BOUND PROPAGATION FOR LINEAR INEQUALITIES REVISITED

KEVIN K. H. CHEUNG

Abstract. In 2011, Korovin and Voronkov (*Proceedings of the 23rd International Conference on Automated Deduction*, vol. 6803 of *Lecture Notes in Computer Science*, pp. 369–383) proposed a method based on bound propagation for solving systems of linear inequalities. In this paper, an alternate description of their algorithm which also incorporates an addition that returns a certificate of infeasibility in the case when the system has no solution is provided. It is shown that the running time of the algorithm is exponential in the number of variables in the worst case. A method for using the algorithm to solve linear programming problems without having to specify the dual is proposed.

1. Introduction

In 2011, Korovin and Voronkov [4] introduced an algorithm, which we refer to as BPA-KV in this paper, for solving systems of linear inequalities based on bound-propagation. Subsequently, a description of an implementation as part of a first-order theorem prover called Vampire [5] appeared in [2], which also provides a more intuitive description of the algorithm.

The three key ingredients of BPA-KV are: bound propagation, variable fixing, and conflict analysis. The algorithm searches for a solution by recursively fixing variables to values chosen within the currently known bounds. Before each value is fixed, the bounds are improved (if possible) via bound propagation. When bound propagation results in a variable with an upper bound contradicted by a lower bound, the algorithm backtracks and adjust the bounds for a previously (but not necessarily the latest) fixed variable. Such a conflict always results in an improved bound.

These three ingredients correspond to ideas for resolution algorithms for Boolean satisfiability. Korovin and Voronkov [4] gave an exact correspondence of the parallel ideas. However, a nontrivial technicality that arises in the case of linear inequalities is that bound propagation could fail to terminate. To guarantee termination, BPA-KV stops bound propagation as soon as a bound has improved D times for some positive integer D chosen ahead of time.

In this paper, we provide an alternate description of BPA-KV and prove that the running time of the algorithm is exponential in the worst case. We also propose an extended mode which turns the algorithm into a linear programming solver without the need to specify the dual problem. Our description of BPA-KV makes the recursive nature of the algorithm apparent by removing the explicit use of a stack in the original description. It also contains a minor refinement that returns explicitly a certificate of infeasibility in the case when the system has no solution. To this end, we describe all inferences performed by the algorithm as taking nonnegative linear combinations of the original inequalities in the system. As a byproduct, our proofs of termination and correction appear somewhat simpler than the original proofs and cast in a language that is perhaps more familiar to readers with a background in mathematical programming rather than in computational logic.

Key words. bound propagation, linear inequalities, linear programming, Farkas certificate, duality.

2. Preliminaries

Let $\overline{\mathbb{R}}$ denote $\mathbb{R} \cup \{-M, M\}$ where M represents ∞ or a very large unspecified constant. We use the following arithmetic conventions for $\pm\infty$:

- $\infty + a = a + \infty = \infty$ for all $a \in \mathbb{R} \cup \{\infty\}$.
- $-\infty + a = a + (-\infty) = -\infty$ for all $a \in \mathbb{R} \cup \{-\infty\}$.
- $0 \cdot \infty = \infty \cdot 0 = 0$.
- $\alpha \cdot \infty = \infty \cdot \alpha = \infty$ and $\alpha \cdot (-\infty) = (-\infty) \cdot \alpha = -\infty$ for all positive $a \in \mathbb{R}$.

Note that if α and β are of opposite signs, then $\alpha\infty + \beta\infty$ is undefined and such an operation is forbidden. Hence, the above list is all the possible operations involving $\pm\infty$ in this paper.

When M represents a very large unspecified constant, we will be working with numbers of the form $\alpha M + \beta$ where $\alpha, \beta \in \mathbb{R}$ with the convention that $\gamma < \delta M$ for all $\gamma, \delta \in \mathbb{R}$ where $\delta > 0$. Such numbers can be implemented as pairs of real numbers; that is, $\alpha M + \beta$ can be represented as (α, β), in which case $(\alpha_1, \beta_1) < (\alpha_2, \beta_2)$ if and only if $(\alpha_1 < \alpha_2) \vee (\alpha_1 = \alpha_2 \wedge \beta_1 < \beta_2)$. In other words, the total order is given by the lexicographic order. Since we will not be multiplying two numbers of the form $\alpha M + \beta$ with $\alpha \neq 0$, scalar multiplication and tuple addition are sufficient to capture the arithmetic operations involving these extended real numbers in this paper.

Our algorithm will operate in either *regular mode* or *extended mode* in the following sense: In the former, only real values can be assigned to variables. In the latter, values of the form $\alpha M + \beta$ where $\alpha, \beta \in \mathbb{R}$ can be assigned to variables and be involved in addition and multiplication by a real number. As will be discussed in Section 6, the extended mode will be useful when the algorithm is used to solve a linear programming problem.

Define $0_{m\times n}$ to be the $m \times n$ matrix whose entries are all 0 and $I_{m\times n}$ to be the $m \times n$ matrix such that the (i,i)-entry is 1 for all $1 = 1, \ldots, \min(m, n)$ and all other entries are 0. For simplicity, we use I_n to denote $I_{n\times n}$.

Vectors in $\mathbb{R}^n$ are column vectors.

A literal is either a variable or the negation of a variable. For an inequality $a^\mathsf{T} x \geq \beta$ where $a \in \mathbb{R}^n$ and $\beta \in \mathbb{R}$, the left-hand side can be written as a linear combination of literals using only positive coefficients. For example, $x_1 - 2x_2 - 3x_3 \geq -1$ can be written as $x_1 + 2(-x_2) + 3(-x_3) \geq -1$ in which the literals are x_1, $-x_2$, and $-x_3$.

A *bound inequality* is an inequality of the form $l \geq \beta$ where l is a literal and $\beta \in \mathbb{R}$. If l is a variable, then the inequality provides a lower bound on the variable. If l is the negation of a variable, then the inequality provides an upper bound on the variable. For example, $-x_2 \geq -3$ imposes the upper bound 3 on the variable x_2.

In this paper, we will be working with systems of the form $Ax \geq b$ with $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^{m-2n} \times \overline{\mathbb{R}}^{2n}$ where $m > 2n$ and A is of the form $\begin{bmatrix} B \\ I_n \\ -I_n \end{bmatrix}$. Hence, the last $2n$ inequalities are bound inequalities involving the literals.

To facilitate discussion, we index the columns of A by the associated variables and the last $2n$ rows by the literals associated with the bound inequalities. The first $m - 2n$ rows of A will be indexed by the row numbers. For a variable v, A_v denotes the column of A indexed by v. Similarly, the last $2n$ entries of b are indexed by the literals associated with the bound inequalities. For a literal l, b_l denotes the entry of b indexed by l. The first $m - 2n$ entries will be indexed by the entry positions. We now illustrate these conventions.

