On Chubanov's method for Linear Programming

Amitabh Basu, Jesús A. De Loera, Mark Junod
Department of Mathematics, University of California, Davis {abasu@math.ucdavis.edu, deloera@math.ucdavis.edu, mjunod@math.ucdavis.edu}

We discuss the method recently proposed by S. Chubanov for the linear feasibility problem. We present new, concise proofs and geometric interpretations of some of his results. From our ideas we derive the first strongly polynomial time algorithm based on relaxation method techniques for special classes of linear feasibility problems. Under certain conditions, these results provide new proofs of classical results obtained by Tardos for combinatorial linear programs. The paper ends with some experimental investigations.

Key words: linear programming; relaxation methods; strongly polynomial time algorithms
History:

---

1. Introduction

In their now classical papers, Agmon [1] and Motzkin and Schoenberg [15] introduced the so called *relaxation method* to determine the feasibility of a system of linear inequalities (it is well-known that optimization and the feasibility problem are polynomially equivalent to one another). Starting from any initial point, a sequence of points is generated. If the current point z_i is feasible we stop, else there must be at least one constraint $c^T x \leq d$ that is violated. We will denote the corresponding hyperplane by (c, d). Let $p_{(c,d)}(z_i)$ be the orthogonal projection of z_i onto the hyperplane (c, d), choose a number λ (usually chosen between 0 and 2), and define the new point z_{i+1} by $z_{i+1} = z_i + \lambda(p_{(c,d)}(z_i) - z_i)$. Agmon, Motzkin and Schoenberg showed that if the original system of inequalities is feasible, then this procedure generates a sequence of points which converges, in the limit, to a feasible point. So in practice, we stop either when we find a feasible point, or when we are close enough, i.e., any violated constraint is violated by a very small (predetermined) amount.

Many different versions of the relaxation method have been proposed, depending on how the step-length multiplier λ is chosen and which violated hyperplane is used. For example, the well-known perceptron algorithms [3] can be thought of as members of this family of methods. In addition to linear programming feasibility, similar iterative ideas have been used

in the solution of overdetermined system of linear equations as in the Kaczmarz's method where iterated projections into hyperplanes are used to generate a solution (see [12, 19, 16]).

The original relaxation method was shown early on to have a poor practical convergence to a solution (and in fact, finiteness could only be proved in some cases), thus relaxation methods took second place behind other techniques for years. During the height of the fame of the ellipsoid method, the relaxation method was revisited with interest because the two algorithms share a lot in common in their structure (see [2, 9, 21] and references therein) with the result that one can show that the relaxation method is finite in all cases when using rational data, and thus can handle infeasible systems. In some special cases the method did give a polynomial time algorithm [14], but in general it was shown to be an exponential time algorithm (see [10, 21]). Most recently in 2004, Betke gave a version that had polynomial guarantee in some cases and reported on experiments [4]. In late 2010 Sergei Chubanov presented a variant of the relaxation algorithm, that was based on the divide-and-conquer paradigm [6]. We will refer to this algorithm as the *Chubanov Relaxation algorithm*. This algorithm takes as input an $m \times n$ matrix A, an $l \times n$ matrix C, $b \in \mathbb{R}^m$, and $d \in \mathbb{R}^l$, where the elements of A, b, C, and d are integers, and either gives a feasible solution in $\mathbb{R}^n$ for the following system, or reports that the system has no solutions in the integers $\mathbb{Z}^n$.

$$\begin{aligned} Ax &= b, \\ Cx &\leq d \end{aligned} \tag{1}$$

Note that the Chubanov Relaxation algorithm is *not* a linear feasibility algorithm, because it only reports integer infeasibility. Another curious aspect is that even if it returns a feasible solution in $\mathbb{R}^n$, the system could still be integer infeasible in $\mathbb{Z}^n$ and thus the two output states of the algorithm are not mutually exclusive. Nevertheless, the advantage of Chubanov's algorithm is that when the inequalities take the form $\mathbf{0} \leq x \leq \mathbf{1}$, then the algorithm runs in *strongly* polynomial time. More formally,

Theorem 1.1. *[see Theorem 5.1 in [6]] Chubanov's Relaxation algorithm either finds a solution to the system*

$$\begin{aligned} Ax &= b, \\ \mathbf{0} \leq x &\leq \mathbf{1}, \end{aligned} \tag{2}$$

or decides that there are no integer solutions to this system. Moreover, the algorithm runs in strongly polynomial time.

This interesting theorem leads to a new polynomial time algorithm for the linear feasibility problem which appears in an unpublished manuscript [7]. After we submitted this paper

for publication, we learned from Sergei Chubanov that he now has another paper along these lines [8]. His new paper is not available online as of this writing, but he kindly provided a copy. His interesting techniques give another new linear feasibility algorithm based on Theorem 1.1 hat is different from the one presented in [7]. However, this new algorithm is also not strongly polynomial in complexity, even when restricted to certain special cases as discussed in the manuscript.

Our Results

Our key contributions are the first strongly polynomial time algorithm based on relaxation methods for some families of linear programs, and the first ever implementation of Chubanov's Relaxation algorithm as described in [6]. We provide simpler and more geometric analyses of Chubanov's key ideas, which forms the basis of our strongly polynomial time algorithm. More precisely, in Section 2, we explain some aspects of Chubanov's Relaxation algorithm relevant to the results of this paper. We will not discuss his full algorithm; instead, we concentrate on the main "Divide and Conquer" subroutine of Chubanov's Relaxation algorithm. We will refer to this subroutine as *Chubanov's D&C subroutine* in the rest of the paper.

Simplified Analysis of Chubanov's D&C subroutine The D&C subroutine is the main ingredient in the Chubanov Relaxation algorithm. We provide a simplified analysis of this subroutine in Section 3 where we prove some key lemmas about the D&C subroutine whose content can be summarized in the following theorem.

Theorem 1.2. *Chubanov's D&C subroutine can be used to infer one of the following statements about the system* (1) *:*

(i) A feasible solution to (1) *exists, and can be found using the output of the D&C subroutine.*

(ii) One of the inequalities in $Cx \leq d$ is an implied equality, i.e., there exists $k \in \{1, \ldots, l\}$ such that $c_k x = d_k$ for all solutions to (1)*.*

(iii) There exists $k \in \{1, \ldots, l\}$ such that $c_k x = d_k$ for all integer *solutions to* (1)*.*

A constructive proof for the above theorem can be obtained using results in Chubanov's original paper [6]. Our contribution here is to provide a different, albeit existential, proof

of this theorem. We feel our proof is more geometric and simpler than Chubanov's original proof. We hope this will help to expose more clearly the main intuition behind Chubanov's D&C subroutine, which is the workhorse behind Chubanov's results. Of course, Chubanov's constructive proof is more powerful in that it enables him to prove Theorem 1.1 using the D&C subroutine.

New Algorithms for Linear Feasibility Using the intuition behind our own proofs of Theorem 1.2, we are able to demonstrate how Chubanov's D&C subroutine can be used to give a strongly polynomial algorithm for deciding the feasibility or infeasibility of a system like (2), under certain additional assumptions. We show the following result about bounded linear feasibility problems in Section 4.1.

Theorem 1.3. *Consider the linear program given by*

$$\begin{aligned} &Ax = b, \\ &\mathbf{0} \leq x \leq \lambda \mathbf{1}. \end{aligned} \tag{3}$$

Suppose A is a totally unimodular matrix and λ is bounded by a polynomial in n, m (the latter happens, for instance, when $\lambda = 1$). Furthermore, suppose we know that if (3) is feasible, it has a strictly feasible solution. Then there exists a strongly polynomial time algorithm that either finds a feasible solution of (3), or correctly decides that the system is infeasible. The running time for this algorithm is $O\left(m^3 + m^2 n + n^2 m + n^2 (2n\lambda\sqrt{2n+1})^{\frac{1}{\log_2\left(\frac{7}{5}\right)}}\right)$. If $\lambda = 1$, then the running time can be upper bounded by $O(m^3 + m^2 n + n^2 m + n^{5.1})$.

Our Theorem 1.3 proves a version of Tardos' theorem on combinatorial LPs [20] using a completely different set of tools. Tardos' result is stronger because she does not assume any upper bounds on the variables ($\lambda = \infty$), does not assume strictly feasible solutions, and only assumes that the entries of A are polynomially bounded by n, m. Nevertheless our result has interest as the techniques are different and we preserve strong polynomiality.

Other authors have obtained similar results before: Vavasis and Ye proved Tardos' result using interior point methods [22]. Interestingly, two sets of authors have done work using relaxation methods: Maurras et al. [14] used relaxation methods to prove a polynomial time algorithm for totally unimodular constraint matrices, but not a strongly polynomial time algorithm. Chubanov, in his recently announced manuscript [8], attempts to improve [14]. Alas, he does not obtain a strongly polynomial time result either. Thus our result is the first strongly polynomial time algorithm based on relaxation method techniques.

To conclude Section 4 we also show that Chubanov's D&C subroutine can be used to construct a new algorithm for solving general linear programs. This is completely different from Chubanov's linear programming algorithm in [7] or [8]. However, this general algorithm that we present in our paper is not guaranteed to run in polynomial time.

In the final section of the paper we investigate the computational performance of three relaxation methods mentioned in this paper. One of them is the first ever implementation of Chubanov's Relaxation algorithm as in Theorem 1.1. The purpose of these experiments is not to compare with the running times of current commercial software like CPLEX, which would require a more sophisticated implementation. Rather, we want to determine how the new relaxation-type algorithms compare with the original algorithm by Agmon, and Motzkin and Schoenberg with respect to the number of iterative/recursive calls and time.

2. Chubanov's Divide and Conquer Subroutine

In this section we will outline Chubanov's main subroutines for the D&C subroutine as presented in [6].

