A specialized, incremental solved form algorithm for systems of linear inequalities

Christian Holzbaur
Austrian Research Institute for Artificial Intelligence, and
Department of Medical Cybernetics and Artificial Intelligence
University of Vienna
Freyung 6, A-1010 Vienna, Austria
email: christian@ai.univie.ac.at
voice: +43 1 53532810, fax: +43 1 5320652

TR-94-07

Abstract

We present a computationally improved incremental solved form algorithm for systems of linear equations and inequalities. The algorithm is of the pivotal algebra type. It benefits (computationally) from a specialization of the classical Simplex algorithm that treats inequalities of dimension one, i.e. of the shape $kx + d \leq 0$, special. In particular, the introduction of a slack variables is avoided in this case, which results in a basis that consists of higher dimensional inequality constraints only. Although the classical results concerning the complexity results for the Simplex algorithm apply, in particular in the worst case, the specialization is justified on the basis that even in the unlikely case that the special cases should not occur in practical programs, the average complexity is not higher than that of the classical algorithm. The proposed algorithm matches and advances current activities in the CLP area that try to restrict the use of general and expensive decision methods to the cases where they are unavoidable.

Keywords: Constraint Logic Programming, Implementation, Linear Programming

1 Introduction

[Imbert et al. 93] propose a syntactically partitioned solved form for the incremental solution for systems of linear equalities, inequalities (strict or non-strict), and disequalities in a CLP context. The basic idea of the approach is the restriction of the use of general and expensive decision methods to the cases where they are unavoidable. The method features efficient dereferencing by considering subclasses of solved forms and efficient testing for inconsistencies between equality and disequality constraints, where the detection of fixed variables plays an important role. Our work, as described in this paper, continues and complements the very same idea.

We introduce an incremental, specialized variant of the Simplex algorithm [Dantzig 63], that also exploits a partitioned solved form. The idea proper has been realized a long time ago in the area of linear programming under the name of *bounded variable linear programs* [Murty 76].

In bounded variable linear programs, some or all variables are restricted to lie within individual finite lower and upper bounds. Such problems can of course be solved by including all bound restrictions as constraints, i.e. rows in the simplex tableau. The advantage of keeping them out of the tableau is that the size of the working basis is smaller.

Our motivation for developing specialized versions of general decision methods is the same as in [Imbert et al. 93]: We hope that the syntactically restricted forms of constraints will indeed occur frequently in actual CLP programs. Complementing this common sense argument, we claim justification for the specialization by the fact that Branch and Bound algorithms, which address mixed integer linear optimization problems, add such simple constraints in the course of their execution.

2 Solved form for inequalities over bounded variables

We formalize bounded variable linear programs as:

$$\begin{array}{rll} Minimize & cx & \\ subject\ to\ (1.a) & Ax = b & \\ (1.b) & l_j \leq x_j \leq u_j & for\ j \in J \\ (1.c) & x_i\ unrestricted & for\ i \notin J \end{array} \tag{1}$$

Where $Ax = b$ denotes the subset of the constraints $\{c_i | dim(c_i) > 1\}$, and inequalities have been transformed into equations through the introduction of slack variables. Strict inequalities are dealt with the same way as in [Imbert et al. 93]: For the Simplex algorithm all inequalities are non-strict. The strictness is enforced through appropriate disequations. Sign restrictions on slack variables are recorded in (1.b). J is the index set for bounded

variables. Note that if a variable has only one bound instead of two, it is still in (1.b). In order to decide whether a set of linear inequalities has an non-empty solution set, we can either employ what has been termed Simplex Phase 1, which works with an artificial objective function cx, or we can utilize a result by [Orden 71], who gives an equivalent algorithm that avoids the artificial row. Before we describe the application of Orden's method to our problem, we need a basic result from polyhedral theory [Murty 76]: A feasible solution $\vec{x}$ of (1) is a BFS[1] iff the set

$$\{A_{,j} : j \in J, l_i \leq x_j \leq u_j\} \cup \{A_{,j} : j \notin J\}$$

is linearly independent. A working basis for (1) is a square, nonsingular sub matrix of A of order m. Variables associated with column vectors of the working basis will be called *basic* variables. All other variables will be called *non-basic* variables. It is clear that a feasible solution $\vec{x}$ is an extreme point in the solution space, iff there exists a corresponding working basis with the property that:

1. all non-basic variables are either *at* their lower or upper bound
2. the basic variables are *within* their bounds

The working basis, together with conditions 1 and 2 from above, constitutes our proposed solved form for linear inequalities over bounded variables. Note that the solved form is *not* unique.