Example 2.1. For the system

$$\begin{aligned} 3x_1 - 2x_2 &\geq 5 \\ x_1 + 4x_2 &\geq 4 \\ x_1 &\geq 1 \\ x_2 &\geq -\infty \\ -x_1 &\geq -2 \\ -x_2 &\geq -5, \end{aligned}$$

we have $A = \begin{bmatrix} 3 & -2 \\ 1 & 4 \\ 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix}$ and $b = \begin{bmatrix} 5 \\ 4 \\ 1 \\ -\infty \\ -2 \\ -5. \end{bmatrix}$. Then row 2 of A is simply the second row of A, row $-x_1$ is the second-last row of A. A_{x_2} is the second column of A, $b_{x_2} = -\infty$, and $b_{-x_1} = -2$.

The following lemma follows from the well-known Farkas' Lemma (see pp. 89 in [6], for example) :

Lemma 2.1. *Let $A \in \mathbb{R}^{m\times n}$ where $b \in \overline{\mathbb{R}}^m$. Then $Ax \geq b$ has no solution if there exists $y \in \mathbb{R}^m$ such that $y \geq 0$, $y^\mathsf{T} A = 0$, and $y^\mathsf{T} b \in \mathbb{R}$ with $y^\mathsf{T} b > 0$.*

The y in Lemma 2 is called a *certificate of infeasibility* or a *Farkas certificate.* The converse of Lemma 2 also holds as in the ordinary Farkas' Lemma but we will not such a result in this paper.

3. Bound propagation revisited

An important procedure in BPA-KV is bound propagation. The original description of bound propagation can be found in [4]. In this section, we reformulate bound propagation as a special form of linear inequality inference that results in a bound inequality.

Consider a system $Ax \geq b$ where $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^{m-2n} \times \overline{\mathbb{R}}^{2n}$, and A is of the form $\begin{bmatrix} B \\ I_n \\ -I_n \end{bmatrix}$.

We call $d \in \mathbb{R}^m$ a *literal-inducing vector with respect to A* if $d \geq 0$ and $d^\mathsf{T} Ax$ is a literal. If the matrix A is clear from the context, we simply call d a literal-inducing vector. The literal given by $d^\mathsf{T} Ax$ is said to be *induced* by d. Bound propagation can be seen as a procedure for obtaining bound inequalities through literal-inducing vectors described as follows:

Let $i \in \{1, \ldots, m-2n\}, j \in \{1, \ldots, n\}$ be such that $A_{ij} \neq 0$. Let $S^+ = \{j' : A_{ij'} > 0,\ j' \neq j\}$ and $S^- = \{j' : A_{ij} < 0,\ j' \neq j\}$. Note that S^+ and/or S^- could be empty.

Let $d \in \mathbb{R}^m$ be such that

$$d_r = \frac{1}{|A_{ij}|} \begin{cases} 1 & \text{if } r = i \\ A_{is} & \text{if } s = r - (m-n) \in S^+ \\ -A_{is} & \text{if } s = r - (m-2n) \in S^- \\ 0 & \text{otherwise.} \end{cases}$$

As $d \geq 0$, we can infer the inequality $d^\mathsf{T} Ax \geq d^\mathsf{T} b$ which is of the form $l_j \geq \gamma$ for some $\gamma \in \overline{\mathbb{R}}$ where l_j is the literal x_j or $-x_j$. Hence, d is a literal-inducing vector and we call it the vector associated with the *bound-resulting inference on x_j from row i*. (In practice, we would not perform an inference that results in γ being $-\infty$. But γ could be of the form $\alpha M + \beta$ in the extended mode.) For example, if we take $i = 1$ and $j = 2$ in Example 1, we have $S^+ = \{1\}$ and $S^- = \emptyset$ and

$$d = \frac{1}{2} \begin{bmatrix} 1 & 0 & 0 & 0 & 3 & 0 \end{bmatrix}^\mathsf{T}.$$

Then, $d^\mathsf{T} Ax \geq d^\mathsf{T} b$ is $-x_2 \geq -\frac{1}{2}$ and the literal induced by d is $-x_2$.

Bound propagation is the process of repeatedly performing bound-resulting inferences that result in improved bounds. It is shown in [4] that bound propagation can fail to terminate. To ensure termination, no bound is permitted to be improved more than D times where D is a positive integer specified in advance. Algorithm 1 performs limited bound propagation through generating literal-inducing vectors.

Algorithm 1: Limited Bound Propagation

```
Input: $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^{m-2n} \times \overline{\mathbb{R}}^{2n}$, where $A$ is of the form $\begin{bmatrix} B \\ I_n \\ -I_n \end{bmatrix}$ with
       columns indexed by variables.
Output: $b^p \in \mathbb{R}^{m-2n} \times \overline{\mathbb{R}}^{2n}$, $U \in \mathbb{R}^{m\times 2n}$ such that $b^p \geq b$, $U^\mathsf{T} A = \begin{bmatrix} I_n \\ -I_n \end{bmatrix}$,
        and $\begin{bmatrix} I_{m\times(m-2n)} & U \end{bmatrix}^\mathsf{T} b = b^p$.
1  $U \leftarrow \begin{bmatrix} 0_{(m-2n)\times 2n} \\ I_{2n} \end{bmatrix}$ where columns of $U$ are indexed by literals
2  $b^p \leftarrow b$
3  repeat
4      for $i \leftarrow 1$ to $m - 2n$, $j \leftarrow 1$ to $n$ do
5          if $A_{ij} \neq 0$ then
6              let $d \in \mathbb{R}^m$ be the vector associated with the bound-resulting
                 inference on $x_j$ from row $i$.
7              $\beta \leftarrow d^\mathsf{T} b$
8              let $l$ be the literal induced by $d$
9              if $\beta > b^p_l$ then
10                 $b^p_l \leftarrow \beta$
11                 $U_l \leftarrow \begin{bmatrix} I_{m\times(m-2n)} & U \end{bmatrix} d$
12             end
13         end
14     end
15 until no bound has improved or a bound has improved $D$ times or there is a
     variable $v$ with conflicting bounds (i.e. $b^p_v + b^p_{-v} > 0$);
```

Before we state and prove some properties of Algorithm 1, we make two observations:

(1) $b^p_i = b_i$ for all $i \in \{1, \ldots, m - 2n\}$ throughout the algorithm.
(2) The algorithm repeatedly updates the right-hand side values in the last $2n$ inequalities, which are bound inequalities on all the variables. Since

we want to be able to produce certificates of infeasibility with respect to the original inequalities, the algorithm keeps track of how each new bound can be inferred from the original inequalities. Line 11 is to make sure that the new bound inequality involving the literal l can still be written as a nonnegative linear combination of the original inequalities given by the literal-inducing vector U_l, the column of U indexed by the literal l. Corollary 4 below makes this precise.

To facilitate discussion, for a literal l, we define the 0-1 row vector e_l as follows:

- if $l = x_j$ for some j, then e_l has value 1 in entry j and 0 everywhere else.
- if $l = -x_j$ for some j, then e_l has value -1 in entry j and 0 everywhere else.

In other words, e_l is the row of A indexed by the literal l.

