The Impact of Scaling on Simplex Type Algorithms

Nikolaos Ploskas
Department of Applied Informatics,
University of Macedonia,
Economic and Social Sciences
156 Egnatia Str.,
Thessaloniki 54006, Greece
Tel.: (+30) 2310 891824
ploskas@uom.gr

Nikolaos Samaras
Department of Applied Informatics,
University of Macedonia,
Economic and Social Sciences
156 Egnatia Str.,
Thessaloniki 54006, Greece
Tel.: (+30) 2310 891866
samaras@uom.gr

ABSTRACT

One of the most significant and well-studied optimization problems is the Linear Programming problem (LP). Many algorithms have been proposed for the solution of Linear Programming problems (LPs); the main categories of them are: (i) simplex-type or pivoting algorithms, (ii) interior-point methods and (iii) exterior point simplex type algorithms. Prior to the application of these algorithms, some preconditioning techniques are executed in order to improve the computational properties of LPs. Scaling is the most widely used preconditioning technique and is used to reduce the condition number of the constraint matrix, reduce the number of iterations required to solve LPs and improve the numerical behavior of the algorithms.

The aim of this paper is to present a computational study of the impact of scaling prior to the application of the aforementioned algorithms. In the computational study that we have conducted, we calculate both the CPU time and the number of iterations with and without scaling for a set of sparse randomly generated optimal LPs. The scaling techniques that we applied to the above mentioned algorithms are: (i) arithmetic mean, (ii) equilibration, and (iii) geometric mean scaling techniques. Computational results showed that equilibration is the best scaling technique and that the effect of scaling is significant to IPM and revised simplex algorithm, while EPSA is scaling invariant.

Categories and Subject Descriptors

G.1.6 [**Optimization**]: Linear Programming

General Terms

Algorithms, Operations Research, Linear Programming.

Keywords

Operations Research, Linear Programming, Simplex Algorithm, Scaling Techniques, Computational Study.

1. INTRODUCTION

Linear Programming (LP) is a significant area in the field of operations research, where a linear function (1) is optimized.

$$z = \sum_{i=1}^{n} c_i x_i \qquad (1)$$

Several methods are available for solving LP problems, among which the simplex algorithm (Dantzig, 1949) is the most widely used. Although the computational complexity of the simplex method is not polynomial in the number of equations (Klee, 1992), it performs sufficiently well in practice, especially on linear problems of small and medium size.

Consider the following LP in its general form (LP.1):

$$\begin{array}{ll} \min & c^T x \\ s.t. & Ax = b \\ & x \geq 0 \end{array} \qquad (LP.1)$$

where $A \in R^{mxn}$, $(c, x) \in R^n$, $b \in R^m$, and T denotes transposition. We assume that A has full rank. The simplex algorithm searches for an optimal solution by moving from one feasible solution to another, along the edges of the feasible set. The dual problem associated with the (LP.1) is presented in (DP.1):

$$\begin{array}{ll} \min & b^T w \\ s.t. & A^T w + s = c \\ & s \geq 0 \end{array} \qquad (DP.1)$$

where $w \in R^m$ and $s \in R^n$,

Since Dantzig's original contribution, new algorithms have been proposed. Two proposed types of algorithms were interior point methods (IPM) and exterior point simplex type algorithms (EPSA). IPMs move inside the feasible region, while EPSA moves in the exterior of the feasible region and its basic solutions are not feasible (Paparrizos, 1997; Paparrizos, 2001). Both IPM's and EPSA's performance outperforms the original simplex algorithm (Karmarkar, 1984; Paparrizos, 2003a). It has been reported that the most effective type of algorithms, in computational terms, for each category are the primal-dual versions (Gondzio, 1996; Paparrizos, 2003b).

The increasing size of real LPs demands more computational power and numerical accuracy. Many preconditioning techniques have been applied to LPs prior to the application of a simplex-type algorithm in order to improve their computational properties.

