Algorithms for Biproportional Apportionment

Sebastian Maier
Institut für Mathematik, Universität Augsburg

Abstract For the biproportional apportionment problem two algorithms are discussed, that are implemented in the Augsburg BAZI program, the alternating scaling algorithm and the tie-and-transfer algorithm of Balinski and Demange (1989b). The goal is to determine an integer-valued apportionment matrix that is "proportional to" a matrix of input weights (e.g. vote counts) and that at the same time achieves prespecified row and column marginals. The alternating scaling algorithm finds the solution of most of the practical problems very efficiently. However, it is possible to create examples for which the procedure fails. The tie-and-transfer algorithm converges always, though convergence may be slow. In order to make use of the benefits of both algorithms, a hybrid version is proposed.

Keywords: Biproportional divisor method; biproportional rounding algorithm; discrete alternating scaling; tie-and-transfer algorithm; BAZI.

1. Introduction

The Zurich Canton parliament is composed of seats that represent electoral districts as well as political parties (Pukelsheim, 2004b; Balinski and Pukelsheim, 2006; Pukelsheim, 2006). Each district $i = 1, \dots, k$ is represented by a number of seats r_i proportional to its population, and each political party $j = 1 \dots l$ gets c_j seats proportional to its total number of votes. The vote count in district i of party j is denoted by v_{ij}. Altogether the vote counts are assembled into a vote matrix $V \in \mathbb{N}^{k \times l}$ (see Box on page 106).

Box: Biproportional divisor method with standard rounding (Neues Zürcher Zuteilungsverfahren).

The seats per district are apportioned in the middle of the legislature period on the basis of the population counts. Party seats are allocated on election day on the basis of the total party ballots in the whole electoral region. As the Zurich electoral law provides each voter with as many ballots as the district has seats, we need to compute the support for a party in a district. This is done by dividing the raw data that are returned by the polling stations by the district magnitude, and by rounding the resulting quotient to the closest integer. For each party, these district support sizes are summed over all districts leading to the overall support size for a party. The support size may be interpreted as number of people supporting a party. It is used to compute the superapportionment, this is, the allocation of the seats to the parties across the whole electoral region.

The final step is the subapportionment, the allocation of the seats to the parties within the districts. It provides a two-way proportionality, achieving the prespecified district magnitudes and the party seats. To compute the apportionment we need two sets of divisors, the district divisors and the party divisor. Each vote count of a party in a district is divided by its corresponding district divisor and party divisor; this quotient is rounded in the usual way to obtain the seat-number. A more detailed description of the Zurich apportionment method can be found in Balinski and Pukelsheim (2006) and Pukelsheim (2006).

Zurich City Parliament election of 12 February 2006, Superapportionment:

		SP	SVP	FDP	Greens	CVP	EVP	AL	SD	City divisor
Support size		23180	12633	10300	7501	5418	3088	2517	1692	530
Seats	125	44	24	19	14	10	6	5	3	

Zurich City Parliament election of 12 February 2006, Subapportionment:

		SP	SVP	FDP	Greens	CVP	EVP	AL	SD	District-
	125	44	24	19	14	10	6	5	3	divisor
"1+2"	12	28518-4	15305-2	21833-3	12401-2	7318-1	2829-0	2413-0	1651-0	7000
"3"	16	45541-7	22060-3	10450-1	17319-3	8661-1	2816-0	7418-1	3173-0	6900
"4+5"	13	26673-5	8174-2	4536-1	10221-2	4099-1	1029-0	9086-2	1406-0	5000
"6"	10	24092-4	9676-1	10919-2	8420-1	4399-1	3422-1	2304-0	1106-0	6600
"7+8"	17	61738-5	27906-2	51252-5	25486-2	14223-1	10508-1	5483-1	2454-0	11200
"9"	16	42044-6	31559-4	12060-2	9154-1	11333-1	9841-1	2465-0	5333-1	7580
"10"	12	35259-4	19557-3	15267-2	9689-1	8347-1	4690-1	2539-0	1490-0	7800
"11"	19	56547-6	40144-4	19744-2	12559-1	14762-2	11998-2	3623-1	6226-1	9000
"12"	10	13215-3	10248-3	3066-1	2187-1	4941-1	0-0	429-0	2078-1	4000
Partydivisor		1.006	1.002	1.01	0.97	11333-1	0.88	0.8	1	

