvable.*

Proof: Assume that $S \models c$, the affine hull of S is included in the set of solutions of c, i.e., we have *Aff(S)* $\subseteq \Gamma_c$. Since $E_S \cup P$ doesn't contain any implicit equality, we have *Aff*(S) =*Aff*($E_A \cup E_S$) and then *Aff*($E_A \cup E_S$) $\subseteq \Gamma_c$. Consequently, $E_A \cup E_S \models c$ and, from property 2, c must be trivial and solvable. The other direction is straightforward. □

Determining wether an equation is entailed or not is thus very simple: it is sufficient to maintain it in dereferenced form and to wait until it becomes trivial and then check its solvability.

4.3. Inequation

¿From Property 1, an inequation is not entailed if it contains an arbitrary variable. Additionaly, the following property allows us to consider only the subsystem of slack equations of the normal form (i.e., the system $E_S \cup P$ of the normal form 1) for deciding entailment of inequations in dereferenced form.

Property 4 *Let S be a constraint system in normal form, and c be an inequation over slack variables in dereferenced form for S. We have $S \models c$ if and only if $E_S \cup P \models c$.*

Proof: Let C be the convex polyhedron defined with $E_S \cup P$, if $S \models c$ then $C \subset \Gamma_c$ and since c is not trivialy entailed and of a strictly lower dimension than E, we have $E_S \cup P \cup D \models c$. Since c is an inequality and D has a finite number of contraints, we have $E_S \cup P \models c$. The other direction is straightforward.□

Obviously, the simplex algorithm is an appropriate algorithm for deciding entailment of an inequality. Consider a system S in normal form 1, an inequation c of the form $ax_N + \beta \leq 0$ (over slack variables and in dereferenced form for S) and the simplex solved form $x_B = Ax_N + b$ of the subsystem $E_S \cup P$. If the basic solution of this system is not a solution of the constraint c then the constraint is not entailed. Otherwise, maximizing the objective function $ax_N + \beta$ (i.e. minimizing $-ax_N - \beta$) can lead to one of the three following results:

1. the objective function is unbounded;
2. the optimal solution gives a strictly positive value to the objective function.
3. the optimal solution gives a negative value to the objective function;

In cases (1) and (2) the constraint is not entailed while in case (3) it is entailed. Closely to constraint addition in simplex solved form we can stop the iterations of the simplex algorithm as soon as an entering variable x_e is chosen in the actual objective function $ux_N + \lambda$ such that the possible variation α is infinite or satisfies

$$\lambda + \alpha u_e < 0$$

where u_e is the coefficient of x_e in the objective function.

It is important to underline the fact that deciding entailment of an inequality subject to a system of equations over slack variables and the problem of solving a linear program are polynomialy equivalent. Thus the simplex algorithm seems to be the most appropriate and efficient[1] approach for deciding entailment of inequalities.

In an incremental context, when a new constraint is added to the normal form, every ask inequality is mapped to dereferenced form and two cases has to be considered:

- the new constraint is a slack constraint and the simplex algorithm must be applied to each ask inequation over slack variables;
- the new constraint is an arbitrary constraint and the simplex algorithm has to be applied to each ask inequation that has become a slack inequation.

[1]The number of pivoting operations of the simplex algorithm is exponential in the worst case but quasi linear on practical problems.