One of the most well-known preconditioning techniques is scaling. Scaling is applied to a matrix in order to harmonize the magnitudes of its nonzero elements, so its nonzero elements are multiplied by positive scalars. Scaling is applied prior to the simplex algorithm for three reasons: (i) to produce a compact representation of the variables' bounds, (ii) to reduce the iterations needed by the algorithm to solve the LP, and (iii) to reduce the condition number of the constrained matrix (Tomlin, 1975).

Tomlin (Tomlin, 1975) performed a computational study comparing arithmetic mean, geometric mean, equilibration, Curtis and Reid scaling technique (Curtis and Reid, 1972), Fulkerson and Wolfe scaling technique (Fulkerson and Wolfe, 1962), and various combinations on six test problems. The conclusion of Tomlin's comparative study was that geometric mean scaling method, optionally followed by equilibration or Curtis and Reid scaling technique are the best combined scaling techniques. Larsson (Larsson, 1993) extended Tomlin's study by comparing entropy (Larsson, 1993), $L_p$norm (Hamming, 1971) and de Buchet (de Buchet, 1966) scaling techniques over a dataset of 135 randomly generated LPs. Larsson concluded that the entropy scaling technique can improve the conditioning number of the constraint matrix. Elble and Sahinidis (Elble and Sahinidis, 2012) expanded on Tomlin's and Larsson's studies by conducting a computational study comparing arithmetic mean, de Buchet, entropy, equilibration, geometric mean, IBM MPSX (Benichou, 1977), Lpnorm, binormilization, and various combinations of the aforementioned scaling techniques over Netlib and Kennington set. They used four measures to evaluate each scaling technique: (i) scaling time, (ii) solution time, (iii) number of iterations for the solution of the LP, and (iv) maximum conditioning number of the constraint matrix. Elble and Sahinidis concluded that equilibration is the best scaling technique.

In a previous paper (Ploskas and Samaras, 2013), we reviewed and compared both the CPU- and GPU-based implementations of seven scaling techniques, namely: (i) arithmetic mean, (ii) de Buchet, (iii) entropy, (iv) equilibration, (v) geometric mean, (vi) IBM MPSX, and (vii) Lp-norm scaling methods. We have performed a computational study over Netlib and Kennington set and concluded that arithmetic mean, equilibration and geometric mean are the best serial scaling techniques. In this paper a computational study is performed over a set of sparse randomly generated LPs in order to highlight the impact of scaling prior to the application of IPM, EPSA and simplex algorithms. To the best of our knowledge, this is the first paper that investigates the effect of scaling on IPM, EPSA and simplex algorithms.

The structure of the paper is as follows. Section 2 includes the presentation of the three scaling techniques that will be compared. Section 3 briefly presents the IPM, EPSA and simplex algorithm, while in Section 4 the computational study is presented. Finally, the conclusions of this paper are outlined in Section 5.

2. SCALING METHODS

2.1 Mathematical Preliminaries and Notations

Prior to the presentation of the three scaling techniques, some mathematical preliminaries and notations are being introduced in this subsection. Let A be an m x n matrix. Let r_i be the row scaling factor for row i and s_j be the column scaling factor for column j. Let $N_i = \left\{ j \mid A_{ij} \neq 0 \right\}$, where i = 1, ...,m, and $M_j = \left\{ i \mid A_{ij} \neq 0 \right\}$, where j = 1, ..., n. Let n_i and m_j be the cardinality numbers of the sets N_i and M_j, respectively. The scaled matrix is expressed as X = RAS, where R = diag(r_1...r_m) and S = diag(s_1...s_n). All scaling methods presented in this paper, perform first a scaling of the rows and then a scaling of the columns.
Furthermore, it is important to introduce some notations that are used in the pseudocodes of the following subsections. Let r be an 1xm vector with row scaling factors and s be an 1xn vector with column scaling factors. Let sum_row be an 1xm vector with the sum of each row's elements, and sum_col be an 1xn vector with the sum of each column's elements. Furthermore, let row_max be an 1xm vector with each row's maximum element, row_min be an 1xm vector with each row's minimum element, col_max be an 1xn vector with each column's maximum element, and row_min be an 1xn vector with each column's minimum element. Finally, let count_row be an 1xm vector with the number of each row's nonzero elements, and count_col be an 1xn vector with the sum of each column's nonzero elements.

