Propagating Dense Systems of Integer Linear Equations

Thibaut Feydy and Peter J. Stuckey
NICTA Victoria Laboratory
Department of Computer Science & Software Engineering
University of Melbourne, Australia
{tfeydy,pjs}@cs.mu.oz.au

ABSTRACT

In interval propagation approaches to solving non-linear constraints over reals it is common to build stronger propagators from systems of linear equations. This, as far as we are aware, is not pursued for integer finite domain propagation. In this paper we show how we can add preconditioning Gauss-Seidel based propagators to an integer propagation solver. The Gauss-Seidel based propagators make use of interval arithmetic which is substantially slower than integer arithmetic. We show how we can build new integer propagators from the result of preconditioning that no longer require interval arithmetic to be performed. Although the resulting propagators may be slightly weaker than the original Gauss-Seidel propagation, they are substantially faster. We show on standard integer benchmarks how these new propagators can substantially improve propagation performance, in terms of strength of propagation and speed.

Categories and Subject Descriptors

D.3.2 [**Programming Languages**]: Language Classifications—*Constraint and logic languages*; D.3.3 [**Programming Languages**]: Language Constructs and Features—*Constraints*

General Terms

Algorithms

Keywords

Linear equations, Gaussian elimination, constraint programming, constraint Propagation

1. INTRODUCTION

Linear equations are one of the most important constraints in any integer finite domain propagation system. Efficient bounds propagation of individual linear equations is well understood [4] and available in all constraint programming systems. But in other solving approaches systems of linear equations (and inequalities) are not treated individually but together as a system.

Example 1. Consider the linear equations $x_1 + x_3 - x_4 = 3 \wedge x_1 + x_2 + 2x_3 - x_4 = 4$ with initial domains $[0, 4]$ then individually no propagation is possible, but the equivalent system $x_1 + x_3 - x_4 = 3 \wedge x_2 + x_3 = 1$ can reduce the domains of all variables. Clearly if we can treat the system together there is more scope for propagation

In linear programming the real relaxation of all integer linear equations and inequalities is treated together by the linear programming algorithm. Even though linear programming based propagators [9] are available in constraint programming toolkits (for example OPL [8] and ECLiPSe [2]), the linear programming propagator does not provide new bounds information for the integer variables involved, rather it is used to bound the objective function (and possibly do reduced cost variable fixing).

Alternatively one can also use linear programming to build propagators that take into account all linear constraints simultaneously by minimizing and maximizing every variable in turn, and rounding the resulting bounds to integers [10]. Unfortunately this requires repeatedly performing $2n$ LP optimizations, where n is the number of variables, to propagate the linear constraints. This is very expensive compared to the cost of treating the constraints as integer linear propagators.

In non-linear interval solvers, systems of linear equations are treated together as propagators using Gauss-Seidel iteration. Typically they are first preconditioned by Gaussian elimination in order to speed up convergence. While it is possible to define Gaussian elimination over integer linear equations, in practice the integer coefficients typically quickly explode making it impractical.

In this paper we explore using a real interval hybrid approach to propagate systems of integer linear equations. Preconditioning of the linear equations is performed using interval arithmetic, and the resulting interval arithmetic propagator is applied to variables (taking into account their integrality). This requires a hybrid interval and integer finite domain solver.

Example 2. Consider the following constraints which are almost collinear to those of Example 1: $100x_1 + 99x_3 - 101x_4 = 301 \wedge 101x_1 + 99x_2 + 199x_3 - 100x_4 = 399$, and initial domains [0,4]. Again no propagation is possible. The integer Gaussian elimination yields $100 \times 99x_2 + (100 \times 199 - 101 \times 99)x_3 - (100 \times 100 - 101 \times 101)x_4 = 100 \times 399 - 101 \times 301$,

SAC'07 March 11-15, 2007, Seoul, Korea.

illustrating the increase in coefficients. The interval version of the elimination yields $x_2 + \overline{\frac{9901}{9900}}x_3 - \overline{\frac{201}{9900}}x_3 = \overline{\frac{9499}{9900}}$ where $\overline{a}$ represents a "tight" floating point interval around a. Both the integer and interval propagators resulting from Gaussian elimination detect that there is no solution.

We then show that we can remove the requirement for the hybrid solver altogether by weakening the resulting interval linear equations to integer linear inequalities.