Lemma 3.1. *Let A and b be as in the input for Algorithm 1. Let $U \in \mathbb{R}^{m\times 2n}$ and $b^p \in \mathbb{R}^{m-2n} \times \overline{\mathbb{R}}^{2n}$ be such that $U^\mathsf{T} A = \begin{bmatrix} I_n \\ -I_n \end{bmatrix}$, and $\begin{bmatrix} I_{m\times(m-2n)} & U \end{bmatrix}^\mathsf{T} b = b^p$. If $d \in \mathbb{R}^m$ is a literal-inducing vector inducing the literal l (i.e. $d^\mathsf{T} A = e_l$), then $d' = \begin{bmatrix} I_{m\times(m-2n)} & U \end{bmatrix} d$ is a literal-inducing vector satisfying ${d'}^\mathsf{T} A = e_l$ and ${d'}^\mathsf{T} b = d^\mathsf{T} b^p$.*

Proof. Indeed,

$$\begin{aligned} {d'}^\mathsf{T} A &= \left(\begin{bmatrix} I_{m\times(m-2n)} & U \end{bmatrix} d\right)^\mathsf{T} A \\ &= d^\mathsf{T} \begin{bmatrix} I_{(m-2n)\times m} A \\ U^\mathsf{T} A \end{bmatrix} \\ &= d^\mathsf{T} \begin{bmatrix} B \\ I_n \\ -I_n \end{bmatrix} = d^\mathsf{T} A = e_l \end{aligned}$$

and

$$\begin{aligned} {d'}^\mathsf{T} b &= \left(\begin{bmatrix} I_{m\times(m-2n)} & U \end{bmatrix} d\right)^\mathsf{T} b \\ &= d^\mathsf{T} \begin{bmatrix} I_{m\times(m-2n)} & U \end{bmatrix}^\mathsf{T} b \\ &= d^\mathsf{T} b^p. \end{aligned}$$

□

Corollary 3.1. *At the end of each iteration of Algorithm 1, $U^\mathsf{T} A = \begin{bmatrix} I_n \\ -I_n \end{bmatrix}$, and $\begin{bmatrix} I_{m\times(m-2n)} & U \end{bmatrix}^\mathsf{T} b = b^p$. In particular, $U_l^\mathsf{T} A = e_l$ and $U_l^\mathsf{T} b = b_l^p$ for each literal l.*

Proof. Right after line 1, we have $U^\mathsf{T} A = \begin{bmatrix} 0_{2n\times(m-2n)} & I_{2n} \end{bmatrix} \begin{bmatrix} B \\ I_n \\ -I_n \end{bmatrix} = I_{2n} \begin{bmatrix} I_n \\ -I_n \end{bmatrix} = \begin{bmatrix} I_n \\ -I_n \end{bmatrix}$ and $\begin{bmatrix} I_{m\times(m-2n)} & U \end{bmatrix}^\mathsf{T} b = I_m b^p = b^p$. Thus, $U_l^\mathsf{T} A = e_l$ for each literal l and $U_l^\mathsf{T} b = b_l^p$ for each literal l since $b^p = \begin{bmatrix} I_{m\times(m-2n)} & U \end{bmatrix}^\mathsf{T} b = \begin{bmatrix} I_{(m-2n)\times m} b \\ U^\mathsf{T} b \end{bmatrix}$.

Assume that $U^\mathsf{T} A = \begin{bmatrix} I_n \\ -I_n \end{bmatrix}$, and $\begin{bmatrix} I_{m\times(m-2n)} & U \end{bmatrix}^\mathsf{T} b = b^p$ at the beginning of an iteration. Note that within each iteration, only U_l and b^p_l are updated where l is the literal induced by d in line 8. Let U' and b'^p denote the updated versions of U and b^p at the end of the iteration. It suffices to show that ${U'}_l^\mathsf{T} A = e_l$ and ${U'}_l^\mathsf{T} b = {b'}^p_l$. Indeed, by Lemma 3,

$$ {U'}_l^\mathsf{T} A = \left(\begin{bmatrix} I_{m\times(m-2n)} & U \end{bmatrix} d \right)^\mathsf{T} A = e_l $$

and

$$ {U'}_l^\mathsf{T} b = \left(\begin{bmatrix} I_{m\times(m-2n)} & U \end{bmatrix} d \right)^\mathsf{T} b = d^\mathsf{T} b^p = {b'}^p_l. $$

□

Proposition 3.1. *For any given positive integer D, Algorithm 1 terminates with b^p and U satisfying $b^p \geq b$, $U^\mathsf{T} A = \begin{bmatrix} I_n \\ -I_n \end{bmatrix}$, and $\begin{bmatrix} I_{m\times(m-2n)} & U \end{bmatrix}^\mathsf{T} b = b^p$. Furthermore, U belongs to a finite set dependent only on A and D.*

Proof. Since the loop can only repeat if a bound is improved and there are $2n$ bound inequalities, the loop must execute fewer than $2nD$ times and the algorithm must therefore terminate. Since the entries in b^p can only increase, we have $b^p \geq b$. By Corollary 4, $U^\mathsf{T} A = \begin{bmatrix} I_n \\ -I_n \end{bmatrix}$, and $\begin{bmatrix} I_{m\times(m-2n)} & U \end{bmatrix}^\mathsf{T} b = b^p$.

In line 6, d depends solely on A by definition and so only finitely many such d's are generated within the for-loop. As U is updated based solely on d, there exists a finite set dependent only on A and D that contains every possible U encountered during the algorithm. □

4. BPA-KV redacted

In this section, we present a version of BPA-KV in a language perhaps more familiar to readers with a background in mathematical programming than its original description in [4]. (A more intuitive verbal description of BPA-KV is given in [2] though without all the technical details for implementation and proofs of correctness and termination.) Our version makes use of recursion in place of a stack in the original algorithm, hence applying in reverse a standard technique for replacing recursion with a stack and a loop. As byproducts, the proofs of correctness and termination appear to be more intuitive.

Example 4.1. We use Algorithm 2 to obtain a certificate of infeasibility for the system

$$ x_1 - 2x_2 \geq 0,\ \ -2x_1 + x_2 \geq 0,\ \ x_1 + x_2 \geq 2, $$
$$ x_1 \geq -\infty,\ \ x_2, \geq -\infty,\ \ -x_1, \geq -\infty,\ \ -x_2 \geq -\infty $$

Note that adding the first three inequalities gives the contradiction $0 \geq 2$.

We set $D = 1$.

The input to the algorithm is given by $A = \begin{bmatrix} 1 & -2 \\ -2 & 1 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix}$ and $b = \begin{bmatrix} 0 \\ 0 \\ 2 \\ -\infty \\ -\infty \\ -\infty \\ -\infty \end{bmatrix}$.