2.2 Arithmetic Mean

Arithmetic mean scaling method aims to reduce the variance between the nonzero elements of the coefficient matrix A. Each row and column is divided by the arithmetic mean of the elements in that row and column, respectively. The row and column scaling factors are presented in equation 2 and equation 3, respectively:

$$r_i = \left(\frac{n_i}{\sum_{j \in N_i} A_{ij}} \right) \tag{2}$$

$$s_j = \left(\frac{m_j}{\sum_{i \in M_j} A_{ij}} \right) \tag{3}$$

Table 1 shows the pseudocode of the implementation of the arithmetic mean scaling technique. In the first for-loop (lines 1:13), the row scaling factors are computed as the number of nonzero elements of each row to the sum of the same row (line 9). Finally, in the second for-loop (lines 14:26)~~,~~ the column scaling factors are calculated as the number of nonzero elements of each column to the sum of the same column (line 22).

Table 1. Arithmetic Mean

```
1. for i=1:m
2.    for j=1:n
3.       if A[i][j] != 0
4.          sum_row[i] = sum_row[i] + |A[i][j]|
5.          count_row[i] = count_row[i] + 1
6.       end if
7.    end for
8.    if count_row[i] != 0 AND sum_row[i] != 0
9.       r[i] = count_row[i] / sum_row[i]
10.      A[i][:] = A[i][:] * r[i]
11.      b[i] = b[i] * r[i]
12.   end if
13. end for
14. for i=1:n
15.    for j=1:m
16.       if A[i][j] != 0
17.          sum_col[i] = sum_col[i] + |A[i][j]|
18.          count_col[i] = count_col[i] + 1
```

```
19.       end if
20.    end
21.    if count_col[i] != 0 AND sum_col[i] != 0
22.       s[i] = count_col[i] / sum_col[i]
23.       A[:][i] = A[:][i] * s[i]
24.       c[i] = c[i] * s[i]
25.    end if
26. end for
```

2.3 Equilibration

In this scaling technique, all elements of the coefficient matrix A have values between -1 and 1 after the appliance of scaling. Table 2 shows the pseudocode of the implementation of the equilibration scaling technique. In the first for-loop (lines 1:8), the row scaling factors are computed as the inverse of the maximum element of each row (line 4). Finally, in the second for-loop (lines 9:16), the column scaling factors are calculated as the inverse of the maximum element of each column (line 12).

Table 2. Equilibration

```
1.  for i=1:m
2.     find the maximum element in row i and store it to
       row_max[i]
3.     if row_max[i] != 0
4.        r[i] = 1 / row_max[i];
5.        A[i][:] = A[i][:] * r[i]
6.        b[i] = b[i] * r[i]
7.     end if
8.  end for
9.  for i=1:n
10.    find the maximum element in column i and store it to
       col_max[i]
11.    if col_max[i] != 0
12.       s[i] = 1 / col_max[i]
13.       A[:][i] = A[:][i] * s[i]
14.       c[i] = c[i] * s[i]
15.    end if
16. end for
```

2.4 Geometric Mean

Like arithmetic mean scaling method, geometric mean also aims to reduce the variance between the nonzero elements of the coefficient matrix A. Each row and column is divided by the product of the square root of the maximum and minimum element in that row and column, respectively. The row and column scaling factors are presented in equation 4 and equation 5, respectively:

$$r_i = \left(\max_{j \in N_i} A_{ij} \times \min_{j \in N_i} A_{ij}\right)^{-\frac{1}{2}} \qquad (4)$$

$$s_j = \left(\max_{i \in M_j} A_{ij} \times \min_{i \in M_j} A_{ij}\right)^{-\frac{1}{2}} \qquad (5)$$

Table 3 shows the pseudocode of the implementation of the geometric mean scaling technique. In the first for-loop (lines 1:9), the row scaling factors are computed as the product of the square root of the maximum element in a row and the square root of the minimum element in a row (line 5). Finally, in the second for-loop (lines 10:18), the column scaling factors are calculated as the product of the square root of the maximum element in a column and the square root of the minimum element in a column (line 14).