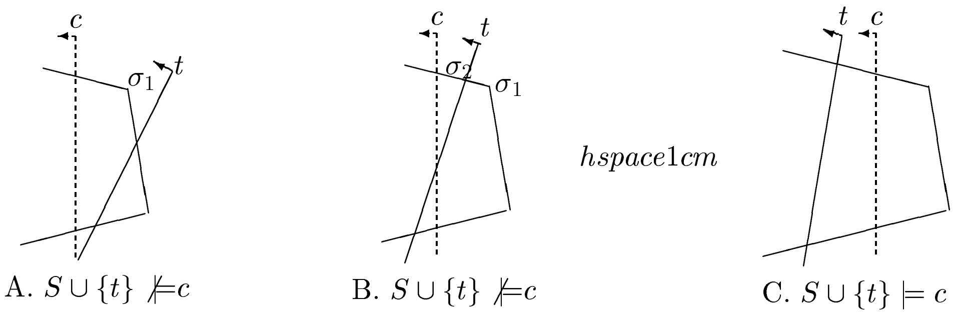

Figure 1: The three cases for an inequation

This improves the naive method presented before, but it is still far from the incremental method, since we systematically apply the simplex algorithm to each ask inequality over slack variables each time a constraint is added to the simplex solved form. The ideal method would check a number of ask constraint which depend on the number of entailed constraints rather than on the total number of ask constraints. A weaker approach would be to determine at a low computational cost the greatest possible number of constraints that are clearly *not entailed* by the new normal form in order to apply the entailment test to the remaining constraints only.

Consider the entailment test described above, when the constraint is not entailed (i.e., in cases 1 and 2) there exists a solution of $E_S \cup P$ which is not a solution of the ask constraint. In [Nil93] it is suggested to compute such a solution and to keep it with the ask inequality. When the ask constraint is proved to be not entailed in cases 1 and 2 above, it is possible to compute a solution σ_1 by giving x_e a value which gives a strictly positive value to the objective function. This solution is a non-entailment test for the ask inequality. Suppose that an inequality t over slack variables is added to the solved form, if σ_1 satisfies t then the constraint $ax_N \leq \beta$ is clearly not entailed by the new system (case A. in figure 1). On the contrary, if σ_1 does not satisfies t the simplex algorithm is applied to find another solution σ_2 of $E_S \cup P \cup \{ax > \beta\}$ (case B. in figure 1) or to prove entailment (case C. in figure 1).

In practice, it should be better to choose the solution σ_1 as closest as possible to the supporting hyperplane of the inequation since a solution far away form the supporting hyperplane would have more chance to be excluded by a new constraint while the ask constraint is not entailed. Indeed it is difficult to know what a *close solution* is since it depends on the problem considered and it is a good policy to construct σ_1 on the supporting hyperplane of the constraint. This solution is no more a solution of $ax_N + \beta > 0$ but very close to. When an inequation t is added to the normal form, the ask constraint has to be reconsidered either if σ_1 is not a solution of t or if it is on the supporting hyperplane of t.

In the following, every ask inequality involving slack variables only is associated to a solution on the supporting hyperplane of the constraint. A similar heuristic can be applied to disequations and strict inequations involving slack variables only.

4.4. Disequation

Deciding entailment of a disequation is more complex. Let S be a system in normal form, let $ax+\beta \neq 0$ be a disequation and reconsider the naive method: $S \models ax+\beta \neq 0$ if and only if the system $S \cup \{ax+\beta = 0\}$ is unsolvable. So when the disequation is not entailed it is possible to map the system $S \cup \{ax+\beta = 0\}$ to a system in normal form S'. This is very close to the *precise* form of a disequation introduced in [LM89] and for which the definition is recalled.

Definition 4 (Precise Form) *Let S be a system in normal form, let $ax+\beta \neq 0$ be a disequation and let S' be the system $S \cup \{ax+\beta = 0\}$ in normal form 1. The precise form of $ax+\beta \neq 0$ is the negation on the subsystem of equations of S', denoted by $\overline{E_A \cup E_S}$.*

A disequation that cannot be mapped to precise form for a system S is obviously entailed by S. This representation has a nice property for entailment: let S be a system of equations and inequations in normal form, let $D = \{d_1, \ldots, d_n\}$ be a set of disequations in precise form for S and let c be a disequation in precise form for S, $S \cup D \models c$ if and only if we have $D \models c$ (see [LM89]).

However, it is not clear how this entailment test can be performed efficiently and, as mentionned before, in terms of space complexity, we cannot afford to maintain $n+p$ systems in normal form when we have n disequations in the set D and p ask disequations.