First, a couple of assumptions and some notation. We assume the matrices $A \in \mathbb{R}^{m\times n}$ and $C \in \mathbb{R}^{l\times n}$ have no zero rows and that A is of full rank. Note if A does not have full rank we can easily transform the system into another, $A'x = b'$, $Cx \leq d$, such that A' has full rank without affecting the set of feasible solutions. Let a_i denote the i-th row of A for $i \in \{1, \ldots, m\}$ and c_k denote the k-th row of C for $k \in \{1, \ldots, l\}$. P will denote the set of feasible solutions of (1). Finally, $B(z, r)$ will denote the open ball centered at z of radius r in $\mathbb{R}^n$.

One new idea of Chubanov's algorithm is its use of new induced inequalities. Unlike Motzkin and Schoenberg [15], who only projected onto the original hyperplanes that describe the polyhedron P (see top of Figure 1), Chubanov constructs new valid inequalities along the way and projects onto them too (bottom part of Figure 1).

The aim of Chubanov's D&C subroutine is to achieve the following. Given a current guess z, a radius r and an error bound $\epsilon > 0$, the algorithm will either:

(1) Find an ϵ-approximate solution $x^* \in B(z, r)$ to (1), i.e. some x^* such that

$$Ax^* = b,\ Cx^* \leq d + \epsilon\mathbf{1},$$

(2) Or find an induced hyperplane $hx = \delta$ that separates $B(z, r)$ from P.

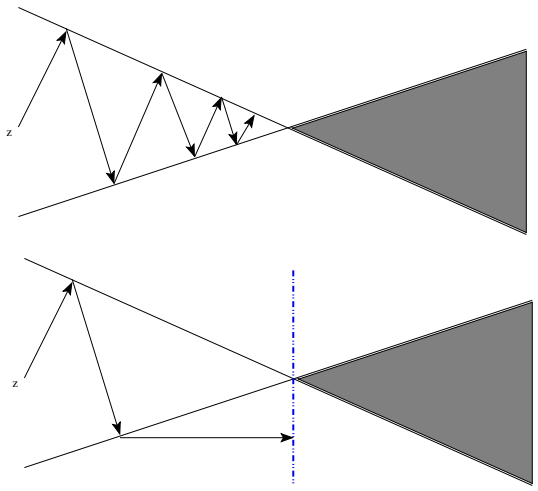

Figure 1: Chubanov's method generates new inequalities on the way.

This task is achieved using a recursive algorithm. In the base case, Chubanov uses a subroutine called *Chubanov's Elementary Procedure* (EP), which achieves the above goal if r is small enough; namely, when $r \leq \frac{\epsilon}{2\|c_{\max}\|}$, where $\|c_{\max}\| = \max_{1\leq k\leq l}(\|c_k\|)$. Let $p_{(A,b)}(z)$ denote the projection of z onto the affine subspace defined by $Ax = b$.

Algorithm 2.1. THE ELEMENTARY PROCEDURE
Input: A system $Ax = b$, $Cx \leq d$ and the triple (z, r, ϵ) where $r \leq \frac{\epsilon}{2\|c_{\max}\|}$.
Output: Either an ϵ-approximate solution x^* or a separating hyperplane $hx = \delta$.

If $\|p_{(A,b)}(z) - z\| < r$ and $\frac{c_k z - d_k}{\|c_k\|} < r$ for all k
  Then $x^* = p_{(A,b)}(z)$ is an ϵ-approximate solution (see Figure 2)
Else If $\|p_{(A,b)}(z) - z\| \geq r$
  Then let $h = (z - p_{(A,b)}(z))^T$ and $\delta = h \cdot p_{(A,b)}(z)$ (see Figure 3)
Else $\frac{c_{k_0} z - d_{k_0}}{\|c_{k_0}\|} \geq r$ for some index k_0
  Then let $h = c_{k_0}$ and $\delta = d_{k_0}$ (see Figure 4)
End If

Note $\frac{c_k z - d_k}{\|c_k\|}$ tells us how far z is from the hyperplane $c_k x = d_k$, and it is in fact negative if z satisfies the inequality $c_k x \leq d_k$. Thus if $\frac{c_k z - d_k}{\|c_k\|} < r$, then any point in $B(z, r)$ is an ϵ-approximation of $c_k x \leq d_k$. This simple observation is enough to see that the EP procedure solves the task when $r \leq \frac{\epsilon}{2\|c_{\max}\|}$. See Chubanov [6] Section 2 for more details and proofs.

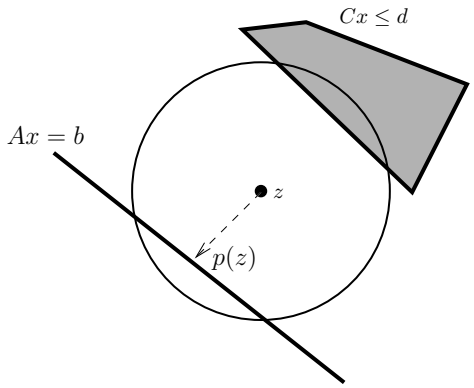

Figure 2: The projection $p(z)$ is an ϵ-approximate solution.

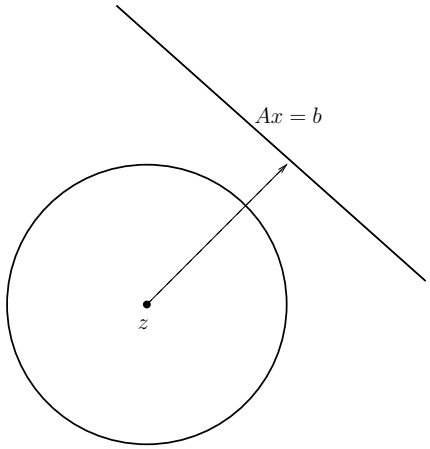

Figure 3: The separating hyperplane is the supporting hyperplane of $Ax = b$ defined by the projection direction $(z - p_{(A,b)}(z))^T$.

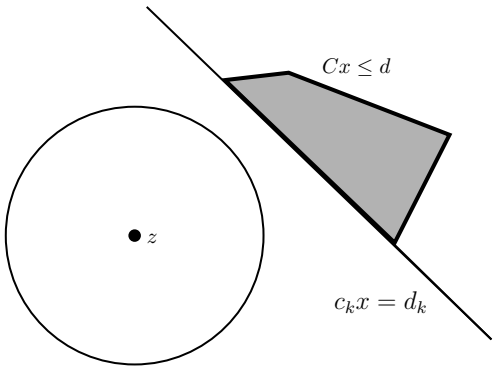

Figure 4: A separating hyperplane is given by a violated constraint.

The elementary procedure achieves the goal when r is small enough, but what happens when $r > \frac{\epsilon}{2\|c_{\max}\|}$? Then the D&C subroutine makes recursive steps with smaller values of r. To complete these recursive steps, the D&C subroutine uses additional projections and separating hyperplanes. To choose the smaller values of r, a rational number $0 < \theta < \sqrt{2}-1$ is fixed and r is replaced by $\frac{1}{1+\theta}r$ in the recursive steps. We give more details below. Figure 5 illustrates some of the steps in this recursion. A sample recursion tree is shown in Figure 6.

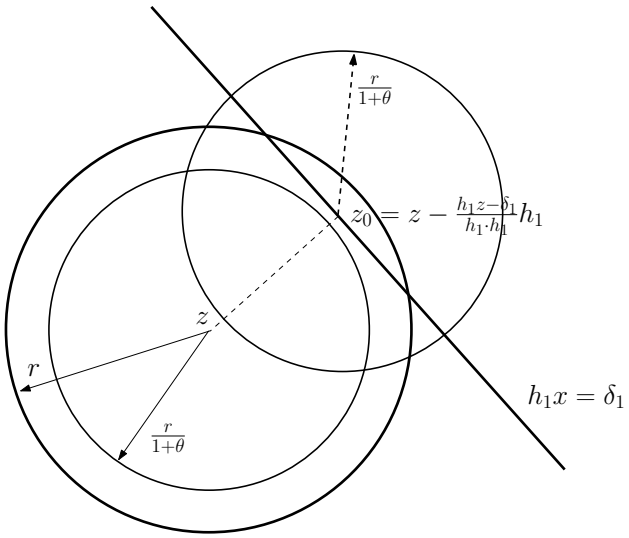

Figure 5: The recursive step in D&C works on two smaller balls with centers z and z_0.

Algorithm 2.2. THE D&C subroutine

Input: A system $Ax = b$, $Cx \leq d$, a triple (z, r, ϵ) and a rational number $0 < \theta < \sqrt{2} - 1$.

Output: Either an ϵ-approximate solution x^*, or a separating hyperplane $hx = \delta$ that separates $B(z, r)$ from $P = \{x \in \mathbb{R}^n \mid Ax = b, Cx \leq d\}$, or report a "Failure".