Table 3. Geometric Mean

```
1.  for i=1:m
2.     find the maximum element in row i and store it to
       row_max[i]
3.     find the minimum element in row i and store it to
       row_min[i]
4.     if row_max[i] != 0 AND row_min[i] != 0
5.        r[i] = 1/(sqrt(row_max[i]) * sqrt(row_min[i])))
6.        A[i][:] = A[i][:] * r[i]
7.        b[i] = b[i] * r[i]
8.     end if
9.  end for
10. for i=1:n
11.    find the maximum element in column i and store it to
       col_max[i]
12.    find the minimum element in column i and store it to
       col_min[i]
13.    if col_max[i] != 0 AND col_min[i] != 0
14.       s[i] = 1/(sqrt(col_max[i]) * sqrt(col_min[i])))
15.       A[i][:] = A[i][:] * s[i]
16.       c[i] = c[i] * s[i]
17.    end if
18. end for
```

3. ALGORITHMS

This paper aims to highlight the impact of scaling prior to the application of IPM, EPSA and simplex algorithms. The IPM that was used in the computational study is MATLAB's large-scale linprog built-in function. MATLAB's IPM algorithm is based on Interior Point Solver (Zhang, 1998), a primal-dual interior point algorithm, that takes advantage of MATLAB's sparse matrix functions. The EPSA algorithm that we used has been proposed by Paparrizos (2003b). This algorithm outperforms simplex algorithm as the problem size increases and the density decreases. Finally, we have also included revised simplex algorithm in the computational study proposed by Dantzig (1953). The basis inverse step is performed using a Modification of the Product Form of the Inverse (MPFI), proposed by Benhamadou (2002). MPFI updating scheme is faster than other updating schemes (Ploskas, 2013; Badr, 2005), like Product Form of the Inverse (PFI).

4. COMPUTATIONAL STUDY

Computational studies have been widely used, in order to examine the practical efficiency of an algorithm or even compare algorithms. The computational comparison has been performed on a quad-processor Intel Core i7 3.4 GHz with 32 Gbyte of main memory running under Microsoft Windows 7 64-bit. The algorithms have been implemented using MATLAB Professional R2012b.

The test set used in the computational study was randomly generated. Problem instances have the same number of constraints and variables. The largest problem tested has 3000 constraints and 3000 variables. We have generated these instances with 10% and 20% density. For each problem type of a particular size, 10 instances were generated, using a different seed number. For each instance, we averaged times over 10 runs. All times in the following tables are measured in seconds.

In this computational study we study the impact of scaling prior to the application of IPM, EPSA and simplex algorithm. So, we have executed these algorithms with and without scaling. Three different scaling techniques have been used: (i) arithmetic mean, (ii) equilibration, and (iii) geometric mean scaling techniques.

Table 4, Table 5 and Table 6 present the results from the execution of MATLAB's IPM algorithm, EPSA algorithm and revised simplex algorithm, respectively, over randomly optimal generated problems with density 10%. Table 7, Table 8 and Table 9 present the results from the execution of MATLAB's IPM algorithm, EPSA algorithm and revised simplex algorithm, respectively, over randomly optimal generated problems with density 20%. In Tables 4 – 9 the following abbreviations are used: (i) AM – arithmetic mean, (ii) EQ – equilibration, and (iii) GM – geometric mean. From the results, we observe that: (i) equilibration is the best scaling technique on all algorithms, (ii) the effect of scaling is more significant on IPM and revised simplex algorithm, and (iii) EPSA is scaling invariant.