This is why a weaker form for disequations has to be considered. Constructing the precise form above requires essentially to find all implicit equalities of the system $E_S \cup P \cup \{ax+\beta = 0\}$. Recall that an inequality, and thus, an implicit equality, in a system in normal form is always of the form $x_i \geq 0$. We introduce a *partial* precise form for disequations which associates to the disequation, the set of index of slack variables that are newly fixed to zero in the system $E_S \cup P \cup \{ax+\beta = 0\}$.

Definition 5 (Partial Precise Form) *Let S be a system in normal form 1, a disequation is in* partial precise form *if it is of the form*

$$\langle ax+\beta \neq 0, F\rangle$$

where $ax+\beta \neq 0$ is non trivial and in dereferenced form for S, $E_S \cup P \cup \{ax+\beta = 0\}$ is solvable and F is the set of index variables that are fixed to zero in the system $E_S \cup P \cup \{ax+\beta = 0\}$ and not fixed in the system $E_S \cup P$.

Note that for an arbitrary disequation, the set F is empty. Every disequation in partial precise form can be mapped to the precise form and vice-versa. The precise form of the constraint $\langle ax+\beta \neq 0, F\rangle$ for S is $\overline{E}$ where E is the system $E_A \cup E_S \cup \{x_F = 0\}$ mapped to normal form. This mapping requires the substitution of every variable x_f ($f \in F$) by the zero value. The partial precise form is much more interesting in terms of space complexity. Moreover, a disequation that cannot be mapped to this form is entailed.

Property 5 *Let S be a system in normal form, if a disequation c cannot be mapped to partial precise form for S, then S entails c.*

The proof of this property is a direct consequence of Definition 5. We can now introduce two properties for entailment of disequations which allows to map only ask disequations to partial precise form. The first property hold for disequations whose associated set of index of variables is not empty. Note that such a disequation can be only a slack disequation.

Property 6 *Let S be a system in normal form 1 and $\langle ax+\beta \neq 0, F\rangle$ be a disequation in partial precise form such that $F \neq \emptyset$, the system S entails the constraint $ax+\beta \neq 0$ if and only if there exists a slack disequation $ex+\gamma \neq 0$ from S such that $E_S \cup \{x_F = 0\} \models ex+\gamma = 0$.*

Proof: Assume that S is of the form 1, from the definition of entailment, S entails $ax+\beta \neq 0$ if and only if the system $E_A \cup E_S \cup P \cup D \cup \{ax+\beta = 0\}$ is unsolvable. From the theorem of independence of disequations (see [LMA]), there exist a disequation $ex+\gamma \neq 0$ form D such that the system $E_A \cup E_S \cup P \cup \{ex+\gamma \neq 0,\ ax+\beta = 0\}$ is unsolvable. Since the system $E_A \cup E_S \cup P \cup \{ax+\beta = 0\}$ is solvable, this is equivalent to $E_A \cup E_S \cup P \cup \{ax+\beta = 0\} \models ex+\gamma = 0$ and hence to $\mathit{Aff}(E_A \cup E_S \cup P \cup \{ax+\beta = 0\}) \subset \mathit{Aff}(ex+\gamma = 0)$. Since the affine closure of the first system is determined by its implicit equalities, we have $E_A \cup E_S \cup \{x_F = 0\} \models ex+\gamma = 0$. Note that the entailment of this equation is equivalent to the entailment of the two inequations $ex+\gamma \leq 0$ and $ex+\gamma \geq 0$ and from property 4 we have $E_S \cup \{x_F = 0\} \models ex+\gamma = 0$. □

Note that from property 3 we have $E_S \cup \{x_F = 0\} \models ex+\gamma = 0$ if and only if $ex+\gamma = 0$ is reduced to $0 = 0$. Consequently, deciding entailment of an ask disequation in partial precise form whose associated set of index of fixed variables is not empty requires the substitution of every variable $x_f, f \in F$ in the subsystem $E_A \cup E_S \cup D$. If a disequation becomes $0 \neq 0$ then the ask disequation is entailed. The next property deal with the case where the set F is empty.