Example:

Input constraints	Solved form
$x1 + x2 + 2x3 \leq 4,$ $3x2 + 4x3 \leq 6,$ $0 \leq x1, x1 \leq 2,$ $0 \leq x2, x2 \leq 9,$ $0 \leq x3$	$x1_{[0,2]} = 1 + \frac{1}{2}x2 + \frac{1}{2}s2 - s1$ $x3_{[0,\_]} = \frac{3}{2} - \frac{3}{4}x2 - \frac{1}{4}s2$ } $basis$ $s1_{[\overline{0},\_]}$ $s2_{[\overline{0},\_]}$ $x2_{[\overline{0},9]}$

Notational conventions: $x_{[0,2]}$ means that $x1$ has a (non-strict) lower bound of zero and a (non-strict) upper bound of two. An unspecified bound is denoted as in $x3_{[0,\_]}$, where we have no finite upper bound. The active bound of non-basic variables is denoted by overlining as in $x2_{[\overline{0},9]}$. If you insert the values for the active bounds into the rhs of the basic

[1] Basic Feasible Solution

variables $x1, x3$, you will find that the resulting values for $x1, x3$ are within the respective bounds. Note that only the two higher dimensional inequalities led to the introduction of slack variables $s1, s2$.

Finding the solved form of a bounded variable linear program can be rephrased as a search problem, where we have:

1. A given initial state, consisting of a system where the solved form invariants may be violated.
2. The specification of a solution state through the solved form invariants.
3. The operators:
   (a) $pivot(x_i, x_j)$
   (b) $toggle\_active\_bound(x_i)$ for non-basic variables

The non-determinism in the selection of the the operators and their arguments can be removed by the same rules that are employed in the original Simplex algorithm: A variable to enter the basis must have the proper sign in the objective function, and the leaving variable corresponds to the row in the working basis that imposes the tightest constraint on the entering variable.

A violation of the solved form is always detected by locating a basic variable that is out of its bounds. To repair such a row, we interpret the linear combination of variables that defines the basic variable as objective function [Orden 71]. If the current evaluation of the rhs is beyond the upper bound of the basic variable, we will try to decrease the objective function. If the rhs evaluation is below the lower bound of the basic variable, we will increase the objective function until it is inside the bound.

3 The incremental solved form algorithm

The purpose of the algorithm to be developed in this section is to maintain the solved form invariants which will be threatened through the addition of constraints to the current solved form. The constraints fed sequentually to our algorithm are arithmetically simplified into one of the two following forms:

$$\sum k_i x_i + b \leq 0$$
$$\sum k_i x_i + b = 0$$

Definition 1 *The variables x_i will be of the following types:*

1. $x_{[\_,\_]}$
2. $x_{[lower,\_]}$
3. $x_{[\_,upper]}$
4. $x_{[lower,upper]}$
5. $x_{[\overline{lower},\_]}$
6. $x_{[\_,\overline{upper}]}$
7. $x_{[\overline{lower},upper]}$
8. $x_{[lower,\overline{upper}]}$

Variables start their lives as of type $x_{[\_,\_]}$. Eventually they acquire a lower and an upper bound. Once they have bounds, these might get activated individually and exclusively as the variable plays the role of an independent variable in the solved form.

The algorithm performs of the following steps with a new constraint c_i that is to be added to a set of constraints in solved form C_{sf}. Assume that the solved form C_{sf} is available to all procedures, i.e. all procedures are curried with regard to this parameter.

Definition 2 *add_constraint*(c_i)

1. *If* $dim(c_i) = 0$, c_i *is trivially decidable and does not change the solved form.*
2. *If* c_i *is an inequality perform enter_inequality*(c_i)
3. *If* c_i *is an equation perform enter_equation*(c_i)

Definition 3 *enter_equation*(c_i)*:*

1. *Let* $c_i' = dereference(c_i)$
2. *If* $dim(c_i) = 0$, c_i *is trivially decidable and does not change the solved form.*
3. *If there is an unbounded (type 1) variable among the* x_i*, solve for it:*

$$x_i = -\frac{1}{k_i} \sum_{i \neq j} k_j x_j + b$$

Occurrences of x_i *on the rhs of the solved form are replaced with this definition. The constraint* $x_i = -\frac{1}{k_i} \sum_{i \neq j} k_j x_j + b$ *is added to the solved form.*