Algorithm 2: BPA (Bound Propagation Algorithm for Linear Inequalities)

```
Input: $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^{m-2n} \times \overline{\mathbb{R}}^{2n}$, where $A$ is of the form $\begin{bmatrix} B \\ I_n \\ -I_n \end{bmatrix}$ with
       columns indexed by variables.
Output: A solution $x$ satisfying $Ax \geq b$ or a certificate of infeasibility
        $y \in \mathbb{R}^m$ such that $y^\top A = 0,\ y^\top b > 0$.
1  call Algorithm 1 with input $A, b$ to obtain $b^p$ and $U$
2  if no variable $v$ has conflicting upper and lower bounds (i.e. $b^p_v + b^p_{-v} > 0$) then
3      choose a variable $v$
4      repeat
5          $v \leftarrow c$ where $c$ is within the bounds for $v$ (i.e. $c \geq b^p_v$ and $-c \geq b^p_{-v}$)
6          if $v$ is the only variable then
7              return $c$
8          else
9              call BPA on $A', b'$ where $A'$ is obtained from $A$ by removing $A_v$
               and the rows indexed by $v$ and $-v$, and $b'$ is obtained from
               $b^p - A_v c$ by removing the entries indexed by $v$ and $-v$
10             if a solution $x'$ is returned then
11                 return $x'$ extended with $v = c$
12             else
13                 let $y'$ be the certificate of infeasibility returned
14                 form $y \in \mathbb{R}^m$ such that $y_i = y'_i$ for $i = 1, \ldots, m-2n$,
                   $y_v = y_{-v} = 0$, and $y_l = y'_l$ for all literals $l$ other than $v$ and
                   $-v$
15                 $\alpha \leftarrow y^\top A_v$
16                 if $\alpha = 0$ then
17                     return $\begin{bmatrix} I_{m\times(m-2n)} & U \end{bmatrix} y$
18                 end
19                 $d \leftarrow \frac{1}{|\alpha|} y$
20                 let $l$ be the literal induced by $d$
21                 $b^p_l \leftarrow d^\top b^p$
22                 $U_l \leftarrow \begin{bmatrix} I_{m\times(m-2n)} & U \end{bmatrix} d$
23             end
24         end
25     until upper and lower bounds on $v$ conflict (i.e. $b^p_v + b^p_{-v} > 0$);
26 end
27 choose a variable $v$ with conflicting upper and lower bounds (i.e.
   $b^p_v + b^p_{-v} > 0$)
28 return $U_v + U_{-v}$
```

Line 1 does not update any bounds and so $b^p = b$ and $U = \begin{bmatrix} 0_{3\times 4} \\ I_4 \end{bmatrix}$.

No variable has conflicting bounds and we can choose $v = x_1$ in line 3 and set it to 0 in line 5.

In line 9, $A' = \begin{bmatrix} -2 \\ 1 \\ 1 \\ 1 \\ -1 \end{bmatrix}$ and $b = \begin{bmatrix} 0 \\ 0 \\ 2 \\ -\infty \\ -\infty \end{bmatrix}$. We now follow the execution of the recursive call:

Line 1 returns $b^p = \begin{bmatrix} 0 \\ 0 \\ 2 \\ 2 \\ 0 \end{bmatrix}$ and $U = \begin{bmatrix} 0 & \frac{1}{2} \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$ since the system corresponding to A', b' is $-2x_2 \geq 0,\ x_2 \geq 0,\ x_2 \geq 2,\ x_2 \geq -\infty,\ -x_2 \geq -\infty$ and so the upper bound comes from the first inequality and the lower bound comes from the third inequality.

The bounds on the variable x_2 conflict. In line 28, $\begin{bmatrix} \frac{1}{2} & 0 & 1 & 0 & 0 \end{bmatrix}^\mathsf{T}$ is returned.

In line 14, we have $y = \begin{bmatrix} \frac{1}{2} & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}^\mathsf{T}$. So $\alpha = y^\mathsf{T} A_{x_1} = \frac{3}{2}$, giving $d = \begin{bmatrix} \frac{1}{3} & 0 & \frac{2}{3} & 0 & 0 & 0 & 0 \end{bmatrix}^\mathsf{T}$ in line 19. We have the new $b^p = \begin{bmatrix} 0 \\ 0 \\ 2 \\ \frac{4}{3} \\ -\infty \\ -\infty \\ -\infty \end{bmatrix}$ and the new $U =$

$$\begin{bmatrix} \frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{2}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Going back to line 5, we can set v to $\frac{4}{3}$. In line 9, $A' = \begin{bmatrix} -2 \\ 1 \\ 1 \\ 1 \\ -1 \end{bmatrix}$ and $b = \begin{bmatrix} -\frac{4}{3} \\ \frac{8}{3} \\ \frac{2}{3} \\ -\infty \\ -\infty \end{bmatrix}$.

We now follow the execution of the recursive call:

Line 1 returns $b^p = \begin{bmatrix} -\frac{4}{3} \\ \frac{8}{3} \\ \frac{2}{3} \\ \frac{8}{3} \\ -\frac{2}{3} \end{bmatrix}$ and $U = \begin{bmatrix} 0 & \frac{1}{2} \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$. The bounds on the variable x_2 conflict. In line 28, $\begin{bmatrix} \frac{1}{2} & 1 & 0 & 0 & 0 \end{bmatrix}^\mathsf{T}$ is returned.

In line 14, we have $y = \begin{bmatrix} \frac{1}{2} & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^\mathsf{T}$. So $\alpha = y^\mathsf{T} A_{x_1} = -\frac{3}{2}$, giving $d = \begin{bmatrix} \frac{1}{3} & \frac{2}{3} & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^\mathsf{T}$ in line 19. We have the new $b^p = \begin{bmatrix} 0 \\ 0 \\ 2 \\ \frac{4}{3} \\ -\infty \\ 0 \\ -\infty \end{bmatrix}$ and the new $U = \begin{bmatrix} \frac{1}{3} & 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{2}{3} & 0 \\ \frac{2}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$.

Now, x_1 is the only variable with conflicting bounds. In line 28, the algorithm returns

$$U_{x_1} + U_{-x_1} = \begin{bmatrix} \frac{1}{3} & 0 & \frac{2}{3} & 0 & 0 & 0 & 0 \end{bmatrix}^\mathsf{T} + \begin{bmatrix} \frac{1}{3} & \frac{2}{3} & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^\mathsf{T} = \begin{bmatrix} \frac{2}{3} & \frac{2}{3} & \frac{2}{3} & 0 & 0 & 0 & 0 \end{bmatrix}^\mathsf{T}.$$

We now prove some properties of Algorithm 2.

Lemma 4.1. *Line 21 strictly improves either the lower or upper bound on v.*

Proof. Note that y' in line 13 satisfies ${y'}^\mathsf{T} A' = 0$ and ${y'}^\mathsf{T} b' > 0$. Hence, with α in line 15, we have $y^\mathsf{T} Ax = \alpha v$. Now, $0 < {y'}^\mathsf{T} b' = y^\mathsf{T}(b^p - A_v c)$ implies that $y^\mathsf{T} b^p > y^\mathsf{T} A_v c = \alpha c$. Since $y \geq 0$, we can infer from $Ax \geq b^p$ that $y^\mathsf{T} Ax \geq y^\mathsf{T} b^p$, implying that $\alpha v > \alpha c$. Since the value c assigned to v violates this inequality, the bound in line 21 must strictly improve either the lower or upper bound on v. □

Lemma 4.2. *At the beginning of each iteration starting at line 4, $U^\mathsf{T} A = \begin{bmatrix} I_n \\ -I_n \end{bmatrix}$, and $\begin{bmatrix} I_{m\times(m-2n)} & U \end{bmatrix}^\mathsf{T} b = b^p$. In particular, $U_l^\mathsf{T} A = e_l$ and $U_l^\mathsf{T} b = b_l^p$ for each literal l.*

Proof. The statement of the lemma is true right after line 1 by Corollary 4.