If $r \leq \frac{\epsilon}{2\|c_{\max}\|}$

Then run the EP on the system

Else recursively run the D&C subroutine with $(z, \frac{1}{1+\theta}r, \epsilon)$

End If

If the recursive call returns a solution x^*

Return x^*

Else let $h_1 x = \delta_1$ be returned by the recursive call

End If

Set $z_0 = z - \frac{h_1 z - \delta_1}{h_1 \cdot h_1} \cdot h_1$, i.e., project z onto (h_1, δ_1) (see Figure 5)
Run the D&C subroutine with $(z_0, \frac{1}{1+\theta} r, \epsilon)$

If the recursive call returns a solution x^*

Return x^*

Else let $h_2 x = \delta_2$ be returned by the recursive call

End If

If $h_1 = -\gamma h_2$ for some $\gamma > 0$

Then STOP, the algorithm fails

Else Find α such that $h = \alpha h_1 + (1-\alpha) h_2$, $\delta = \alpha \delta_1 + (1-\alpha)\delta_2$, and $\frac{hz - \delta}{\|h\|} \geq r$

Return $hx = \delta$ **End If**

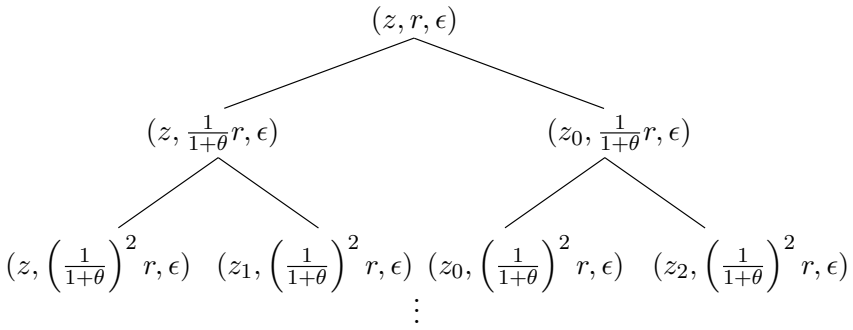

Figure 6: D&C Recursion Tree

Remark 2.1. *The fact that a number α can be found in the second to last line of the subroutine is guaranteed by the choice of $0 < \theta < \sqrt{2} - 1$. This argument can be found in [6] (see Proposition 2.3 and the related discussion in Section 3.1 in [6]).*

The D&C subroutine and its three possible outputs will be discussed at greater length in Section 3. We conclude this section by stating the running time of the Chubanov D&C subroutine.

Proposition 2.1. *[Theorem 3.1 in [6]] The matrix A is assumed to have m rows and n columns and let N be the number of non-zero entries in the matrix C. Let $\mu = \frac{2\epsilon}{28n\|c_{max}\|^2}$*

where c_{max} is the row of C with maximum norm. Let ρ be a number such that $|z_j| \leq \rho$ and ρz_j is an integer for all components z_j of the center z in the input to the D&C subroutine. Let $K = \left(\frac{r\|c_{max}\|}{\epsilon}\right)^{\frac{1}{log_2(7/5)}}$.

Chubanov's D&C subroutine with $\theta = \frac{2}{5} \leq \sqrt{2} - 1$ performs at most

$$O(m^3 + m^2 n + n^2 m + K\big(n \log(\frac{\rho + log(K)(r + n\mu)}{\mu}) + n^2 + N\big)) \tag{4}$$

arithmetic operations.

3. A Simplified Analysis of Chubanov's D&C subroutine

One possible way to exploit the D&C subroutine is the following. Since D&C returns ϵ-approximate solutions, we can try to run it on the system $Ax = b, Cx \leq d - \epsilon\mathbf{1}$; if the algorithm returns an ϵ-approximate solution, we will have an *exact* solution for our original system $Ax = b, Cx \leq d$. However, we need a z and an r as input for D&C. To get around this, we can appeal to some results from classical linear programming theory. Suppose one can, a priori, find a real number r^* such that if the system $Ax = b,\ Cx \leq d - \epsilon\mathbf{1}$ is feasible, then it has a solution with norm at most r^*. In other words, there exists a solution in $B(0, r^*)$, if the system is feasible. Such bounds are known in the linear programming literature (see Corollary 10.2b in Schrijver [18]), where r^* depends on the entries of A, b, C, d and ϵ. Then one can choose $z = 0$ and $r = r^*$ for the D&C subroutine. If the algorithm returns an ϵ-approximate solution, we will have an *exact* solution for our original system $Ax = b, Cx \leq d$; whereas, if the algorithm returns a separating hyerplane, we know that $Ax = b, Cx \leq d - \epsilon\mathbf{1}$ is infeasible by our choice of r.

This strategy suffers from three problems.

1. There is a strange outcome in the D&C procedure - when it "fails" and stops. This occurs when the two recursive branches return hyperplanes with normal vectors h_1, h_2 with $h_1 = -\gamma h_2$ for some $\gamma > 0$. It is not clear what we can learn about the problem from this outcome. Later in this paper, we will interpret this outcome in a manner that is different from Chubanov's interpretation.

2. It might happen that $Ax = b, Cx \leq d - \epsilon\mathbf{1}$ is infeasible, even if the original system $Ax = b, Cx \leq d$ is feasible. In this case the algorithm may return a separating

hyperplane, but we cannot get any information about our original system. All we learn is that $Ax = b, Cx \leq d - \epsilon\mathbf{1}$ is infeasible.

3. Finally, the running time of the D&C subroutine is a polynomial in n, m and $r = r^*$. Typically, the classical bounds on r^* are exponential in the data. This would mean the running time of the algorithm is not polynomial in the input data.

Let us concentrate on tackling the first two problems, to progress towards a correct linear programming algorithm. This requires a second important idea in Chubanov's work. We address the first two problems above by homogenizing, or parameterizing, our original system, and we show how this helps in the rest of this section. Geometrically this turns the original polyhedron into an unbounded polyhedron, defined by

$$\begin{aligned} Ax - bt &= \mathbf{0}, \\ Cx - dt &\leq \mathbf{0}, \\ -t &\leq -1. \end{aligned} \tag{5}$$

Note this system (5) is feasible if and only if (1) is feasible. Let (x^*, t^*) be a solution to (5). Then $\frac{x^*}{t^*}$ is a solution of (1). Similarly, if x^* is a solution of (1), then $(x^*, 1)$ is a solution of (5). Thus we can apply the D&C to a strengthened parameterized system

$$\begin{aligned} Ax - bt &= \mathbf{0}, \\ Cx - dt &\leq -\epsilon\mathbf{1}, \\ -t &\leq -1 - \epsilon \end{aligned} \tag{6}$$

and any ϵ-approximate solution will be an exact solution of (5), and thus will give us an exact solution of (1). We still need to figure out what z and r to use. For the rest of the paper, we will use $\epsilon = 1$ as done by Chubanov in his paper. It turns out that if we choose the appropriate z and r, we can get interesting information about the original system $Ax = b, Cx \leq d$ when D&C fails, or returns a separating hyperplane. We explain this next.

Let us summarize before we proceed. Given a system (1) we parameterize and then strengthen with $\epsilon = 1$ (to obtain a system in the form (6)). Then we apply the D&C to (6) with appropriately chosen z and r. Our three possible outcomes are:

(I) The D&C gives us a solution (x^*, t^*) which is an $\epsilon = 1$-approximate solution to (6). This is the best possible outcome, because we can then return the exact solution $\frac{x^*}{t^*}$ to (1).

(II) The D&C fails. The reader can look ahead to our Proposition 3.3 for a simple, geometric interpretation of this outcome. The reader can compare this to Chubanov's analysis of this case, which is more complicated in our opinion, and comes with no geometric intuition - see Section 4.2 in [6].

(III) The D&C returns a separating hyperplane $hx = \delta$. Our Proposition 3.4 tells us how to interpret this outcome. Again, we feel our analysis is simpler than Chubanov's original ideas, and gives a very clear geometric interpretation which is missing from Chubanov's paper. See Section 4.3 of [6].

In the rest of this section, we give some more geometric intuition behind the process of homogenizing the polyhedron and why it is useful. Our goal will be to prove Theorem 1.2.

3.1. Meaning of "Failure" Outcome in D&C

First we show that if the Chubanov D&C subroutine fails on a particular system, then in fact that system is infeasible. This observation is never made in the original paper by Chubanov [6] and, as far as we know, is new.

Proposition 3.1. *Suppose Chubanov's D&C subroutine fails on the system $Ax = b, Cx \leq d$, i.e., it returns two hyperplanes $h_1x = \delta_1$ and $h_2x = \delta_2$ with $h_1 = -\gamma h_2$ with $\gamma > 0$. Then the system $Ax = b, Cx \leq d$ is infeasible.*

Proof. Let $P = \{x \in \mathbb{R}^n \mid Ax = b, Cx \leq d\}$. If Chubanov's algorithm fails, then there exists $z \in \mathbb{R}^n, r > 0$ such that the following two things happen :

(i) $h_1x \leq \delta_1$ is valid for P and for all $y \in B(z, \frac{1}{1+\theta}r)$, $h_1y > \delta_1$.

(i) $h_2x \leq \delta_2$ is valid for P and for all $y \in B(z_0, \frac{1}{1+\theta}r)$, $h_2y > \delta_2$, where $z_0 = z - \frac{h_1z-\delta_1}{h_1 \cdot h_1} \cdot h_1$.

Since $z_0 - \frac{r}{2(1+\theta)} \frac{h_2}{\|h_2\|} \in B(z_0, \frac{1}{1+\theta}r)$, $h_2 \cdot (z_0 - \frac{r}{2(1+\theta)} \frac{h_2}{\|h_2\|}) > \delta_2$. Therefore,

$$h_2z_0 - \frac{r\|h_2\|}{2(1+\theta)} > \delta_2. \tag{7}$$

Now we use the fact that $h_1 = -\gamma h_2$ and so $-\frac{1}{\gamma}h_1 = h_2$ which we substitute into (7) to get $-\frac{1}{\gamma}h_1z_0 - \frac{r\|h_2\|}{2(1+\theta)} > \delta_2$. From the definition of z_0, we have $h_1z_0 = \delta_1$. Therefore, $-\frac{1}{\gamma}\delta_1 > \frac{r\|h_2\|}{2(1+\theta)} + \delta_2 > \delta_2$. So $\delta_1 < -\gamma\delta_2$. Now $h_1x \leq \delta_1$ is valid for P using (i) above. Using

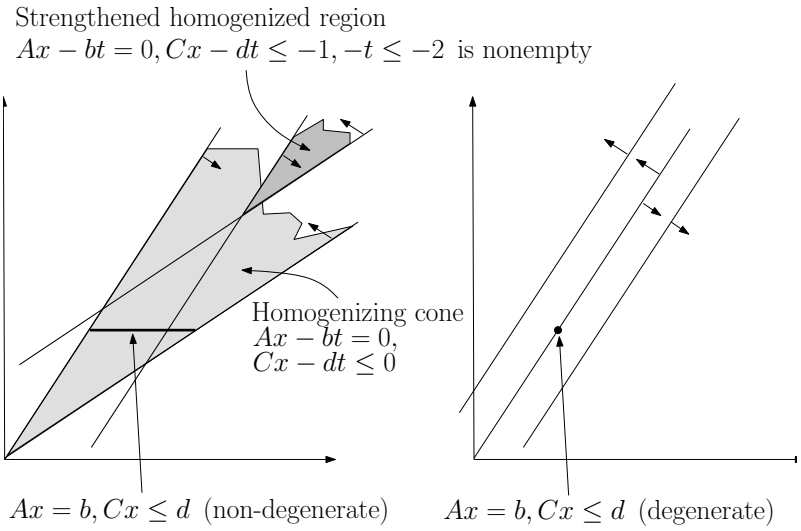

Figure 7: This figure illustrates our Lemma 3.2. In the left figure, the original feasible region is non-degenerate and so the strengthened, homogenized cone is nonempty. The figure on the right shows an example where the original system is degenerate; the strengthened cone is empty because we have two parallel hyperplanes which get pushed away from each other, creating infeasibility in the strengthened system.

