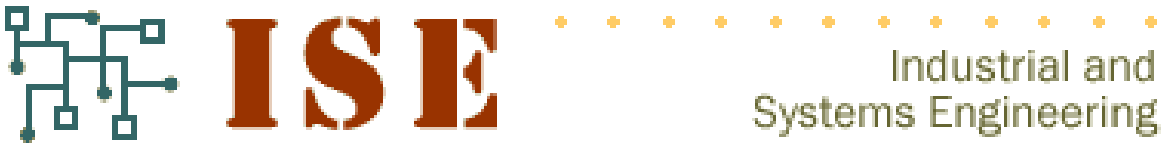

A Polynomial Column-wise Rescaling von Neumann Algorithm

Dan Li

Department of Industrial and Systems Engineering, Lehigh University, USA

Cornelis Roos

Department of Information Systems and Algorithms, Delft University of Technology, Netherlands

Tamás Terlaky

Department of Industrial and Systems Engineering, Lehigh University, USA

ISE Technical Report 15T-010

A Polynomial Column-wise Rescaling von Neumann Algorithm

Dan Li, Cornelis Roos, and Tamás Terlaky

July 2015

Recently Chubanov proposed a method which solves homogeneous linear equality systems with positive variables in polynomial time. Chubanov's method can be considered as a column-wise rescaling procedure. We adapt Chubanov's method to the von Neumann problem, and so we design a polynomial time column-wise rescaling von Neumann algorithm. This algorithm is the first variant of the von Neumann algorithm with polynomial complexity.

1 Introduction

The von Neumann problem is a Linear Feasibility Problem (LFP) with the following form:

$$Ax = 0, \quad e^T x = 1, \quad x \geq 0, \tag{1}$$

where A is an $m \times n$ matrix, $x \in \mathbb{R}^n$, and e denotes the all one vector in $\mathbb{R}^n$. In this paper, we only consider matrices A with integral entries. Without loss of generality, we may assume that rank(A)=m. With a matrix A given as in (1) we also consider the system

$$Ax = 0, \quad 0 \neq x \geq 0. \tag{2}$$

Any solution of (2) can be transferred to a feasible solution of (1) by the normalization $x := \frac{x}{e^T x}$.

As the alternative system [15] of the von Neumann problem (1), the perceptron problem is also an LFP in the following specific form

$$A^T y > 0, \tag{3}$$

where $y \in \mathbb{R}^m$. Being a pair of alternative systems, exactly one of the two LFPs (1) and (3) is feasible, by Farkas Lemma. In other words, problem (1) has a solution if and only if (3) has no solution.

Several algorithms have been proposed for solving LFPs, such as simplex methods [2,21], ellipsoid method [13, 14], interior point methods [19], Chubanov's method [5, 6], variants of the perceptron algorithm [3, 4, 22], and variants of the von Neumann algorithm [7, 8]. All of these algorithms aim to find a feasible solution to a linear optimization problem, or equivalently to an LFP. They either deliver a feasible solution, or provide an evidence of infeasibility.

The perceptron algorithm [20] was originally invented in the field of machine learning, i.e., it solves data classification problems by learning a linear threshold function. It solves the perceptron problem in the form (3). In its original form, it is not a polynomial-time algorithm. Its complexity is $O(\frac{1}{\rho_p^2})$, where ρ_p is the width of the cone of feasible solutions, which can be exponentially small. In order to speed up the perceptron algorithm, two rescaling variants were proposed. Dunagan and Vempala [10] purposed a randomized rescaling perceptron algorithm. It expands the width of the feasible region ρ_p with high probability by periodically rescaling the linear system. The total complexity of the randomized rescaling perceptron algorithm is $O(n^5 \log n \log(\frac{1}{\rho_p}))$ with high probability. Observe that even though the complexity bound becomes polynomial, the algorithm has a stochastic component, and the bound holds only with high probability. Recently, Peña and Sohèili [17] purposed a deterministic rescaling perceptron algorithm, which enjoys an $O(n^6 \log(\frac{1}{\rho_p}))$ polynomial-time complexity. Observe that the complexity of the stochastic version is better by a factor of $\frac{n}{\log n}$.