Table entries are of the form v-a, where v is the number of party votes in the district and a is the number of seats apportioned to that party's list in the district. The party ballot v is divided by the associated district and party divisors, and then rounded in the standard way to obtain a. In district "1+2" the Greens had 12401 ballots and were awarded by 2 seats, since $12401/(7000 \times 0.97) = 1.83 \nearrow 2$.

This leads to the following proportional matrix problem (cf. Balinski and Demange 1989a,b; Pukelsheim, 2004; Balinski and Pukelsheim, 2006; Pukelsheim, 2006). Find a matrix apportionment $A \in \mathbb{N}^{k \times l}$ and row divisors ρ_i, $i = 1 \dots k$ and column divisors γ_j, $j = 1 \dots l$ which satisfy the following conditions:

$$a_{ij} = \left[\frac{v_{ij}}{\rho_i \cdot \gamma_j}\right]_s \tag{1}$$

$$a_{i+} = \sum_{j \le l} a_{ij} = r_i, \ i = 1, \dots, k \tag{2}$$

$$a_{+j} = \sum_{i \le k} a_{ij} = c_j, \ j = 1, \dots, l. \tag{3}$$

The rounding $[x]_s$ of a positive number $x \in [n, n+1]$, $n \in \mathbb{N}$ depends on a dividing point $s(n) \in [n, n+1]$. We have $[x]_s = n+1$, if $x > s(n)$, and $[x]_s = n$, if $x < s(n)$. In the case x hits the signpost $s(n)$, x can be either rounded down or up. Thus the commonly known divisor methods for vector apportionments are extended to the matrix case. Matrix apportionments which are computed using divisor methods share nice properties, e.g. uniformity and homogeneity, and are unique up to ties (Balinski and Demange 1989a,b).

Since the matrix problem cannot be solved in one step, we need iterative procedures. In Section 2 we review two algorithms for computing the matrix apportionment. In Section 3 we investigate the runtime and the error functional. This leads us to suggest merging the advantages of both algorithms to a hybrid algorithm in Section 4.

2. Algorithms

The two algorithms to be reviewed are the alternating scaling algorithm, a discrete version of the commonly known iterative proportional fitting algorithms from statistics, and the tie-and-transfer algorithm proposed in (Balinski and Demange, 1989b).

For both algorithms, a measure for the improvement is the error count in step t which corresponds to an interim apportionment $A(t)$:

$$f(t) := \frac{1}{2} \sum_{i \le k} |a_{i+}(t) - r_i| + \frac{1}{2} \sum_{j \le l} |a_{+j}(t) - c_j|$$

This is the number of seat transfers necessary to achieve the solution, that is, the number of "wrong" allocations within the apportionment matrix. The procedure stops as soon as the error count is zero.

Before starting the computation, existence of a solution can be checked by using a max-flow min-cut algorithm (Joas, 2005). For the continuous case, existence is investigated for example in Bacharach (1970) or Pretzel (1980). In the sequel, we assume existence. Also we do not pay attention to multiplicities that are possible when the scaled weights hit a signpost.

2.1 Alternating Scaling Algorithm (AS)

The continuous alternating scaling algorithm was proposed by Deming and Stephan (1940) and has many applications in statistics, such as fitting contingency tables or fitting loglinear models (Fienberg and Meyer, 1983). This procedure, also known as the RAS algorithm, is extensively studied in literature (Ireland and Kullback, 1968; Marshall and Olkin, 1968; Bacharach, 1970). A more extensive and detailed overview on the literature can be found in Balinski and Pukelsheim (2006). The discrete version of the alternating scaling procedure and its properties is described in Pukelsheim (2004). The idea is to solve the vector problem for rows in odd steps and for columns in even steps. Hence, either rows or columns are fitted. If the rows are fitted, errors may be left in the columns, and if the columns are fitted, errors may be left in the rows. Thus the matrix problem is reduced to many vector problems, either by solving a set of row problems, or by solving a set of column problems. To solve such a vector problem, the algorithm described in Dorfleitner and Klein (1999) is used in updating the divisors in each step. The algorithm succeeds in presenting row and column divisors fulfilling (1) – (3). A Java implementation of the following algorithm can be downloaded from `www.uni-augsburg.de/bazi`.