4. *If there is no unbounded (type 1) variable among the x_i, solve for an arbitrary bounded variable: $x_i = -\frac{1}{k_i}\sum_{i \neq j} k_j x_j + b$. x_i will now become a basic variable, which has to be reflected by its type. A type 6 variable would be changed into a type 3 variable, for example. Perform activate_rhs_dec(x_i). Backsubstitute as in the previous step. Some of the affected basic variables x_J might be out of their range now — perform repair(x_j) on them.*

Definition 4 *repair(x_j):*

1. *If eval_rhs(x_j) > upper(x_j) perform dec_till_below(x_j, upper(x_j))*
2. *If eval_rhs(x_j) < lower(x_j) perform inc_till_above(x_j, lower(x_j))*

We only give the definition of *dec_till_below* here, *inc_till_above* is its dual.

Definition 5 *dec_till_below(x_j, Bound):*

1. *If eval_rhs(x_j) < Bound return true*
2. *else let Status = dec_step(x_j).*
   (a) *Status = applied, return dec_till_below(x_j, Bound).*
   (b) *Status = unlimited(x_k), perform pivot(x_k, x_j), return true.*
   (c) *Status = optimum:*
      i. *eval_rhs(x_j) > Bound, return false.*
      ii. *eval_rhs(x_j) = Bound, perform enter_equation(x_j = Bound), return true.*
      iii. *eval_rhs(x_j) < Bound, record the tightened bound on x_j, return true.*

Definition 6 *dec_step(x_j): the basic variable x_j is defined as: $x_j = \sum k_i x_i + b$, look at each x_i in turn:*

1. *If type(x_i) = [_, _], return unlimited(x_i)*
2. *If type(x_i) = [_, $\overline{upper}$] and $k_i > 0$ let $x_j = lb(x_i)$ be the basic variable imposing the tightest lower bound on x_i:*
   (a) *If there is no lower bound imposed, return unlimited(x_i).*
   (b) *Else perform pivot(x_j, x_i), return applied.*
3. *If type(x_i) = [lower, $\overline{upper}$] and $k_i > 0$ let $x_j = lb(x_i)$ be the basic variable imposing the tightest lower bound on x_i:*
   (a) *If $i = j$, transform x_i into $x_{[\overline{lower}, upper]}$, return applied.*

(b) *Else perform* $pivot(x_j, x_i)$, *return applied.*

4. *If* $type(x_i) = [\overline{lower}, \_]$ *and* $k_i < 0$ *let* $x_j = ub(x_i)$ *be the basic variable imposing the tightest upper bound on* x_i*:*

   (a) *If there is no upper bound imposed, return* $unlimited(x_i)$.

   (b) *Else perform* $pivot(x_j, x_i)$, *return applied.*

5. *If* $type(x_i) = [\overline{lower}, upper]$ *and* $k_i < 0$ *let* $x_j = ub(x_i)$ *be the basic variable imposing the tightest upper bound on* x_i*:*

   (a) *If* $i = j$, *transform* x_i *into* $x_{[lower,\overline{upper}]}$, *return applied.*

   (b) *Else perform* $pivot(x_j, x_i)$, *return applied.*

6. *If none of the previous cases applied to any* x_i, *return optimum.*

Definition 7 $lb(x_i)$*: determine the basic variable that imposes the tightest lower bound on* x_i. *Map over all basic variables, return* x_b *where the quantity lb has its maximum. Let* k *be the coefficient of* x_i *in* $x_b = \sum k_j x_j + b$ *and* $r = eval\_rhs(x_b)$

1. x *is of type* $x_{[lower,\_]}$ *and* $k > 0$ *let* $lb = \frac{lower - r}{k}$
2. x *is of type* $x_{[\_,upper]}$ *and* $k < 0$ *let* $lb = \frac{upper - r}{k}$
3. x *is of type* $x_{[lower,uper]}$

   (a) $k > 0$ *let* $lb = \frac{lower - r}{k}$

   (b) $k < 0$ *let* $lb = \frac{upper - r}{k}$

Note that in order to guarantee finite termination of the modified Simplex algorithm, we brake ties in the standard way by extending the definition of maximum to that of a lexicographic maximum [Murty 76]. If x_b later leaves the basis, it gets a non-basic variable at the bound that determined lb above. For variables of type $x_{[lower,uper]}$ this depends on the sign of k.