The von Neumann algorithm, originally published by Dantzig [7, 8], can also be considered as a special case of the Frank-Wolfe algorithm [1, 11], which is an iterative method for solving linearly constrained convex optimization problems. The von Neumann algorithm is not a polynomial-time

algorithm. Its complexity of finding an ϵ-approximate solution is $O(\frac{1}{\rho_v^2}\ln\frac{1}{\epsilon})$. Analogous to ρ_p, the quantity ρ_v quantifies how deep the origin is in the convex hull defined by the columns of matrix A. Inspired by the success of rescaling perceptron algorithms, we aim to design a rescaling variant of the von Neumann algorithm with deterministic polynomial-time complexity.

Chubanov [5,6] has recently proposed novel polynomial-time algorithms for solving homogeneous linear systems with positive variables. It is a divide-and-conquer algorithm which can be considered as a generalization of the relaxation method [16]. The so-called Basic Procedure is the core of the method. If it neither finds a feasible solution, nor identifies the infeasibility of the system, then the basic procedure identifies an upper bound for at least one coordinate of any possible feasible solution. According to this upper bound for the identified coordinates, the corresponding columns of the coefficient matrix are multiplied by a scalar. Therefore, Chubanov's method can be also considered as a rescaling procedure. Utilizing this idea, we propose a deterministic column-wise rescaling von Neumann algorithm and prove its polynomial-time complexity. We rename the basic procedure as von Neumann Procedure (vNP) because it uses von Neumann-like update steps and it is a subroutine of this variant of the von Neumann algorithm. The outline of this paper is as follows. In the next section we introduce some notations and important lemmas that serve as the foundation of Chubanov's method. In Section 4, we present the details of the column-wise rescaling von Neumann algorithm. In Section 3.1, we introduce different ways to compute upper bounds for some coordinates of any feasible x, if problem (2) has feasible solutions. These bounds are utilized to construct a rescaling matrix. The complexity analysis is presented in Section 5, and computational results are given in Section 6.

2 Preliminaries

Before presenting the details of the column-wise rescaling perceptron algorithm, we first introduce important notations and lemmas which are the foundation of Chubanov's method. Let $\mathcal{N}_A$ denote

the null space of the matrix A and $\mathcal{R}_A$ its row space, i.e.,

$$\mathcal{N}_A := \{x \in \mathbb{R}^n : Ax = 0\}, \quad \mathcal{R}_A := \{A^T y : y \in \mathbb{R}^m\}.$$

We define matrices P_A and Q_A as the orthogonal projection matrices of $\mathbb{R}^n$ onto $\mathcal{N}_A$ and $\mathcal{R}_A$, respectively, as follows:

$$P_A := I - A^T(AA^T)^{-1}A, \quad Q_A := A^T(AA^T)^{-1}A = I - P_A.$$

Our assumption that matrix A is full rank guarantees that AA^T is invertible. So P_A and Q_A are well defined. Let $x^{\mathcal{N}}$ and $x^{\mathcal{R}}$ denote the orthogonal decomposition of vector x in the spaces $\mathcal{N}_A$ and $\mathcal{R}_A$, respectively, i.e.,

$$x^{\mathcal{N}} := P_A x, \quad x^{\mathcal{R}} := Q_A x.$$

Obviously we have

$$AP_A = 0, \quad P_A Q_A = 0, \quad x = x^{\mathcal{N}} + x^{\mathcal{R}}.$$

According to the properties of the orthogonal decomposition [12], $P_A x = 0$ holds if and only if $x \in \mathcal{R}_A$, i.e., $x = A^T y$ holds for some y. In other words, problem (3) is equivalently solvable to the following problem

$$P_A x = 0, \quad x > 0. \tag{4}$$

Since problem (4) is homogeneous and x is strictly positive, without loss of generality, we may assume that $e^T x = 1$. The concept of the orthogonal decomposition plays a crucial role in Chubanov's method. The following lemma summarize the relationship between the orthogonal components and the solutions of problems.