$h_1 = -\gamma h_2$ and $\delta_1 < -\gamma\delta_2$, we get $-\gamma h_2 x < -\gamma\delta_2$ is valid for P, i.e., $h_2 x > \delta_2$ is valid for P. But we also know that $h_2 x \leq \delta_2$ is valid for P from (ii) above. This implies that $P = \emptyset$ and the system $Ax = b, Cx \leq d$ is infeasible. □

We now interpret the "failure" outcome of the D&C subroutine on the strengthened parameterized system. First we prove the following useful lemma.

Lemma 3.2. *If the system $Ax - bt = 0, Cx - dt \leq -1, -t \leq -2$ is infeasible, then there exists $k \in \{1, \ldots, l\}$ such that $c_k \cdot x = d_k$ for all x satisfying $Ax = b, Cx \leq d$.*

Proof. We prove the contrapositive. So suppose that for all $k \in \{1, \ldots, l\}$, there exists x_k satisfying $Ax_k = b, Cx_k \leq d$ with $c_k \cdot x_k < d_k$. Then using $\bar{x} = \frac{1}{l}(x_1 + \ldots + x_l)$, we get that $A\bar{x} = b$ and $c_k\bar{x} < d_k$ for all $k \in \{1, \ldots, l\}$. Let $\eta_k = d_k - c_k \cdot \bar{x} > 0$ and let $\eta = \min\{\frac{1}{2}, \eta_1, \ldots, \eta_l\}$. Therefore, $\eta > 0$. Let $x^* = \frac{\bar{x}}{\eta}$ and $t^* = \frac{1}{\eta}$. Then

$$Ax^* - bt^* = \frac{1}{\eta}(A\bar{x} - b) = 0.$$

For every $k \in \{1, \ldots, l\}$,

$$d_k t^* - c_k x^* = \frac{1}{\eta}(d_k - c_k\bar{x}) = \frac{\eta_k}{\eta} \geq 1.$$

Therefore, $c_k x^* - d_k t^* \leq -1$ for every $k \in \{1, \ldots, l\}$. Finally, since $t = \frac{1}{\eta} \geq 2$ by definition of η, we have $-t \leq -2$. □

An illustration of the above lemma appears in Figure 7. The following is the important conclusion one makes if the D&C subroutine "fails" on the strengthened parameterized system.

Proposition 3.3. *If Chubanov's D&C subroutine fails on the system $Ax - bt = 0, Cx - dt \leq -1, -t \leq -2$, then there exists $k \in \{1, \ldots, l\}$ such that $c_k \cdot x = d_k$ for all x satisfying $Ax = b, Cx \leq d$.*

Proof. Follows from Proposition 3.1 and Lemma 3.2. □

3.2. The meaning of when a separating hyperplane is returned by D&C

We now make the second useful observation about the strengthened parameterized system $Ax - bt = 0, Cx - dt \leq -1, -t \leq -2$. Suppose we know that all solutions to $Ax = b, Cx \leq d$ have norm at most r^*. Then we will show that if all solutions to the strengthened system are "too far" from the origin, then the original system is "very thin" (this intuition is illustrated in Figure 8). More precisely, we show that if all solutions to the strengthened parameterized system have norm greater than $2l(r^* + 1)$, then there exists an inequality $c_k x \leq d_k$ such that all solutions to $Ax = b, Cx \leq d$ satisfy $d_k - \frac{1}{2} \leq c_k x$. That is, the original polyhedron lies in a "thin strip" $d_k - \frac{1}{2} < c_k x \leq d_k$. This would then imply that all integer solutions to $Ax = b, Cx \leq d$ satisfy $c_k x = d_k$ since c_k, d_k have integer entries. Here is the precise statement of this observation.

Proposition 3.4. *Suppose $r^* \in \mathbb{R}$ is such that $\|x\| \leq r^*$ for all x satisfying $Ax = b, Cx \leq d$. If $\|(x, t)\| > 2l(r^* + 1)$ for all (x, t) satisfying $Ax - bt = 0, Cx - dt \leq -1, -t \leq -2$, then there exists $k \in \{1, \ldots, l\}$ such that $d_k - \frac{1}{2} < c_k x \leq d_k$ for all x satisfying $Ax = b, Cx \leq d$.*

Proof. Suppose to the contrary that for all $j \in \{1, \ldots, l\}$, there exists x^j such that $Ax^j = b, Cx^j \leq d$, and $c_j x^j \leq d^j - \frac{1}{2}$, i.e., $c_j(2x^j) - 2d_j \leq -1$. This implies that the following equations hold for every $j \in \{1, \ldots, l\}$

$$\begin{array}{ll} c_j(2x^j) - 2d_j \leq -1, & \\ c_j(2x^i) - 2d_j \leq 0 & \forall i \neq j. \end{array} \tag{8}$$

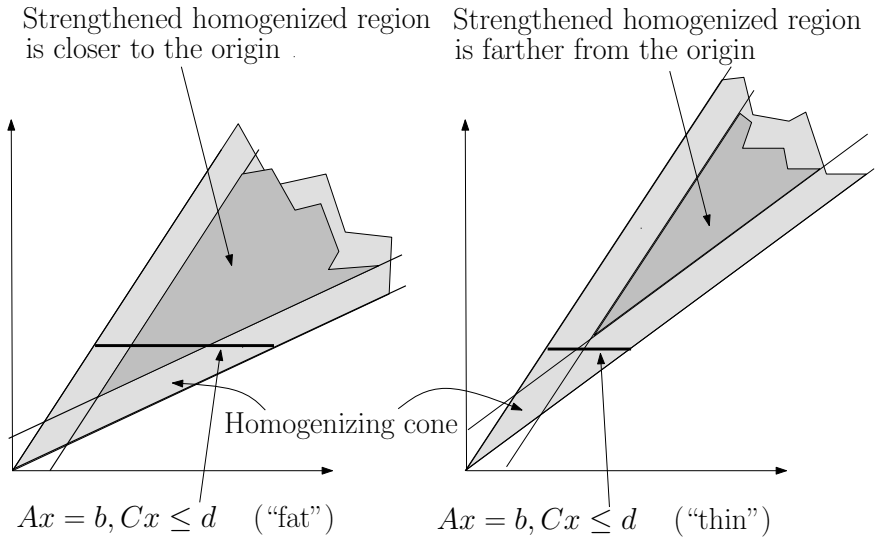

Figure 8: This Figure illustrates our Proposition 3.4. When the original feasible region is "thinner", the strengthened homogenized cone is pushed farther away from the origin.

Now consider $\hat{x} = \sum_{j=1}^{l} 2x^j$ and $\hat{t} = 2l$. It is easily verified $A\hat{x} - b\hat{t} = 0$ since $Ax^j = b$ for all $j \in \{1, \ldots, l\}$. Moreover, adding together the inequalities in (8), we get that $c_j\hat{x} - d_j\hat{t} \leq -1$ for all $j \in \{1, \ldots, l\}$. Therefore, $(\hat{x}, \hat{t})$ satisfies the constraints $Ax - bt = 0, Cx - dt \leq -1, -t \leq -2$.

Finally, $\|(\hat{x}, \hat{t})\| \leq \|\hat{x}\| + 2l \leq \sum_{j=1}^{l} 2\|x^j\| + 2l \leq 2l(r^* + 1)$, where the last inequality follows from the fact that all solutions to $Ax = b, Cx \leq d$ have norm at most r^*. We have thus reached a contradiction with the hypothesis that $\|(x, t)\| > 2l(r^* + 1)$ for all (x, t) satisfying $Ax - bt = 0, Cx - dt \leq -1, -t \leq -2$. □

The above proposition shows that if Chubanov's D&C subroutine returns a separating hyperplane separating the feasible region of $Ax - bt = 0, Cx - dt \leq -1, -t \leq -2$ from the ball $B(0, 2l(r^* + 1))$, one can infer that there exists an inequality $c_k x \leq d_k$ that is satisfied at equality by all integer solutions to $Ax = b, Cx \leq d$.

We now have all the tools to prove Theorem 1.2.