Definition 8 $pivot(x_b, x_i)$*:* x_b *leaves the basis,* x_i *enters.* $x_b = \sum k_j x_j + b$, *therefore*

$$x_i = -\frac{1}{k_i} \sum_{i \neq j} k_j x_j - x_b + b$$

Record this definition for x_i *and backsubstitute into he solved form. Note that the pivot operation is not defined iff* x_i *is not among the* x_j, *i.e. has a zero coefficient in the linear form.*

Definition 9 *enter_inequality*(c_i)*:*

1. *If* $dim(c_i) = 1$*, i.e.* c_i *is of the form* $kx + b \leq 0$*, let* $Bound = -b/k$
   (a) *If* $k < 0$ *perform* $update\_lower(x, Bound)$
   (b) *If* $k > 0$ *perform* $update\_upper(x, Bound)$
2. *else let* $c_i' = dereference(c_i)$
   (a) *If* $dim(c_i') = 0$, c_i' *is trivially decidable and does not change the solved form.*
   (b) *If* $dim(c_i') = 1$*, i.e.* c_i' *is of the form* $kx + b \leq 0$*, let* $Bound = -b/k$
      i. *If* $k < 0$ *perform* $update\_lower(x, Bound)$
      ii. *If* $k > 0$ *perform* $update\_upper(x, Bound)$
   (c) *else if* c_i'*:* $\sum k_i x_i + b \leq 0$ *contains an unbounded (type 1) variable* x_i*, define*

$$x_i = -\frac{1}{k_i} \sum_{i \neq j} k_j x_j + s_{[0,\_]} + b$$

   where s *is a fresh slack variable. Record this definition in the solved form and backsubstitute* x_i*.*
   (d) *otherwise define* $s_{[\_,0]} = \sum k_i x_i + b$*, where* s *is a fresh slack variable. No need to backsubstitute because the slack variable just introduced cannot occur on any rhs in the solved form. Check if the evaluated rhs is within the slack's bound: Perform* $activate\_rhs\_dec(s)$ *and* $dec\_till\_below(s, 0)$*.*

We will give only the definition of *update_lower* here, *update_upper* is it's dual.

Definition 10 *update_lower*$(x, Bound)$*:*

1. *If* x *is a basic variable, perform* $update\_lower\_basic(x, Bound)$
2. *else perform* $update\_lower\_nonbasic(x, Bound)$

Definition 11 *update_lower_basic*$(x, Bound)$*:*

1. x *of type* $x_{[\_,\_]}$ *is transformed into* $x_{[Bound,\_]}$*.*
   (a) *If there is an unbounded variable among the* x_i *solve for it:* $x_i = -\frac{1}{k_i} \sum_{i \neq j} k_j x_j + b$ *and backsubstitute.*
   (b) *else perform* $activate\_rhs\_inc(x)$*, perform* $repair(x)$*.*
2. x *of type* $x_{[lower,\_]}$ *is transformed into* $x_{[Bound,\_]}$ *if the new bound is tighter than the old one, perform* $repair(x)$*.*

3. *x of type* $x_{[\_,upper]}$ *is transformed into:* $x_{[Bound,upper]}$ *if the lower bound does not exceed the upper bound, perform* $repair(x)$*, else fail. If the lower bound meets the upper bound, perform* $enter\_equation(x = Bound)$.

4. *x of type* $x_{[lower,upper]}$*: perform* $repair(x)$ *is the new bound is tighter than the old one and does not exceed the upper bound. If the lower bound meets the upper bound, perform* $enter\_equation(x = Bound)$.

Definition 12 $activate\_rhs\_inc(x)$*: for each* x_i *in* $x = \sum k_i x_i + b$*:*

1. *if* x_i *is of type* $x_{[lower,\_]}$*, change the type of* x_i *into* $x_{[\overline{lower},\_]}$
2. *if* x_i *is of type* $x_{[\_,upper]}$*, change the type of* x_i *into* $x_{[\_,\overline{upper}]}$
3. *if* x_i *is of type* $x_{[lower,upper]}$
   (a) *If* $k_i > 0$ *change the type of* x_i *into* $x_{[lower,\overline{upper}]}$
   (b) *If* $k_i < 0$ *change the type of* x_i *into* $x_{\overline{[lower},upper]}$

Definition 13 $activate\_rhs\_dec(x)$*: for each* x_i *in* $x = \sum k_i x_i + b$*:*

