A family of linear programming algorithms based on an algorithm by von Neumann

Goncalves, JPM, Storer, RH & Gondzio, J 2009, 'A family of linear programming algorithms based on an algorithm by von Neumann' Optimization methods & software, vol 24, no. 3, pp. 461-478.

A Family of Linear Programming Algorithms Based on an Algorithm by von Neumann

João P. M. Gonçalves

Mathematical Sciences Department, IBM T. J. Watson Research Center,

Yorktown Heights, NY 10598, USA,

jpgoncal@us.ibm.com

Robert H. Storer

Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA 18015, USA,

rhs2@lehigh.edu

Jacek Gondzio

School of Mathematics, The University of Edinburgh, Edinburgh, EH9 3JZ, UK,

J.Gondzio@ed.ac.uk

Abstract

In this paper, we present a family of algorithms for linear programming based on an algorithm proposed by von Neumann. The von Neumann algorithm is very attractive due to its simplicity but is not practical for solving most linear programs to optimality due to its slow convergence. Our algorithms were developed with the objective of improving the practical convergence of the von Neumann algorithm while maintaining its attractive features. We present results from computational experiments on a set of linear programming problems that show significant improvements over the von Neumann algorithm.

Keywords: Linear programming; Elementary algorithms; Von Neumann algorithm

1 Introduction

In 1948, von Neumann proposed to Dantzig, in a private communication, an algorithm for linear programming. The algorithm was first published by Dantzig in the early 1990's [5, 6] and was later studied by Epelman and Freund [9, 10] and Beck and Teboulle [2]. Although Dantzig introduces it in [5, 6] as an algorithm for finding a feasible solution to a linear program with a convexity constraint, the von Neumann algorithm can be more generally viewed as an algorithm for solving systems of linear inequalities. Epelman and Freund [9, 10] refer

to this algorithm as "elementary", in the sense that it performs only simple computations at each iteration and consequently it is very unsophisticated, especially when compared to modern interior point algorithms. Attractive properties of the von Neumann algorithm are its low computational cost per iteration, which is dominated by a matrix-vector multiplication, and the possibility of exploiting the sparsity of the original problem data. As pointed out by Epelman and Freund [9, 10], these properties are shared by other elementary algorithms for finding a point in a convex set, such as the relaxation method for systems of linear inequalities [1, 19, 8, 15] and the perceptron algorithm [20, 21]. A description and analysis of the von Neumann algorithm can also be found in [3].

As shown in this paper, the von Neumann algorithm is impractical for solving linear programs to a high degree of optimality due to its slow overall convergence. However, it usually has a fast initial convergence rate that, combined with the other nice properties mentioned above, can make it attractive in some contexts. For example, it could possibly be used to provide a starting solution to another linear programming algorithm such as an interior point method. As given by Epelman and Freund [9, 10], a generalization of the von Neumann algorithm could also, for example, be used for solving conic linear systems. Their study is theoretical and the practical viability of their algorithm still remains to be seen.

In this paper, we propose three new algorithms designed to overcome some of the convergence difficulties of the original von Neumann method. Through computational experiments on a set of linear programming problems, we show that our algorithms provide very significant improvements.

The outline of the paper is as follows. In section 2, we describe the von Neumann algorithm and discuss its computational complexity. We also present a review of the literature focusing on ideas for improving the algorithm. In section 3, we present our new algorithms for linear programming based on the von Neumann algorithm. Section 4 includes some implementation details and in section 5 we describe our computational experiments and present the results. Finally, we discuss the main contributions of this paper in section 6. In the appendix we give more details related to the computational experiments.

2 The von Neumann Algorithm

We consider the problem of finding a feasible solution to the following set of linear constraints:

$$\begin{aligned} \mathbf{P}\mathbf{x} &= \mathbf{0}, \\ \mathbf{e}^T\mathbf{x} &= 1, \\ \mathbf{x} &\geq \mathbf{0}, \end{aligned} \tag{1}$$

where $\mathbf{P} \in \mathbb{R}^{m\times n}$, $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{e} \in \mathbb{R}^n$ is the vector of all ones, and the columns of $\mathbf{P}$ have norm one, i.e., $\|\mathbf{P}_j\| = 1, j = 1, \ldots, n$. Geometrically, the columns $\mathbf{P}_j$ can be viewed as points lying on the m-dimensional hypersphere with unit radius and center at the origin (see figure 1). The above problem can then be described as that of assigning nonnegative weights x_j to the points $\mathbf{P}_j$ so that their weighted center of gravity is the origin $\mathbf{0}$. Note that any linear programming problem can be reduced to problem (1). For the details of the necessary transformations, the reader is referred to [16].

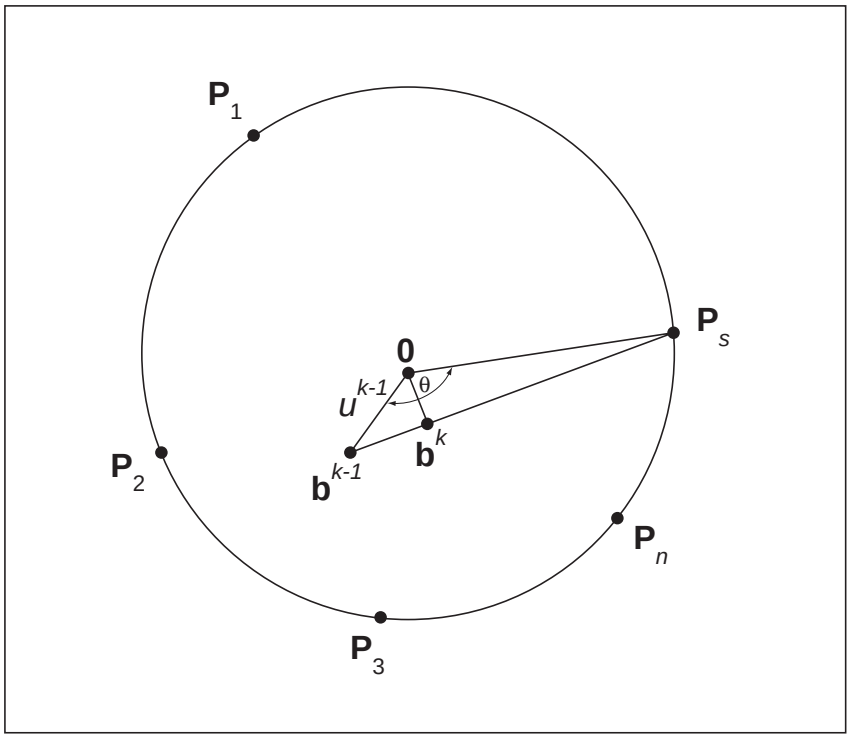

Figure 1: Illustration of the von Neumann algorithm.

The von Neumann algorithm can be stated as follows:

1. (Initialization) The algorithm can be initialized with any approximation to the origin, i.e., $\mathbf{b}^0 = \mathbf{P}\mathbf{x}^0, \mathbf{e}^T\mathbf{x}^0 = 1, \mathbf{x}^0 \geq \mathbf{0}$, where $\mathbf{x}^0$ is arbitrary (e.g., $x_j = 1/n, j = 1, \ldots, n$).

2. (Computation of direction) At the start of iteration $k, k \geq 1$, we have an approximate solution $\mathbf{x} = \mathbf{x}^{k-1}$, such that $\mathbf{x} \geq \mathbf{0}$ and $\mathbf{e}^T\mathbf{x} = 1$. Let

$$\mathbf{b}^{k-1} = \mathbf{P}\mathbf{x}^{k-1}, \qquad u_{k-1} = \|\mathbf{b}^{k-1}\|.$$

   Among all vectors $\mathbf{P}_j, j = 1, \ldots, n$, find a vector $\mathbf{P}_s$ which makes the largest angle (θ in figure 1) with the vector $\mathbf{b}^{k-1}$:

$$s = \text{argmin}_{j=1,\ldots,n}\, \mathbf{P}_j^T\mathbf{b}^{k-1}.$$

3. (Check for infeasibility) Let $v_{k-1} = \mathbf{P}_s^T\mathbf{b}^{k-1}$. If $v_{k-1} > 0$, stop; the problem (1) is infeasible.

4. (Computation of new approximation) The next approximation $\mathbf{b}^k$ is chosen as the closest point to the origin on the line segment joining $\mathbf{b}^{k-1}$ and $\mathbf{P}_s$ (see figure 1). This is done by letting

$$\begin{aligned}
\lambda &= \frac{1 - v_{k-1}}{u_{k-1}^2 - 2v_{k-1} + 1}, \\
\mathbf{b}^k &= \lambda\mathbf{b}^{k-1} + (1-\lambda)\mathbf{P}_s, \\
u_k^2 &= \lambda v_{k-1} + (1-\lambda), \\
\mathbf{x}^k &= \lambda\mathbf{x}^{k-1} + (1-\lambda)\mathbf{e}_s.
\end{aligned}$$

   where $\mathbf{e}_s$ is the unit vector corresponding to index s. Let $k := k + 1$ and go to Step 2.