Proof of Theorem 1.2. As discussed earlier, we can assume that there exists r^* such that $\|x\| \leq r^*$ for all x satisfying $Ax = b, Cx \leq d$. We then run the D&C subroutine on the strengthened system (6) with $\epsilon = 1$, $z = 0$ and $r = 2l(r^* + 1)$. Recall the three possible outcomes of this algorithm. In the first outcome, we can find an exact solution of $Ax = b, Cx \leq d$ - this is (i) in the statement of the theorem. In the second outcome, when D&C fails, Proposition 3.3 tells that we have an implied equality, which is (ii) in

the statement of the theorem. Finally, in the third outcome, D&C returns a separating hyperplane. By Proposition 3.4, we know that there exists an inequality $c_k x \leq d_k$ that is satisfied at equality by all integer solutions to $Ax = b, Cx \leq d$. This is (iii) in the statement of the theorem. □

3.3. Chubanov's proof of Theorem 1.1

Chubanov is able to convert the existential results of Propositions 3.3 and 3.4 into constructive procedures, wherein he can find the corresponding implied equalities in strongly polynomial time. He then proceeds to apply the D&C procedure iteratively and reduce the number of inequalities by one at every iteration. One needs at most l such iterations and at the end one is left with a system of linear equations. This system can be tested for feasibility in strongly polynomial time by standard linear algebraic procedures. This is the main idea behind the Chubanov Relaxation algorithm and the proof of Theorem 1.1 that appears in [6].

4. New Algorithms for Linear Feasibility Problems

4.1. Linear feasibility problems with strictly feasible solutions

We will now demonstrate that using the lemmas we proved in Section 3, we can actually give strongly polynomial time algorithms for certain classes of linear feasibility problems. More concretely, we aim to prove Theorem 1.3. Consider a linear feasibility problem in the following standard form.

$$\begin{aligned} Ax &= b, \\ -x &\leq \mathbf{0}, \end{aligned} \tag{9}$$

where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. We will assume that the entries of A and b are integers and that A has full row rank. Let

$$\Delta_A = \max\{|det(B)| \mid B \text{ is an } m \times m \text{ submatrix of } A\}$$

be the maximum $m \times m$ subdeterminant of the matrix A. Let $P(A, b) = \{x \in \mathbb{R}^n \mid Ax = b, -x \leq \mathbf{0}\}$ denote the feasible region of (9).

Lemma 4.1. *Let x be any vertex of $P(A, b)$. If $x_i > 0$ for some $i \in \{1, \ldots, n\}$, then $x_i \geq \frac{1}{\Delta_A}$.*

Proof. Since x is a vertex, there exists a nonsingular $m \times m$ submatrix B of A such that x is the basic feasible solution corresponding to the basis B. That is, the non basic variables are 0 and the basic variable values are given in the vector $B^{-1}b$. If for some i, $x_i > 0$ then this is a basic variable and therefore its value is at least $\frac{1}{|det(B)|}$ since b is an integer vector. Since $|det(B)| \leq \Delta_A$, we have that $x_i \geq \frac{1}{\Delta_A}$. □

Lemma 4.2. *Suppose we know that* (9) *has a strictly feasible solution, i.e. there exists* $\bar{x} \in \mathbb{R}^n$ *such that* $A\bar{x} = b$ *and* $\bar{x}_i > 0$ *for all* $i \in \{1, \ldots, n\}$. *If* $P(A, b)$ *is bounded, then there exists a solution* $x^* \in \mathbb{R}^n$ *such that* $Ax^* = b$ *and* $x_i^* \geq \frac{1}{n\Delta_A}$ *for all* $i \in \{1, \ldots, n\}$.

Proof. Since we have a strictly feasible solution and $P(A, b)$ is a polytope, then for every $i \in \{1, \ldots, n\}$ there exists a vertex $\bar{x}^i$ of P such that $\bar{x}_i^i > 0$. By Lemma 4.1, we have that $\bar{x}_i^i \geq \frac{1}{\Delta_A}$. Therefore, if we consider the solution

$$x^* = \frac{1}{n} \sum_{i=1}^{n} \bar{x}^i,$$

i.e., the convex hull of all these n vertices, then $x_i^* \geq \frac{1}{n\Delta_A}$ for all $i \in \{1, \ldots, n\}$. □

We now consider linear feasibility problems of the form (9) such that either it is infeasible or has a strictly feasible solution. We will present an algorithm to decide if such linear feasibility problems are feasible or infeasible. We call this algorithm the **LFS Algorithm**, as an acronym for **L**inear **F**easibility problems with **S**trict solutions. As part of the input, we will take a parameter $\Delta \geq \Delta_A$, as well as a real number r such that $P(A, b) \subset B(0, r)$, i.e., $\|x\| < r$ for all feasible solutions x. The running time of our algorithm will depend on Δ, r, m and n.

Algorithm 4.1. THE **LFS ALGORITHM**

Input: $Ax = b$, $-x \leq \mathbf{0}$, such that either this system is infeasible, or there exists a *strictly* feasible solution. We are also given a real number r such that $P(A, b) \subset B(0, r)$, i.e., $\|x\| < r$ for all feasible solutions x. Moreover, we are given a parameter Δ as part of the input.

Output: A feasible point x^* or the decision that the system is infeasible.

Parameterize (9):

$$\begin{aligned} &Ax - bt = 0, \\ &-x \leq \mathbf{0}, \\ &-t \leq -1. \end{aligned} \tag{10}$$

Run Chubanov's D&C subroutine on the strengthened version of (10):

$$\begin{aligned} Ax - bt &= 0, \\ -x &\leq -\mathbf{1}, \\ -t &\leq -2. \end{aligned} \tag{11}$$

with $z = \mathbf{0}$, $\hat{r} = 2n\Delta\sqrt{r^2+1}$, and $\epsilon = 1$

If Chubanov's D&C subroutine finds an ϵ-feasible solution (x^*, t^*)

Return $\hat{x} = \frac{x^*}{t^*}$ as a feasible solution for (9).

Else (Chubanov's D&C subroutine fails or returns a separating hyperplane)

Return "The system (9) is INFEASIBLE"

Theorem 4.3. *The LFS Algorithm, when executed with $\Delta \geq \Delta_A$, correctly determines a feasible point x^* of (9) or determines the system is infeasible. The running time is*

$$O\left(m^3 + m^2 n + n^2 m + (2n\Delta\sqrt{r^2+1})^{\frac{1}{\log_2\left(\frac{7}{5}\right)}} \left(n^2 + n\log\Delta + n\log(r)\right)\right)$$

Proof. We first confirm the running time. We will use Proposition 2.1. Observe that for our input to the D&C subroutine, $\|c_{max}\| = 1$, $\epsilon = 1$ and $\hat{r} = 2n\Delta\sqrt{r^2+1}$ and $N = n$. Therefore, $K = (2n\Delta\sqrt{r^2+1})^{\frac{1}{\log_2\left(\frac{7}{5}\right)}}$, $\rho = 0$ since we use the origin as the initial center for the D&C subroutine, $\mu = \frac{1}{14n}$. Substituting this into (4), we get the stated running time for the LFS algorithm.

To prove the correctness of the algorithm, we need to look at each of the three cases Chubanov's D&C can return.

CASE 1: An ϵ-approximate solution (x^*, t^*) is found for (11). Then (x^*, t^*) is an exact solution to (10), and $\frac{x^*}{t^*}$ is a solution to (9).

CASE 2: The D&C subroutine fails to complete, returning consecutive separating hyperplanes (h_1, δ_1) and (h_2, δ_2) such that $h_1 = -\gamma h_2$ for some $\gamma > 0$. Then Proposition 3.3 says that there exists $i \in \{1, \ldots, n\}$ such that $x_i = 0$ for all feasible solutions to (9). But then the original system (9) has no strictly feasible solution, and by our assumption, is therefore actually infeasible. Therefore if the D&C fails, our original system $Ax = b$, $-x \leq \mathbf{0}$ is infeasible.

CASE 3: The final case is when the D&C subroutine returns a separating hyperplane (h, δ), separating $B(0, \hat{r})$ from the feasible set of (11). We now show that this implies the original system $Ax = b$, $-x \leq \mathbf{0}$ is infeasible.

If not, then from our assumption, we have a strictly feasible solution x^* for (9). By Lemma 4.2, we know that $x_i^* \geq \frac{1}{n\Delta_A}$ for all $i \in \{1, \ldots, n\}$. Consider the point $(\bar{x}, \bar{t}) = (2n\Delta_A x^*, 2n\Delta_A)$ in $\mathbb{R}^{n+1}$. We show that $(\bar{x}, \bar{t})$ is feasible to (11). Since $Ax^* = b$, we have that $A\bar{x} - b\bar{t} = 0$. Moreover, since $x_i^* \geq \frac{1}{n\Delta_A}$ for all $i \in \{1, \ldots, n\}$, we have that $\bar{x} = 2n\Delta_A x^* \geq \mathbf{1}$, i.e., $-\bar{x} \leq -\mathbf{1}$. Finally, $\bar{t} = 2n\Delta_A \geq 2$, since $\Delta_A \geq 1$ and $n \geq 1$. Therefore, $-\bar{t} \leq -2$. Now we check the norm of this point $\|(\bar{x}, \bar{t})\| = \|(2n\Delta_A x^*, 2n\Delta_A)\| = 2n\Delta_A \|(x^*, 1)\|$. Since $x^* \in P(A, b) \subset B(0, r)$, we know that $\|x^*\| < r$. Also, $\Delta_A \leq \Delta$. Therefore, $\|(\bar{x}, \bar{t})\| < \hat{r}$. So $(\bar{x}, \bar{t})$ is a feasible solution to (11) and $(\bar{x}, \bar{t}) \in B(0, \hat{r})$. But D&C returned a separating hyperplane separating $B(0, \hat{r})$ from the feasible set of (11). This is a contradiction. Hence, we conclude that $Ax = b$, $-x \leq \mathbf{0}$ is infeasible. □

Remark 4.1. *We proved the correctness of the LFS algorithm with a parameter $\Delta \geq \Delta_A$. If one executes the LFS algorithm with a parameter $\Delta < \Delta_A$, the algorithm could still return a feasible solution after its execution. Of course, if it reports infeasibility when executed with $\Delta < \Delta_A$, then we cannot be sure that it is indeed infeasible. However, this observation can be used to find feasible solutions to a linear feasibility problem without explicitly computing Δ_A. This idea is used in Section 5 for running our experiments on the LFS algorithm.*