1. *if* x_i *is of type* $x_{[lower,\_]}$*, change the type of* x_i *into* $x_{[\overline{lower},\_]}$
2. *if* x_i *is of type* $x_{[\_,upper]}$*, change the type of* x_i *into* $x_{[\_,\overline{upper}]}$
3. *if* x_i *is of type* $x_{[lower,upper]}$
   (a) *If* $k_i < 0$ *change the type of* x_i *into* $x_{[lower,\overline{upper}]}$
   (b) *If* $k_i > 0$ *change the type of* x_i *into* $x_{\overline{[lower},upper]}$

Definition 14 $update\_lower\_nonbasic(x, Bound)$*:*

1. *x of type* $x_{[\_,\_]}$ *is transformed into:* $x_{[Bound,\_]}$
2. *x of type* $x_{[lower,\_]}$ *is transformed into:* $x_{[Bound,\_]}$ *if the new bound is tighter than the old one.*
3. *x of type* $x_{[\_,upper]}$ *is transformed into:* $x_{[Bound,upper]}$ *if the lower bound does not exceed the upper bound, else fail. If the lower bound meets the upper bound, perform* $enter\_equation(x = Bound)$.
4. *x of type* $x_{[lower,upper]}$ *is transformed into:* $x_{[Bound,upper]}$ *if the new bound is tighter than the old one. Fail in case the new bound exceeds the upper bound. If the lower bound meets the upper bound, perform* $enter\_equation(x = Bound)$.

5. *x of type* $x_{[\_,\overline{upper}]}$ *is transformed into* $x_{[lower,\overline{upper}]}$ *if the lower bound does not exceed the upper bound, else fail. If the lower bound meets the upper bound, perform* $enter\_equation(x = Bound)$.

6. *x of type* $x_{[lower,\overline{upper}]}$ *is transformed into* $x_{[Bound,\overline{upper}]}$ *if the new bound is tighter than the old one. Fail in case the new bound exceeds the upper bound. If the lower bound meets the upper bound, perform* $enter\_equation(x = Bound)$.

7. *x of type* $x_{[\overline{lower},\_]}$*: If the new bound is tighter than the old one, determine the tightest upper bound imposed through the basic variables:* $x_j = ub(x)$. *If the new bound on x is within the bound imposed by* x_j*, we are done*[2]*. Otherwise perform* $pivot(x_j, x)$ *and perform* $inc\_till\_above(x, Bound)$.

8. *x of type* $x_{[\overline{lower},upper]}$*: Perform the same steps as for type* $[\overline{lower}, \_]$. *If the lower bound meets the upper bound, perform* $enter\_equation(x = Bound)$.

3.1 Remarks

3.1.1 Termination

All iterations mentioned in the algorithms either run over a linear form $\sum k_i x_i + b$, or over the basis of the solved form. Both data structures a apparently finite. The pivot and backsubstitution steps individually terminate because A in $Ax = b$ is of finite order. The pivot selection rule, in particular the selection of a lexicographic minimum/maximum, guarantees productive pivot steps, and hence finite termination.

3.1.2 Complexity

The Simplex algorithm is known to be of exponential complexity in the worst case. [Orden 71] used a simple statistical model to explain the long observed fact that the number of pivot steps to be *expected* per row is in the order of the number of rows. Specifically he notes that the expected ratio of the number of pivots to m, the number of rows in the system, is $\log_e m$.

In our variant of the Simplex algorithm, a series of full pivot steps are the worst thing that can happen with regard to complexity. Our design is such that we can avoid them frequently. The type system facilitates the minimization of the number of tests to be performed to detect trivial redundancy and feasibility. Updates on inactive bounds of non-basic variables are of $O(1)$. Trivially non-redundant, feasible updates on active bounds of non-basic variables are at least of $O(n)$ where n is the size of the basis, maybe followed by a series of pivot steps. This is the reason for delaying the activation of the bounds of

[2] $max(x) \geq$ the upper bound imposed by x_j

non-basic variables. An eager strategy would fix the activity as soon a variable is known to be non-basic, i.e. it appears on the rhs of a basic variable.

3.1.3 Detection of fixed variables

There is no need for algorithmic twists in order make our algorithm detect fixed variables. The trivial detection, where a lower bound meets an upper bound, is of $O(1)$. The more subtle and expensive detections take place in the procedures *dec_till_below* and *inc_till_above* where we try to decrement/increment the rhs of a basic variable in order to satisfy the bounds. Note that the test in step 1 is strict, i.e. the procedure does not stop to decrement/increment *at* the bound, which would be sufficient regarding satisfiability, but goes ahead at least one step further. It turns out that this is sufficient to find all fixed variables. In oder words, if we do not have to identify fixed variables, we can save a few pivot steps, and in the worst case exponential many, by using a non-strict test in step 1 in *dec_till_below* and *inc_till_above*.