Example 3. The floating point interval equation resulting in Example 2 can be weakened to the integer inequalities $9899x_2 + 9900x_3 - 202x_4 \leq 9500$ and $9901x_2 + 9902x_3 - 200x_4 \geq 9498$. This is at least strong enough to reduce the domains of x_2 and x_3 to [0,1], and with the original equation x_1 obtains [1,4] and x_4 obtains [0,2].

Note that x_4 is almost irrelevant in these equations since its coefficient is much smaller than the other coefficients. We can remove it altogether, using its bounds, to simplify the inequalities without losing much propagation. We obtain $9899x_2 + 9900x_3 \leq 10308$ and $9901x_2 + 9902x_3 \geq 9498$. We lose no initial propagation with this simplification.

This paper examines the use of Gauss-Seidel methods with preconditioning for integer linear equations. We show how to achieve this using a hybrid interval and integer propagation solver, and how to map the problem back entirely to integer propagation. Our experiments on standard benchmarks show that the technique can lead to substantial improvements in propagation efficiency.

2. INTERVAL GAUSSIAN ELIMINATION

2.1 Interval Arithmetic

Let $\mathbb{R}$ be the set of real numbers, and let $\mathbb{R}^\infty$ be $\mathbb{R} \cup \{+\infty, -\infty\}$. Let $\mathbb{F}$ be the subset of $\mathbb{R}$ of the representable floating-point numbers in a given format, and let $\mathbb{F}^\infty$ be $\mathbb{F} \cup \{+\infty, -\infty\}$. Given two numbers $a \in \mathbb{F} \cup \{+\infty\}$ and $b \in \mathbb{F} \cup \{-\infty\}$, the *interval* $[a, b]$ is the set $\{x \in \mathbb{R} \mid a \leq x \leq b\}$. We will use $\mathbb{I}$ to represent the set of intervals, which is closed under intersection.

Given an interval I we will define $\lfloor I \rfloor$ (resp. $\lceil I \rceil$) as the smallest (resp. largest) element of I.

Given an interval $I = [a, b]$, let $ceil(I)$ (respectively $floor(I)$) be the smallest (resp. largest) integer such that $ceil(I) \geq b$ (resp. $floor(I) \leq a$).

Given an interval $I = [a, b]$, let $mid(I)$ be the approximated midpoint of I:

$$\forall y \in \mathbb{F},\ \left(\frac{a+b}{2} - mid(I)\right)^2 \leq \left(\frac{a+b}{2} - y\right)^2$$

Interval arithmetic [6] was designed to tackle two problems of numerical analysis: uncertainty of data and roundoff error. It has the advantage of being a sound approximation of problems over $\mathbb{R}$, and interval analysis methods, based on iterative contraction of intervals, are easily implemented in a constraint programming framework.

Given two intervals D_x and D_y, the interval operator $\diamond$ associated to the real operator $\diamond$ is defined by:

$$D_x \diamond D_y = \bigcap_{\mathbb{I}} \{D_z \mid \forall x \in D_x, \forall y \in D_y, x \diamond y \in D_z\}$$

As an example, the multiplication operator is defined by:

$$[a, b] \times [c, d] = [\downarrow \min(ac, ad, bc, bd), \uparrow \max(ac, ad, bc, bd)]$$

where $\downarrow(r)$ and $\uparrow(r)$ are respectively the downward and upward roundings to $\mathbb{F}$ of a real number r.

An *interval extension* of a function $f : \mathbb{R}^n \to \mathbb{R}$ is a mapping $F : \mathbb{I}^n \to \mathbb{I}$ such that:
$\forall(D_1, \ldots, D_n) \in \mathbb{I}^n, \forall(x_1, \ldots, x_n) \in (D_1, \ldots, D_n)$
$f(x_1, \ldots, x_n) \in F(D_1, \ldots, D_n)$

The following standard result give the basis of interval arithmetic.

Definition 1. Given an arithmetic real function f, the mapping obtained by replacing the real vector x by the interval vector D_x, each constant r by the interval $[\lfloor r \rfloor, \lceil r \rceil]$ and each operator by the associated interval operator is an interval extension of f, called its *natural extension.*

2.2 Gauss-Seidel

Given a system $Ax = b$, where A is a $\mathbb{F}^{n \times n}$ matrix, b is a $\mathbb{F}^n$ vector, and given an initial box $D^{(0)}$, Gauss-Seidel will compute iteratively smaller boxes containing the real solutions of the system contained in $D^{(0)}$.