Corollary 4.4. *Consider the following system.*

$$\begin{aligned} &Ax = b, \\ &\boldsymbol{0} \leq x \leq \lambda \boldsymbol{1}. \end{aligned} \tag{12}$$

Suppose that we know that the above system is either infeasible, or has a strictly feasible solution, i.e., there exists $\bar{x}$ such that $A\bar{x} = b$ and $\boldsymbol{0} < \bar{x} < \lambda \boldsymbol{1}$. Then there exists an algorithm which either returns a feasible solution to (12)*, or correctly decides that the system is infeasible, with running time*

$$O\left(m^3 + m^2 n + n^2 m + (2n\Delta_A \lambda \sqrt{2n+1})^{\frac{1}{\log_2\left(\frac{7}{5}\right)}} \left(n^2 + n \log \Delta_A + n \log(\lambda)\right)\right).$$

Proof. We first put (12) a standard form.

$$\begin{aligned} &Ax = b, \\ &x + y = \lambda \mathbf{1}, \\ &-x \leq \mathbf{0}, \\ &-y \leq \mathbf{0}. \end{aligned} \tag{13}$$

We will use the constraint matrix of (13) :

$$\tilde{A} = \begin{bmatrix} A & \mathbf{0}_{m \times n} \\ I_n & I_n \end{bmatrix},$$

where $\mathbf{0}_{m\times n}$ denotes the $m \times n$ matrix with all 0 entires, and I_n is the $n \times n$ identity matrix. Therefore, $P(\tilde{A}, [b, \lambda\mathbf{1}])$ is the feasible set for (13). Also, note that $\Delta_{\tilde{A}} = \Delta_A$. Since $\mathbf{0} \leq x, y \leq \lambda\mathbf{1}$ for all $(x, y) \in P(\tilde{A}, [b, \lambda\mathbf{1}])$, we know that $P(\tilde{A}, [b, \mathbf{1}]) \subset B(0, \lambda\sqrt{2n})$. Therefore, we run LFS with $r = \lambda\sqrt{2n}$ and the system (13) as input. Observe that since (12) has a strictly feasible solution, so does (13). By Theorem 4.3, when LFS is exectued with $\Delta = \Delta_{\tilde{A}} = \Delta_A$, it either returns a feasible solution to (13) which immediately gives a feasible solution to (12), or correctly decides that (13) is infeasible and hence (12) is infeasible. Moreover, the running time for LFS is

$$\begin{aligned} &O\left((m^3 + m^2n + n^2m + (2n\Delta_{\tilde{A}}\sqrt{r^2+1})^{\frac{1}{\log_2(\frac{7}{5})}}\,(n^2 + n\log\Delta_{\tilde{A}} + n\log(r))\right) \\ &= O\left(m^3 + m^2n + n^2m + (2n\Delta_A\lambda\sqrt{2n+1})^{\frac{1}{\log_2(\frac{7}{5})}}\,(n^2 + n\log\Delta_A + n\log(\lambda))\right). \end{aligned}$$

□

Corollary 4.4 is related to the following theorem of Schrijver. Let $\tilde{\Delta}_A = \max\{|det(B^{-1})| \mid B \text{ is a nonsingular submatrix of A}\}$.

Theorem 4.5 (Theorem 12.3 in [18])**.** *A combination of the relaxation method and the simultaneous diophantine approximation method solves a system $Ax \leq b$ of rational linear inequalities in time polynomially bounded by $size(A, b)$ and by $\tilde{\Delta}_A$.*

On one hand, we have the additional assumptions of being bounded and having strictly feasible solutions. On the other hand, we can get rid of the dependence of the running time on $size(A, b)$ and the use of the simultaneous diophantine approximation method, which utilizes non-trivial lattice algorithms. It is also not immediately clear how Δ_A is related to $\tilde{\Delta}_A$. We finally prove Theorem 1.3.

Proof of Theorem 1.3. If A is totally unimodular, then $\Delta_A = 1$. The result now follows from Corollary 4.4. □

4.2. A General Algorithm for Linear Feasibility Problems

In this subsection, we describe an algorithm for solving general linear feasibility problems using Chubanov's D&C subroutine and the ideas developed in Section 3. We call this algorithm the **LFG Algorithm**, as an acronym for **L**inear **F**easibility problems in **G**eneral. Before we can state our algorithm and prove its correctness, we need a couple of other pieces.

Lemma 4.6. *[Schrijver Corollary 10.2b] Let P be a rational polyhedron in $\mathbb{R}^n$ of facet complexity ϕ. Define*

$$Q = P \cap \{x \in \mathbb{R}^n | -2^{5n^2\phi} \leq x_i \leq 2^{5n^2\phi} \text{ for } i = 1, \ldots, n\}.$$

Then dim(P) = dim(Q).

Thus, if we use $\hat{r} = 2^{5n^2\phi}\sqrt{n}$ then $B(\mathbf{0}, \hat{r})$ will circumscribe the hypercube in the above lemma from Schrijver. We will also have $\dim(P) = \dim(P \cap B(\mathbf{0}, \hat{r}))$.

The second piece we need comes from Papadimitriou and Stieglitz [17]. Simply put, the theorem states that (1) is feasible if and only if some other system is strictly feasible (i.e. all the inequalities are strict inequalities).

Lemma 4.7. *[Lemma 8.7 in [17]] The system $Ax = b, -x \leq \mathbf{0}$ is feasible if and only if*

$$\begin{aligned} &Ax = b, \\ &-x < \nu\mathbf{1}. \end{aligned} \tag{14}$$

is feasible, where $\nu = 2^{-2T}$ when T is the size of the input data. Moreover, given a solution to (14)*, we can construct a solution to* (1)*.*

Algorithm 4.2. THE LFG ALGORITHM

Input: $Ax = b$, $-x \leq \mathbf{0}$.

Output: A feasible point x^* or the decision the system is infeasible.

Set $\nu = 2^{-2T}$ where T is the bit length of the input data

Set a new system

$$\begin{aligned} &Ax = b, \\ &-x \leq \tfrac{\nu}{2}\mathbf{1}. \end{aligned} \tag{15}$$

Parameterize (15):

$$\begin{aligned} &Ax - bt = 0, \\ &-x - (\tfrac{\nu}{2}\mathbf{1})t \leq \mathbf{0}, \\ &-t \leq -1. \end{aligned} \tag{16}$$

Run Chubanov's D&C subroutine on the strengthened version of (16)

$$\begin{aligned} &Ax - bt = 0, \\ &-x - (\tfrac{\nu}{2}\mathbf{1})t \leq -\mathbf{1}, \\ &-t \leq -2. \end{aligned} \tag{17}$$

with $z = \mathbf{0}$, $r = 2^{5(n+1)^2\phi}\sqrt{n+1}$ where ϕ is the facet complexity of (17), and $\epsilon = 1$.

If a feasible solution (x^*, t^*) is found

Return $\hat{x}$ a feasible solution to $Ax = b, -x \leq \mathbf{0}$ obtained by an application of Lemma 4.7, using the feasible solution $\frac{x^*}{t^*}$ for (14)

Else Else (Chubanov's D&C subroutine fails or returns a separating hyperplane)

Return "The system (15) is INFEASIBLE"

Theorem 4.8. *The LFG Algorithm correctly determines a feasible point x^* of (1) or determines the system is infeasible in a finite number of steps.*

Proof. To prove the correctness of the algorithm, we need to look at each of the three cases Chubanov's D&C subroutine can return.

CASE 1: An ϵ-approximate solution (x^*, t^*) is found for (17). Then (x^*, t^*) is an exact solution to (16), and $\frac{x^*}{t^*}$ is a solution to (15), and hence is also a solution to (14). Using Lemma 4.7, we can construct a feasible solution to our original system (1).

CASE 2: The D&C subroutine fails to complete, returning consecutive separating hyperplanes (h_1, δ_1) and (h_2, δ_2) such that $h_1 = -\gamma h_2$ for some $\gamma > 0$. By Proposition 3.3, we know that there exists $k \in \{1, \ldots, l\}$ such that $-x_k = \frac{\nu}{2}$ for all solutions to $Ax = b, -x \leq \frac{\nu}{2}\mathbf{1}$. But this simply implies that $Ax = b, -x \leq \mathbf{0}$ is infeasible.

CASE 3: The final case is when the D&C returns a separating hyperplane (h, δ). Note that due to the $\hat{r}$ we use, Lemma 4.6 implies (17) is infeasible. Then by Lemma 3.2, we know there exists some $k \in \{1, \ldots l\}$ such that $-x_k = \frac{\nu}{2}$ for all x satisfying $Ax = b, -x \leq \frac{\nu}{2}\mathbf{1}$. Again, as in Case 2, this implies that $Ax = b$, $-x \leq \mathbf{0}$ is infeasible. □

Since the running time of the D&C subroutine is a polynomial in $\hat{r}$, and $\hat{r}$ is exponential in the input data, our algorithm is not guaranteed to run in polynomial time. However, as mentioned in the Introduction, it has certain advantages over Chubanov's polynomial time LP algorithm from [7]. First, it avoids the complicated reformulations used by Chubanov. Second, the Chubanov Relaxation algorithm requires multiple iterations of the D&C subroutine, whereas the LFG algorithm uses only one single application of the D&C subroutine.

5. Computational Experiments

In this final section, we compare the computational performance of the Chubanov Relaxation algorithm and the LFS algorithm, which we mention in the preceding sections, with the original relaxation algorithm by Motzkin and Schoenberg. We also revisit some of the experiments done more than sixty years ago by Hoffman et al. [11]. The first subsection

describes the details of the algorithm implementations as well as the problems used in the experiments. The second subsection discusses the results of our experiments, and in the final subsection we draw some conclusions.