4 Implementation

The algorithm as described in this article has been implemented in Prolog. It now replaces the linear solver kernel of the CLP($\Re$) and CLP($\mathcal{Q}$) systems distributed with DMCAI-Clp, a CLP system based on extensible unification [Holzbaur 92, Holzbaur 92, Holzbaur 93], which initially employed a classical Simplex algorithm without the specialization for lower dimensional constraints. The rich type system in the new algorithm and the choice of Prolog as implementation language led to a very tabular coding style. In fact, the executable specifications corresponding to the abstract procedures in this text, are somewhat more compact than the text proper. This positive experience seems to justify a further type refinement: We will use extra types for variables with lower or upper bounds of zero in the future.

In the preceding text we spoke of the evaluation of the rhs of basic variables, where the active bounds of the non-basic variables would be substituted for the non-basic variables. This evaluation is performed relatively frequently. In the actual implementation, this value is therefore initialized when a variable gets a bound activated and maintained across pivots and other operations that change the solved form. Technically this is achieved with no algorithmic fuzz through an artificial column in the tableau, the same way the constant term in the linear forms is dealt with.

Acknowledgments

This research was sponsored by the Austrian *Fonds zur Förderung der Wissenschaftlichen Forschung* under grant P9426-PHY. Financial support for the Austrian Research Institute for Artificial Intelligence is provided by the Austrian Federal Ministry for Science and Research.

References

[Dantzig 63] Dantzig G.B.: Linear Programming and Extensions, Princeton University Press, Princeton, NJ, 1963.

[Holzbaur 92] Holzbaur C.: Metastructures vs. Attributed Variables in the Context of Extensible Unification, in Bruynooghe M. & Wirsing M.(eds.), Programming Language Implementation and Logic Programming, Springer, LNCS 631, pp.260-268, 1992.

[Holzbaur 92] Holzbaur C.: A High-Level Approach to the Realization of CLP Languages, in Proceedings of the JICSLP92 Post-Conference Workshop on Constraint Logic Programming Systems, Washington D.C., 1992.

[Holzbaur 93] Holzbaur C.: Extensible Unification as Basis for the Implementation of CLP Languages, in Baader F., et al., Proceedings of the Sixth International Workshop on Unification, Boston University, MA, TR-93-004, pp.56-60, 1993.

[Imbert et al. 93] Imbert J.-L., Cohen J., Weeger M.-D.: An Algorithm for Linear Constraint Solving: Its Incorporation in a Prolog Meta-Interpreter for CLP, in Special Issue: Constraint Logic Programming, Journal of Logic Programming, 16(3&4), 235-253, 1993.

[Murty 76] Murty K.G.: Linear and Combinatorial Programming, Wiley, New York, 1976.

[Orden 71] Orden A.: On the Solution of Linear Equation/Inequality Systems, Mathematical Programming (1)137-152, 1971.

Appendix

We step through a slightly larger example:

$$\begin{aligned}
x1 + x3 - x4 + x5 + 2x6 + x7 &\leq 6,\\
x2 + x4 - 2x5 + x6 - 2x7 &\leq 4,\\
x3 - x4 + 2x6 + x7 &\leq 1,\\
0 \leq x1, x1 &\leq 6,\\
0 \leq x2, x2 &\leq 6,\\
0 \leq x3,&\\
0 \leq x4, x4 &\leq 4,\\
0 \leq x5, x5 &\leq 2,\\
0 \leq x6, x6 &\leq 10,\\
0 \leq x7,&\\
minimize(-3x1 + 4x2 + 2x3 - 2x4 - 14x5 + 11x6 - 5x7)&
\end{aligned}$$

The first three inequalities lead to the introduction of (positive) slack variables. There are three basic variables.

$$\begin{aligned}
&x5_{[_,_]} = 5 - x1 + s3 - s1\\
&x6_{[_,_]} = \tfrac{16}{5} + \tfrac{1}{5}x4 - \tfrac{2}{5}x3 - \tfrac{1}{5}x2 - \tfrac{2}{5}x1 - \tfrac{1}{5}s2 - \tfrac{2}{5}s1\\
&x7_{[_,_]} = -\tfrac{27}{5} + \tfrac{3}{5}x4 - \tfrac{1}{5}x3 + \tfrac{2}{5}x2 + \tfrac{4}{5}x1 - s3 + \tfrac{2}{5}s2 + \tfrac{4}{5}s1\\
&s1_{[0,_]}, s2_{[0,_]}, s3_{[0,_]},\\
&x1_{[_,_]}, x2_{[_,_]}, x3_{[_,_]}, x4_{[_,_]}
\end{aligned}$$