In step 3 of the algorithm, if $v_{k-1} > 0$, then all points $\mathbf{P}_j$ lie on one side of the hyperplane that passes through the origin and is perpendicular to the direction $\mathbf{b}^{k-1}$. This means that no convex combination of the points $\mathbf{P}_j$ can be found having the origin as center of gravity. Thus, in such a case, we can conclude that problem (1) is infeasible.

In step 4 of the algorithm, note that, since $v_{k-1} = \mathbf{P}_s^T\mathbf{b}^{k-1} \leq 0$, we have $0 < 1 - v_{k-1} < u_{k-1}^2 - 2v_{k-1} + 1$, and therefore, $0 < \lambda < 1$. Also note that the new approximation is guaranteed to be closer to the origin than the previous one, i.e., $u_k < u_{k-1}$. This can be easily understood from figure 1, where we see that in the right triangle $\mathbf{0}\mathbf{b}^{k-1}\mathbf{b}^k$ the hypotenuse is $u_{k-1} = \mathbf{0}\mathbf{b}^{k-1}$ and a leg is $u_k = \mathbf{0}\mathbf{b}^k$.

The von Neumann algorithm performs only simple computations at each iteration. The most expensive computation is the matrix-vector multiplication required to select the column $\mathbf{P}_s$ in step 2 of the algorithm which is $O(mn)$. Note that the number of computations required to perform this multiplication can be significantly reduced if $\mathbf{P}$ is sparse.

The rate of convergence of the von Neumann algorithm was studied by Dantzig [5, 6], by Epelman and Freund [9, 10], and by Beck and Teboulle [2]. Before presenting their convergence results, we define an ϵ-solution of (1) as an approximate solution $\mathbf{x}^k$ such that, $\mathbf{x}^k \geq \mathbf{0}$, $\mathbf{e}^T\mathbf{x}^k = 1$, and $u_k = \|\mathbf{b}^k\| = \|\mathbf{P}\mathbf{x}^k\| \leq \epsilon$. We can now state the convergence result by Dantzig.

Theorem 2.1 (Dantzig [6]) *For $\epsilon > 0$, if problem (1) is feasible, the von Neumann algorithm obtains an ϵ-solution of (1) in at most $\lceil 1/\epsilon^2 \rceil$ iterations.*

Note that the complexity bound in theorem 2.1 is independent of the number of rows m and columns n, which is potentially advantageous. Note also that theorem 2.1 only treats the case when problem (1) is feasible.

The analysis by Epelman and Freund [9, 10] covers both the feasible and infeasible cases. It is based on the quantity r that, when problem (1) has a feasible solution, is defined as the radius of the largest ball centered at the origin $\mathbf{0}$ that is entirely contained in the convex hull of the columns of $\mathbf{P}$. If (1) does not have a feasible solution, then r is the distance from the origin $\mathbf{0}$ to the convex hull of the columns of $\mathbf{P}$.

Theorem 2.2 (Epelman and Freund [9]) *Suppose that $r > 0$ and let $\epsilon > 0$. If problem (1) is feasible, then the von Neumann algorithm obtains an ϵ-solution of (1) in at most*

$$\lceil \frac{2}{r^2} \ln \frac{1}{\epsilon} \rceil$$

iterations. If (1) is infeasible, then the von Neumann algorithm proves infeasibility in at most $\lfloor 1/r^2 \rfloor$ iterations.

The result of Beck and Teboulle [2] applies to the case when problem (1) is feasible. It differs from the Epelman and Freund result for the feasible case only in that r is substituted by another quantity R that depends on the distance between a feasible point and the boundary of $S = \{\mathbf{e}^T\mathbf{x} = 1, \mathbf{x} \geq \mathbf{0}\}$. According to the authors, the inequality $r \geq R$ holds for any feasible point.

In practice, the von Neumann algorithm is usually fast during the early iterations but then its convergence rate becomes slow. The practical slow convergence was observed by Dantzig [6], who developed a variant of the von Neumann algorithm that yields an exact solution to (1). Dantzig's algorithm is based on the assumption that the value of r is known. However, in general, we do not know r in advance, which makes the algorithm impractical.

Other algorithms that can be seen as variants of the von Neumann algorithm have been proposed in the literature. They were developed in the context of the Frank-Wolfe algorithm [12], which reduces to the von Neumann algorithm when applied to a particular problem form. We implemented and tested three of those algorithms, namely the away step introduced by Wolfe [22], the parallel tangents (PARTAN) method [11, 18], and the algorithm introduced by Fukushima [13]. We briefly describe the basic idea of each of these algorithms. The reader is referred to [16] for a full description.

The basic idea of the modification proposed by Wolfe is to consider an alternative feasible direction from the current iterate. This direction is called an "away direction" since it is determined by the vector $\mathbf{P}_t$ that makes the smallest (rather than largest) angle with the vector $\mathbf{b}^{k-1}$. If $|\mathbf{P}_t^T \mathbf{b}^{k-1}| > |v_{k-1}|$ and $x_t^{k-1} > 0$, the algorithm performs the away step, which consists of finding the point $\mathbf{b}^k$ that is closest to the origin along the line connecting $\mathbf{P}_t$ and $\mathbf{b}^{k-1}$. Otherwise, the algorithm performs the normal von Neumann iteration.

The PARTAN method aims at correcting the zigzag behavior responsible for the slow convergence of the von Neumann algorithm. This behavior is characterized by the zigzag movement of successive iterates of the algorithm, making small progress towards the solution. The basic idea of the PARTAN method is to define a feasible direction (PARTAN direction) by connecting the current iterate $\mathbf{b}^{k-1}$ and the iterate from two iterations ago $\mathbf{b}^{k-3}$. The algorithm alternates between the original von Neumann direction and the PARTAN direction.

The Fukushima algorithm considers at each iteration an alternative feasible direction formed by the current iterate $\mathbf{b}^{k-1}$ and a convex combination of vectors $\mathbf{P}_s$ that have been selected in previous iterations. The number of vectors $\mathbf{P}_s$ from previous iterations used in the convex combination is chosen by the user and the weights are the same for all vectors. The direction actually used in each iteration is the best of the above direction and the von Neumann direction.

3 New Algorithms

In this section, we describe three new algorithms that are based on the von Neumann algorithm and that were developed in an attempt to improve its convergence. These algorithms have been named weight-reduction, optimal pair adjustment, and projection. They all apply to problem (1).

Our main focus is on the optimal pair adjustment algorithm. This is the algorithm that performed better in our computational experiments. Also, it is a generalization of the von Neumann and weight-reduction algorithms. The other algorithms are given with different levels of detail. In particular, the projection algorithm is described very briefly and the reader is referred elsewhere for its details.

3.1 The Weight-Reduction Algorithm

The weight-reduction algorithm is based on the idea that a current approximation $\mathbf{b}^{k-1}$ can be moved closer to the origin $\mathbf{0}$ by increasing the weights x_j assigned to some of the columns $\mathbf{P}_j$ and decreasing the weights x_j assigned to other columns $\mathbf{P}_j$. In particular, we expect the new approximation $\mathbf{b}^k$ to be closer to the origin $\mathbf{0}$ than the previous one, if we increase the weight corresponding to the vector $\mathbf{P}_s$ that has the largest angle with $\mathbf{b}^{k-1}$ and decrease the weight assigned to the vector $\mathbf{P}_t$ that has the smallest angle with $\mathbf{b}^{k-1}$. This corresponds to moving from $\mathbf{b}^{k-1}$ in the direction $\mathbf{P}_s - \mathbf{P}_t$. The new point $\mathbf{b}^k$ is the one that minimizes the distance to the origin $\mathbf{0}$ along that line. Of course, the minimization of the distance to the origin is constrained on the maximum possible decrease of x_t. Since we have $x_j \geq 0,\ \forall j$, we can only decrease x_t until it becomes zero.

We now state the weight-reduction algorithm by specifying the steps that are different from the von Neumann algorithm described in the previous section. In step 2, in addition to finding the vector $\mathbf{P}_s$ which makes the largest angle with the vector $\mathbf{b}^{k-1}$, we also find the vector $\mathbf{P}_t$ which makes the smallest angle with the vector $\mathbf{b}^{k-1}$ and such that $x_t > 0$:

$$t = \text{argmax}_{\substack{j=1,\ldots,n \\ x_j>0}} \mathbf{P}_j^T \mathbf{b}^{k-1}.$$

In step 4, we let $\mathbf{d} = \mathbf{P}_s - \mathbf{P}_t$ and

$$\lambda = \min\{\frac{-\mathbf{d}^T \mathbf{b}^{k-1}}{\|\mathbf{d}\|^2}, x_t\}$$

The next approximation is computed as

$$\begin{aligned}\mathbf{b}^k &= \mathbf{b}^{k-1} + \lambda \mathbf{d},\\ u_k &= \|\mathbf{b}^k\|,\\ \mathbf{x}^k &= \mathbf{x}^{k-1} + \lambda(\mathbf{e}_s - \mathbf{e}_t),\end{aligned}$$

where $\mathbf{e}_s$ and $\mathbf{e}_t$ are unit vectors with one in position $j = s$ and $j = t$, respectively.