Algorithm (Discrete Alternating Scaling)

(0) Initialize start divisors $P_i(0) = 1, i = 1\ldots, k$ *and* $\Gamma_j(1) = 1, j = 1, \ldots, l$. *At every step t, the scaled weights will be of the form* $v_{ij}(t) = \frac{v_{ij}}{P_i(t)\cdot\Gamma_j(t)}$.

(i) For odd steps, find row divisors $\rho_i(t)$ *such that, with updated divisors* $P_i(t) = \rho_i(1)\rho_i(3)\ldots\rho_i(t)$, *the apportionment* $a_{ij}(t) = [v_{ij}(t)]_s$ *satisfies (2).*

(ii) For even steps, find column divisors $\gamma_j(t)$ *such that, with updated divisors* $\Gamma_j(t) = \gamma_j(2)\gamma_j(4)\ldots\gamma_j(t)$, *the apportionment* $a_{ij}(t) = [v_{ij}(t)]_s$ *satisfies (3).*

The algorithm terminates successfully after finitely many steps, T say, when (1) – (3) are satisfied with divisors $\rho_i = P_i(T)$ *and* $\gamma_j = \Gamma_j(T)$.

Note that row divisors are updated only in odd steps, and column divisors in even steps. Therefore the updated divisors are of the form

$$P_i(t) = \rho_i(1)\rho_i(3)\cdots\rho_i\left(2\lceil\frac{t}{2}\rceil - 1\right),\ \Gamma_j(t) = \gamma_j(2)\gamma_j(4)\cdots\gamma_j\left(2\lfloor\frac{t}{2}\rfloor\right).$$

2.2 Tie-and-Transfer (TT)

The tie-and-transfer algorithm was first described in Balinski and Demange (1989b), in a more general form dealing with inequality constraints for rows and columns. The case of equality constraints can be found in Balinski and Rachev (1997). The main idea is to transform the biproportional problem into a bipartite graph. This graph is used to find a feasible flow corresponding to a biproportional apportionment (Balinski and Demange, 1989b; Balinski and Rachev, 1997; Zachariasen, 2006).

Algorithm (Tie-and-Transfer)

(0) Start with an initial apportionment exhausting the housesize. This initial apportionment is obtained by fitting all columns as proposed in Balinski and Rachev (1997). Then the following labelling procedure is established.

(i) Either determine subsets of rows and columns to modify the row and column divisors. The modification is done in such a way that at least one more of the rescaled weights is either scaled down to the previous signpost, or scaled up to the next signpost. This means that the rescaled weight can be rounded either down or up without affecting the divisors.

(ii) Else determine a path from an underrepresented row to an overrepresented row in the graph. The path is along an interim apportionment on rescaled weights hitting a signpost alternating being rounded down and rounded up. Along this path the direction of rounding is changed, that is, rescaled weights which were rounded down will now be rounded up, and rescaled weights which were rounded up will now be rounded down. This procedure increases the number of seats in the underrepresented row and decreases the number of seats in the overrepresented row. The net effect of the transfer is a decrease of the error function by exactly one unit. The transfer does not affect any other row or column sums.

The algorithm will terminate after finitely many steps.

Note that the apportionment is only modified on arcs that correspond to rescaled weights on signposts. This ensures at every step an apportionment which can be obtained by the current divisors. The error count decrease is

Table 1. Runtimes, iterations and starting errors: AS seems to perform better than TT, except for examples MOC3T and MOD3T.