The next inequalities lead to applications of:

1. $update\_lower\_nonbasic(x1, 0)$
2. $update\_upper\_nonbasic(x1, 6)$
3. $update\_lower\_nonbasic(x2, 0)$
4. $update\_upper\_nonbasic(x2, 6)$
5. $update\_lower\_nonbasic(x3, 0)$
6. $update\_lower\_nonbasic(x4, 0)$
7. $update\_upper\_nonbasic(x4, 4)$

The next interesting event is $update\_lower\_basic(x5, 0)$, leading to $activate\_rhs\_inc(x5)$, which activates the lower bounds of $x1, s3, s1$, and $5 = eval\_rhs(x5) > 0$, the new lower bound on $x5$.

$$\begin{aligned}
&x5_{[0,_]} = 5 - x1 + s3 - s1\\
&x6_{[_,_]} = \tfrac{16}{5} + \tfrac{1}{5}x4 - \tfrac{2}{5}x3 - \tfrac{1}{5}x2 - \tfrac{2}{5}x1 - \tfrac{1}{5}s2 - \tfrac{2}{5}s1\\
&x7_{[_,_]} = -\tfrac{27}{5} + \tfrac{3}{5}x4 - \tfrac{1}{5}x3 + \tfrac{2}{5}x2 + \tfrac{4}{5}x1 - s3 + \tfrac{2}{5}s2 + \tfrac{4}{5}s1\\
&s1_{[\overline{0},_]}, s2_{[0,_]}, s3_{[\overline{0},_]},\\
&x1_{[\overline{0},6]}, x2_{[0,6]}, x3_{[0,_]}, x4_{[0,4]}
\end{aligned}$$

$update\_lower\_basic(x5, 2)$ necessitates a call to $dec\_till\_below(x5, 2)$. After one pivot step $pivot(x5, x1)$, the rhs of $x5$ evaluates to zero:

$$\begin{array}{l} x1_{[0,6]} = 5 - x5 + s3 - s1 \\ x6_{[\_,\_]} = \frac{6}{5} + \frac{2}{5}x5 + \frac{1}{5}x4 - \frac{2}{5}x3 - \frac{1}{5}x2 - \frac{2}{5}s3 - \frac{1}{5}s2 \\ x7_{[\_,\_]} = -\frac{7}{5} - \frac{4}{5}x5 + \frac{3}{5}x4 - \frac{1}{5}x3 + \frac{2}{5}x2 - \frac{1}{5}s3 + \frac{2}{5}s2 \\ s1_{[\overline{0},\_]}, s2_{[0,\_]}, s3_{[\overline{0},\_]}, \\ x2_{[0,6]}, x3_{[0,\_]}, x4_{[0,4]}, x5_{[\overline{0},2]} \end{array}$$

$update\_lower\_basic(x6, 0)$ necessitates a call to $inc\_till\_above(x6, 0)$. The upper bound of $x4$ and the lower bounds of $x3, x2, s2$ are activated, the rhs of $x6$ evaluates to $2 > 0$. The bound set by $update\_upper\_basic(x6, 10)$ is likewise satisfied by this rhs.

$update\_lower\_basic(x7, 0)$ needs not activate any non-basic variables, and we have the solved form:

$$\begin{array}{l} x1_{[0,6]} = 5 - x5 + s3 - s1 \\ x6_{[0,10]} = \frac{6}{5} + \frac{2}{5}x5 + \frac{1}{5}x4 - \frac{2}{5}x3 - \frac{1}{5}x2 - \frac{2}{5}s3 - \frac{1}{5}s2 \\ x7_{[0,\_]} = -\frac{7}{5} - \frac{4}{5}x5 + \frac{3}{5}x4 - \frac{1}{5}x3 + \frac{2}{5}x2 - \frac{1}{5}s3 + \frac{2}{5}s2 \\ s1_{[\overline{0},\_]}, s2_{[\overline{0},\_]}, s3_{[\overline{0},\_]}, \\ x2_{[\overline{0},6]}, x3_{[\overline{0},\_]}, x4_{[0,\overline{4}]}, x5_{[\overline{0},2]} \end{array}$$