5.1. Implementation Setup

We implemented the following three algorithms using MATLAB 7.12.0:

1. *The Chubanov Relaxation algorithm*, exactly as described in [6]. Apart from the input data A, b, C, d, this algorithm requires as input a real number r such that the feasible region is contained in $B(0, r)$. In binary problems r is taken as $\sqrt{n+1}$, while in general linear feasibility problems it is taken as the input dependent bound given in Lemma 4.6.

   Just as in [6], we say this algorithm finishes when it returns either a feasible solution or a message saying no integer solutions exist. The times in the tables below reflect only the time it takes for the algorithm to finish. Note that in [7], Chubanov presented a polynomial reduction that transforms an arbitrary linear system of the form

   $$Ax = b,\ \mathbf{0} \leq x \leq u,$$

   where all coefficients are integer and u is assumed to be strictly positive, into a binary problem

   $$A'x = b',\ \mathbf{0} \leq x \leq \mathbf{1},$$

   such that the linear system has a solution if and only if the binary problem has a 0-1 solution. The binary problem is then solved via the Chubanov Relaxation algorithm. But for this to be a practical solution for arbitrary linear systems, the Chubanov Relaxation algorithm must be practical first. This is why we only test the Chubanov Relaxation algorithm and did not implement the linear system reduction in [7].

2. *Our LFS algorithm*, as described in Section 4.1. This algorithm also requires as input a real number r such that the feasible region is contained in $B(0, r)$. As with the Chubanov Relaxation algorithm above, r is taken as $\sqrt{n+1}$ for binary problems, and for general linear feasibility problems it is taken as the input dependent bound given in Lemma 4.6. Moreover, we require as input Δ_A, the maximum subdeterminant of the matrix A.

This algorithm finishes when it returns either a feasible solution or a message saying no strictly feasible solutions exist. The times in the tables below only report the run time of the algorithm, and do not account for any preprocessing such as computing the parameter Δ_A.

3. *The Motzkin Relaxation algorithm*, as described in [15]. In our implementation, we use the normal convention and transform equality constraints into two inequality constraints since the original Motzkin's algorithm was only formulated for systems of linear inequalities. Along with the input data A, b, C, d, this algorithm needs an under-/over-projection constant λ and an error constant ϵ, as the algorithm may only converge to a feasible solution. We use $\lambda = 1.9$ and $\epsilon = 10^{-6}$. This algorithm only ends when an ϵ-approximate feasible solution is found. The times reported in the tables below are the run times for feasible problems.

At this point it is important to note that of the three algorithms we are testing, only our LFS is an actual Linear Feasibility algorithm. The Motzkin Relaxation algorithm has no way to determine when a linear system is infeasible. If a system is feasible, the algorithm will either hit a feasible point or converge to one. But if the system is not feasible, the algorithm will keep cycling indefinitely. On the other hand, the Chubanov Relaxation algorithm only decides if a solution to the linear system exists or decides *no integer* solutions exist. Thus, it may determine no integer solutions exist when the system is in fact feasible (or it could find a feasible solution when no integer solutions exist). The algorithm of [8] was not included in the present experiments as we completed our study well before his new paper went through the referee process.

To allow for a fair comparison, in our selection of problems instances we were careful to choose only problems with feasible LP relaxations. This way we know the Motzkin Relaxation algorithm only timed out due to the difficulty of the problem, and not simply because it was cycling on an infeasible problem. Also, since the Chubanov Relaxation algorithm and the LFS algorithm perform best within a cube, we primarily tested problems that, if feasible, only have solutions in the 0-1 cube. More precisely we used problem instances from the following three sources in our experiments:

1. *Examples from Telgen and Goffin [10, 21]:* These problems have the form

$$\begin{pmatrix} -1 & 2^\alpha \\ 0 & -1 \end{pmatrix} x \leq \begin{pmatrix} -2^\alpha \\ 0 \end{pmatrix}$$

and were first used to prove the Motzkin Relaxation algorithm is not a polynomial algorithm. We set $r = 2^{\alpha+1}$ when running this example through the Chubanov Relaxation algorithm and the LFS. This r ensured the initial ball contained a reasonable portion of the unbounded feasible region as well as an integer solution.

These are the only instances we tested where the feasible region of the experimental problems was not bounded by the 0-1 cube, but they are feasible and have integral solutions. We experimented on this set in order to see what impact induced hyperplanes may have when dealing with constraints that intersect at a very acute angle. The work of Goffin and Telgen used the variation of the angle by α to demonstrate the exponential behavior of Motzkin's algorithm. The results of the experiments using this problem set can be seen in Table 1.

2. *Examples from Hoffman et al. [11]:* We ran experiments using all six versions of the example in Hoffman et al. [11]. In this paper the authors start with the problem

$$\begin{pmatrix} 0 & 1 & -2 & -1 & 3 & -2 & -1 & -4 & 1 & -2 \\ -1 & 0 & -1 & 1 & 2 & -2 & 1 & 1 & -1 & -1 \\ 2 & 1 & 0 & -3 & 1 & 1 & 3 & -3 & 1 & -1 \\ 1 & -1 & 3 & 0 & 1 & -1 & 4 & 2 & -1 & 5 \\ -3 & -2 & -1 & -1 & 0 & -1 & -5 & 6 & 1 & 6 \\ 2 & 2 & -1 & 1 & 1 & 0 & -2 & -1 & -1 & 3 \\ 1 & -1 & -3 & -4 & 5 & 2 & 0 & 2 & 1 & -4 \\ 4 & -1 & 3 & -2 & -6 & 1 & -2 & 0 & 1 & -1 \\ -1 & 1 & -1 & 1 & -1 & 1 & -1 & -1 & 0 & -5 \\ 2 & 1 & 1 & -5 & -6 & -3 & 4 & 1 & 5 & 0 \end{pmatrix} x \leq \mathbf{0}, \quad x \geq \mathbf{0}, \quad \|x\|_1 = 1.$$

The 5-dimensional problem is then created by removing the last five rows and columns of the above matrix, the 6-dimensional problem is created by removing the last four rows and columns, and so on. We used $r = \sqrt{n+1}$ in our experiments since the final two constraints above ensure the feasible region is restricted to the 0-1 cube (in fact it is restricted to the unit simplex). The results of these experiments can be seen in Table 2.

3. *MIPLIB Problems [5, 13]:* We experimented with the LP relaxations of some binary problems from the MIPLIB repository in order to ensure the feasible regions of the LP relaxations are in the 0-1 cube. As a result we used $r = \sqrt{n+1}$ in the experiments. We used the smaller binary instances available in MIPLIB so that MATLAB is able to easily deal with each problem. The results can be seen in Table 3.

As noted above, the algorithms and experiment scripts were all developed in MATLAB 7.12.0. No parallelization was incorporated into the algorithms. The computational experiments were run on a personal computer with an Intel Core i5 M560 2.67 GHz processor. The code and the problems used are available at the following webpage :
`https://www.math.ucdavis.edu/~deloera/chubanovcode/code.html`.

5.2. Performance

The tables below show the results of each of the experiments we ran. A dash, "–", in any of the tables indicates the corresponding algorithm took longer than three hours to finish running on that particular experiment. The following list talks about the results recorded in the corresponding table.

1. *Table 1 - Telgen Goffin Results:* These experiments best show the difference induced hyperplanes can make. We can easily see that as α increases, the Chubanov Relaxation algorithm and the LFS outperform the Motzkin Relaxation algorithm. This is because the Motzkin Relaxation algorithm can only project from one constraint to the other, while the other algorithms create induced hyperplanes to which they can project.

   We can also see the LFS algorithm consistently outperforms the Chubanov Relaxation algorithm. This is because the Chubanov Relaxation algorithm must go through several iterations of the D&C subroutine, as it adds new equality constraints (see [6] and Section 2), in order to find a feasible solution. Note that in these simple examples the Chubanov Relaxation will always find a feasible solution to the linear feasibility problem. This is because there always exists an integer feasible solution, namely $(2^\alpha, 0)$ and thus, it cannot report that no integer solutions were found. In contrast, our LFS uses a single application of the D&C subroutine.

2. *Table 2 - Hoffman Results:* The Motzkin Relaxation algorithm outperformed the other two algorithms every time, except in the 9-dimensional experiment, where the Motzkin Relaxation algorithm took longer than our three hour limit. This happened in spite of the fact the Hoffman problems, as described above, are bounded to the 0-1 cube (in fact the feasible region is a subset of the standard $(d-1)$ dimensional simplex in $\mathbb{R}^d$). And this type of region is precisely where the Chubanov Relaxation algorithm runs in strongly polynomial time.

We can also see the Chubanov Relaxation algorithm outperformed the LFS in every case. In fact the LFS reached our three hour time limit in the larger Hoffman experiments. This was due to a couple of different reasons. First, the Chubanov Relaxation algorithm did not actually find a feasible solution. Instead, it finished as soon as it determines no integer solutions are possible. Part of the extra running time of the LFS came from finding an actual feasible point. The second reason the Chubanov Relaxation algorithm outperformed the LFS is the much larger radius $\hat{r}$ the LFS must use in its algorithm, $\hat{r} = 2n\Delta\sqrt{r^2+1}$ with $\Delta \geq \Delta_A$, as opposed to the $\hat{r} = (2l+3)(r+1)$ that is used in the Chubanov Relaxation algorithm. The larger $\hat{r}$ came primarily from a large Δ_A. This $\hat{r}$ used by the LFS resulted in a greater recursion depth, which in turn meant the LFS spent more time traversing these recursions.

3. *Table 3 - MIPLIB Results:* First, an explanation of how we ran these experiments with the LFS algorithm. The algorithm needed as input the parameter $\Delta \geq \Delta_A$ for each problem, and we were unable to compute Δ_A in a reasonable amount of time. We decided to simply run each instance with larger and larger values of Δ, until we were able to find a solution (we knew the problems are all feasible) - see Remark 4.1. In particular, we ran the LFS algorithm on the MIPLIB problem using $\Delta = 1$, $\Delta = 2$, $\Delta = 4$, ... , and so on until a feasible point was found. In every case that did not time out, $\Delta = 1$ was sufficient to find a solution (even though A is not totally unimodular and $\Delta_A > 1$). Timing out after three hours in these experiments means the final iteration with $\Delta = 2^k$, where k is some nonnegative integer, took longer than three hours.