	Tie-and-Transfer		Alternating Scaling	
	time (sec.)	start error (count)	time (sec.)	iterations (count)
KRW1995	2	30	1	11
KRW1999	2	29	2	75
KRW2003	2	29	1	18
MOB3T	6	499	2	177
MOC3T	6	500	25	2294
MOD3T	6	500	34	3472
MOB3M	7674	499092	5	395
MOC3M	7756	499902	92	6535
MOD3M	7043	499999	112	9468

$o(lkf(0))$ (Balinski and Demange, 1989b). In contrary to the alternating scaling algorithm, the tie-and-transfer-algorithm in every step either modifies the divisors, or changes the apportionment.

3. Properties and Data

The BAZI-program (`www.-augsburg.de/bazi`) includes not only various apportionment methods, but also an extensive data-base. This data-base includes examples for vector problems and for matrix problems, empirical examples, and academic ones. To study the performance of the algorithms on empirical data, we choose the last three elections for the Zurich Canton parliament, KRW1995, KRW1999, KRW2003. There are 18 districts, and 13 or 14 participating parties. The housesize is 180, the district magnitudes vary from 4 to 16, the parties get 1 to 55 seats.

We also study 3×3 weight matrices that are motivated by the literature on the continuous iterative proportional fitting algorithm (Marshall and Olkin, 1968). These matrices have large "housesizes" of 3000 (MOB3T, MOC3T, MOD3T), or 3000000 (MOB3M, MOC3M, MOD3M).

Table 1 summarizes the observed runtimes of the computation. Since the error count function is linearly decreasing for the tie-and-transfer algorithm, only the starting error count is given. For the alternating scaling algorithm the number of iterations is shown. Both algorithms are quite fast for the Zurich Canton Parliament Election data, taking about 1 to 2 seconds. For MOB3T both algorithms are very fast, but for MOC3T and MOD3T alternating scaling takes about four to six times longer than tie-and-transfer. The three examples

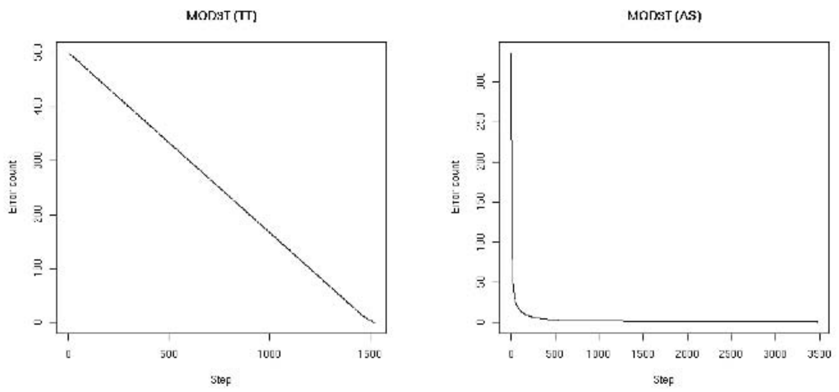

Fig. 1. Error count for MOD3T: For TT, the decrease is constant. For AS, the decrease is fast at the beginning and slow towards the end. It reaches 0 after 3472 iterations.

with housesizes 3000000 exhibit an extreme behavior: tie-and-transfer takes hours, while the alternating scaling algorithm ends within seconds or minutes. To explain this behavior, we take a closer look at the development of the error counts.

Figure 1 shows the error count function for MOD3T for the tie-and-transfer algorithm on the left hand side, and for the alternating scaling algorithm on the right hand side. The running times are 6.5 seconds for tie-and-transfer, and 34.4 seconds for alternating scaling. For the alternating scaling algorithm, the decrease is very fast in the beginning, but very slow towards the end. It takes 225 steps for a decrease from three to two, 511 steps from two to one, and another 2200 steps to end at zero. It is known from Fienberg and Meyer (1983) that the convergence of the continuous iterative scaling procedure may be very slow. The tie-and-transfer algorithm shows the predicted linear behavior, by reducing the error count one by one. Starting with an error of 500, it is faster than alternating scaling which looses a lot of time towards the end.