Property 7 *Let S be a system in normal form 1 and $\langle ax+\beta \neq 0, \emptyset\rangle$ be a disequation in partial precise form, the system S entails the constraint $ax+\beta \neq 0$ if and only if there exists a disequation $ex+\gamma \neq 0$ from S and a non-zero real number λ such that $a = \lambda e$ and $\beta = \lambda\gamma$.*

Proof: Since the system $E_A \cup E_S \cup P \cup \{ax+\beta = 0\}$ is solvable without implicit equalities we have $\mathit{Aff}(ax+\beta = 0) \subset \mathit{Aff}(E_A \cup E_S \cup P)$ and then there exists a disequation $ex+\gamma \neq 0$ such that $\{ax+\beta = 0\} \models ex+\gamma = 0$. If we map $\{ax+\beta = 0\}$ to normal form, we would have to choose a variable x_i such that $a_i \neq 0$ to enter the basis. Mapping $ex+\gamma \neq 0$ to dereferenced form requires dereference x_i in that constraint. This dereferencing amounts to compute $e'x+\gamma' = 0$ where $e' = e - a \times \frac{e_i}{-a_i}$ and $\gamma' = \gamma - \beta \times \frac{e_i}{-a_i}$. Consequently, $ex+\gamma = 0$ is entaild if and only if $e' = 0$ and $\gamma' = 0$ and so iff $e = a \times \lambda$ and $\gamma = \beta \times \lambda$ where $\lambda = \frac{e_i}{-a_i}$. □

When the ask disequation in partial precise form has an empty set of index variables, this property allows to consider the disequations of the normal form one by one for deciding entailment.

When a new disequation $ex+\gamma \neq 0$ is added to the system in normal form, an ask disequation $\langle ax+\beta \neq 0, F\rangle$ is entailed if one the two following cases hold:

- disequations $ex+\gamma \neq 0$ and $ax+\beta \neq 0$ have the same supporting hyperplane (property 7);
- $ex+\gamma \neq 0$ is a slack disequation, F is not empty and $E_S \cup \{x_F = 0\} \models ex+\gamma = 0$ (property 6).

Disequations must also be incrementaly maintained in partial precise form. So, when a new equation is added to the normal form, two case have to be considered :

- the new equation is a slack equation and each slack disequation as to be reconsidered to extend the set of associated fixed variables or to prove that it cannot be mapped to partial precise form for the new normal form.

- the new equation is an arbitrary equation and every disequation that has become slack must be reconsidered for the same reasons.

To extend its set of associated fixed variables, we have to reconsider a disequation when the new constraints can introduce fixed variables. This can be detected by associating to each disequation a solution on its supporting hyperplane. A disequation has to be reconsidered only if this solution is excluded by, or on the supporting hyperplane of the new constraint.

4.5. Strict Inequation

Let S be a system in normal form and and let $ax_N + \beta < 0$ be a strict inequality in dereferenced form for the system S. From property 1, this strict inequalitiy is not entailed when it contains at least one arbitrary variable, so consider the case where it contains only slack variables. Closely to constraint addition, deciding entailment of this strict inequality amounts to decide the entailment of both the inequality $ax_N + \beta \leq 0$ and the disequation $ax_N + \beta \neq 0$. However, such a decomposition is not efficient and can be improved, but we use it in the following to describe and clarify the improvement.

Consider the inequality $ax_N + \beta \leq 0$, we have seen previously that its entailment can be decided by minimizing $-ax_N - \beta$ subject to $E_S \cup P$ in simplex solved form. This leads to on of the three following results and we detail the consequences for the strict inequality:

1. The optimal solution is undounded or gives a strictly positive value to $ax_N + \beta$, and hence neither $ax_N + \beta \leq 0$ nor $ax_N + \beta < 0$ is entailed.
2. The optimal solution gives a strictly negative value to $ax_N + \beta$ and the system $E_S \cup P$ entails the constraint $ax_N + \beta \leq 0$ and also the constraint $ax_N + \beta < 0$.
3. The optimal solution gives the zero value to $ax_N + \beta$ and $E_S \cup P$ entails $ax_N + \beta \leq 0$ and in that case we have $S \models ax_N + \beta < 0$ if and only if we have $S \models ax_N + \beta \neq 0$.