An iteration k of the Gauss-Seidel algorithm consists of computing n projections in sequence. The computation of the i-th projection is given by the formula:

$$D_i^{(k)} = D_i^{(k-1)} \cap \left(\frac{b_i - \sum_{j<i} a_{ij} D_j^{(k)} - \sum_{l>i} a_{il} D_l^{(k-1)}}{a_{ii}} \right)$$

where D_i is the interval for the i-th variable in x.

The *hull* of the solution set of a system of linear equations is the tightest intervals that encloses the solutions of the system. The *mignitude* of an interval $I = [a, b]$ is defined as $mig(I) = \min(|a|, |b|)$ while its *magnitude* is defined as $mag(I) = \max(|a|, |b|)$. A matrix $A = (a_{ij})$ is *diagonally dominant* if:

$$\forall i \in 1 \ldots n,\ mig(a_{ii}) > \sum_{k \neq i} mag(a_i k)$$

The power of interval Gauss-Seidel is captured by the following result:

THEOREM 1. [7] *The Interval Gauss-Seidel method applied to a system $Ax = b$ where A is diagonally dominant is guaranteed to converge to the hull of the solution set of the system.*

2.3 Preconditioning

The Interval Gauss-Seidel method is not guaranteed to converge in a general system. In order to obtain tighter bounds and to accelerate the convergence of the algorithm, a transformation is usually applied to the system before using Gauss-Seidel. This is generally done by multiplying the original system $Ax = b$ by a suitable matrix, or *preconditioner*, P. A preconditioner widely used and often considered optimal is the midpoint inverse of A:

$$\tilde{A}^{-1} = (mid(a_{ij}))^{-1}$$

$\tilde{A}^{-1}$ can be computed approximatively using floating-point inversion algorithms, such as Gaussian Elimination or LU decomposition. Preconditioning involves $O(n^3)$ interval multiplications, thus the resulting system will have a wider

hull than the original one. An inconsistent system of equation may become consistent after preconditioning, however *soundness* is guaranteed (Definition 1).

The coefficient matrix resulting from this preconditioning is a matrix whose diagonal elements are centered around one, while non-diagonal elements are centered around zero. When the initial coefficient matrix A is a floating-point matrix, the width of the coefficients after preconditioning result from rounding during computations. We may then expect these intervals to be tight and we will refer to them as *quasi-zeros* for non-diagonal coefficients and *quasi-ones* for diagonal coefficients. We will refer to the resulting matrix as a quasi-identity matrix.

Example 4. Consider the variables x_1, x_2, x_3, with the domains $[-10, 10]$, and the following linear constraints:

$$C_1 : \begin{bmatrix} 5 & 3 & 4 \\ -1 & 2 & -2 \\ 1 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 7 \\ -2 \end{bmatrix}$$

None of these constraints perform any pruning. Applying the midpoint inverse preconditioner yields the system:

$$C_1' : \begin{bmatrix} \overline{1} & \overline{0} & \overline{0} \\ \overline{0} & \overline{1} & \overline{0} \\ \overline{0} & \overline{0} & \overline{1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \overline{-7} \\ \overline{5} \\ \overline{5} \end{bmatrix}$$

In double precision floating-point arithmetic, each interval of this system have a width smaller than 10^{-10}. Given any reasonable domains, these constraints fix the values of x_1, x_2 and x_3 during the first iteration.

2.4 Rectangular matrix preconditioning

Many constraint programming problems contain linear equalities, however the number of linear constraints may be less than the number of variables involved in these constraints. In this case only a partial Gaussian elimination [1] is performed, which may still lead to better propagation.

Given an $m \times n$ matrix A, with m the number of equations and $n > m$ the number of variables, the result of the preconditioning of a system $Ax = b$ is a system $(I_m\, A')\, x = b'$, where I_m is a quasi-identity matrix. Since the coefficient matrix is not diagonally dominant (A' is a general matrix, not a quasi-zeros matrix), Gauss-Seidel applied to the resulting system is not guaranteed to converge to the hull of the system anymore. For that reason a general linear constraint is posted for each linear equality, rather than one Gauss-Seidel projection. Another approach is to apply a modified Gauss-Seidel procedure which computes $(n-m)+1$ projections, one for each non-zero coefficient. We will see later in this paper a method similar to a modified Gauss-Seidel procedure which is implemented using the usual linear equality constraint.