Table 4. Results of MATLAB's IPM Algorithm over Randomly Optimal Generated Problems with Density 10%

Density 10%	IPM		IPM with AM scaling		IPM with EQ scaling		IPM with GM scaling	
	time (sec)	iterations	time (sec)	iterations	time (sec)	iterations	time (sec)	iterations
1000x1000	8.7	22.0	7.68	20.2	7.69	20.5	7.69	20.0
1500x1500	216.81	26.1	96.72	24.7	99.54	24.8	84.44	22.3
2000x2000	321.88	28.4	162.68	24.0	150.42	23.5	150.51	23.6
2500x2500	1,089.31	29.8	399.74	28.1	395.48	28.1	451.47	29.7
3000x3000	1,512.48	24.3	670.84	25.9	399.08	25.6	574.04	25.8
Average	**629.84**	**26.1**	**267.53**	**24.6**	**210.44**	**24.5**	**253.63**	**24.3**

Table 5. Results of EPSA Algorithm over Randomly Optimal Generated Problems with Density 10%

Density 10%	EPSA		EPSA with AM scaling		EPSA with EQ scaling		EPSA with GM scaling	
	time (sec)	iterations	time (sec)	iterations	time (sec)	iterations	time (sec)	iterations
1000x1000	6.27	947.5	5.69	944.3	5.85	965.3	5.99	970.3
1500x1500	14.4	1,353.5	14.35	1,333.4	14.35	1,350.4	14.21	1.310.1
2000x2000	28.53	4,397.3	27.67	4,284.5	27.28	3,770.9	27.69	4.202.9
2500x2500	35.8	2,007.9	34.69	1,957.6	34.69	2,007.5	34.69	1.973.7
3000x3000	51.9	2,051.2	51.26	2,036.7	51.07	2,104.1	50.83	2.018.2
Average	**27.38**	**2,151.5**	**26.73**	**2,111.3**	**26.65**	**2,039.6**	**26.68**	**970.3**

Table 6. Results of Revised Simplex Algorithm over Randomly Optimal Generated Problems with Density 10%

Density 10%	RSM		RSM with AM scaling		RSM with EQ scaling		RSM with GM scaling	
	time (sec)	iterations	time (sec)	iterations	time (sec)	iterations	time (sec)	iterations
1000x1000	348.02	33,792.4	222.38	19,787.8	114.74	10,046.1	318.49	32,061.3
1500x1500	1,375.35	60,393.3	864.65	37,145.6	495.85	21,911.4	1,124.91	53,154.4
2000x2000	3,651.17	94,639.7	2,512.86	63,071.7	1,768.99	41,846.5	3,245.30	82,613.5
2500x2500	7,945.33	135,998.5	5,119.14	84,925.1	3,657.21	57,218.9	6,514.19	115,013.6
3000x3000	15,450.59	173,233.6	9,858.00	111,358.2	6,680.26	73,303.1	12,005.41	146,513.1
Average	**5,754.09**	**99,611.5**	**3,715.41**	**63,257.7**	**2,543.41**	**40,865.2**	**4,641.66**	**85,871.2**

Table 7. Results of MATLAB's IPM Algorithm over Randomly Optimal Generated Problems with Density 20%

Density	IPM	IPM with AM scaling	IPM with EQ scaling	IPM with GM scaling

20%								
	time (sec)	**iterations**	**time (sec)**	**iterations**	**time (sec)**	**iterations**	**time (sec)**	**iterations**
1000x1000	69.40	23.0	26.89	22.3	21.81	22.1	28.30	23.0
1500x1500	131.46	19.1	51.50	23.5	51.22	24.9	56.50	22.1
2000x2000	260.74	24.4	111.45	26.0	110.68	26.8	96.60	25.6
2500x2500	633.91	23.7	271.46	25.1	246.72	26.4	312.45	25.0
3000x3000	840.08	24.2	328.90	25.3	271.82	27.6	269.13	26.4
Average	**387.12**	**22.8**	**158.04**	**24.4**	**140.45**	**25.6**	**152.60**	**24.4**