   The final table shows that, with one single exception, the Motzkin Relaxation algorithm far outperformed both the Chubanov Relaxation algorithm and the LFS, while the LFS is shown to be the worst performer. This time the performance of the LFS is not due to a large Δ_A but because of an increase in the dimension n, caused by the introduction of a large number of slack variables. The creation of these new variables and equality constraints significantly slowed down MATLAB's ability to calculate the projection onto the affine space defined by $Ax = b$.

5.3. Conclusions

As we can see from the results discussed above, in most problems the Motzkin Relaxation algorithm outperformed the other two relaxation algorithms we tested, with the exception of the Telgen-Goffin instances, which were designed to expose the weaknesses of Motzkin. Even with the binary MIPLIB problems, specifically chosen to try and highlight the strengths of the Chubanov Relaxation algorithm and the LFS algorithm, the simplicity of the original Motzkin Relaxation algorithm consistently outperformed the other two.

While this is the first ever implementation of the Chubanov Relaxation algorithm and the LFS algorithm to be attempted, it appears the theoretical power of induced hyperplanes does not translate into practice. The Motzkin Relaxation algorithm avoids computing matrix inverses, and this allows it to complete many more iterations than the other two algorithms in the same amount of time. In addition, the geometry of the D&C subroutine indicates that because all the inequality constraints are of the form $x \geq \mathbf{0}$, the ability to create induced hyperplanes gives the LFS no advantage over the Motzkin Relaxation algorithm, let alone the Chubanov Relaxation algorithm. Of course, one must remember that the LFS is the only linear feasibility algorithm, and the other two may not detect infeasibility correctly.

Finally, given our experience with the parameter $\Delta \geq \Delta_A$ we feel that algorithms whose time complexity relies on this parameter might not be competitive in practice.

Table 1: Results of the Telgen Experiments

	Chubanov Relaxation		LFS		Motzkin Relaxation	
Experiment	Recursions	Time (Sec)	Recursions	Time (Sec)	Iterations	Time (Sec)
Telgen ($\alpha = 1$)	135	0.0155	79	0.0227	**6**	**0.0008**
Telgen ($\alpha = 2$)	301	0.0269	123	0.0139	**13**	**0.0007**
Telgen ($\alpha = 3$)	616	0.0445	355	0.0367	**27**	**0.0014**
Telgen ($\alpha = 4$)	1,232	0.0873	484	0.0450	**93**	**0.0042**
Telgen ($\alpha = 5$)	1,774	0.1289	**790**	0.0668	1,591	**0.0658**
Telgen ($\alpha = 6$)	3,834	0.2661	**372**	**0.0350**	6,813	0.2769
Telgen ($\alpha = 7$)	782	0.0531	**352**	**0.0288**	27,665	1.1077
Telgen ($\alpha = 8$)	6,506	0.4343	**1,431**	**0.1102**	111,059	4.4575
Telgen ($\alpha = 9$)	6,133	0.4288	**5,015**	**0.4115**	444,631	17.7989
Telgen ($\alpha = 10$)	29,434	1.8719	**5,262**	**0.4790**	1,778,925	71.5001

Acknowledgements The second author was supported by NSF grant DMS-0914107. The third author was supported by the NSF-VIGRE grant DMS-0636297. We are grateful to

Table 2: Results of the Hoffman et al. Experiments

	Chubanov Relaxation		LFS		Motzkin Relaxation	
Experiment	Recursions	Time (Sec)	Recursions	Time (Sec)	Iterations	Time (Sec)
5-Dim Hoffman	19,025	1.7969	93,070	7.7065	**296**	**0.0137**
6-Dim Hoffman	29,588	2.4028	1,037,271	84.340	**509**	**0.0222**
7-Dim Hoffman	74,187	5.7864	41,715,915	3,304.47	**609**	**0.0270**
8-Dim Hoffman	163,010	13.0156	–	–	**4,248**	**0.1816**
9-Dim Hoffman	**108,649**	**8.1142**	–	–	–	–
10-Dim Hoffman	423,338	33.3376	–	–	**16,284**	**0.6921**

Table 3: Results of the MIPLIB Experiments

	Chubanov Relaxation		LFS		Motzkin Relaxation	
Experiment	Recursions	Time (Sec)	Recursions	Time (Sec)	Iterations	Time (Sec)
COV1075	1,688	0.3210	11,245	399.797	**1**	**0.0008**
IIS-100-0-COV	953	0.9470	1,911	9,798.78	**20**	**0.0250**
LSEU	10,330	1.0228	278,287	131.696	**24**	**0.0035**
M100N500K4R1	10,944	15.6991	168,739	6,288.44	**0**	**0.0011**
MOD008	5,244	1.1101	430,528	2,749.04	**1**	**0.0008**
NEOS-1440225	–	–	–	–	**16,261**	**143.741**
NEOS-1616732	2,387	2.8928	3,520	3,789.41	**128**	**0.1728**
NEOS788725	5,751	2.2866	72,405	3,340.31	**2,113**	**1.8194**
NEOS-807456	–	–	–	–	**376,250**	**6,580.68**
NEOS-820146	8,685	17.6941	–	–	**2,130**	**6.2027**
NEOS-820157	**7,102**	**16.4914**	–	–	10,050	90.5661
NEOS-849702	–	–	–	–	**41,378**	**725.148**
NEOS858960	4,146	0.7315	20,191	52.191	**1**	**0.0014**
P2M2P1M1P0N100	22,573	2.0216	11,974	4.8804	**86**	**0.0158**
QUEENS-30	16,019	33.2726	–	–	**0**	**0.0038**
REBLOCK67	1,345,129	1,062.71	–	–	**0**	**0.0042**
SEYMOUR	–	–	–	–	**15,783**	**345.084**
STEIN27	73	0.0076	92	0.0424	**1**	**0.0005**
STEIN45	392	0.0649	196	0.7682	**1**	**0.0008**

Sergei Chubanov for sharing his new article with us, and to Kurt Anstreicher, Antoine Deza, and two anonymous referees for their useful comments. We are grateful to Dan Steffy and Matthias Walter for sharing their software and MIPLIB insights.

References

[1] S. Agmon. The relaxation method for linear inequalities. *Canadian J. Math.*, 6:382–392, 1954.

[2] E. Amaldi and R. Hauser. Boundedness theorems for the relaxation method. *Math. Oper. Res.*, 30(4):939–955, 2005.

[3] A. Belloni, R. Freund, and S. Vempala. An efficient re-scaled perceptron algorithm for conic systems. *Mathematics of Operations Research*, 34(3):621–641, 2009.

[4] U. Betke. Relaxation, new combinatorial and polynomial algorithms for the linear feasibility problem. *Discrete Comput. Geom.*, 32(3):317–338, 2004.

[5] R. E. Bixby, S. Ceria, C. M. McZeal, and M. W. P Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, 58:12–15, 1998.

[6] S. Chubanov. A strongly polynomial algorithm for linear systems having a binary solution. *Mathematical Programming*, 134:533–570, 2012.

[7] S. Chubanov. A polynomial relaxation-type algorithm for linear programming, unpublished manuscript (2011).

[8] S. Chubanov. A polynomial relaxation-type algorithm for linear programming, unpublished manuscript (2012).

[9] J.-L. Goffin. The relaxation method for solving systems of linear inequalities. *Math. Oper. Res.*, 5(3):388–414, 1980.

[10] J.-L. Goffin. On the nonpolynomiality of the relaxation method for systems of linear inequalities. *Math. Programming*, 22(1):93–103, 1982.

[11] A. Hoffman, M. Mannos, D. Sokolowsky, and N. Wiegmann. Computational experience in solving linear programs. *Journal of the Society for Industrial and Applied Mathematics*, 1(1):pp. 17–33, 1953.

[12] S. Kaczmarz. Approximate solution of systems of linear equations. *Internat. J. Control*, 57(6):1269–1271, 1993. Translated from the German original of 1933.

[13] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. E. Bixby, E. Danna, G. Gamrath, A. M. Gleixner, S. Heinz, A. Lodi, H. Mittelmann, T. Ralphs, D. Salvagnin, D. E. Steffy, and K. Wolter. MIPLIB 2010. *Mathematical Programming Computation*, 3(2):103–163, 2011.

[14] J.-F. Maurras, K. Truemper, and M. Akgül. Polynomial algorithms for a class of linear programs. *Math. Programming*, 21(2):121–136, 1981.

[15] T. S. Motzkin and I. J. Schoenberg. The relaxation method for linear inequalities. *Canadian J. Math.*, 6:393–404, 1954.

[16] D. Needell. Randomized Kaczmarz solver for noisy linear systems. *BIT*, 50(2):395–403, 2010.

[17] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization : Algorithms and Complexity*. Dover Books on Computer Science. Courier Dover Publications, 1998.

[18] A. Schrijver. *Theory of linear and integer programming*. Wiley-Interscience Series in Discrete Mathematics. John Wiley & Sons Ltd., 1986. A Wiley-Interscience Publication.

[19] T. Strohmer and R. Vershynin. A randomized Kaczmarz algorithm with exponential convergence. *J. Fourier Anal. Appl.*, 15(2):262–278, 2009.

[20] E. Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Math. of Oper. Res.*, 34(2):250–256, 1986.

[21] J. Telgen. On relaxation methods for systems of linear inequalities. *European J. Oper. Res.*, 9(2):184–189, 1982.

[22] S. Vavasis and Y. Ye. A primal-dual interior point method whose running time depends only on the constraint matrix. *Mathematical Programming*, 74:79–120, 1996.