Lemma 1. *For a vector $x \in \mathbb{R}^n$, if we have $0 \neq P_A x \geq 0$, then $x^{\mathcal{N}}$ is a solution to problem (2) and problem (1) is also solvable; if $P_A x = 0$ for some $x > 0$, i.e., x is a solution to problem (4), then problem (3) is solvable, i.e., $x = A^T y$ holds for some y.*

Proof. The first statement immediately follows from the definitions of $\mathcal{N}_A$ and P_A. For the second statement, if x is a solution to problem (4), then we have $x = x^{\mathcal{R}} + x^{\mathcal{N}} = x^{\mathcal{R}} \in \mathcal{R}_A$, which implies $x = A^T y > 0$ has a solution, i.e., problem (3) is feasible. By Farkas Lemma, problem (1) has no solution. □

Lemma 1 shows that the value of $P_A x$ for some x can be used to solve problem (1) when $0 \neq P_A x \geq 0$ or identify the feasibility when $P_A x = 0$. Therefore, as we show in the next section, Lemmas 1 serves as stopping criteria for the vNP in Chubanov's method.

3 The von Neumann Procedure

Chubanov's method is solving the homogeneous linear inequality system

$$Ax = 0, \quad x > 0. \tag{5}$$

Since this system is homogeneous, we may assume that $0 < x \leq e$, where e denotes the all-one vector. Thus, we may equivalently consider the problem

$$Ax = 0, \quad x \in (0, 1]^n. \tag{6}$$

The solution set of problem (6) is a subset of the unit cube. The major difference between (1) and (6) is that every solution of (6) has to be strictly positive, while solutions of (1) still may have zero coordinates.

3.1 Bounds for feasible solutions

The core of Chubanov's method is the vNP. The vNP is a von Neumann-like algorithm and works on problem (4). For the purpose of clarification, vector $u \in \mathbb{R}^n$ is used to denote the variable in problem (4) in the rest of this paper. Vector x is only used in the problems whose coefficient matrix is A, such as problems (1), (2), and (6). With the new notation, problem (4) can be rewritten as follows.

$$P_A u = 0, \quad e^T u = 1, \quad u > 0. \tag{7}$$

Recall that $u^{\mathcal{N}} = P_A u$ and $u^{\mathcal{R}} = Q_A u$. Due to the fact that P_A and Q_A are orthogonal projection matrices, with the assumption that problem (1) or (6) is feasible, an upper bound of every feasible solution x may be obtained from a given vector u. Let vector $d > 0$ denote this upper bound for x and its i-th coordinate d_i represent the upper bound for x_i, i.e., $x_i \leq d_i$ holds for every feasible solution x and every coordinate i. We will show later that vector d is crucial for rescaling. First we have the following observation.

Lemma 2. *Let vector d be an upper bound for every feasible solution x of problem (6). If $\max(d) < 1$, then problem (6) is infeasible.*

Proof. Observe that problem (6) is homogenous. Assume that it is feasible and x' is a feasible solution, then $x = \frac{x}{\max(x)}$ is another feasible solution which has at least one coordinate equal to 1. In other words, we have $x_j = 1$ for some j. According to the definition of vector d, $x_i \leq d_i$ holds for every feasible solution x and every coordinate i. Therefore, if problem (6) is feasible, then d_j has to be at least 1 for some j. □

Lemma 2 is utilized as an evidence of infeasibility in Algorithm 3 in Section 4. There are several ways to compute such an upper bound for x. Chubanov's original method uses the bound [5, 6]

$$x_i \leq d_i = \frac{\sqrt{n}\|u^{\mathcal{N}}\|}{u_i}, \tag{8}$$

where the subscript i is the index for coordinates. This inequality provides a useful bound only if the right hand side expression is smaller than 1, because our assumption is that x is in the unit cube for problem (6). To determine if there is a bound (8) not greater than $\frac{1}{2}$ for some i, we can simply test the inequality

$$2\sqrt{n}\|u^{\mathcal{N}}\| \leq \max_i(u_i). \tag{9}$$

Chubanov proved the following lemma.