Finally, we let $k := k + 1$ and go to Step 2.

An iteration of the weight-reduction algorithm is not guaranteed to improve as much as an iteration of the von Neumann algorithm. However, the weight-reduction algorithm can easily be modified such that a weight-reduction iteration is replaced by a von Neumann iteration when the latter provides a larger improvement.

The work per iteration of the weight-reduction algorithm is dominated by the matrix-vector multiplication required for the selection of the columns $\mathbf{P}_s$ and $\mathbf{P}_t$ which is $O(mn)$. This is the same bound as in the von Neumann algorithm.

3.2 The Optimal Pair Adjustment Algorithm

The optimal pair adjustment algorithm is a generalization of the weight-reduction algorithm designed to give the maximum possible freedom to two of the weights x_j. Similar to the weight-reduction algorithm, we start by identifying the vectors $\mathbf{P}_s$ and $\mathbf{P}_t$ that have the largest and the smallest angle with $\mathbf{b}^{k-1}$, respectively. We then find the values of x_s^k, x_t^k, and λ, where $x_j^k = \lambda x_j^{k-1}$ for all $j \neq s$ and $j \neq t$, that minimize the distance from $\mathbf{b}^k$ to the origin $\mathbf{0}$ while satisfying the convexity and nonnegativity constraints. This optimization problem has an easily computable solution found by examination of the Karush-Kuhn-Tucker (KKT) conditions. The main difference between the weight-reduction algorithm and the optimal pair adjustment algorithm is that in the former only the weights of $\mathbf{P}_s$ and $\mathbf{P}_t$ are changed while in the latter all other weights are also changed.

The optimal pair adjustment algorithm differs from the von Neumann algorithm in steps 2 and 4. Step 2 is the same as for the weight-reduction algorithm.

In step 4, which is the computation of the new approximation, we solve the problem

$$\begin{aligned}\text{minimize} \quad & \|\mathbf{b}^k\|^2 = \|\lambda_1(\mathbf{b}^{k-1} - x_s^{k-1}\mathbf{P}_s - x_t^{k-1}\mathbf{P}_t) + \lambda_2\mathbf{P}_s + \lambda_3\mathbf{P}_t\|^2\\ \text{subject to} \quad & \lambda_1(1 - x_s^{k-1} - x_t^{k-1}) + \lambda_2 + \lambda_3 = 1,\\ & \lambda_1 \geq 0,\ \lambda_2 \geq 0,\ \lambda_3 \geq 0.\end{aligned} \tag{2}$$

The next approximation is now computed as

$$\begin{aligned}
\mathbf{b}^k &= \lambda_1(\mathbf{b}^{k-1} - x_s^{k-1}\mathbf{P}_s - x_t^{k-1}\mathbf{P}_t) + \lambda_2\mathbf{P}_s + \lambda_3\mathbf{P}_t, \\
u_k &= \|\mathbf{b}^k\|, \\
x_j^k &= \begin{cases} \lambda_1 x_j^{k-1}, & j \neq s \text{ and } j \neq t, \\ \lambda_2, & j = s, \\ \lambda_3, & j = t. \end{cases}
\end{aligned}$$

We finally let $k := k + 1$ and go to Step 2.

In order to solve problem (2), we first simplify it by eliminating the variable λ_1. We do this by rewriting the equality constraint as

$$\lambda_1 = \frac{1 - \lambda_2 - \lambda_3}{1 - x_s^{k-1} - x_t^{k-1}}$$

and substituting this expression where appropriate. The problem reduces to

$$\begin{aligned}
\text{minimize} \quad & \|\mathbf{b}^k\|^2 = \|\frac{1 - \lambda_2 - \lambda_3}{1 - x_s^{k-1} - x_t^{k-1}}(\mathbf{b}^{k-1} - x_s^{k-1}\mathbf{P}_s - x_t^{k-1}\mathbf{P}_t) + \lambda_2\mathbf{P}_s + \lambda_3\mathbf{P}_t\|^2 \\
\text{subject to} \quad & 1 - \lambda_2 - \lambda_3 \geq 0, \\
& \lambda_2 \geq 0, \ \lambda_3 \geq 0.
\end{aligned} \tag{3}$$

This problem can be easily solved by writing the Karush-Kuhn-Tucker (KKT) necessary and sufficient conditions and finding a feasible solution that satisfies those conditions. The details of this process are given in [16].

The work per iteration of the optimal pair adjustment algorithm is of the same order as the work per iteration of the von Neumann algorithm. Moreover, the improvement in the former is at least as good as the improvement in the latter as it is shown in the next theorem.

Theorem 3.1 *Suppose that $\mathbf{b}^{k-1}$ is the residual at the beginning of iteration $k, k \geq 1$. Also, suppose that $\mathbf{b}_{OPA}^k$ is the residual after an iteration of the optimal pair adjustment algorithm and $\mathbf{b}_{VN}^k$ is the residual after an iteration of the von Neumann algorithm. Then,*

$$\|\mathbf{b}_{OPA}^k\| \leq \|\mathbf{b}_{VN}^k\|.$$

Proof. Let $k, k \geq 1$ be given and let $\mathbf{b}^{k-1}$ be the residual at the beginning of iteration k. Let $\mathbf{P}_s$ and $\mathbf{P}_t$ be the vectors that make the largest and smallest angle with $\mathbf{b}^{k-1}$, respectively. After iteration k of the optimal pair adjustment algorithm we will have

$$\mathbf{b}_{\mathrm{OPA}}^k = \bar{\lambda}_1(\mathbf{b}^{k-1} - x_s^{k-1}\mathbf{P}_s - x_t^{k-1}\mathbf{P}_t) + \bar{\lambda}_2\mathbf{P}_s + \bar{\lambda}_3\mathbf{P}_t,$$

where $(\bar{\lambda}_1, \bar{\lambda}_2, \bar{\lambda}_3)$ is the optimal solution to problem (2). Let $(\lambda_{\text{VN}}, \lambda_{\text{VN}} x_s^{k-1} + 1 - \lambda_{\text{VN}}, \lambda_{\text{VN}} x_t^{k-1})$, where λ_{VN} is the λ of a von Neumann iteration, be a feasible solution to (2). Then, we can write

$$\|\lambda_{\text{VN}} \mathbf{b}^{k-1} + (1 - \lambda_{\text{VN}}) \mathbf{P}_s\| = \|\mathbf{b}^k_{\text{VN}}\| \geq \|\mathbf{b}^k_{\text{OPA}}\|.$$

■

The above theorem allows us to show that the convergence results for the von Neumann algorithm presented in section 2 also apply to the optimal pair adjustment algorithm. As an example, we show next that the convergence result by Epelman and Freund (see theorem 2.2) when problem (1) is feasible is valid for the optimal pair adjustment algorithm.

We start by stating the following proposition derived by Epelman and Freund for the von Neumann algorithm.

Proposition 3.1 (Epelman and Freund [9]) *Suppose that problem* (1) *has a feasible solution, and that* $r > 0$. *At every iteration* $k, k \geq 1$, *of the von Neumann algorithm*

$$\|\mathbf{b}^k\|^2 \leq \|\mathbf{b}^{k-1}\|^2 e^{-r^2}.$$

Given theorem 3.1, proposition 3.1 is also valid if k is an iteration of the optimal pair adjustment algorithm. Applying this inequality inductively, we can bound the size of the residual $\|\mathbf{b}^k\|$ by

$$\|\mathbf{b}^k\| \leq \|\mathbf{b}^0\| e^{-kr^2/2} \leq e^{-kr^2/2}.$$

Recall that for an ϵ-solution, $\|\mathbf{b}^k\| \leq \epsilon$. Given the above bound for the size of the residual $\|\mathbf{b}^k\|$, we are guaranteed to have an ϵ-solution for

$$e^{-kr^2/2} \leq \epsilon.$$

Rearranging the above expression, we obtain

$$k \geq \frac{2}{r^2} \ln \frac{1}{\epsilon}.$$

Thus, if (1) is feasible, the optimal pair adjustment algorithm needs only

$$\lceil \frac{2}{r^2} \ln \frac{1}{\epsilon} \rceil$$

iterations to find an ϵ-solution.