Figure 2 conveys the same information. A constant decrease for the tie-and-transfer algorithm, but it has to start with an error count of 499092, and it takes rather long (about 2 hours) to work this down to zero. Alternating scaling is fast, but again the last steps take excessively long. It takes 491 steps for the decrease from three to two, 46 steps from two to one, and another 1577 steps from one to zero.

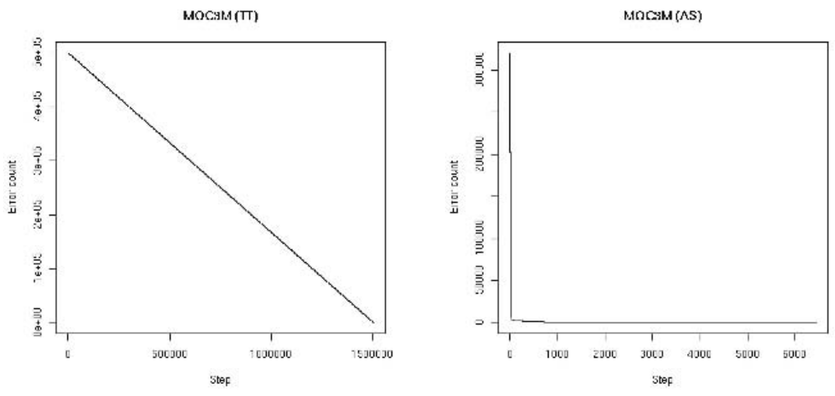

Fig. 2. Error count for MOC3M: The decrease is constant for TT with starting error 499902. For AS, the decrease is fast at the beginning and slow towards the end, reaching 0 after 653 iterations.

4. Hybrid Algorithm

On the basis of these examples we may summarize the runtime properties of the two algorithms: the alternating scaling algorithm is very fast in the beginning, but may take very long towards the end. The tie-and-transfer algorithm processes the error count one by one, even if there is a big error count left.

To combine the advantages of both algorithms, we suggest a hybrid version, starting with the alternating scaling algorithm and finishing with the tie-and-transfer algorithm. The challenge is to implement an appropriate time to switch.

At the end of the column adjustment there is a check for a switch of the method. We have experimented with two switching rules.

1 *Fast switch:* The error count stays the same for two iterations.

2 *Adaptive switch:* The error count stays the same as many iterations as it has digits, e.g. six iterations for an error count of 499902.

Figure 3 shows the error count decrease of the hybrid algorithm with adaptive switch for MOD3T and MOC3M. The decrease is fast in the beginning on the left hand side of the vertical line that marks the switching point. After the switch on the right hand side the error count function is linear like the error count function of the tie-and-transfer algorithm. To compare the decrease of the hybrid algorithm, the decrease of the alternating scaling algorithm is also plotted. The error count function of the hybrid algorithm decreases faster than that for the alternating scaling algorithm.

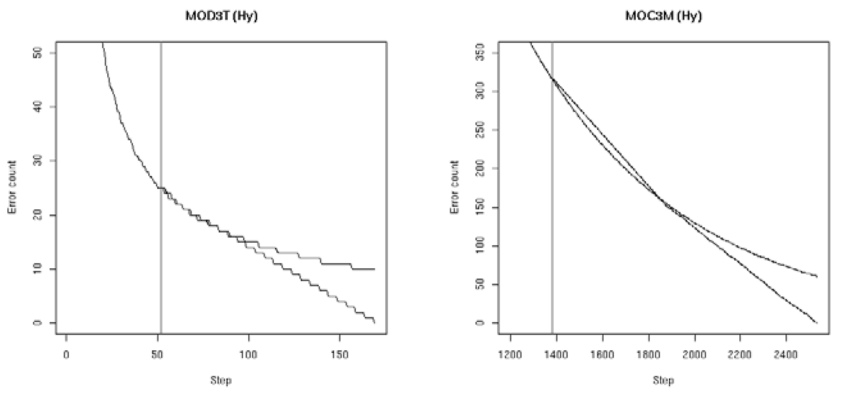

Fig. 3. Error count for MOD3T and MOC3M using the hybrid algorithm with adaptive switch: On the left hand side of the vertical line the decrease of AS can be seen. After the switching point the error count decreases linearly using the hybrid algorithm.