Lemma 3. *[5, 6] Let u satisfy $0 \neq u \geq 0$ and (9), and let j be such that $u_j = \max_i(u_i)$. Let x be a solution for (6). Then x_j is bounded above by $d_j = \frac{1}{2}$.*

It will be convenient to call u *small* if it satisfies (9), and *large* otherwise. Note that $u^{\mathcal{N}} \neq 0$ if u is large, and u is small if (4) is feasible. For future use we also state the following result.

Lemma 4. *[18] If u satisfies $2\sqrt{n}\|u^{\mathcal{N}}\| \leq e^T u$, then u is small.*

Lemma 3 shows that any small vector u induces a bound $x_j \leq \frac{1}{2}$ for some j for problem (6). Recently more study shows that some large vectors u may also provide available bounds for x. Roos [18] proposed a modified vNP using the following bound.

Lemma 5. *[18] Let x be a solution for (6). Then x_i is bounded by*

$$x_i \leq d_i = \min\left\{1, e^T\left[\frac{u^{\mathcal{R}}}{-u_i^{\mathcal{R}}}\right]^+\right\}, \text{ for } i = 1, \cdots, n, \tag{10}$$

where $[a]^+$ arises from a by replacing its negative entries by zero, i.e., $[a]_j^+ = \max\{0, a_j\}$

By using the duality theorem of LO, Chubanov [6] also derived another bound in Lemma 6.

Lemma 6. *[18] Let x be a solution for (6). Then x_i is bounded by*

$$x_i \leq d_i = \min\left\{1, e^T\left[e_i - \frac{u^{\mathcal{R}}}{u_i}\right]^+\right\}, \text{ for } i = 1, \cdots, n, \tag{11}$$

where e_i is the i-th unit vector.

Among these three bounds (8), (11), and (10), Roos [18] concludes that for each nonzero $u \geq 0$ and for each i, one has

$$\min\left\{1, e^T\left[\frac{u^{\mathcal{R}}}{-u_i^{\mathcal{R}}}\right]^+\right\} \leq \min\left\{1, e^T\left[e_i - \frac{u^{\mathcal{R}}}{u_i}\right]^+\right\} \leq \min\left\{1, \frac{\sqrt{n}\|u^{\mathcal{N}}\|}{u_i}\right\}. \tag{12}$$

Bound (10) is the tightest upper bound for x. In the vNP, we only need to compute the smallest bound among all coordinates, so we define

$$d_{\min} \quad := \quad \min_i d_i = d_j,$$

where j is as follows: for the bounds (11) and (10), j is the index such that $u_j^{\mathcal{R}} = \max_i(u_i^{\mathcal{R}})$ if $e^T u^{\mathcal{R}} > 0$ and $u_j^{\mathcal{R}} = \min_i(u_i^{\mathcal{R}})$ if $e^T u^{\mathcal{R}} < 0$; for the bound (8), j is the index such that $u_j = \max_i(u_i)$. Note that j might not be unique.

3.2 The von Neumann Procedure (vNP)

By iteratively updating vector u and the value of $P_A u$, the vNP aims to find a vector u which either satisfies one of the two conditions in Lemma 1, or if such an u is not found, then $u^{\mathcal{N}} \neq 0$ and there is at least one nonpositive coordinate of $u^{\mathcal{N}}$. Let K denote a nonempty set of indices such that

$$\sum_{k \in K} u_k^{\mathcal{N}} \leq 0.$$

Let p_k denote the k-th column of P_A, i.e., $p_k = P_A e_k$. We define

$$e_K := \frac{1}{|K|}\sum_{k \in K} e_k, \quad p_K := P_A e_K = \frac{1}{|K|}\sum_{k \in K} p_k.$$

The vNP is shown in Algorithm 1. Recall that vector u in problem (7) is analogous to vector x in problem (1). To solve problem (7), Algorithm 1 starts with u as a point from the unit simplex. Vector $u^{\mathcal{N}} = P_A u$ is analogous to vector $b = Ax$ in the von Neumann algorithm [15]. Line 12-15 in Algorithm 1 is the update step. It moves $u^{\mathcal{N}}$ along a direction, which is a combination of one

Algorithm 1 [$\bar{u}$, u, $u^{\mathcal{N}}$, $\tilde{J}$, $\tilde{d}$, CASE]=von Neumann Procedure(P_A, u)