3.3 The Projection Algorithm

The structure of the projection algorithm is similar to the von Neumann algorithm. The main difference is that, at each iteration of the projection algorithm, the new approximation $\mathbf{b}^k$ is computed as a convex combination of the previous approximation $\mathbf{b}^{k-1}$ and of a point $\bar{\mathbf{b}}$ that is itself a convex combination of some of the columns of the matrix $\mathbf{P}$. Recall that in the von Neumann algorithm, the new approximation $\mathbf{b}^k$ is a convex combination of the previous approximation $\mathbf{b}^{k-1}$ and of the vector $\mathbf{P}_s$. The motivation for using a vector $\bar{\mathbf{b}}$ instead of $\mathbf{P}_s$ is to try to make more progress at each iteration. The vector $\bar{\mathbf{b}}$ is constructed by solving an auxiliary problem using the von Neumann algorithm. The auxiliary problem is created as follows:

1. We define a hyperplane through the origin and orthogonal to the vector $\mathbf{b}^{k-1}$.

2. We take the vectors $\mathbf{P}_j$ that lie on the opposite side of the above hyperplane (in relation to $\mathbf{b}^{k-1}$) and project them onto the same hyperplane.

3. We create a linear programming feasibility problem using the projected vectors and the origin.

Any approximate solution to the auxiliary problem can be mapped back to the original problem, i.e., the weights that define the convex combination of the projected points can be used to define a convex combination of the points in the original problem (i.e., before projecting). The point resulting from that convex combination is designated by $\bar{\mathbf{b}}$ and is used to compute the new approximate solution to the original problem. When the approximate solution in the auxiliary problem is close enough to the origin $\mathbf{0}$, we expect the corresponding point in the original problem $\bar{\mathbf{b}}$ to be better than $\mathbf{P}_s$, in the sense that it will produce a smaller $\|\mathbf{b}^k\|$.

The details of the algorithm are given in [16]. The work per iteration depends on the work done solving the auxiliary problem. In practice, it is of the same order as the von Neumann algorithm. The convergence bounds of the von Neumann algorithm presented in section 2 are also valid for the projection algorithm.

4 Implementation

The von Neumann algorithm, the three algorithms presented in section 2 (away step, PARTAN, and Fukushima) resulting from the modifications proposed in the literature to the Frank-Wolfe algorithm, and all the new algorithms described in the previous section have been implemented in ANSI-standard Fortran 77. The codes use routines from the linear programming solver HOPDM developed by Gondzio [17]. In particular, they use the routines to read the problem data in MPS format, to perform presolve analysis, and to scale the problem. For efficiency, the upper bound constraints in the primal problem are treated separately from the other constraints.

4.1 Acceleration Strategies

For all but one of the algorithms implemented, the selection of the column(s) $\mathbf{P}_j$ to use in each iteration is the most time-consuming computation. An obvious way to reduce the computation associated with the selection of the column(s) is to consider only a subset of the columns at each iteration. We have implemented two strategies based on ideas used in practical implementations of the simplex method known as partial and multiple pricing. Since the number of columns that we need to select at each iteration is not the same for all algorithms, the actual implementation of these strategies depends on the algorithm. However, the main concept of these strategies is the same throughout and therefore we focus only on the implementation of these strategies for the von Neumann algorithm.

4.1.1 Partial Pricing

The idea of partial pricing is to divide the matrix $\mathbf{P}$ into blocks of columns and consider only the columns from one of those blocks at each iteration. More specifically, in step 2 of the algorithm (see section 2), the column $\mathbf{P}_s$ is chosen from among a subset of the columns of $\mathbf{P}$, rather than among all its columns. In our implementation, we divide the matrix $\mathbf{P}$ in ten blocks. Each block contains a subset of the columns associated with each set of variables. For example, the set of variables x_j is divided into ten subsets and the columns associated with each subset are assigned to a different block.

At each iteration, if there is not a column from the current block for which $\mathbf{P}_j^T \mathbf{b}^{k-1} \leq 0$, then we move on to consider the columns of the following block. At every new iteration we start by considering the block following the last block used in the previous iteration. In the

first iteration, we consider all columns from matrix $\mathbf{P}$. We do that because we have observed that the improvement of $\|\mathbf{b}^k\|$ in the first iteration of the von Neumann algorithm when considering all columns is often very good.

Note that we divide the matrix $\mathbf{P}$ into a fixed number of blocks for all problems. This is a simple way of dividing the matrix but it goes without saying that one could use other ways which would possibly lead to better results.

4.1.2 Multiple Pricing

The multiple pricing strategy uses the same division of the matrix $\mathbf{P}$ in blocks of columns as partial pricing. In addition, a list of candidate columns is kept from one iteration to another. At each iteration, we consider first the columns in the candidate list. If for all the columns in the list we have $\mathbf{P}_j^T\mathbf{b}^{k-1} > 0$, then we switch to the partial pricing strategy and look for a suitable column in one of the blocks of columns. The strategy for choosing the blocks is exactly as described in the previous section. After we find a suitable column, we replace the columns in the candidate list by columns j from the last block examined for which $\mathbf{P}_j^T\mathbf{b}^{k-1}$ is smallest. In our implementation, the candidate list contains ten columns.

For the algorithms that, at each iteration, require the columns that make the largest and smallest angles with $\mathbf{b}^{k-1}$, we fill the candidate list with the five columns for which $\mathbf{P}_j^T\mathbf{b}^{k-1}$ is smallest and the five columns for which $\mathbf{P}_j^T\mathbf{b}^{k-1}$ is largest.

5 Computational Experiments

In our computational experiments, we used a collection of 145 linear programming instances. The set is divided into 91 Netlib instances [14], 15 Kennington instances [4], and 39 other instances which are not available publicly but can be made available upon request. Note that four Netlib instances (scsd1, scsd6, wood1p, woodw) and one Kennington instance (pds-20) were removed from this study because at least one of the algorithms stopped prematurely on those problems. That happened because some of our codes do not avoid all possible solutions where the variable corresponding to the last column of matrix $\mathbf{P}$ becomes zero. If that happens, we have a solution for problem (1) but not for the original primal and dual problems (see the problem transformations in [16]). Since we only observed these difficulties for a few instances, we did not correct our codes in order to avoid them. However, the changes needed are fairly straightforward and should not affect the performance of the algorithms.

The names of the instances in the three subsets are given in a table in the appendix where the subsets appear ordered as above. In that same table, we also give the sizes of all the problems after presolve, as well as $\|\mathbf{b}^0\|$, i.e., the norm of the vector of residuals for the starting solution. The starting solution is the same for all algorithms and corresponds to setting all variables equal to $1/N$, where N is the total number of variables in the problem.

The main objective of our computational experiments was to compare the performance of the new algorithms proposed in this paper with the performance of the von Neumann algorithm and of those resulting from the modifications to the Frank-Wolfe algorithm proposed in the literature. We recall that the three modifications to the Frank-Wolfe algorithm that we have applied to the von Neumann algorithm are the away step introduced by Wolfe [22], the parallel tangents (PARTAN) method [11, 18], and the idea introduced by Fukushima [13]. In terms of the algorithms that we propose in this paper, we tested the weight-reduction algorithm as described in section 3.1 and also a version where at each iteration we select the best step between the weight-reduction step and the von Neumann step. We tested the other algorithms (optimal pair adjustment algorithm and projection algorithm) as described in sections 3.2 and 3.3. In the case of the projection algorithm, we stop the auxiliary problem when the relative improvement in two consecutive iterations is less than a certain percentage (rd) specified by the user. We chose to use $rd = 50\%, 5\%$, and 0.5%. In addition to testing the original algorithms, we also tested the versions that use partial pricing and multiple pricing.

In our experiments, we first ran the von Neumann algorithm on all test problems and, for each problem, recorded the time t_1 (CPU seconds) and the norm of the vector of residuals $\|\mathbf{b}^k\|$ when the relative difference between $\|\mathbf{b}^{k-1}\|$ and $\|\mathbf{b}^k\|$ was less than 0.5%. We also recorded $\|\mathbf{b}^k\|$ at four other times t_2, t_3, t_4 and t_5 (CPU seconds). Times t_2, t_3, t_4 and t_5 correspond to 3, 5, 10 and 20 times the number of iterations at t_1. We then ran all other algorithms and, for each problem, recorded $\|\mathbf{b}^k\|$ at times $t_i, i = 1, \ldots, 5$. In table 1, we give the percentage of problems that were winning, i.e., that had smaller $\|\mathbf{b}^k\|$, at times t_1 through t_5. For each algorithm, we give the results for the original version, as well as for the versions with partial and multiple pricing. Note that the measure of time used is CPU seconds. The algorithm that wins for a larger percentage of the problems at times t_1 through t_5 is the optimal pair adjustment algorithm with multiple pricing. It is followed by the optimal pair adjustment with partial pricing. In third place is the original optimal pair adjustment algorithm except for time t_5 where the weight-reduction algorithm with multiple

pricing has a larger number of winnings.