Example 5. Consider the last two equations of the constraint C_1:

$$C_2 : \begin{bmatrix} -1 & 2 & -2 \\ 1 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ -2 \end{bmatrix}$$

A partial Gaussian elimination yields the constraint:

$$C_2' : \begin{bmatrix} \overline{1} & \overline{0} & \overline{0.5} \\ \overline{0} & \overline{1} & \overline{0} \end{bmatrix} \begin{bmatrix} x_3 \\ x_2 \\ x_1 \end{bmatrix} = \begin{bmatrix} \overline{1.5} \\ \overline{5} \end{bmatrix}$$

The original constraints will again not perform any pruning whereas the redundant constraints introduced by C_2' will reduce the domain of x_1 to $[-9, 9]$, the domain of x_3 to $[-3, 6]$, and x_2 will be fixed to 5. x_1 and x_3 are also strongly connected by a binary constraint.

However, partial Gaussian elimination can result in costly propagators providing weak information if applied to a sparse matrix when $n >> m$: in the worst case, performing a partial Gaussian elimination on a system of m equations and n variables will result in a system of m equations, each with $(n-m)+1$ non quasi-zero coefficients. If $n >> m$ and if the original equations only involved a few variables each, it is unlikely that we will gain stronger information from preconditioning, while each resulting propagator will have a execution cost in $O(n-m)$.

Example 6. Consider the system $x_1 + x_2 + x_3 = 0 \wedge x_1 + x_4 + x_5 = 0 \wedge x_5 + x_6 + x_7 = 0$. Preconditioning this system results in the system $x_1 + x_2 + x_3 = 0 \wedge x_4 + x_5 - x_2 - x_3 = 0 \wedge x_6 + x_7 - x_4 + x_2 + x_3 = 0$. No information is gained from the resulting, more expensive, propagators.

3. PROPAGATING SYSTEMS OF INTEGER LINEAR CONSTRAINTS

3.1 Systems of linear equalities

Clearly we can transport the interval Gauss-Seidel method with pre-conditioning unchanged to an integer finite domain propagation system. In order to do so we simply build an appropriate hybrid propagator. Note that the interval system is vital for performing the preconditioning steps (using doubles) rather than using integers which quickly leads to overflows (or highly expensive computation if using infinite precision integers). Examples 4 and 5 illustrate the benefits over standard integer linear propagation. Note that we always execute the original integer linear equation propagators as well to maintain inconsistency detection.

This hybrid provides us with capabilities not available in any other solver, since we can use Gauss-Seidel with preconditioning, available in interval solvers, and `alldifferent` propagators, available in integer propagation solvers.

3.2 Handling linear inequalities

Linear inequalities can be used during Gaussian elimination, since a linear inequality can be written as an equality constraint by adding a slack variable. However, this is usually less efficient as an inequality constraint can be satisfied (and thrown away) before its variables are ground, which is not true for an equality constraint. Furthermore, using an inequality for pivoting makes the slack variable of this inequality appear in other constraints, which will likely provide very weak propagation.

However, it is still possible to efficiently apply Gaussian elimination to a system with linear equalities and inequalities as long as inequalities are not chosen for pivoting.

Let us consider a system $Ax = b \wedge Cx \leq d$ of m_A equalities and m_C inequalities. We can rewrite this system by introducing m_C slack variables:

$$\begin{pmatrix} I_{m_C} & C \\ 0 & A \end{pmatrix} x \;=\; \begin{pmatrix} d \\ b \end{pmatrix}$$

where I_{m_C} is the $m_C \times m_C$ identity matrix. This system has the form of a general system of equations after m_C pivoting

step, and we can start variable elimination at the $m_C + 1$ row. The result of the next m_A elimination is:

$$\begin{pmatrix} I_{m_C} & Q_0 & C' \\ 0 & Q_{I\,m_A} & A' \end{pmatrix} x \;=\; \begin{pmatrix} d' \\ b' \end{pmatrix}$$

where Q_{Im_A} is a quasi-identity matrix and Q_0 is a matrix of quasi-zeros. We can then go back to a reduced system with both equalities and inequalities:
$(Q_0\, C')\, x \leq d' \wedge (Q_{Im_A}\, A')\, x = b'$.