Initialize: $\bar{u} = 0$, $u^{\mathcal{N}} = P_A u$, $K = \tilde{J} = \emptyset$, $d_{\min} = 1$, CASE $= 0$, $\frac{1}{2} < \theta < 1$ (e.g. $\theta = 0.8$).
while $d_{\min} > \frac{1}{2}$ **and** CASE $= 0$ **do**
  if $0 \neq u^{\mathcal{N}} \geq 0$ **then**
    CASE $= 1$ ▷ Problem (1) is feasible.
    Return
  else
    if $u^{\mathcal{N}} = 0$ **then**
      CASE $= 2$ ▷ Problem (1) is infeasible.
      Return
    else
      $\bar{u} = u$
      Find K such that $\sum_{k \in K} u_k^{\mathcal{N}} \leq 0$
      $\alpha = \frac{p_K^T (p_K - u^{\mathcal{N}})}{\|u^{\mathcal{N}} - p_K\|^2}$
      $u = \alpha u + (1 - \alpha) e_K$
      $u^{\mathcal{N}} = \alpha u^{\mathcal{N}} + (1 - \alpha) p_K$
    end if
  end if
  Compute $d_{\min}$ by using (10)
end while
if CASE $= 0$ **then**
  Compute d_i for all i by using (10)
  $\tilde{d} = \{d_i : d_i \leq \theta\}$ ▷ Upper bound(s).
  $\tilde{J} = \{i : d_i \leq \theta\}$ ▷ Corresponding coordinate index (indices).
end if

or more columns of P_A, with step size α. The updating maintains at every iteration the conditions that u is from the unit simplex and the corresponding $u^{\mathcal{N}}$ is a convex combination of columns of P_A, i.e., $u \in \Delta_n$ and $u^{\mathcal{N}} \in \text{conv}(P_A)$. Since this update step is analogous to the von Neumann update step in the von Neumann algorithm, we name this procedure as the vNP. As you will learn later in Section 4 that the vNP is the core subroutine of this proposed rescaling algorithm, therefore, we classify it as a variant of the von Neumann algorithm. Line 12-15 in Algorithm 1 is the von Neumann update steps. It updates vector u until one of the following three cases occurs:

CASE = 1: Find $u^{\mathcal{N}} = P_A u$ as a solution of problem (6);

CASE = 2: If $u^{\mathcal{N}} = 0$, then problem (6) is infeasible, and u is a certificate of infeasibility;

CASE = 0: Find an index set $\tilde{J}$ and the corresponding bounds $\tilde{d}$ such that $x_{\tilde{J}} < \tilde{d}$ for every feasible solution x for problem (6), and $\tilde{d} \leq \frac{1}{2}$. In other words, find at least one coordinate j of x such that $x_j \leq \frac{1}{2}$.

As we will show in Section 4, $\tilde{d}$ is going to be used as the rescaling factor. In the case of rescaling, the vNP terminates when $d_{\min} \leq \frac{1}{2}$, i.e. the minimum value in $\tilde{d}$ is less than $\frac{1}{2}$. We also require that the maximum value of $\tilde{d}$ should not exceed a threshold $\theta \in (\frac{1}{2}, 1)$. Therefore, the vNP only records those $d_i \leq \theta$ into $\tilde{d}$ and their corresponding indices into $\tilde{J}$.

3.3 Complexity of vNP

Roos has proved that the vNP in Algorithm 1 has strong polynomial-time complexity.

Theorem 3.1. *[18] After at most $4n^2$ iterations, the vNP either (a) provides a solution to problem (6), or (b) provides an evidence of infeasibility, or (c) identifies at least one coordinate of x which is smaller than or equal to $\frac{1}{2}$ in every feasible solution of (6).*

Each vNP iteration needs $O(n)$ arithmetic operations. Therefore, the vNP has $O(n^3)$ time complexity. Note that this is a strongly polynomial-time complexity.

4 The column-wise rescaling von Neumann algorithm

In Section 3, we introduced the vNP to calculate an upper bound d for every feasible solution x for problem (6). In this section, we start with the idea of utilizing this upper bound as a rescaling vector. Then the column-wise rescaling von Neumann algorithm is discussed in details.