Table 1: Percentage of winning problems for each algorithm at five different times.

Algorithm	t_1	t_2	t_3	t_4	t_5
Von Neumann (VN)	0.7%	0.0%	0.0%	0.0%	0.0%
w/ pp	0.0%	0.0%	0.0%	0.0%	0.0%
w/ mp	0.0%	0.0%	0.0%	0.0%	0.0%
VN w/ away step	0.0%	0.0%	0.0%	0.0%	0.0%
w/ pp	0.0%	0.0%	0.0%	0.0%	0.7%
w/ mp	0.0%	0.0%	0.0%	0.0%	0.0%
PARTAN	0.0%	0.0%	0.0%	0.7%	0.7%
w/ pp	0.0%	0.7%	0.7%	0.0%	0.0%
w/ mp	0.0%	1.4%	2.1%	2.1%	2.8%
Fukushima	0.0%	0.0%	0.0%	0.0%	0.0%
w/ pp	0.0%	0.0%	0.0%	0.0%	0.0%
w/ mp	1.4%	0.0%	0.0%	0.0%	0.0%
Weight-reduction	0.0%	0.0%	0.0%	0.0%	0.7%
w/ pp	0.0%	0.7%	0.0%	1.4%	2.8%
w/ mp	2.8%	9.7%	11.0%	14.5%	18.6%
Weight-reduction w/ VN	0.0%	0.7%	0.7%	0.7%	0.7%
w/ pp	0.7%	2.1%	2.1%	1.4%	2.1%
w/ mp	2.1%	1.4%	2.1%	3.4%	2.8%
Projection ($rd = 50\%$)	4.1%	3.4%	4.1%	4.1%	4.8%
w/ pp	2.8%	1.4%	2.8%	0.7%	0.0%
w/ mp	0.0%	0.7%	0.0%	0.0%	0.0%
Projection ($rd = 5\%$)	2.8%	2.8%	3.4%	2.1%	0.7%
w/ pp	1.4%	0.7%	0.0%	0.0%	0.7%
w/ mp	0.0%	0.0%	0.0%	0.0%	0.0%
Projection ($rd = 0.5\%$)	0.0%	1.4%	0.0%	0.7%	0.7%
w/ pp	0.0%	1.4%	0.0%	0.0%	0.0%
w/ mp	0.0%	0.0%	0.0%	0.0%	0.0%
Optimal pair adjustment	14.5%	11.7%	13.1%	15.2%	13.1%
w/ pp	22.8%	21.4%	20.0%	18.6%	20.7%
w/ mp	44.1%	38.6%	37.9%	34.5%	27.6%

We also analyze the performance of the algorithms using performance profiles, which were introduced by Dolan and Moré [7] as a tool for comparing optimization software. Dolan and Moré call the performance profile for a solver "the distribution function of a performance metric". It basically provides a measure of the performance of a solver s as compared to a group of solvers S on a set of problems P. In order to construct a performance profile, we first select $\gamma_{p,s}$, which is a performance measure of solver s on problem p. The performance on

problem p by solver s is compared with the best performance by any solver on this problem using the performance ratio

$$r_{p,s} = \frac{\gamma_{p,s}}{\min_{s \in S} \gamma_{p,s}}.$$

The performance profile for solver s is given by

$$\rho_s(\tau) = \frac{|\{p \in P | r_{p,s} \leq \tau\}|}{|P|},$$

i.e., it is the fraction of instances for which the performance ratio $r_{p,s}$ is within a factor of τ of the best ratio. The comparison of the plots of $\rho_s(\tau)$ for the different solvers gives a way of comparing the relative performance between solvers. The performance profile plots that we present in this paper have τ as the x-axis and $\rho_s(\tau)$ as the y-axis. The solvers that perform better are those for which the plots are "higher".

In our case, we construct performance profiles for the different algorithms at each time $t_i,\ i = 1, \ldots, 5$. Our performance measure of algorithm s on problem p ($\gamma_{p,s}$) is the distance to the origin $\|\mathbf{b}^k\|$ at time t_i. In figure 2, we give the performance profiles for the von Neumann algorithm and our algorithms at time t_1. For the optimal pair adjustment algorithm, we plot the performance profiles for the three versions tested, i.e., the original version and the versions with partial and multiple pricing. For the other algorithms, we just plot one of the versions that is representative of their performance. The versions chosen are: the original von Neumann algorithm, the weight-reduction algorithm with multiple pricing, the multiple pricing version of the algorithm where at each iteration we select the best of the weight-reduction and the von Neumann steps, and the projection algorithm with $rd = 50\%$ and multiple pricing. In this graph, $\rho_s(1)$ is the percentage of problems that were winning at time t_1. It is clear from the graph that the three versions of the optimal pair adjustment algorithm perform much better than any of the other algorithms. It can also be seen that the von Neumann algorithm performs at least as well as any of the other algorithms that we have developed.

When we compare each of our algorithms with the von Neumann algorithm, we conclude that the three versions of the optimal pair adjustment algorithm perform better than any of the other algorithms. For example, at time t_1, there are 91.7%, 97.2%, and 95.2% of winnings for the original version of the optimal pair adjustment, the version with partial pricing, and the version with multiple pricing, respectively. At the same time, the weight-reduction algorithm with multiple pricing and the projection algorithm ($rd = 50\%$) win

only for 31% and 40.7% of the problems, respectively. Furthermore, in the cases where the von Neumann algorithm performs better than some version of the optimal pair adjustment algorithm, the value of $\|\mathbf{b}^k\|$ obtained with the latter is at most 2.4 times larger than the value of $\|\mathbf{b}^k\|$ obtained with the former (i.e., $\|\mathbf{b}^k\|_{\mathrm{OPA}} \leq 2.4\|\mathbf{b}\|_{\mathrm{VN}}$). In contrast, when the weight-reduction algorithm with multiple pricing and the projection ($rd = 50\%$) lose against the von Neumann algorithm, the values of $\|\mathbf{b}^k\|$ can be within a factor of up to 156 and 124, respectively, of the value of $\|\mathbf{b}^k\|$ obtained with the von Neumann algorithm.

At time t_5, the three versions of the optimal pair adjustment algorithm win for 97.2% of the problems when comparing with the von Neumann algorithm. When the optimal pair adjustment loses, the norm of the residuals vector is at most twice as large as that corresponding to the solution obtained with the von Neumann algorithm.

The percentage of winnings of the weight-reduction algorithm with multiple pricing against the von Neumann algorithm increases with time. At time t_5, it wins for 78.6% of the problems. However, when it looses, the ratio of the norms of the vectors of residuals can still be very large (up to 162).

The performance of the projection algorithm ($rd = 50\%$) also improves with time when compared to the von Neumann algorithm. In this case, not only the number of winnings

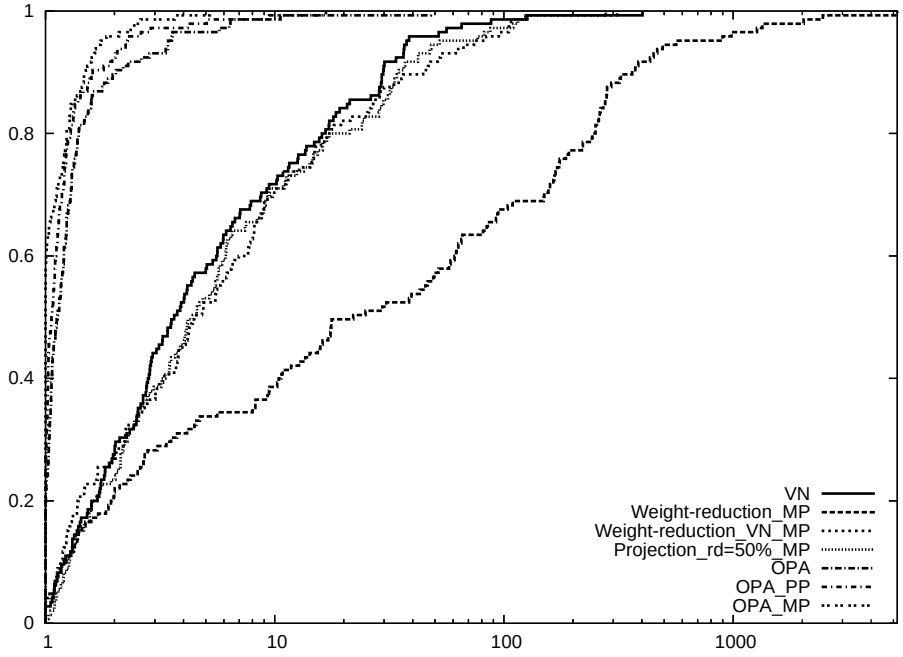

Figure 2: Performance profile of best algorithms tested at time t_1 (x-axis: τ; y-axis: $\rho_s(\tau)$).

increases (up to 95.9% at time t_5) but also the ratio of the norms of the vectors of residuals ($\|\mathbf{b}^k\|_{\text{Projection}}/\|\mathbf{b}\|_{\text{VN}}$) decreases (at time t_5, the maximum ratio is smaller than 1.1).