Observe that within each iteration, only U_l and b_l^p are updated where l is the literal induced by d in line 19. It suffices to show that if $U_l^\mathsf{T} A = e_l$ and $U_l^\mathsf{T} b = b_l^p$ at the beginning of an iteration, then the same also hold at the end of the iteration. The proof now proceeds as in the proof for Corollary 4. □

Theorem 4.1. *If Algorithm 2 terminates, the result it returns is correct.*

Proof. The proof is by induction on n.

Suppose that $n = 1$. Let v denote the only variable in the system. Note that every constraint in the system is equivalent to a bound inequality. Hence, after line 1, the system has no solution if and only if $b_v^p + b_{-v}^p > 0$. If the condition in line 2 is true, then the c returned must be a solution to the system. Otherwise, the system has no solution and the algorithm returns $d = U_v + U_{-v}$ at line 28. By Corollary 4, we have $d^\mathsf{T} A = U_v^\mathsf{T} A + U_{-v}^\mathsf{T} A = e_v + e_{-v} = 0$ and $d^\mathsf{T} b = U_v^\mathsf{T} b + U_{-v}^\mathsf{T} b = b_v^p + b_{-v}^p > 0$. Hence, d is a certificate of infeasibility.

Suppose that $n \geq 2$. If the algorithm terminates at line 11, then by induction hypothesis, $A'x' \geq b'$. Since c satisfies the bounds for v given by b^p and $b_i' =$

$b^p - A_{iv}c$ for $i = 1, \ldots, m-2n$ and $b'_l = b^p - A_{lv}c$ for all literals l other than v and $-v$, we see that the extension of x' with $v = c$, call it x^*, satisfies $Ax^* \geq b^p \geq b$ since $b^p \geq b$ after line 1 and the algorithm never decreases the entries of b^p by Lemma 7. Thus, x^* is a feasible solution.

Suppose that the algorithm terminates at line 17. Let $d = \begin{bmatrix} I_{m\times(m-2n)} & U \end{bmatrix} y$, which is the vector returned. Clearly, $d^\mathsf{T} A = 0$. Now,

$$\begin{aligned} d^\mathsf{T} b &= \left(\begin{bmatrix} I_{m\times(m-2n)} & U \end{bmatrix} y \right)^T b \\ &= y^\mathsf{T} \begin{bmatrix} I_{m\times(m-2n)} & U \end{bmatrix}^\mathsf{T} b \\ &= y^\mathsf{T} b^p \end{aligned}$$

by Lemma 8. Since $y^\mathsf{T} A_v = 0$, we have

$$y^\mathsf{T} b^p = y^\mathsf{T}(b^p - A_v c) = y'^\mathsf{T} b' > 0.$$

Thus, $d^\mathsf{T} b > 0$, implying that d is a certificate of infeasibility for $Ax \geq b$.

Finally, suppose that the algorithm terminates at line 28. By Lemma 8, $U_v^\mathsf{T} A = e_v$, $U_{-v}^\mathsf{T} A = e_{-v}$, $U_v^\mathsf{T} b = b_v^p$, and $U_{-v}^\mathsf{T} b = b_{-v}^p$. It follows that if $d = U_v + U_{-v}$ is returned where there bounds on v given by b^p are conflicting, we have $d^\mathsf{T} A = U_v^\mathsf{T} A + U_{-v}^\mathsf{T} A = e_v + e_{-v} = 0$ and $d^\mathsf{T} b = U_v^\mathsf{T} b + U_{-v}^\mathsf{T} b = b_v^p + b_{-v}^p > 0$. Hence, d is a certificate of infeasibility for $Ax \geq b$. □

Next, we prove that Algorithm 2 indeed terminates.

Theorem 4.2. *Algorithm 2 terminates.*

Proof. Note that if at every recursion depth, the loop starting at line 4 is iterated a finite number of times, the algorithm will terminate. Hence, if the algorithm fails to terminate, there must be a recursive call in which the loop executes an infinite number of times. This is only possible if every iteration obtains a certificate of infeasibility in line 13. By Lemma 7, a bound on v is improved in each iteration, implying that the bounds on v are being improved infinitely many times. We show that this is impossible.

In particular, we prove by induction on n that for $A \in \mathbb{R}^{m\times n}$ of the form $\begin{bmatrix} B \\ I_n \\ -I_n \end{bmatrix}$, there exists a finite set $\mathcal{S}(A, D)$ dependent only on A and D such that for every $b \in \mathbb{R}^{m-2n} \times \overline{\mathbb{R}}^{2n}$ such that $Ax \geq b$ has no solution, Algorithm 2 terminates and returns a certificate of infeasibility in $\mathcal{S}(A, D)$.

Suppose that $n = 1$. If b is such that $Ax \geq b$ has no solution, the body of the conditional in line 2 is not executed. Note that every certificate of infeasibility returned in line 28 is the sum of the two columns of U since U has exactly two columns when $n = 1$. By Proposition 5, U comes from a finite set that depends only A and D, we see that the possibilities for the certificate of infeasibility returned come from a finite set dependent only on A and D.

Suppose that $n \geq 2$. By Proposition 5, the U returned in line 1 comes from a finite set dependent only on A and D. Suppose that the loop starting at line 4 is entered. If $b \in \mathbb{R}^{m-2n} \times \overline{\mathbb{R}}^{2n}$ is such that $Ax \geq b$ has no solution, line 9 will return a certificate of infeasibility y' which is, by the induction hypothesis, in some finite set $\mathcal{S}(A', D)$ that depends only on A' and D. Therefore, in line 22, there can only be a finite number of possible updates to U, regardless of what b^p is. The finiteness of $\mathcal{S}(A', D)$ also implies that the loop can only execute a finite number of times since no y' is returned twice because, by Lemma 7, a bound on v must be improved for each y' returned. As A' remains unchanged throughout all iterations of the loop, the total number of possible updates to U is finite, regardless of what b^p is. Hence, every certificate of infeasibility returned in line 28 comes from a finite set $\mathcal{S}(A, D)$ dependent only on A and D. □

5. Exponential worst-case running time

In this section, we show that the number of variable assignments that Algorithm 2 makes can be exponential in the number of variables.

Define the sequence of integers $F_1, F_2, \ldots$ as follows:

$$\begin{aligned} F_1 &= 0 \\ F_{i+1} &= 2F_i + 2 \quad \text{for } i \geq 1. \end{aligned}$$

A simple induction shows that $F_i = 2^i - 2$ for all $i \in \{1, 2, \ldots\}$.