4.1 Rescaling

Since $x \leq d \leq e$ holds for every feasible solution x of problem (6), then $x'_i = \frac{x_i}{d_i} \leq 1$. This means that x' is a feasible solution to the following problem:

$$ADx = 0, \quad x \in (0, 1]^n, \tag{13}$$

where $D = \text{diag}(d)$, i.e., D is the diagonal matrix whose i-th diagonal entry is d_i. Observe that problems (6) and (13) are the same, if we replace A by AD. Since D is a diagonal matrix, AD is a rescaled version of A, where the i-th column of A is scaled by the factor d_i. This rescaling preserves the problem's form because e remains the upper bound for the variables.

When the vNP stops with an upper bound d, then the columns of A are rescaled by their corresponding d_i bound, respectively. The condition $d_{\min} \leq \frac{1}{2}$ ensures that at least one column is divided by at least a factor of $\frac{1}{2}$. This fact is used when proving the complexity result. Note that in Algorithm 1, the vNP only records the bounds which are less than a threshold θ, e.g., 0.8. After rescaling the vNP is called again to solve the rescaled problem, which has the same form but a different coefficient matrix. By repeating this vNP-rescaling procedure, a sequence of vectors d is constructed. The coordinate wise multiplication of these d vectors is denoted by $\hat{d}$ in Algorithm 3, as the final upper bound for every feasible solutions of problem (6).

It is well known that if problem (6) has rational data, there exists a positive number τ such that it is a lower bound for the positive coordinates in all basic solutions. Due to [19], we may assume $\tau^{-1} = O(2^L)$, where L denotes the binary input size of matrix A [13]. After calling the vNP-rescaling procedure at most $O(nL)$ times, the upper bound for at least one coordinate of x will become smaller than τ, which is not possible if the problem has positive solution. Therefore,

we can conclude that then problem (6) is infeasible.

4.2 Removing columns

Compare the von Neuman problem (2) and problem (6). Every solution of (6) is restricted to be strictly positive. However, solutions of (2) may have zero coordinates. This difference leads to different conclusions in the case of $x_i < \tau$ for some i. As we stated in the previous section, when solving problem (6), we can conclude that if $x_i < \tau$, then problem (6) is infeasible. When solving problem (2), in such a case x_i has to be zero if the problem is feasible. We call such i a "must-be-zero" coordinate. Once a "must-be-zero" coordinate is identified, x_i is fixed to 0, and the corresponding column is removed from A without changing the feasibility of the problem.

Recall that in order to guarantee that P_A is well defined, we assume that matrix A has full row rank. Removing columns from A may destroy this assumption. Therefore, a preprocessing step is needed before running the vNP-rescaling procedure again on the new problem. The preprocessing procedure eliminates any redundant rows to bring A back to a full rank matrix and reduces problem (6) to a similar problem with A replaced by a reduced matrix of A. The preprocessing procedure is stated as Algorithm 2. There are three possible outcomes of the preprocessing procedure:

Algorithm 2 [A,CASE]=PreProcessing(A, J)

if $J \neq \emptyset$ **then**
  $A = A \setminus A_J$ ▷ Remove column(s) J from matrix A.
  if rank(A)=0 **then**
    Return CASE = 3 ▷ $x_{I \setminus J_0}$ (Line 35 in Algorithm 3) can be any positive numbers.
  end if
  if A is not full rank **then**
    Remove redundant row(s) of A to make it of full row rank
  end if
end if
Return A, CASE = 0

CASE = 0: A is full rank and not a square matrix;

CASE = 2: A is full rank and square. In this case problem (2) is infeasible;

CASE = 3: rank(A) = 0. In this case problem (2) is feasible.

If the preprocessing procedure returns CASE = 3, then the non-zero coordinates of a feasible solution x can be any positive numbers. If CASE = 0, no action is needed.