For detailed results, including the values of $\|\mathbf{b}^k\|$ at different points in time for several algorithms, the reader is referred to the appendix and to [16].

In figure 3, we give the convergence for some of the algorithms tested when applied to the Netlib problem 80bau3b which was selected as representative of performance as a whole. Similar to the behavior of the von Neumann algorithm, our algorithms start with a fast initial convergence (some faster than others) but later the convergence becomes slow. However, the significant improvement of the optimal pair adjustment algorithm when compared to the von Neumann algorithm can clearly be seen in the figure.

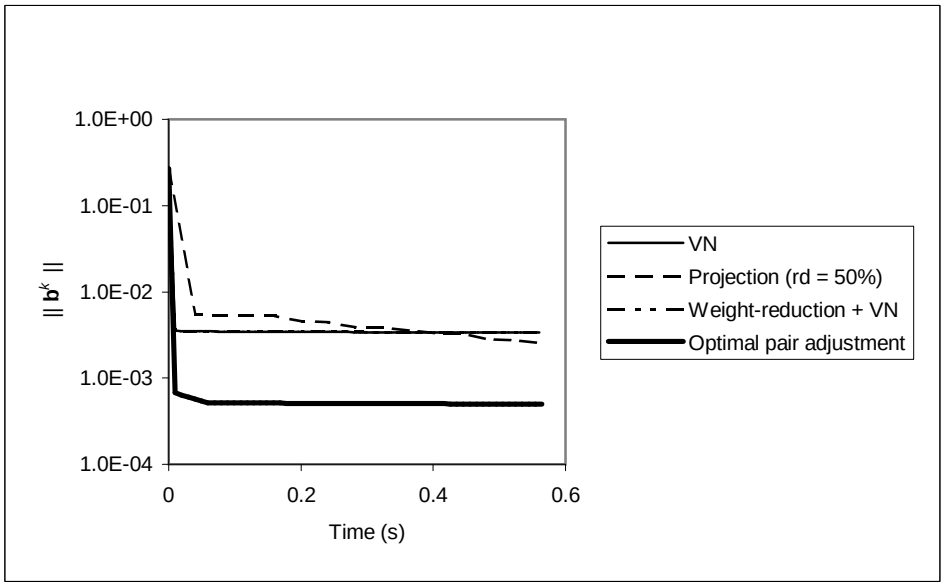

Figure 3: Comparison between the von Neumann algorithm and some of the other algorithms tested when applied to problem 80bau3b.

In figure 4, we present performance profiles for the von Neumann algorithm, the three algorithms described in the literature that were developed in the context of the Frank-Wolfe algorithm, and the optimal pair adjustment algorithm at time t_1. This figure illustrates the consistent improvement of the optimal pair adjustment algorithm over those presented in the literature.

We end this section with an illustration of the typical residuals obtained after transform-

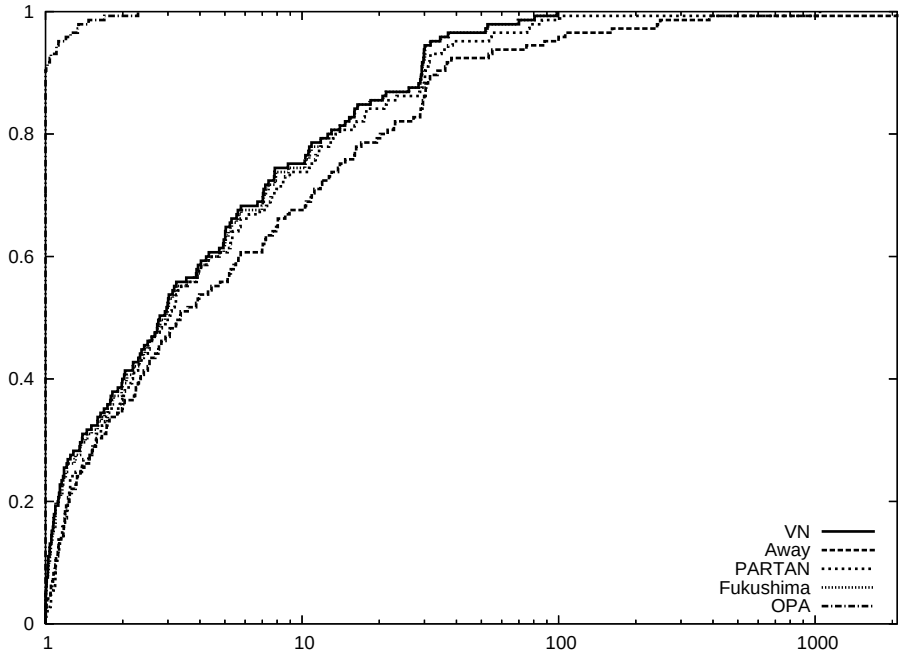

Figure 4: Performance profiles of the von Neumann algorithm, the three algorithms developed to improve the Frank-Wolfe algorithm, and the optimal pair adjustment algorithm, tested at time t_1 (x-axis: τ; y-axis: $\rho_s(\tau)$).

ing the approximate solutions for problem (1) to the original form of the linear programs. In table 2 we show the norms of the residuals of the primal constraints, upper bounds, dual constraints, and duality gap for a sample of problem instances selected to illustrate the range of behavior observed. In the first four lines we show the norms of the residuals for the initial solution. In the following four lines we show the norms of the residuals after running the von Neumann algorithm, and in the last four lines we show the norms of the residuals after running the optimal pair adjustment algorithm with multiple pricing. The results presented in the table give a good idea of the range of accuracies that can be achieved with the algorithms studied. As it can be seen, the accuracies can vary considerably. For example, for the solutions obtained by the optimal pair adjustment, the norm of the primal residual ranges from 10^{-1} in problem kb2 to 10^{6} in problem CO5. Nevertheless, the final accuracies obtained by the optimal pair adjustment algorithm represent, in most cases, an improvement over the accuracies of the initial solutions of at least two orders of magnitude.

Table 2: Norms of residuals in the original problem form at time t_5 for several instances tested.

Algorithm	Norm of residual	25fv47	kb2	ship12l	tuff	CO5
Initial	Primal	5.56E+05	4.03E+03	2.69E+05	1.65E+05	2.16E+08
	Upper bound	2.28E+05	2.41E+03	1.61E+05	3.64E+04	3.11E+07
	Dual	3.65E+05	5.64E+03	1.09E+05	3.03E+04	8.52E+07
	Duality gap	8.91E+06	7.14E+03	1.33E+07	5.21E+05	3.62E+09
VN	Primal	5.44E+03	3.57E+01	2.45E+03	8.79E+02	1.79E+07
	Upper bound	3.30E+03	8.85E+00	1.23E+03	4.72E+02	1.70E+07
	Dual	5.24E+03	1.18E+02	7.24E+02	6.95E+02	2.04E+07
	Duality gap	8.53E+01	1.19E-02	5.72E+01	3.70E+01	2.38E+05
OPA	Primal	5.07E+03	7.90E-01	3.84E+02	2.00E+01	5.18E+06
w/ mp	Upper bound	9.01E+02	2.43E-01	2.64E+02	3.43E+00	4.33E+06
	Dual	3.56E+03	3.80E+00	8.47E+02	3.84E+01	5.93E+06
	Duality gap	2.76E+01	7.09E-04	2.66E+00	7.48E-01	5.28E+03

6 Conclusions

In this paper, we presented three new algorithms for linear programming based on the von Neumann algorithm. These algorithms can be considered elementary since they perform only simple computations.

We presented computational results that showed that our algorithms can provide significant improvements when compared to the von Neumann algorithm. In particular, the optimal pair adjustment algorithm consistently provides solutions significantly closer to optimal than the von Neumann algorithm in the same amount of time.

In spite of the improvements over the von Neumann algorithm, our algorithms are still impractical for solving linear programs to optimality. However, they could be useful in some situations and future research is needed to understand the practical impact that these algorithms can have. Although we have presented results on some quite large and very sparse linear programming instance (e.g., ken-18 and osa-60), more research should be done on even larger instances where the simplicity of these methods may give them an advantage over interior point methods and the simplex method. Also, the fast initial convergence rate of these methods could be used to help enhance the performance of interior point methods. This idea is especially attractive when considering the use of our algorithms in conjunction with an infeasible primal-dual path following algorithm, which is the type of interior point method

most commonly implemented in software. Since our algorithms provide an infeasible solution and since those interior point methods start with an infeasible solution and, in general, reduce the infeasibility at each iteration, we could easily switch between our algorithms and the interior point method. Finally, generalizations of these algorithms, such as the one studied by Epelman and Freund [9, 10] for solving conic linear systems, could be studied.