Table 8. Results of EPSA Algorithm over Randomly Optimal Generated Problems with Density 20%

Density 20%	**EPSA**		**EPSA with AM scaling**		**EPSA with EQ scaling**		**EPSA with GM scaling**	
	time (sec)	**iterations**	**time (sec)**	**iterations**	**time (sec)**	**iterations**	**time (sec)**	**iterations**
1000x1000	2.29	612.4	2.23	618.4	2.28	620.1	2.26	641.1
1500x1500	6.91	819.7	6.87	840.5	6.88	824.4	6.33	771.4
2000x2000	22.14	1,361.2	21.81	1,354.7	21.84	1,350.2	21.9	1,373.3
2500x2500	35.05	1,433.4	34.9	1,466.1	33.62	1,407.7	34.76	1,445.8
3000x3000	73.65	2,032.7	72.48	2,028.8	72.71	2,032.9	72.66	2,027.0
Average	**28.01**	**1,251.7**	**27.66**	**1,261.3**	**27.47**	**1,246.7**	**27.58**	**1,251.5**

Table 9. Results of Revised Simplex Algorithm over Randomly Optimal Generated Problems with Density 20%

Density 20%	**RSM**		**RSM with AM scaling**		**RSM with EQ scaling**		**RSM with GM scaling**	
	time (sec)	**iterations**	**time (sec)**	**iterations**	**time (sec)**	**iterations**	**time (sec)**	**iterations**
1000x1000	179.9	19,907.1	139.54	13,235.1	73.51	7,462.1	197.09	18,858.5
1500x1500	899.75	41,427.4	468.39	22,501.6	395.71	17,265.7	781.89	35,868.3
2000x2000	2,136.92	58,260.8	1,975.36	48,373.4	1,316.99	32,272.2	2,010.17	55,272.8
2500x2500	5,671.15	105,135.2	4,561.93	70,193.4	3,813.18	50,145.4	5,103.51	91,031.1
3000x3000	12,549.91	145,013.4	7,984.15	100,451.7	6,198.10	70,135.1	9,814.13	125,012.3
Average	**4,287.53**	**73,948.8**	**3,025.87**	**50,951.0**	**2,359.50**	**35,456.1**	**3,581.36**	**65,208.6**

5. CONCLUSIONS

Scaling is a prerequisite step that is applied prior to the application of simplex-type algorithms in order to reduce: (i) the condition number of the constrained matrix, (ii) the number of iterations, and (iii) the solution time of the algorithm. In this paper, we studied the impact of scaling prior to the application of the interior point methods, exterior point methods and revised simplex algorithm. We performed a computational study and found that equilibration is the best scaling technique on all types of algorithms. Furthermore, scaling is more important on IPM and revised simplex algorithm achieving to reduce noticeably both the solution time and the number of iterations needed by the algorithms. Finally, EPSA is proved to be scaling invariant.

6. REFERENCES

[1] Badr, E. S., Moussa, M., Papparrizos, K., Samaras, N., and Sifaleras, A. 2006. Some computational results on MPI parallel implementations of dense simplex method. In Proceedings of World Academy of Science, Engineering and Technology 23, 39-32 (CISE 2006, Cairo, Egypt, 8–10 December)

[2] Benhamadou, M. 2002. On the simplex algorithm ’revised form’. *Advances in Engineering Software* 33, 11-12, 769-777. DOI= http://dx.doi.org/10.1016/S0965-9978(02)00037-6

[3] Benichou, M., Gauthier, J. M., Hentges, G., and Ribiere, G. 1977. The efficient solution of large-scale linear programming problems-Some algorithmic techniques and computational results. *Mathematical Programming* 13, 1, 280-322. DOI= http://dx.doi.org/10.1007/BF01584344

[4] Curtis, A. R. and Reid, J. K. 1972. On the automatic scaling of matrices for Gaussian elimination. *IMA Journal of Applied Mathematics* 10, 1, 118-124. DOI= http://dx.doi.org/10.1093/imamat/10.1.118

[5] Dantzig, G. B. 1949. Programming of Interdependent Activities: II Mathematical Model. *Econometrica* 17, 3, 200-211. DOI= http://dx.doi.org/10.2307/1905523.