4.3 The column-wise rescaling von Neumann algorithm

The column-wise rescaling von Neumann algorithm is stated as Algorithm 3. For convenience, the while loop in lines 9-32 is called inner loop, the while loop in lines 2-37 is called outer loop. The inner loop is the vNP-rescaling procedure for the actual matrix A. Once it identifies "must-be-zero" coordinates, the algorithm removes the corresponding columns from A, calls the preprocessing procedure, updates matrix A and P_A, and starts the vNP-rescaling procedure again.

5 Complexity

The following complexity result for the column-wise rescaling von Neumann algorithm shows that this is a polynomial time variant of the von Neumann algorithm.

Theorem 5.1. *After at most $O(n^5 \log_2 \tau^{-1}) = O(n^5 L)$ arithmetic operations, the column-wise rescaling von Neumann algorithm, as stated in Algorithm 3, either finds a solution to the von Neumann problem (2) or provides an evidence of its infeasibility.*

Proof. The number of inner-loop iterations is $O(n \log_2 \tau^{-1})$ for a given A. For each inner-loop iteration, the complexity of the vNP is $O(n^3)$ arithmetic operations. For each time calling the vNP, $O(n^3)$ arithmetic operations are needed for computing P_A. Therefore, the complexity of executing the inner loop is $O(n^4 \log_2 \tau^{-1})$. The complexity of the preprocessing procedure is $O(n^3)$. The total number of executions of the outer loop is $O(n)$. Therefore, the total complexity of Algorithm 3 is $O(n^5 \log_2 \tau^{-1}) = O(n^5 L)$. □

6 Computational results

The performance of the column-wise rescaling von Neumann algorithm is compared with those of SeDuMi and Linprog. The bound used in the implementation is bound (10). For each size of A,

Algorithm 3 The Column-wise Rescaling von Neumann Algorithm

Initialize: CASE = 0, $J_0 = J = \emptyset$
while CASE = 0 **do**
  $[A, \text{CASE}]$=PreProcessing(A, J) ▷ Check if A is full rank
  if A is square **then**
    CASE=2 ▷ System (2) is infeasible
    Break
  end if
  Set $\hat{d} = e$, $y = \frac{e}{n}$, $x = 0$ with corresponding dimension
  while CASE = 0 **do**
    $P_A = I - A^T(AA^T)^{-1}A$
    $[\bar{y}, y, z, \tilde{J}, \tilde{d}, \text{CASE}]$=von Neumann Procedure($P_A$, y)
    if CASE=0 **then**
      $\tilde{D} = \text{diag}(\tilde{d})$
      $\hat{d}_{\tilde{J}} = \tilde{D}\hat{d}_{\tilde{J}}$ ▷ d records the rescaling factors
      $A_{\tilde{J}} = \tilde{D}A_{\tilde{J}}$ ▷ Rescale matrix A
      if $\max(\hat{d}) < 1$ **then**
        CASE = 2 ▷ System (2) is infeasible
        Break
      end if
      if exists some coordinates J such that $\hat{d}_J < \tau$ **then**
        $x_J = 0$
        $J_0 = J_0 \cup J$
        Break
      else
        if $\bar{y} \neq 0$ **then**
          $y = \bar{y}$
        end if
        $y_J = y_J/2$
        $y = y/e^T y$
      end if
    end if
  end while
  if CASE = 1 **then** ▷ x is a solution of (1.1).
    $D = \text{diag}(\hat{d})$
    $x_{I\setminus J_0} = Dz$
  end if
end while

Table 1: Comparison of the performance of Algorithm 3, SeDuMi, and Linprog.

Size($m \times n$)	MA		SeDuMi		Linprog	
	Sec.	$\|Ax\|$	Sec.	$\|Ax\|$	Sec.	$\|Ax\|$
5×10	0.0025	3.0e-13	0.0316	2.0e-9	0.0033	5.3e-11
25×50	0.0085	3.6e-12	0.1077	6.3e-9	0.0046	6.7e-11
125×250	0.1102	8.5e-11	0.7439	2.7e-8	0.0456	9.1e-10
250×500	0.2386	3.5e-10	3.8000	1.6e-7	0.3476 (48)	6.7e-9
500×1000	1.0553	8.3e-10	27.7594	4.4e-7	1.0407 (19)	9.4e-9
625×1250	2.4499	2.2e-10	61.6622	3.0e-8	–	–
1000×2000	7.6571	8.4e-10	555.3555	1.8e-7	–	–

we randomly generated 100 von Neumann problems with a dense matrix. The elements of A are randomly chosen in the intervals [-100,100].