The last case is the only one where the disequation has to be considered because the supporting hyperplane of the strict inequality is on the *boundary* of the polyhedron defined by the system $E_S \cup P$. Deciding entailment in that case amounts to decide the entailment of the disequation $ax_N + \beta \neq 0$.

Since a strict inequation is considered as an inequation as soon as it becomes a disequation, we can also maintain a solution of $E_S \cup P$ on the supporting hyperplane of the constraint. In that case it is really a solution of the negation of the constraint; it is to reconsidered when a new constraint excludes this solution.

5. Applications

The entailment algorithms described in the previous section has been integrated to the CLP language `Athena` over rationals numbers in infinite precision, described in [RH96]. Even if the simplex algorithm used is a revised simplex over bounded variables, all the results presented in this paper also hold for bounded variables.

The language includes the implication operator and the possibility to associate a boolean value to a constraint. Roughtly, it allows to write expressions of the form $u = [c]$ where c is a constraint

	Naive Method		Dereferenced Form		Solution		
Problems	Pivot	Time (N)	Pivot	Time	Pivot	Time (S)	N / S
Bridge	322512	3706.2	114038	1749.3	13380	221.7	16.7
CNET	14010	385.1	13852	378.2	8751	159.5	2.4
Circuit	520	312.2	236	102.3	47	38.5	8.1
HSP1	18117	401.0	11258	250.2	6541	119.8	3.3
HSP2	25108	682.2	18523	510.3	11736	192.3	3.5
Ar. Mean							6.8

Table 1: Time Requirement for Entailment.

and u a variable. This expression is equivalent to the following conjunction of constraints and implications but more efficiently implemented:

$$\{0 \leq u \leq 1, u = 0 \rightarrow \neg c, u = 1 \rightarrow c, c \rightarrow u = 1, \neg c \rightarrow u = 0\}$$

We have try various problem form operation research and electrical engineering. The `Bridge` problem is an optimisation problem with disjunctive constraints whose first CLP solution appears in [vH89]. The disjunctive constraints states that two task T_1 and T_2, whose starting date are t_1 and t_2 and whose duration are d_1 and d_2 cannot overlap. This is stated with the following expression:

$$[t_2 + d_2 \leq t_1] + [t_1 + d_1 \leq t_2] = 1$$

The `CNET` problem [Nar94] is a 0-1 mixed integer optimisation problem which amounts to select the position of the concentrators on a network to minimize the distances between concentrators and plugs and the number of concentrators. The 0-1 constraint over a variable x is expressed as follows

$$0 \leq x \leq 1, [x \leq 0] + [1 \leq x] = 1$$

Finaly the `Circuit` problem [TGZ90] is a simulation of the temporal behaviour of an hybrid circuit containing electro-mecanical and digital control devices. The behavior of these devices is easily represented by the high level operators using the implication operator.

The hoist scheduling problem (`HSP`) is a cyclic scheduling problem with disjunctive constraints [BLV92]. The main difficulties of this problem come from the great number of disjunction and from the presence of the same variable (the period variable) in every disjunction. Two intances of this problem from industrial applications has been considered.

In all this problems the presence of high level constraint simplifies the statement but does not avoid the use of enumeration for finding a solution or the best solution.

Table 1 presents the amounts of pivoting operations and the time (in seconds) required for the naive method, for the use of the dereferenced form and for the use of a solution on the supporting hyperplane of each ask constraint. The dereferenced form always improves the number of pivots and the time requirement as well. The additional use of a solution on the supporting hyperplane of the ask constraints improves the naive algorithm by a factor of 6 on the average and up to a factor of 16 for the `Bridge` problem.