Table 2. Runtimes, iterations and error counts for the the tie-and-transfer algorithm, the alternating scaling algorithm, and the hybrid algorithm: The hybrid algorithm performs always better than the other proposed algorithms.

	Tie-and-Transfer		Alternating Scaling		Fast Sw. Hybrid		Adapt. Sw. Hybrid	
	time (sec.)	start error (count)	time (sec.)	iter. (count)	time (sec.)	iter.+err. c. (count)	time (sec.)	iter.+err. c. (count)
KRW1995	2	30	1	11	1	6 + 2	1	6 + 2
KRW1999	2	29	2	75	2	6 + 1	2	6 + 1
KRW2003	2	29	1	18	2	6 + 2	2	6 + 2
MOB3T	6	499	2	177	1	52 + 20	1	52 + 20
MOC3T	6	500	25	2294	0	24 + 4	0	24 + 4
MOD3T	6	500	34	3472	1	52 + 25	1	52 + 25
MOB3M	7674	499092	5	395	4	270 + 24	4	282 + 17
MOC3M	7756	499902	92	6535	24	832 + 709	26	1380 + 317
MOD3M	7043	499999	112	9468	51	922 + 1806	40	1934 + 571

Table 2 summarizes the runtime improvements of the hybrid algorithm. In each line there is a runtime decrease. For empirical examples both switching rules have the same effect, because of the small number of error counts. For larger housesizes there is a runtime decrease from about two hours for the tie-and-transfer algorithm down to one minute. The improvement for the alternating scaling algorithm is also substantial, though not quite so spectacular.

Another way to speed up the calculation is to start with continuous iterative proportional fitting (IPF) and switch according to some rule to one of the discrete algorithms. This approach is proposed in Balinski and Demange (1989b) for finding a good starting point for the tie-and-transfer algorithm.

Table 3 summarizes running times and remaining error counts for the tie-and-transfer algorithm and required iterations for the alternating scaling procedure for a switching barrier of one. Table 4 applies for a switching barrier of ten. For the Canton Zurich data the combination of continuous and discrete alternating scaling is approximately as fast as the discrete algorithm. If there is a higher error count left, e.g. for MOC3T, MOC3M, MOD3T, and MOD3M with barrier ten, the disadvantage of slow convergence towards the end becomes visible. Using barrier one, the combination of continuous iterative proportional fitting and alternating scaling is quite fast. Using iterative proportional fitting together with tie-and-transfer together is a little slower than the other combination (IPF – AS) for the empirical examples. For the artificial examples, it has a very good performance. After the initial fitting of columns, there is no or only a small error count left for barrier one. Especially for MOC3M and MOD3M there are 369 and 445 alternating scaling steps necessary to reduce the error count to zero, but this is processed very fast by the tie and transfer algorithm.

In conclusion we find that the hybrid algorithm performs better than one of the proposed methods alone The choice of the switching rule has practically no influence, hence we decided to implement the fast switching rule in BAZI. The use of the continuous iterative proportional fitting procedure at the beginning would be another option. Another proposol to improve upon the tie-and-transfer algorithm, and a detailed investigation of the runtime properties of the algorithms can be found in Zachariasen (2006).

Acknowledgments

The author is grateful for the support of the Deutsche Forschungsgemeinschaft. I thank Michel Balinski, Bruno Simeone and an anonymous referee for all their helpful comments. A special thank to Friedrich Pukelsheim for his valuable advice and continuous support.

Table 3. Runtimes, iterations and error counts for continuous hybrid algorithm using a switching barrier of one: Starting with IPF speeds up the computation. For larger housesizes switching to TT performs better.