For those randomly generated problems, the rescaling von Neumann algorithm outperforms SeDuMi. The running time shown has a significant reduction. With less than a tenth of the running time, the rescaling von Neumann algorithm returns solutions with higher accuracy then the ones obtained by SeDuMi. Linprog runs faster than the rescaling von Neumann algorithm when problem size is small. However, when the size is getting larger than 250×500, Linprog has a limited ability to solve all the problems. The numbers in the bracket show how many problems out of 100 is solved successfully by Linprog.

The results in Table 1 are obtained by using Matlab R2014a on a Windows 7 desktop (Intel(R) Xeon(R) CPU, 3.07GHz) with 4Gb RAM. For the computation of the projection matrix P_A, we used the factorize function developed by Davis [9].

References

[1] A. Beck and M. Teboulle. A conditional gradient method with linear rate of convergence for solving convex linear systems. *Mathematical Methods of Operations Research*, 59:235–247, 2004.

[2] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

[3] A. Blum and J. Dunagan. Smoothed analysis of the perceptron algorithm for linear programming. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 905–914, 2002.

[4] A. Blum, A. Frieze, R. Kannan, and S. Vempala. A polynomial-time algorithm for learning noisy linear threshold functions. *Algorithmica*, 22(1), 1998.

[5] S. Chubanov. A polynomial relaxation type algorithm for linear programming. `http://www.optimization-online.org/DB_FILE/2011/02/2915.pdf`, 2012.

[6] S. Chubanov. A polynomial projection algorithm for linear feasibility problems. *Mathematical Programming*, pages 1–27, 2014.

[7] G. B. Dantzig. Converting a converging algorithm into a polynomially bounded algorithm. Technical Report SOL 91-5, Stanford University, 1991.

[8] G. B. Dantzig. An ϵ-precise feasible solution to a linear program with a convexity constraint in $1/\epsilon^2$ iterations independent of problem size. Technical Report SOL 92-5, Stanford University, 1992.

[9] T. Davis. http://www.mathworks.com/matlabcentral/fileexchange/24119-don-t-let-that-inv-go-past-your-eyes–to-solve-that-system–factorize-.

[10] J. Dunagan and S. Vempala. A simple polynomial-time rescaling algorithm for solving linear programs. *Mathematical Programming*, 114:101–114, 2008.

[11] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3:95–110, 1956.

[12] P. E. Gill, W. Murray, and M. H. Wright. *Numerical Linear Algebra and Optimization, Volume 1.* Addison-Wesley Publishing Company, 1991.

[13] L. G. Khachiyan. A polynomial algorithm for linear programming. *Soviet Mathematics Doklady*, 20:191–194, 1979.

[14] E. Klafszky and T. Terlaky. On the ellipsoid method. *Radovi Mathematicki*, 8:269–280, 1992.

[15] D. Li and T. Terlaky. The duality between the perceptron algorithm and the von neumann algorithm. In L. F. Zuluaga and T. Terlaky, editors, *Modeling and Optimization: Theory and Applications*, volume 62 of *Springer Proceedings in Mathematics and Statistics*, pages 113–136. Springer New York, 2013.

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

[17] J. Peña and N. Sohèili. A deterministic rescaled perceptron algorithm. 2015. Accepted by Mathematical Programming. Appears online at `http://dx.doi.org/10.1007/s10107-015-0860-y`.

[18] C. Roos. An improved version of chubanov's method for solving a homogeneous feasibility problem. `http://www.optimization-online.org/DB_HTML/2015/01/4750.html`.

[19] C. Roos, T. Terlaky, and J.-P. Vial. *Interior Point Methods for Linear Optimization*. Springer, 2006.

[20] F. Rosenblatt. The perceptron–a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, 1957.

[21] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1998.

[22] J. Shawe-Taylor and N. Cristianini. *Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, 2000.