Lemma 5.1. *Let m, n, k be positive integers with $m > 2k$ and $n > k$. Let $A \in \mathbb{R}^{m\times k}$ and $b \in \mathbb{R}^{m-2k} \times \overline{\mathbb{R}}^{2k}$ be such that A is of the form $\begin{bmatrix} B^{(k,n)} \\ I_k \\ -I_k \end{bmatrix}$ where*

$$B^{(k,n)}_{ij} = \begin{cases} -n & \text{if } j = k - i + 1,\ i \in \{1, \ldots, k\} \\ 1 & \text{otherwise,} \end{cases}$$

$b_i \leq 0$ for $i \in \{1, \ldots, k\}$, $b_i = -\infty$ for all $i \in \{m - 2k + 1, \ldots, m\}$, and there exists $s \in \{k+1, \ldots, m-2k\}$ such that $b_s > 0$, $b_s > b_i$ for all $i \in \{k+1, \ldots, m-2k\} \setminus \{s\}$, and $(n - k + 1)b_s + \sum_{i=1}^{k} b_i > 0$. Then on input A, b, Algorithm 2 can make F_k variable assignments and return the certificate of infeasibility y with

$$y_i = \begin{cases} \frac{n-k+2}{(n+1)(n-k+1)} & \text{if } i \in \{1, \ldots, k\} \\ \frac{n-k+2}{n+1} & \text{if } i = s \\ 0 & \text{otherwise.} \end{cases}$$

Proof. Indeed, y as defined is a certificate of infeasibility by Lemma 2 since $y \geq 0$, $y^\mathsf{T} A = 0$ and

$$y^\mathsf{T} b = \frac{n-k+2}{n+1}\left(b_s + \frac{1}{n-k+1}\sum_{i=1}^{k} b_i\right) = \frac{n-k+2}{(n+1)(n-k+1)}\left((n-k+1)b_s + \sum_{i=1}^{k} b_i\right) > 0.$$

We now prove the rest of the statement of the lemma by induction on $k \geq 1$.

Suppose that $k = 1$. Then every inequality in the system $Ax \geq b$ is either a bound inequality or equivalent to one and all that line 1 does is to update the bounds on the only variable x in the system. The best upper bound on x comes from the first inequality $-nx \geq b_1 > -\infty$ and the best lower bound on x comes from $x \geq b_s$ since, by assumption, $b_s > b_i$ for all $i \in \{2, \ldots, m-2\}$ and $b_{m-1} = b_m = -\infty$. As $nb_s + b_1 > 0$ by assumption, the upper and lower bounds on x conflict. The algorithm skips to line 27 and in line 28 returns the certificate of infeasibility with value $\frac{1}{n}$ in the first entry, 1 in the sth entry, and 0 everywhere else as desired. Note that no variable assignment has been made since line 5 is not executed.

Suppose that $k \geq 2$. Observe that line 1 terminates with b^p and U as initialized in Algorithm 1 since no new bounds can be obtained. Hence, $b^p = b$ and $\begin{bmatrix} I_{m\times(m-2n)} & U \end{bmatrix} = I_{2m}$.

Line 2 detects no conflicting bounds.

In line 3, the algorithm can choose the variable corresponding to the first column of A, which we call u, and assign to it the value $c = 0$ in line 5.

In line 9, with $m' = m-2$ and $k' = k-1$, we have $A' = \begin{bmatrix} B^{(k',n)} \\ I_{k'} \\ -I_{k'} \end{bmatrix}$ and $b' \in \mathbb{R}^{m'-2k'} \times \mathbb{R}^{2k'}$ where $b'_i = b_i$ for all $i \in \{1,\ldots,m'-2k'\}$ (since $m'-2k' = m-2k$) and $b'_i = -\infty$ for all $i \in \{m'-2k'+1,\ldots,m'\}$. Hence, $b'_i \leq 0$ for all $i \in \{1,\ldots,k'\}$, $b'_i = b_i = -\infty$ for all $i \in \{m'-2k'+1,\ldots,m'\}$, $b'_s = b_s > 0 \geq b_k = b'_{k'+1}$, $b'_s = b_s > b_i = b'_i$ for all $i \in \{k'+2,\ldots,m'-2k'\} \setminus \{s\}$, and

$$(n-k'+1)b'_s + \sum_{i=1}^{k'} b'_i = (n-k+2)b_s + \sum_{i=1}^{k-1} b_i = b_s - b_k + (n-k+1)b_s + \sum_{i=1}^{k} b_i > 0.$$

It follows that A' and b' satisfy the conditions of the statement of the lemma and so by the induction hypothesis, the recursive call in line 9 can make F_{k-1} variable assignments and return a certificate of infeasibility y' such that $y'_i = \begin{cases} \frac{n-k+3}{(n+1)(n-k+2)} & \text{if } i \in \{1,\ldots,k-1\} \\ \frac{n-k+3}{n+1} & \text{if } i = s \\ 0 & \text{otherwise.} \end{cases}$

In line 14, y is given by $y_i = \begin{cases} \frac{n-k+3}{(n+1)(n-k+2)} & \text{if } i \in \{1,\ldots,k-1\} \\ \frac{n-k+3}{n+1} & \text{if } i = s \\ 0 & \text{otherwise.} \end{cases}$

Then in line 15, we have $\alpha = y^{\mathsf{T}} A_u = \frac{(n-k+3)(k-1)}{(n+1)(n-k+2)} + \frac{n-k+3}{n+1} = \frac{n-k+3}{n-k+2} > 0$.

So d in line 19 is given by $d_i = \begin{cases} \frac{1}{n+1} & \text{if } i \in \{1,\ldots,k-1\} \\ \frac{n-k+2}{n+1} & \text{if } i = s \\ 0 & \text{otherwise.} \end{cases}$

In line 21, b^p_u is set to $d^{\mathsf{T}} b^p = d^{\mathsf{T}} b = \frac{1}{n+1}\left((n-k+2)b_s + \sum_{i=1}^{k-1} b_i\right)$ and in line 22, U_u is set to d since $\begin{bmatrix} I_{m\times(m-2n)} & U \end{bmatrix} = I_{2m}$.