[6] Dantzig, G. B. 1953. *Computational Algorithm of the Revised Simplex Method*. RAND Report RM-1266, The RAND Corporation, Santa Monica, CA.

[7] de Buchet, J. 1966. Experiments and statistical data on the solving of large-scale linear programs. In *Proceedings of the Fourth International Conference on Operational Research*, Hertz, D. A. and Melese, J., Eds. Wiley-Interscience, New York, 3-13.

[8] Elble, J. M. and Sahinidis, N. V. 2012. Scaling linear optimization problems prior to application of the simplex method. *Computational Optimization and Applications* 52, 2, 345-371. DOI= http://dx.doi.org/10.1007/s10589-011-9420-4

[9] Fulkerson, D. R. and Wolfe, P. 1962. An algorithm for scaling matrices. *SIAM Review* 4, 2, 142-146. DOI= http://dx.doi.org/10.1137/1004032.

[10] Gondzio, J. 1996. Multiple centrality corrections in a primal-dual method for linear programming. *Computational Optimization and Applications* 6, 2, 137-156. DOI= http://dx.doi.org/10.1007/BF00249643.

[11] Hamming, R. W. 1971. *Introduction to Applied Numerical Analysis*. McGraw-Hill, New York.

[12] Karmarkar, N. K. 1984. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4, 4, 373–395. DOI= http://dx.doi.org/10.1007/BF02579150.

[13] Klee, V. and Minty, G. J. 1992. *How good is the simplex algorithm?*. Inequalities III. New York: Academic Press, 159–175.

[14] Larsson, T. 1993. On scaling linear programs-Some experimental results. *Optimization* 27, 4, 335-373. DOI= http://dx.doi.org/10.1080/02331939308843895

[15] Paparrizos, K. 1997. Pivoting algorithms generating two paths. In *Proceedings of the ISMP '97* (Lausanne, EPFL Switzerland, 1997).

[16] Paparrizos, K., Samaras, N., and Tsiplidis, K. 2001. Pivoting algorithms for (LP) generating two paths. In *Encyclopedia of Optimization* 4, Pardalos, M.P and Floudas A.C., Eds. Kluwer Academic Publishers, 302-306. DOI= http://dx.doi.org/10.1007/0-306-48332-7_388.

[17] Paparrizos, K., Samaras, N., and Stephanides, G. 2003. A new efficient primal dual simplex algorithm. *Computers and Operations Research* 30, 9, 1383-1399. DOI= http://dx.doi.org/10.1016/S0305-0548(02)00077-1.

[18] Paparrizos, K., Samaras, N., and Stephanides, G. 2003. An efficient simplex type algorithm for sparse and dense linear programs. *European Journal of Operational Research* 148, 2, 323-334. DOI= http://dx.doi.org/10.1016/S0377-2217(02)00400-9.

[19] Ploskas N. and Samaras N. 2013. A Computational Comparison of Basis Updating Schemes for the Simplex Algorithm on a CPU-GPU System. *Applied Mathematics and Computation*, Paper under review.

[20] Ploskas, N. and Samaras N. 2013. A Computational Comparison of Scaling Techniques for Linear Optimization Problems on a GPU. *Optimization Methods and Software*. Paper under review.

[21] Tomlin, J. A. 1975. On scaling linear programming problems. *Mathematical Programming Studies* 4, 146-166. DOI= http://dx.doi.org/10.1007/BFb0120718.

[22] Zhang, Y. 1998. Solving Large-Scale Linear Programs by Interior-Point Methods under the MATLAB Environment. *Optimization Methods and Software* 10, 1, 1-31. DOI= http://dx.doi.org/10.1080/10556789808805699.