Example 7. Consider the system $C_3 = C_2 \wedge 5x_1 + 3x_2 + 4x_3 \leq 0$. We can transform this system into a system of linear equalities by adding a slack variable s:

$$C_4 : \begin{bmatrix} 1 & 5 & 3 & 4 \\ 0 & -1 & 2 & -2 \\ 0 & 1 & -1 & 2 \end{bmatrix} \begin{bmatrix} s \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 7 \\ -2 \end{bmatrix}$$

A partial elimination performed on C_4 gives us the system

$$C'_4 : \begin{bmatrix} 1 & \overline{0} & \overline{0} & \overline{3} \\ 0 & \overline{1} & \overline{0} & \overline{0.5} \\ 0 & \overline{0} & \overline{1} & \overline{0} \end{bmatrix} \begin{bmatrix} s \\ x_3 \\ x_2 \\ x_1 \end{bmatrix} = \begin{bmatrix} \overline{-21} \\ \overline{3} \\ \overline{5} \end{bmatrix}$$

And since s is a non-negative slack variable, we replace the first equality by an inequality, leading to the constraint $C''_4 = C'_2 \wedge \overline{3}x_1 \leq \overline{-21}$. Again this gives strong information, with the domain of x_1 reduced to $[-9, -7]$, the domain of x_3 reduced to $[5, 6]$ and x_2 fixed to 5.

3.3 Quasi-zeros elimination

The result of Gaussian Elimination on a system of equations $Ax = b$ with m equations and $n > m$ variables is a system $(Q_I\, A')\, x = b'$ where Q_I is a $m \times m$ quasi-identity matrix. In the case of a floating-point matrix $A \in \mathbb{F}^{n\times n}$, we may expect the quasi-zeros of Q_I to be very tight intervals, as they are the result of outward roundings performed during Gaussian Elimination.

Given an interval equation $\sum_{i=1\ldots n} \alpha_i x_i = \alpha_0$, let $\mathcal{Q}$ be the subset of $\{1, \ldots, n\}$ such that $\{\alpha_i \,|\, i \in \mathcal{Q}\}$ are quasi-zeros. The propagator of this equality on the variable x_i $(i \notin \mathcal{Q})$ of domain D_i is:

$$D_i \leftarrow D_i \cap \left(\frac{\alpha_0 - \sum_{j\notin\mathcal{Q}, j\neq i} \alpha_j D_j - \sum_{k\in\mathcal{Q}} \alpha_k D_k}{\alpha_i} \right)$$

Given the assumption that the quasi-zeros are tight and that we have reasonable initial bounds for each variable:

$$v = \frac{\sum_{k\in\mathcal{Q}} \alpha_k D_k}{\alpha_i} << 1$$

So we replace this expression by its initial value v in the propagator above (hence the variables with quasi-zero coefficients are eliminated). The elimination of x_4 in Example 3 illustrates the approach.

3.4 Getting rid of interval constraints

Using an external interval solver may be an issue for FD solvers, for efficiency or portability considerations. Obviously, interval constraints need to be available, at least for linear equality constraints. One also needs a way to channel information between the finite domain and the interval solvers. Secondly, whereas it is easy to write portable rounding functions for the arithmetic operators used for the Gaussian elimination, this is not true for other mathematical functions, making interval solvers more hardware dependent than most FD solvers. And finally, integers operations and constraints are still faster than their floating-point or interval counter-part. That is why we present in this paper a way to use Gaussian elimination as a preprocessing step posting redundant *integer* constraints.

Given an interval linear constraint $c : \sum \alpha_i x_i = \alpha_0$, where the variables x_i are n integers taking values in the box $D = D_1 \times \cdots \times D_n$, and given $2n$ integers $\{a_1, \ldots, a_n\}$ and $\{b_1, \ldots, b_n\}$, the constraint $\sum a_i x_i \leq a_0 \wedge \sum b_i x_i \geq b_0$ where $a_0 = ceil(\alpha_0 + \sum (a_i - \alpha_i) D_i)$ and $b_0 = floor(\alpha_0 - \sum (b_i - \alpha_i) D_i)$ is a safe approximation of c. Indeed $\forall x \in D$, $\sum a_i x_i - a_0 \leq \sum \alpha_i x_i - \alpha_0 \leq \sum b_i x_i - b_0$.