The algorithm then goes back to the beginning of the loop and can assign to u a positive value c chosen large enough so that in line 9, with $m' = m-2$ and $k' = k-1$, $b'_{k'+1} = b_k - A_{ku}c$ is positive and greater than $\max\{b'_{k'+2},\ldots,b'_{m'-2k'}\}$ (possible because $A_{ku} = -n$ and $A_{iu} = 1$ for all $i \in \{k'+2,\ldots,m'-2k'\}$) and that $(n-k'+1)b'_k + \sum_{i=1}^{k'} b'_i > 0$ (possible because

$$\begin{aligned} (n-k'+1)b'_k + \sum_{i=1}^{k'} b'_i &= (n-(k-1)+1)b'_k + \sum_{i=1}^{k-1} b'_i \\ &= (n-k+2)(b_k+nc) + \sum_{i=1}^{k-1}(b_i - c) \\ &\geq b_k + nc - (k-1)c + \sum_{i=1}^{k-1} b_i \\ &= (n-k+1)c + \sum_{i=1}^{k} b_i \\ &\to \infty \end{aligned}$$

as $c \to \infty$ since $n-k+1 > 0$.) Note also that $b'_i = b_i - c < b_i \leq 0$ for all $i \in \{1,\ldots,k'\}$, $b'_i = -\infty$ for all $i \in \{m'-2k'+1,\ldots,m'\}$, and $A' = \begin{bmatrix} B^{(k',n)} \\ I_{k'} \\ -I_{k'} \end{bmatrix}$. Hence,

A' and b' satisfy the conditions of the statement of the lemma and so by the induction hypothesis, the recursive call in line 9 can make F_{k-1} variable assignments and return a certificate of infeasibility y' such that $y'_i = \begin{cases} \frac{n-k+3}{(n+1)(n-k+2)} & \text{if } i \in \{1,\dots,k-1\} \\ \frac{n-k+3}{n+1} & \text{if } i = k \\ 0 & \text{otherwise.} \end{cases}$

In line 14, y is given by $y_i = \begin{cases} \frac{n-k+3}{(n+1)(n-k+2)} & \text{if } i \in \{1,\dots,k-1\} \\ \frac{n-k+3}{n+1} & \text{if } i = k \\ 0 & \text{otherwise.} \end{cases}$

Then in line 15, we have $\alpha = y^\top A_u = \frac{(n-k+3)(k-1)}{(n+1)(n-k+2)} - \frac{n(n-k+3)}{n+1} = -\frac{(n-k+1)(n-k+3)}{n-k+2} < 0$.

So d in line 19 is given by $d_i = \begin{cases} \frac{1}{(n+1)(n-k+1)} & \text{if } i \in \{1,\dots,k-1\} \\ \frac{n-k+2}{(n+1)(n-k+1)} & \text{if } i = k \\ 0 & \text{otherwise.} \end{cases}$

In line 21, b^p_{-u} is set to $d^\top b^p = \frac{1}{(n+1)(n-k+1)}\left((n-k+2)b_k + \sum_{i=1}^{k-1} b_i\right)$ since $b^p_i = b_i$ for all $i \in \{1,\dots,k\}$.

In line 22, U_{-u} is set to d since $\left[I_{m\times(m-2n)} \; U\right] = I_{2m}$.

Now,

$$\begin{aligned} b^p_u + b^p_{-u} &= \frac{1}{n+1}\left((n-k+2)b_s + \sum_{i=1}^{k-1} b_i\right) + \frac{1}{(n+1)(n-k+1)}\left((n-k+2)b_k + \sum_{i=1}^{k-1} b_i\right) \\ &= \frac{1}{n+1}\left((n-k+2)b_s + \left(1 + \frac{1}{n-k+1}\right)\sum_{i=1}^{k-1} b_i + \frac{n-k+2}{n-k+1}b_k\right) \\ &= \frac{n-k+2}{(n+1)(n-k+1)}\left((n-k+1)b_s + \sum_{i=1}^{k} b_i\right) > 0. \end{aligned}$$

Hence, the variable u has conflicting bounds and the algorithm can choose it in line 27.

In line 28, since $s > k$, the algorithm returns $d' = U_u + U_{-u}$ given by $d'_i = \begin{cases} \frac{n-k+2}{(n+1)(n-k+1)} & \text{if } i \in \{1,\dots,k\} \\ \frac{n-k+2}{n+1} & \text{if } i = s \\ 0 & \text{otherwise.} \end{cases}$

The total number of variable assignments is $2F_{k-1} + 2 = F_k$. This completes the induction.

$\square$

Corollary 5.1. *The running time of Algorithm 2 can be exponential in the number of variables of the input system.*

Proof. Let n be a positive integer. Let $A \in \mathbb{R}^{(3n+1)\times n}$ and $b \in \mathbb{R}^{n+1} \times \overline{\mathbb{R}}^{2n}$ be such that $A = \begin{bmatrix} B \\ I_n \\ -I_n \end{bmatrix}$ where $B = \begin{bmatrix} 1 & \cdots & 1 & -n \\ 1 & \cdots & -n & 1 \\ \vdots & \ddots & \vdots & \\ -n & \cdots & 1 & 1 \\ 1 & \cdots & 1 & 1 \end{bmatrix}$ and $b_1 = \cdots = b_n = 0$, $b_{n+1} = 2$, and $b_{n+2} = \cdots = b_{3n+1} = -\infty$. By Lemma 11, Algorithm 2 on input A, b can make $2^n - 2$ variable assignments. (See Example 6 for an illustration of the case $n = 2$.) $\square$

6. Linear programming

There is a well-known trick for converting a linear programming problem to a system of linear constraints. For example, the primal-dual pair $\min\{c^\mathsf{T}x : Ax \geq b\}$ and $\max\{y^\mathsf{T}b : y^\mathsf{T}A = c^\mathsf{T},\ y \geq 0\}$ can be solved by finding a solution to the system

$$\begin{aligned} Ax &\geq b \\ y^\mathsf{T}A &= c^\mathsf{T} \\ y &\geq 0 \\ c^T x &\geq y^\mathsf{T}b. \end{aligned}$$

Thus, any method that solves a system of linear constraints can be used to solve a linear programming problem. However, as with Fourier-Motzkin elimination method (see pp. 155–156 in [6]), Algorithm 2 can be used to solve $\min\{c^\mathsf{T}x : Ax \geq b\}$ directly without having to specify its dual. To this end, consider the equivalent problem

$$\begin{aligned} \min\ & z \\ \text{s.t. }\ & z - c^\mathsf{T}x \geq 0 \\ & Ax \geq b. \end{aligned}$$

Assume as in previous sections that $A \in \mathbb{R}^{m\times n}$ is of the form $\begin{bmatrix} B \\ I_n \\ -I_n \end{bmatrix}$ and $b \in \mathbb{R}^{m-2n} \times \overline{\mathbb{R}}^{2n}$ where $\overline{\mathbb{R}}$ includes $\pm M$ instead of $\pm\infty$. Let $\gamma = \sum_{i=1}^n |c_i|$. We apply Algorithm 2 in *extended mode* to the system

$$\begin{aligned} z - c^\mathsf{T}x &\geq 0 \\ Ax &\geq b, \\ z &\geq -\gamma M \\ -z &\geq -M \end{aligned}$$

by treating M as a very large but unspecified number.

The system can be written in matrix form as $\tilde{A}\tilde{x} \geq \tilde{b}$ with $\tilde{A} = \begin{bmatrix} \tilde{B} \\ I_{n+1} \\ -I_{n+1} \end{bmatrix}$ where $\tilde{B} = \begin{bmatrix} 1 & -c^\mathsf{T} \\ 0 & B \end{bmatrix}$, $\tilde{x} = \begin{bmatrix} z \\ x \end{bmatrix}$ and $\tilde{b}$ is given by $\tilde{b}_i = b_i$ for $i = 1, \ldots, m-2n$, $\tilde{b}_z = -\gamma M$, $\tilde{b}_{-z} = -M$, and $\tilde{b}_l = -M$ for all literals $l \notin \{z, -z\}$.