The results presented in table 2 shows that a significant reduction of the search tree and of the computation time is achieved when using entailment. On the other hand, entailment over linear

		Standard Backtracking		High Level Operators		
Problems	Ask Constraints	Back.	Time (N)	Back.	Time (E)	N / E
`Bridge`	154	8948	221.7	436	205.0	1.08
`CNET`	20	173	159.5	45	280.2	0.56
`Circuit`	46	289	38.5	11	13.4	2.87
`HSP1`	348	965	119.8	52	14.5	8.26
`HSP2`	602	1750	192.3	89	21.2	9.07

Table 2: With and Without High Level Operators

constraint is computationaly difficult. As a consequence, high level operators over linear constraints should not be used systematicaly for solving any combinatorial linear constraint problem and their use must studied because not every problem can benefit of them. For example, in the `CNET` problem the reduction of the search space does not compensate the cost of reducing it.

6. Conclusion

This article has presented two forms for linear constraints, the dereferenced form and the partial precise form for disequations and several properties that improve the entailment test. A method for avoiding to apply systematicaly the entailment test to each ask constraint, when a new constraint is added to the store was also presented and sucessfuly applied. Linear constraint entailment has been integrated in a CLP language and has been applied to real-live problems. The practical results shows that, even if linear constraint entailment is computationaly more expensive than entailment over finite domains constraints and intervals it is complete and can be sucessfuly used for solving practical problems efficiently. The way to improve these results is certainly in making our solver cooperating with an interval constraint solver. Since these solvers are sound they can prove that a constraint is entailed and can avoid sometimes to make this proof with the simplex algorithm. This is the topic of our current research.

References

[BLV92] P. Baptiste, B. Legeard, and C. Varnier. Hoist scheduling problem : an approach based on constraint logic programming. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, pages 1139–1144, Nice, France, May 1992.

[BO96] F. Benhamou and W. Older. Applying interval arithmetic to integer and boolean constraints. *Journal of Logic Programming*, 1996. To appear.

[Chv83] V. Chvatal. *Linear Programming.* W.H. Freeman and Company, New York, 1983.

[Col90] A. Colmerauer. An introduction to Prolog III. *Communications of the ACM*, 33(7):69–91, July 1990.

[DvHS88] M. Dincbas, P. van Hentenryck, and H. Simonis. The constraint logic programming langage CHIP. In *FGCS-88*, Tokyo, Japan, 1988.

[JMSY92] 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.

[Las90a] J-L Lassez. Parametric queries, linear constraints and variable elimination. In *DISCO 90*. Springer-Verlag Lecture Notes in Computer Science, 1990.

[Las90b] J-L Lassez. Querying constraints. In *Proceedings of the ACM Conference on Principles of Database Systems*, Nashville, 1990.

[LM89] J-L Lassez and K. McAloon. A canonical form for generalised linear constraints. Technical report, IBM Research Report, T.J. Watson Research Center, 1989.

[Nar94] G.A. Narboni. Optimization of concentrator positioning in the design of private data networks. In *Conference on the Practical Applications of Prolog*, 1994.

[Nil93] M. Nilsson. Piecewise linear constraint and entailment. In *Constraint Logic Programming Workshop*, Marseille, France, 1993.

[RH96] P. Refalo and P. Van Hentenryck. CLP(R_{lin}) revised. In *Joint International Conference and Symposium of Logic Programming*, Bonn, Germany, September 1996. MIT Press.

[Sar93] V. A. Saraswat. *Concurent Constraint Programming Langages.* MIT Press, Cambridge, Mass., 1993.

[TGZ90] C. Pradelles-Lasserre T. Graf, P. Van Hantenryck and L. Zimmer. Simulation of hybrid circuits in constraint logic programming. *Computers Math. Applic.*, 20(9):45–56, 1990.

[vH89] P. van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, Cambridge, Mass., 1989.

[vHSD99] P. van Hentenryck, V. Saraswat, and Y. Deville. Design, implementation and evaluation of the constraint langage $\mathsf{cc}(\mathcal{FD})$. In *Constraint Programming: Basic and Trends.* Springer Verlag, 1999.