The linear form to be minimized reads after dereferencing:

$$Min = \frac{26}{5} - \frac{13}{5}x5 - \frac{14}{5}x4 - \frac{7}{5}x3 - \frac{1}{5}x2 - \frac{32}{5}s3 - \frac{21}{5}s2 + 3s1$$

which evaluates to -6 at the currently active bounds. $x5$ is suitable to decrement this value further because it occurs with a negative coefficient, and moving it from the currently active lower bound 0 towards the upper bound will yield the desired effect. The most constraining basic variable is $x7$. After $pivot(x7, x5)$, the linear form evaluates to $-\frac{37}{4}$:

$$\begin{array}{l} x1_{[0,6]} = \frac{27}{4} + \frac{5}{4}x7 - \frac{3}{4}x4 + \frac{1}{4}x3 - \frac{1}{2}x2 + \frac{4}{5}s3 - \frac{1}{2}s2 - s1 \\ x5_{[0,2]} = -\frac{7}{4} - \frac{5}{4}x7 + \frac{3}{4}x4 - \frac{1}{4}x3 + \frac{1}{2}x2 - \frac{1}{4}s3 + \frac{1}{2}s2 \\ x6_{[0,10]} = \frac{1}{2} - \frac{1}{2}x7 + \frac{1}{2}x4 - \frac{1}{2}x3 - \frac{1}{2}s3 \\ Min = \frac{39}{4} + \frac{13}{4}x7 - \frac{19}{4}x4 - \frac{3}{4}x3 - \frac{3}{2}x2 - \frac{23}{4}s3 - \frac{11}{2}s2 + 3s1 \\ s1_{[\overline{0},\_]}, s2_{[\overline{0},\_]}, s3_{[\overline{0},\_]}, \\ x2_{[\overline{0},6]}, x3_{[\overline{0},\_]}, x4_{[0,\overline{4}]}, x7_{[\overline{0},\_]} \end{array}$$

Now $x3$ is suitable to further decrease the rhs evaluation of Min. The most constraining row is $x5$: $pivot(x5, x3)$ yields en evaluation of -13. The next pivot is $pivot(x6, x2)$, which does not change the current minimum of -13. The next non-basic candidate for a pivot step is $x5$. It turns out that its own upper bound is the tightest one. Therefore the active

bound is toggled from lower to upper, giving a current minimum of -19:

$$
\begin{array}{l}
x1_{[0,6]} = 5 - x5 + s3 - s1 \\
x2_{[0,6]} = 4 + 2x7 - x6 + 2x5 - x4 - s2 \\
x3_{[0,\_]} = 1 - x7 - 2x6 + x4 - s3 \\
Min = 3 + x7 + 3x6 - 3x5 - 4x4 - 5s3 - 4s2 + 3s1 \\
s1_{[\overline{0},\_]}, s2_{[\overline{0},\_]}, s3_{[\overline{0},\_]}, \\
x4_{[0,\overline{4}]}, x5_{[0,\overline{2}]}, x6_{[\overline{0},10]}, x7_{[\overline{0},\_]}
\end{array}
$$

The next pivots are $pivot(x1, s3)$ with a yield of -34, $pivot(x2, s2)$ with a yield of -50, $pivot(x3, x7)$ with a yield of -64, and $pivot(s3, x1)$ with a yield of -70, optimum:

$$
\begin{array}{l}
s2_{[0,\_]} = 6 - 5x6 + 2x5 + x4 - 2x3 - x2 - 2s3 \\
x1_{[0,6]} = 5 - x5 + s3 - s1 \\
x7_{[0,\_]} = 1 - 2x6 + x4 - x3 - s3 \\
Min = -20 + 21x6 - 11x5 - 7x4 + 7x3 + 4x2 + 2s3 + 3s1 \\
s1_{[\overline{0},\_]}, s3_{[\overline{0},\_]}, \\
x2_{[\overline{0},6]}, x3_{[\overline{0},\_]}, x4_{[0,\overline{4}]}, x5_{[0,\overline{2}]}, x6_{[\overline{0},10]}
\end{array}
$$

Solving $-20 + 21x6 - 11x5 - 7x4 + 7x3 + 4x2 + 2s3 + 3s1 = -70$ fixes the values of all variables:

$$x1 = 3, x2 = 0, x3 = 0, x4 = 4, x5 = 2, x6 = 0, x7 = 5, s1 = 0, s2 = 14, s3 = 0$$