In order to get a tight approximation for the left inequality, the values $\{a_1 \times \cdots \times a_n\}$ are chosen so that the difference $d_l(x) = \sum \alpha_i x_i - \alpha_0 - (\sum a_i x_i - a_0)$ is small over D.

$$\begin{aligned} d_l(x) &= \sum (\alpha_i - a_i) x_i - \alpha_0 + ceil(\alpha_0 + \sum (a_i - \alpha_i) D_i) \\ &\approx \lceil \alpha_0 \rceil - \alpha_0 + \sum ((\alpha_i - a_i) x_i - \lceil (\alpha_i - a_i) D_i \rceil) \end{aligned}$$

This last expression is minimised over D by choosing each a_i as $\min_k(mag((k - \alpha_i) D_i - \lceil (k - \alpha_i) D_i \rceil))$, which is an integer value between $\lfloor \alpha_i \rfloor$ and $\lceil \alpha_i \rceil$. A similar reasoning is applied to choose the coefficents of the right inequality.

In order to avoid overflow when going back to integers, we may need to multiply the original equation $\sum \alpha_i x_i = \alpha_0$ by a suitable value β:

$$\sum \alpha'_i x_i = \alpha'_0 \text{ where } \alpha'_i = \beta \alpha_i,\ i \in 0, \ldots, n$$

However, we choose the largest β which does not cause overflow, if possible $\beta >> 1$, to lose as little information as possible when rounding to integers values, as illustrated by the following example.

Example 8. Consider the preconditioned system C'_2 of example 5, with the initials domains $[-10, 10]$:

$$C'_2 : \begin{bmatrix} \overline{1} & \overline{0} & \overline{0.5} \\ \overline{0} & \overline{1} & \overline{0} \end{bmatrix} \begin{bmatrix} x_3 \\ x_2 \\ x_1 \end{bmatrix} = \begin{bmatrix} \overline{1.5} \\ \overline{5} \end{bmatrix}$$

The equation $\overline{0.5}x_1 + \overline{1}x_3 = \overline{1.5}$ rounded to integers provides the inequalities $x_1 + x_3 \geq -4$ and $x_1 + x_3 \leq 7$, which do not prune the domains of x_1 and x_3. If we first multiply the equality terms by, for example, 100, we get the new equality $\overline{50}x_1 + \overline{100}x_3 = \overline{150}$. This equality provides the inequalities $50x_1 + 100x_3 \geq 149$ and $50x_1 + 100x_3 \leq 151$. This information is strong enough to get the same pruning as the original equation: at fix point the domain of x_1 is reduced to $[-9, 9]$ while the domain of x_3 is reduced to $[-3, 6]$. In the same way, for the second equation we get $x_2 \geq 4 \wedge x_2 \leq 6$ with the original equation and $100x_2 \geq 499 \wedge 100x_2 \leq 501$ (i.e. $x_2 = 5$) if we multiply the original equation by 100.

4. EXPERIMENTAL RESULTS

The finite domain solver and the interval solver used for the experiments are implemented in Mercury [5]. The interval solver uses the Gaol [3] interval library. The tests have been executed on a Pentium 4 1.60GHz running Linux(Sarge).

	Times(ms)				
Problem	fd	ic	ic-qz	ii	ii-qz
eq10	23	10	9	14	11
alpha	277	34	18	15	14
alpha-rev	$+\infty$	79790	28600	5650	5500
ineq-alpha-rev	$+\infty$	79320	29050	6020	5900
ineq-alpha-rev2	$+\infty$	7110	3390	450	430
overlap-a	2470	540	290	270	270
partial-overlap-a	2470	95	55	48	44
crypta	2	6	5	4.4	4
crypta-magic-sqr	78	4	4	2	3
crypta-magic-sqr2	39	4	4	4	4
magic-square-5	408	7250	930	440	450

Table 1: Comparative execution times

	Failures				
Problem	fd	ic	ic-qz	ii	ii-qz
eq10	5	0	0	0	0
alpha	1286	0	0	0	0
alpha-rev	$+\infty$	11779	12512	12472	12472
ineq-alpha-rev	$+\infty$	11643	12444	12335	12335
ineq-alpha-rev2	$+\infty$	499	1025	499	499
overlap-a	11393	0	0	0	0
partial-overlap-a	11393	0	0	0	0
crypta	15	14	14	14	14
crypta-magic-sqr	570	1	1	1	1
crypta-magic-sqr2	304	0	0	0	0
magic-square-5	1704	1075	1704	1704	1704