Acknowledgements

The authors thank the anonymous referees for their suggestions for improving this paper.

A Appendix

In table 3, we give the names of the 145 linear programming instances in our computational experiments. We also give the sizes of all the problems after presolve, as well as $\|\mathbf{b}^0\|$, i.e., the norm of the vector of residuals for the starting solution.

In the last two columns of table 3, we provide the values of the norms of the vectors of residuals ($\|\mathbf{b}^k\|$) at time t_1 obtained with the von Neumann algorithm and with the optimal pair adjustment algorithm. For more detailed results, including the values of $\|\mathbf{b}^k\|$ at other points in time and for other algorithms, the reader is referred to [16].

Table 3: Sizes of problems after presolve, norm of initial residual vectors, and norm of residual vectors at time t_1 for von Neumann and Optimal Pair Adjustment algorithms.

Problem	m	n	nub	nnz	$\|\mathbf{b}^0\|$	$\|\mathbf{b}^k\|_{\text{VN}}$ at t_1	$\|\mathbf{b}^0\|_{\text{OPA}}$ at t_1
25fv47	769	1821	513	10245	1.64E-1	9.40E-3	4.28E-3
80bau3b	1965	10701	5141	21013	2.70E-1	3.49E-3	6.28E-4
adlittle	53	134	60	404	1.86E-1	2.46E-2	4.90E-3
afiro	25	48	6	97	7.69E-2	3.14E-2	2.08E-2
agg	319	404	21	1838	2.19E-2	1.81E-2	1.79E-2
agg2	455	689	16	4351	4.26E-2	1.57E-2	1.33E-2
agg3	455	689	16	4367	4.27E-2	1.57E-2	1.33E-2
bandm	211	366	271	1654	1.31E-1	1.68E-2	6.10E-3
beaconfd	73	148	27	561	2.69E-1	2.15E-2	5.56E-3
blend	66	101	57	416	1.05E-1	2.76E-2	1.20E-2
bnl1	558	1439	1002	4949	2.56E-1	6.97E-3	3.50E-3
bnl2	1848	3800	2303	13251	1.95E-1	4.50E-3	1.16E-3
boeing1	294	660	333	3020	1.37E-1	1.16E-2	1.48E-3

Table 3: continued

Problem	m	n	nub	nnz	$\|\mathbf{b}^0\|$	$\|\mathbf{b}^k\|_{\text{VN}}$ at t_1	$\|\mathbf{b}^0\|_{\text{OPA}}$ at t_1
boeing2	125	264	108	922	6.63E-2	2.08E-2	1.23E-2
bore3d	64	90	60	405	2.11E-1	2.30E-2	2.02E-2
brandy	116	216	154	1557	1.69E-1	2.03E-2	1.06E-2
capri	235	421	239	1448	1.69E-1	1.58E-2	2.24E-3
cycle	1400	2749	1301	14462	6.54E-2	8.59E-3	8.12E-3
czprob	661	2705	2141	5393	1.90E-1	6.63E-3	6.39E-3
d2q06c	2012	5561	1515	30860	1.89E-1	5.77E-3	5.60E-4
d6cube	403	5443	8	32523	4.59E-1	9.32E-3	9.33E-3
degen2	444	757	0	4199	1.56E-1	1.42E-2	1.32E-2
degen3	1503	2604	0	25149	1.11E-1	1.08E-2	1.07E-2
dfl001	5907	12065	5126	35021	1.51E-1	5.66E-3	6.42E-4
e226	161	392	243	2301	1.26E-1	1.70E-2	1.47E-2
etamacro	331	666	411	1972	1.30E-1	1.76E-2	1.65E-3
fffff800	313	817	101	4542	1.08E-1	1.75E-2	1.93E-2
finnis	359	775	174	1809	1.52E-1	1.23E-2	4.11E-3
fit1d	24	1047	1024	13381	2.90E-1	1.32E-2	1.34E-2
fit1p	678	1706	399	9948	2.51E-1	8.11E-3	8.60E-3
fit2d	25	10387	10363	127784	2.80E-1	3.06E-3	3.00E-3
fit2p	3170	13695	7500	50624	3.62E-1	2.36E-3	1.19E-3
forplan	104	411	7	4066	2.66E-1	1.82E-2	1.61E-2
ganges	840	1197	428	5512	7.04E-2	1.46E-2	1.81E-4
gfrd-pnc	590	1134	258	2393	2.76E-1	7.63E-3	2.64E-4
greenbea	1872	4081	581	23334	4.73E-2	1.04E-2	3.56E-3
greenbeb	1865	4065	754	23225	5.77E-2	1.02E-2	3.37E-3
grow15	300	645	600	5620	1.80E-1	1.14E-2	8.35E-3
grow22	440	946	880	8252	1.80E-1	9.54E-3	7.89E-3
grow7	140	301	280	2612	1.81E-1	1.63E-2	8.94E-3
israel	166	307	4	2425	3.81E-2	2.30E-2	2.13E-2
kb2	43	68	9	292	6.21E-2	2.59E-2	3.86E-3
lotfi	117	329	16	643	1.63E-1	2.16E-2	3.05E-3
maros	626	1365	93	6156	6.38E-2	1.30E-2	8.14E-3
maros-r7	2152	6578	0	80167	3.41E-1	3.88E-3	2.76E-4
modszk1	658	1405	0	2863	1.90E-1	9.22E-3	1.18E-3
nesm	646	2850	1560	13100	1.90E-1	1.34E-2	1.72E-3
perold	580	1412	490	6298	9.43E-2	1.27E-2	2.53E-3
pilot	1350	4506	1292	41683	5.95E-2	1.26E-2	1.19E-3
pilot4	389	1069	349	6606	9.39E-2	1.59E-2	1.46E-3
pilot87	1968	6367	1908	72133	6.57E-2	1.06E-2	2.14E-3
pilot_ja	795	1834	713	12032	9.57E-2	1.62E-2	4.37E-4
pilot_we	691	2621	560	8553	5.53E-2	1.84E-2	6.21E-3
pilotnov	830	2089	895	11694	9.62E-2	1.57E-2	2.97E-4
recipe	61	120	56	392	8.48E-2	2.58E-2	1.78E-2
sc105	104	162	0	339	3.23E-2	2.79E-2	2.79E-2
sc205	203	315	13	663	2.56E-2	1.94E-2	7.28E-3
sc50a	49	77	0	159	4.68E-2	3.31E-2	3.23E-2

Table 3: continued

Problem	m	n	nub	nnz	$\|\mathbf{b}^0\|$	$\|\mathbf{b}^k\|_{\text{VN}}$ at t_1	$\|\mathbf{b}^0\|_{\text{OPA}}$ at t_1
sc50b	48	76	0	146	4.72E-2	3.10E-2	2.94E-2
scagr25	344	543	127	1364	1.45E-1	1.16E-2	9.81E-4
scagr7	92	147	37	356	1.53E-1	2.21E-2	3.14E-3
scfxm1	268	526	201	2263	1.41E-1	1.60E-2	2.16E-3
scfxm2	536	1052	402	4531	1.40E-1	1.17E-2	2.32E-3
scfxm3	804	1578	603	6799	1.40E-1	9.60E-3	2.27E-3
scorpion	180	239	28	608	1.53E-1	2.12E-2	1.81E-2
scrs8	418	1183	622	2819	1.87E-1	9.84E-3	3.27E-3
scsd8	397	2750	0	8584	4.25E-1	2.14E-2	1.71E-2
sctap1	269	608	339	1713	2.66E-1	1.70E-2	1.64E-2
sctap2	977	2303	1326	6694	2.75E-1	1.61E-2	1.61E-2
sctap3	1346	3113	1767	8986	2.70E-1	1.65E-2	1.64E-2
seba	2	9	8	12	4.20E-1	4.16E-2	1.64E-2
share1b	107	243	31	1016	7.59E-2	3.01E-2	2.16E-2
share2b	92	158	76	711	1.16E-1	2.75E-2	1.01E-2
shell	487	1450	188	2904	3.33E-1	7.30E-3	7.41E-5
ship04l	292	1905	1672	4290	4.24E-1	8.63E-3	1.75E-3
ship04s	216	1281	1052	2875	4.21E-1	9.77E-3	1.69E-3
ship08l	470	3121	2664	7122	4.28E-1	6.53E-3	1.66E-3
ship08s	276	1604	1155	3644	4.22E-1	7.94E-3	1.67E-3
ship12l	610	4171	3510	9254	3.95E-1	6.75E-3	2.89E-3
ship12s	340	1943	1282	4297	3.89E-1	8.15E-3	2.99E-3
sierra	1129	2618	2008	7566	3.43E-1	4.28E-3	1.64E-4
stair	356	531	42	3811	3.18E-2	1.72E-2	1.34E-2
standata	292	582	358	1167	2.39E-1	1.08E-2	6.19E-3
standgub	292	582	358	1167	2.39E-1	1.08E-2	6.01E-3
standmps	388	1146	984	2491	3.22E-1	7.50E-3	5.52E-3
stocfor1	94	142	80	405	7.77E-2	2.39E-2	1.01E-2
stocfor2	1968	2856	1286	8066	5.58E-2	5.61E-3	2.56E-3
stocfor3	15336	22202	9667	62908	4.62E-2	2.07E-3	4.78E-4
truss	1000	8806	0	27836	4.42E-1	9.42E-3	9.36E-3
tuff	246	553	380	3737	1.60E-1	3.38E-2	3.55E-2
vtp_base	46	82	43	205	1.33E-1	2.79E-2	1.00E-2
cre-a	2994	6692	302	16552	1.89E-1	5.24E-3	3.43E-4
cre-b	5336	36382	506	111637	2.32E-1	8.09E-3	2.55E-4
cre-c	2375	5412	132	13346	1.87E-1	6.23E-3	1.20E-4
cre-d	4102	28601	203	86353	2.57E-1	8.99E-3	2.60E-4
ken-07	1427	2603	2603	5494	5.84E-3	5.51E-3	5.04E-3
ken-11	10061	16709	16709	35578	2.22E-3	2.12E-3	1.94E-3
ken-13	22519	36546	36546	80148	1.58E-3	1.39E-3	1.30E-3
ken-18	78823	128395	128395	286183	9.63E-4	7.51E-4	6.69E-4
osa-07	1047	24911	23864	65138	4.76E-1	8.24E-3	9.89E-3
osa-14	2266	54535	52269	143777	4.76E-1	5.87E-3	8.20E-3
osa-30	4279	103978	99699	276565	4.77E-1	4.37E-3	6.90E-3
osa-60	10209	242411	232202	614537	4.76E-1	3.05E-3	7.00E-3