With $\tilde{A}, \tilde{b}$ as input to Algorithm 2, we make z the very first choice in line 3 and always assign to z the current lower bound on z, b^p_v, in line 5. At termination, if a certificate of infeasibility is returned, then the problem is infeasible. Otherwise, a solution $\begin{bmatrix} z^* \\ x^* \end{bmatrix}$ is returned whose entries are in $\overline{\mathbb{R}}$. Let $d = U_v$ where U is the matrix right before the algorithm terminates. Then $z^* = d^\mathsf{T}b = \alpha M + \beta$ for some $\alpha, \beta \in \mathbb{R}$. Since $d \geq 0$ and no entry in b contains a positive multiple of M, we must have $\alpha \leq 0$. If $\alpha = 0$, z^* as the optimal value and x^* is an optimal solution to the linear programming problem. In this case, a dual optimal solution can be readily constructed from d. If $\alpha < 0$, then the problem is unbounded and $c^\mathsf{T}x^* \to -\infty$ as $M \to \infty$.

Example 6.1. Consider the linear programming problem

$$\begin{aligned} \min\ & u - 2v \\ \text{s.t.}\ & u + v \geq 2 \end{aligned}$$

which is unbounded. We apply Algorithm 2 to the system

$$\begin{array}{r} z - u + 2v \geq 0 \\ u + v \geq 2 \\ \hline z \geq -3M \\ u, v, -z, -u, -v \geq -M. \end{array}$$

The horizontal line merely serves as a reading aid by making the separation of the bound inequalities from the other inequalities apparent. Instead of showing the details of the execution of the algorithm using the matrix notation, we opt for a more intuitive approach and proceed informally. We set $D = 1$.

After limited bound propagation (Algorithm 1), the system with updated bounds is

$$\begin{array}{l} z - u + 2v \geq 0 \\ \quad u + v \geq 2 \\ \hline \qquad\quad z \geq -3M \\ \qquad\quad u \geq -M + 2 \\ \qquad\quad v \geq -M + 2 \\ -z, -u, -v \geq -M \end{array}$$

since adding $u + v \geq 2$ and $-v \geq -M$ gives $u \geq -M + 2$ and adding $u + v \geq 2$ and $-u \geq -M$ gives $v \geq -M + 2$. We can set $z = -3M$ to obtain

$$\begin{array}{l} -u + 2v \geq 3M \\ \quad u + v \geq 2 \\ \hline \qquad u \geq -M + 2 \\ \qquad v \geq -M + 2 \\ \ -u, -v \geq -M \end{array}$$

and make a recursive call on this system. After limited bound propagation, we obtain

$$\begin{array}{l} -u + 2v \geq 3M \\ \quad u + v \geq 2 \\ \hline \qquad u \geq -M + 2 \\ \qquad v \geq M + 1 \\ \quad\ -u \geq M \\ \quad\ -v \geq -M \end{array}$$

since adding $-u + 2v \geq 3M$ and 2 times $-v \geq -M$ gives $-u \geq M$ and adding $-u + 2v \geq 3M$ and $u \geq -M + 2$ gives $2v \geq 2M + 2$, resulting in $v \geq M + 1$ after multiplying by $\frac{1}{2}$. Thus, $U_{-u} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}^\intercal$ and $U_v = \begin{bmatrix} \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \end{bmatrix}^\intercal$. We see that the bounds on u conflict and the algorithm returns the certificate of infeasibility $U_u + U_{-u} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 2 \end{bmatrix}^\intercal$.

Therefore, in line 14, $y = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 2 \end{bmatrix}^\intercal$ after returning from the recursive call and leads to adding $z - u + 2v \geq 0$, $u \geq -M + 2$, and 2 times $-v \geq -M$ to give $z \geq -3M + 2$. We set $z = -3M + 2$ to obtain

$$\begin{array}{r}
-u + 2v \geq 3M - 2 \\
u + v \geq 2 \\
\hline
u \geq -M + 2 \\
v \geq -M + 2 \\
-u, -v \geq -M
\end{array}$$

and make a recursive call on this system. After limited bound propagation, we obtain

$$\begin{array}{r}
-u + 2v \geq 3M - 2 \\
u + v \geq 2 \\
\hline
u \geq -M + 2 \\
v \geq M \\
-u \geq M - 2 \\
-v \geq -M
\end{array}$$

We can set $u = -M + 2$ to obtain

$$\begin{array}{r}
2v \geq 2M \\
v \geq M \\
\hline
v \geq M \\
-v \geq -M
\end{array}$$

and make a recursive call on this system. The system has the unique solution $v = M$ and the algorithm will terminate with the solution $\begin{bmatrix} z \\ u \\ v \end{bmatrix} = \begin{bmatrix} -3M + 2 \\ -M + 2 \\ M \end{bmatrix}$. Note that $\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -M + 2 \\ M \end{bmatrix}$ is a feasible solution to the linear programming problem for all $M \in \mathbb{R}$ and its objective function value is $-3M + 2 \to -\infty$ as $M \to -\infty$.

7. Final remarks

Results from computational experiments in [2] show that BPA-KV performed well in comparison to a selection of linear arithmetic solvers. It remains to be seen how it performs in comparison with mainstream linear programming solvers. We think that there might be problem classes for which it could be advantageous. One possibility is solving problems that require exact arithmetic since the algorithm performs only simple operations and does not require non-trivial techniques used in exact rational solver such as QSOpt-Exact [1] or the SoPlex rational solver [3]. We leave it for future research to explore the practical potential of the algorithm.

References

[1] William Cook, Sanjeeb Dash, and Daniel G. Espinoza. Exact solutions to linear programming problems. *Operations Research Letters*, 35:693–699, 2007.

[2] I. Dragan, K. Korovin, L. Kovács, and A. Voronkov. Bound propagation for arithmetic reasoning in vampire. In *2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 169–176, 2013.

[3] Ambros M. Gleixner, Daniel E. Steffy, and Kati Wolter. Iterative refinement for linear programming. *INFORMS Journal on Computing*, 28(3):449–464, 2016, https://arxiv.org/abs/https://doi.org/10.1287/ijoc.2016.0692.

[4] Konstantin Korovin and Andrei Voronkov. Solving systems of linear inequalities by bound propagation. In *Proceedings of the 23rd International Conference on Automated Deduction*, volume 6803 of *Lecture Notes in Computer Science*, pages 369–383. Springer International Publishing, 2011.

[5] Laura Kovács and Andrei Voronkov. First-order theorem proving and vampire. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification*, pages 1–35, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[6] A. Schrijver. *Theory of Linear and Integer Programming.* Wiley Series in Discrete Mathematics & Optimization. Wiley, 1998.

Carleton University, 1125 Colonel By Dr, Ottawa, ON K1S 5B6, Canada
E-mail address: `kcheung@math.carleton.ca`