Table 2: Comparative numbers of failures

`eq10` is a well known FD benchmark, involving 7 variables and 10 linear equalities. `magic-square-n` is a scalable arithmetic puzzle involving $2n+2$ linear equalities, n^2 variables and an `alldifferent` constraint. `alpha`, `crypta`, `crypta-magic-sqr`, and `crypta-magic-sqr2` are cryptarithmetic puzzles instances involving an `alldifferent` constraint and respectively 20, 3, 7, and 8 linear equalities and 26, 10, 10, and 11 variables. `overlap-a` is a problem consisting of 3 copies of `alpha` sharing few variables. The problem is handled as a whole system of equations whereas in `partial-overlap-a` each `alpha` copy is pre-processed separately. `alpha-rev` is the same as `alpha` but the labeling order of the variables is the inverse of the original one. `ineq-alpha-rev` is the same problem as `alpha-rev` with 3 redundant linear inequalities involving many of the variables (9,9 and 8) added. In `ineq-alpha-rev` Gaussian elimination is only performed on equalities whereas it is also performed on inequalities for `ineq-alpha-rev2`.

We compare finite domain propagation `fd`, preconditioned Gauss-Seidel interval propagation without, `ic`, and with, `ic-qz`, quasi-zero elimination, and transforming `ic` interval constraints to integer inequalities `ii` (and with quasi-zero elimination `ii-qz`). Table 1 and Table 2 shows the comparative times (including all preconditioning and transformation times) and the number of failures for finding the first solution for each of the various methods. We use a fixed labeling order to ensure that the searches are only modified by better pruning. The entry $+\infty$ indicates no solution in 15 minutes.

One can see that the time is substantially reduced as soon as we use preconditioned Gauss-Seidel, and quasi-zero elimination can also give dramatic improvements. The move to integer inequalities is always worthwhile, except for the smallest example `eq10` where the overhead of three propagators for one original constraint is not repaid. Note the mapping to integer inequalities reduces the benefits of quasi-zeros elimination. `ineq-alpha-rev2` shows that applying the approach to inequalities is also beneficial. `partial-overlap-a` illustrates that handling dense subsystems independently can be worthwile.

Our approach is advantageous when the system after preconditioning has not too many more non-(quasi-)zero coefficients than the original system. Then the better pruning can substantially reduce search. In the case of non-dense systems such as `magic-square-n`, the resulting systems have many more non-zero coefficients, and create weak propagators that gain no benefit and may cost significantly.

5. CONCLUSION

In summary, our approach offers substantial benefits when propagating dense systems of integer linear equations. Using the mapping to integer inequalities we do not need to use interval propagation, though it is still required for building the inequalities, and lose little pruning. Quasi-zero elimination is also effective in speeding up propagation without losing much pruning, although less so in the integer case. We need to investigate further exactly when the approach should be applied, and how it should be applied (to what subsystems).

6. REFERENCES

[1] C. K. Chiu and J. H. M. Lee. Interval linear constraint solving using the preconditioned interval gauss-seidel method. In *Twelfth International Conference on Logic Programming*, pages 17–31. MIT Press, 1995.

[2] ECLiPSe. `http://eclipse.crosscoreop.com/eclipse/`.

[3] F. Goualard. Gaol reference manual. `http://sourceforge.net/projects/gaol/`.

[4] W. Harvey and P. Stuckey. Improving linear constraint propagation by changing constraint representation. *Constraints*, 8(2):173–207, 2003.

[5] F. Henderson et al. The mercury language reference manual. `http://www.mercury.cs.mu.oz.au`.

[6] R. Moore. *Interval Arithmetic*. Prentice-Hall, Englewood Cliffs (NJ), USA, 1966.

[7] A. Neumaier. *Interval Methods for Systems of Equations*, volume 37 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, UK, 1990.

[8] OPL Studio. `www.ilog.com/products/oplstudio/`.

[9] K. Shen and J. Schimpf. Eplex: Harnessing mathematical programming solvers for constraint logic programming. In *Proceedings of Principles and Practice of Constraint Programming*, number 3709 in LNCS, pages 622–636. Springer Verlag, 2005.

[10] C. Solnon. Cooperation of LP solvers for solving MILPs. In *Proceedings International Conference on Tools for Artificial Intelligence*, pages 240–247. IEEE Press, 1997.