	TT	AS	IPF–TT (1)		IPF–AS (1)	
	time	time	time	err. c.	time	iter.
	(sec.)	(sec.)	(sec.)	(count)	(sec.)	(count)
KRW1995	2	1	1.3	6	0.9	16
KRW1999	2	2	2.1	8	1.3	39
KRW2003	2	1	1.6	6	0.7	11
MOB3T	6	2	0.2	1	0.2	9
MOC3T	6	25	0.8	0	0.8	2
MOD3T	6	34	1.4	0	1.3	1
MOB3M	7674	5	0.2	0	0.2	2
MOC3M	7756	92	2.1	1	5.8	369
MOD3M	7043	112	3.2	1	7.5	445

Table 4. Runtimes, iterations and error counts for continuous hybrid algorithm using a switching barrier of ten: Starting with IPF speeds up the computation. Switching to AS the disadvantage of slow convergence towards the end becomes visible again.

	TT	AS	IPF–TT (10)		IPF–AS (10)	
	time	time	time	err. c.	time	iter.
	(sec.)	(sec.)	(sec.)	(count)	(sec.)	(count)
KRW1995	2	1	1.3	6	0.7	16
KRW1999	2	2	2	8	1.15	32
KRW2003	2	1	1.8	6	0.76	9
MOB3T	6	2	0.2	5	1	82
MOC3T	6	25	0.2	5	24.76	2286
MOD3T	6	34	0.4	5	30.26	3105
MOB3M	7674	5	0.3	5	0.92	74
MOC3M	7756	92	1.6	5	27.95	2257
MOD3M	7043	112	2.4	5	30.8	2806

References

Bacharach, Michael (1970): *Biproportional Matrices & Input-Output Change*. Cambridge UK.

Balinski, Michel L. and Demange, Gabrielle (1989a): An axiomatic approach to proportionality between matrices. *Mathematics of Operations Research*, 14:700–719.

Balinski, Michel L. and Demange, Gabrielle (1989b): Algorithms for proportional matrices in reals and integers. *Mathematical Programming*, 45:193–210.

Balinski, Michel L. and Pukelsheim, Friedrich (2006): Matrices and politics. In E. Liski, S. Puntanen J. Isotalo, and G. P. H. Styan, editors, *Festschrift for Tarmo Pukkila on His 60th Birthday*. Department of Mathematics, Statistics, and Philosophy: Tampere.

Balinski, Michel L. and Rachev, Svetlozar T. (1997). Rouding proportions: Methods of rounding. *Mathematical Scientist*, 22:1–26.

Deming, W. Edwards and Stephan, Frederick F. (1940). On a least squares adjustment of a sample frequency table when the expected marginal totals are known. *Annals of Mathematical Statistics*, 11:427–440.

Dorfleitner, Gregor and Klein, Thomas (1999). Rounding with multiplier methods: An efficient algorithm and applications in statistics. *Statistical Papers*, 20:143–157.

Fienberg, Stephan S. and Meyer, Michael M. (1983): Iterative proportional fitting. In *Encyclopedia of Statistical Sciences*, volume 4, pages 275–279. John Wiley & Sons.

Ireland, C. T. and Kullback, S. (1968): Contingency tables with given marginals. *Biometrika*, 55(1):179–188.

Joas, Bianca (2005): A graph theoretic solvability check for biproportional multiplier methods. Diploma Thesis, Institute of Mathematics, University of Augsburg.

Marshall, Albert W. and Olkin, Ingram (1968) Scaling of matrices to achieve specified row and column sums. *Numerische Mathematik*, 12:83–90.

Pretzel, Oliver (1980): Convergence of the interative scaling procedure for non-negative matrices. *Journal of the London Mathematical Society*, 21:379–384.

Pukelsheim, Friedrich (2004): BAZI - a java programm for proportional representation. In *Oberwolfach Reports*, volume 1, pages 735–737.

Pukelsheim, Friedrich (2006): Current issues of apportionment methods. In Bruno Simeone and Friedrich Pukelsheim, editors, *Mathematics and Democracy. Recent Advances in Voting Systems and Collective Choice*. New York, 2006.

Pukelsheim, Friedrich and Schuhmacher, Christian (2004): Das neue Zürcher Zuteilungsverfahren bei Parlamentswahlen. *Aktuelle Juristische Praxis - Pratique Juridique Actuelle*, 13: 505–522.

Zachariasen, Martin (2006): Alorithmic aspects of divisor-based biproportional rounding. Typescript, April 2006.