Table 3: continued

Problem	m	n	nub	nnz	$\|\mathbf{b}^0\|$	$\|\mathbf{b}^k\|_{\text{VN}}$ at t_1	$\|\mathbf{b}^0\|_{\text{OPA}}$ at t_1
pds-02	2603	7333	4440	15682	2.50E-1	7.42E-3	1.32E-3
pds-06	9119	28435	18835	60676	2.77E-1	3.19E-3	1.02E-3
pds-10	15587	48719	33076	104038	2.81E-1	2.26E-3	8.70E-4
BL	5468	12038	2253	32699	1.73E-1	3.42E-3	9.66E-4
BL2	5480	12063	2263	32837	1.73E-1	3.46E-3	1.07E-3
CO5	4471	10318	1029	49028	1.34E-1	5.42E-3	3.90E-3
CO9	8510	19276	1844	92450	1.31E-1	4.22E-3	2.36E-3
CQ9	7073	17806	1893	82802	1.08E-1	5.26E-3	3.30E-3
GE	8361	14096	1359	39167	9.58E-2	3.64E-3	9.02E-4
NL	6478	14393	3213	44437	2.03E-1	2.95E-3	1.18E-3
a1	42	73	72	284	1.16E-1	2.21E-2	1.81E-2
fort45	1037	1467	1402	6077	1.26E-1	6.33E-3	4.86E-4
fort46	1037	1467	1402	6077	1.25E-1	6.31E-3	6.16E-4
fort47	1037	1467	1402	6077	1.37E-1	6.52E-3	4.08E-4
fort48	1037	1467	1402	6077	1.29E-1	6.58E-3	2.31E-4
fort49	1037	1467	1402	6077	1.27E-1	6.43E-3	4.01E-4
fort51	1042	1473	1402	8359	1.77E-1	5.84E-3	1.10E-3
fort52	1041	1471	1402	7957	1.57E-1	6.10E-3	2.04E-4
fort53	1041	1471	1402	7957	1.57E-1	5.94E-3	2.02E-4
fort54	1041	1471	1402	7730	1.34E-1	5.74E-3	2.70E-4
fort55	1041	1471	1402	7730	1.34E-1	5.71E-3	2.77E-4
fort56	1041	1471	1402	8027	1.60E-1	6.09E-3	2.04E-4
fort57	1041	1471	1402	8027	1.60E-1	5.88E-3	2.02E-4
fort58	1041	1471	1402	7957	1.25E-1	6.41E-3	2.13E-4
fort59	1041	1471	1402	7957	1.25E-1	6.36E-3	2.17E-4
fort60	1041	1471	1402	7958	1.34E-1	6.31E-3	2.12E-4
fort61	1041	1471	1402	7958	1.34E-1	6.24E-3	2.13E-4
x1	983	1413	1412	5873	1.40E-1	6.42E-3	4.36E-4
x2	983	1413	1412	5873	1.08E-1	6.50E-3	9.03E-4
pata01	122	1241	0	2443	7.19E-2	2.80E-2	2.46E-2
pata02	122	1241	0	2443	7.19E-2	2.80E-2	3.74E-2
patb01	57	143	0	277	7.00E-2	3.02E-2	1.49E-2
patb02	57	143	0	277	7.00E-2	3.02E-2	1.48E-2
vschna02	122	1363	0	2565	6.73E-2	2.96E-2	2.90E-2
vschnb01	57	144	0	278	7.10E-2	2.97E-2	2.36E-3
vschnb02	58	202	0	338	5.93E-2	3.49E-2	2.13E-2
willett	184	588	0	2403	5.02E-2	4.10E-2	1.69E-2
ex01	234	1555	1325	9091	1.95E-1	1.01E-2	1.44E-4
ex02	226	1547	1324	8899	2.13E-1	1.31E-2	1.76E-2
ex05	831	7747	6923	46038	1.93E-1	1.79E-2	3.42E-3
ex06	824	7778	6961	44370	3.36E-1	8.71E-3	5.29E-4
ex09	1818	18120	16309	104559	2.04E-1	2.22E-2	1.20E-3

References

[1] S. Agmon. The relaxation method for linear inequalities. *Canadian Journal of Mathematics*, 6(3):382–392, 1954.

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

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

[4] W. J. Carolan, J. E. Hill, J. L. Kennington, S. Niemi, and S. J. Wichmann. An empirical evaluation of the KORBX algorithms for military airlift applications. *Operations Research*, 38(2):240–248, 1990.

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

[6] 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.

[7] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002.

[8] B. C. Eaves. Piecewise linear retractions by reflexion. *Linear Algebra and its Applications*, 7:93–99, 1973.

[9] M. Epelman and R. M. Freund. Condition number complexity of an elementary algorithm for resolving a conic linear system. Working Paper OR 319-97, Massachusetts Institute of Technology, 1997.

[10] M. Epelman and R. M. Freund. Condition number complexity of an elementary algorithm for computing a reliable solution of a conic linear system. *Mathematical Programming*, 88:451–485, 2000.

[11] M. Florian, J. Guélat, and H. Spiess. An efficient implementation of the PARTAN variant of the linear approximation method of the network equilibrium problem. *Networks*, 17:319–339, 1987.

[12] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1 and 2):95–110, 1956.

[13] M. Fukushima. A modified Frank-Wolfe algorithm for solving the traffic assignment problem. *Transportation Research*, 18B(2):169–177, 1984.

[14] D. M. Gay. Electronic mail distribution of linear programming test problems. *Mathematical Programming Society COAL Newsletter*, 13:10–12, 1985.

[15] J. L. Goffin. The relaxation method for solving systems of linear inequalities. *Mathematics of Operations Research*, 5(3):388–414, 1980.

[16] J. P. M. Gonçalves. *A Family of Linear Programming Algorithms Based on the von Neumann Algorithm*. PhD thesis, Lehigh University, Bethlehem, PA, 2004.

[17] J. Gondzio. HOPDM (version 2.12) - a fast LP solver based on a primal-dual interior point method. *European Journal of Operational Research*, 85:221–225, 1995.

[18] L. J. LeBlanc, R. V. Helgason, and D. E. Boyce. Improved efficiency of the Frank-Wolfe algorithm for convex network programs. *Transportation Science*, 19:445–462, 1985.

[19] T. S. Motzkin and I. J. Schoenberg. The relaxation method for linear inequalities. *Canadian Journal of Mathematics*, 6(3):393–404, 1954.

[20] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

[21] F. Rosenblatt. *Principles of Neurodynamics*. Spartan Books, Washington, DC, 1962.

[22] P. Wolfe. Convergence theory in nonlinear programming. In J. Abadie, editor, *Integer and Nonlinear Programmin*, pages 1–36. North-Holland, Amsterdam, 1970.