Matrix Scaling by Network Flow

Günter Rote* Martin Zachariasen†

July 5, 2006

Abstract

A given nonnegative $n \times n$ matrix $A = (a_{ij})$ is to be scaled, by multiplying its rows and columns by unknown positive multipliers λ_i and μ_j, such that the resulting matrix $(a_{ij}\lambda_i\mu_j)$ has specified row and column sums r_i and s_j.

We give an algorithm that achieves the desired row and column sums with a maximum absolute error ε in $O(n^4(\log n + \log \frac{h}{\varepsilon}))$ steps, where h is the overall total of the result matrix.

Our algorithm is a scaling algorithm. It solves a sequence of more and more refined discretizations. The discretizations are minimum-cost network flow problems with convex piecewise linear costs. These discretizations are interesting in their own right because they arise in proportional elections.

1 Introduction

The Matrix Scaling Problem. The input of the (real-valued) *matrix scaling problem* is a non-negative $n \times n$ matrix $A = (a_{ij})$, and two positive n-vectors $r = (r_i)$ and $s = (s_j)$. The problem is to find positive multipliers λ_i for scaling the rows and μ_j for scaling the columns, such that the resulting matrix $(a_{ij}\lambda_i\mu_j)$ has the specified values r_i and s_j as row and column sums:

$$\sum_{j=1}^{n} a_{ij}\lambda_i\mu_j = r_i, \qquad \sum_{i=1}^{n} a_{ij}\lambda_i\mu_j = s_j \tag{1}$$

Of course, we must have $\sum_{i=1}^{n} r_i = \sum_{j=1}^{n} s_j$. We denote this common value by h, the *total sum* of the desired matrix.

*Freie Universität Berlin, Institut für Informatik, Takustraße 9, 14195 Berlin, Germany, rote@inf.fu-berlin.de.

†University of Copenhagen, Department of Computer Science, Universitetsparken 1, DK-2100 Copenhagen Ø, Denmark, martinz@diku.dk.

A special case is when all r_i and s_j are 1. This is the *doubly stochastic* scaling problem. The problem also makes sense for rectangular matrices, but for simplicity, we restrict our attention to the square case.

The problem (1) is a non-linear system of equations, and there is no hope to solve it exactly in reasonable time. Thus we settle for an ε-approximate solution:

$$\left|\sum_{j=1}^{n} a_{ij}\lambda_i\mu_j - r_i\right| \le \varepsilon, \text{ and } \left|\sum_{i=1}^{n} a_{ij}\lambda_i\mu_j - s_j\right| \le \varepsilon \tag{2}$$

Applications. There is an abundance of literature about the problem, see for example [8] and the references given there. Linial, Samorodnitsky, and Wigderson [7] mention such diverse applications as statistics and image reconstruction. In numerical analysis, matrix scaling is used as a simple preconditioning operation to improve the numerical stability for solving linear equations. Linial et al. [7] have used doubly stochastic matrix scaling as a tool for approximating the permanent of a nonnegative matrix with a multiplicative error of e^n.

A fast *parallel* algorithm for doubly stochastic matrix scaling (with $\varepsilon = 1/n$) could be used for the bipartite matching problem, which is a major open problem in parallel complexity, see [7, Section 5.1] or [4].

Previous Results. The problem (1) has a solution if and only if the system

$$\sum_{j=1}^{n} x_{ij} = r_i, \qquad \sum_{i=1}^{n} x_{ij} = s_j \tag{3}$$

has a solution (x_{ij}) with the same sign structure as A: $x_{ij} > 0$ if $a_{ij} > 0$ and $x_{ij} = 0$ otherwise. This condition can be tested in polynomial time because it is just a network flow problem.

If a solution exists, the values $a_{ij}\lambda_i\mu_j$ are unique. Obviously, one has the freedom of shifting a common factor between the λ_i's and the μ_j's.

There are several different formulations of the matrix scaling problem as a nonlinear optimization problem.

Algorithms. The original paper by Sinkhorn [9] introduced the simple algorithm of alternatively scaling the rows and columns to the desired sums, and he proved (originally only for the case of doubly stochastic scaling and for a positive matrix A) that the process converges to the solution. Relatively little work has been done on the complexity of computing a solution. Most of this research concentrated on investigating the convergence of the *Sinkhorn scaling* approach. If a solution exists, convergence to the correct solution is linear (i. e., the error decreases geometrically), but the rate of convergence deteriorates when entries of the scaled matrix approach zero [10].

For doubly stochastic scaling, an algorithm of Kalantari and Khachian [6] uses the ellipsoid method and takes $O(\log L(A) \cdot n^4 \log \frac{n}{\varepsilon})$ arithmetic operations, where $L(A)$ is the ratio between the largest and the smallest positive entry in the matrix. The only strongly polynomial running time so far is achieved by Linial et al. [7]. It takes $O(n^7 \log \frac{h}{\varepsilon})$ operations.[1] For the case of doubly stochastic scaling, they have a different algorithm which takes $O(n^3 \log n/\varepsilon^2)$ operations.[2]

The Integer Matrix Scaling Problem. In the integer version of the matrix scaling problem, the resulting values are rounded to integers:

$$\sum_{j=1}^{n} [a_{ij}\lambda_i\mu_j] = r_i, \qquad \sum_{i=1}^{n} [a_{ij}\lambda_i\mu_j] = s_j \tag{4}$$

Here, $[\cdot]$ denotes the rounding-down operation: $x = [u]$ denotes an integer value with

$$u - 1 \le x \le u, \tag{5}$$

i. e., $x = \lfloor u \rfloor$ if u is not integral. If u is integral, there is a *tie*: $x = u$ or $x = u - 1$.[3]

This problem arises in *biproportional voting systems*, where the allocated seats are classified by two independent categories: the total number of seats s_j in each *district* j is specified in advance, according to the population; and the total number of seats r_i for each *party* i is calculated in proportion to the total vote for that party. Moreover, the seats should be proportional both in rows and columns. Such a model (with a different rounding function) was recently used for electing the 125 members of the municipal council of the city of Zurich on February 12, 2006.[4]

Previous Results on the Integer Matrix Scaling Problem. The biproportional scaling problem with rounding was first investigated by Balinski and Demange [1]. The conditions for the existence of a solution of the

[1] The running time claimed in [7] has an extra factor of $\log n$ which comes from Theorem 4.3. We don't see how this factor arises in the proof.

[2] This result does not appear explicitly in [7]. A different definition of the error is used there: the squared L_2 norm ε' of the discrepancies in the row and column sums. The time bound is $O(n^3 \log n/\varepsilon')$, and it is stated only for $\varepsilon' = 1/(n \log n)$. An $O(n^5 \log n)$ additive term for preprocessing in the algorithm can be reduced to $O(n^3)$.

[3] Other rounding functions, like rounding to the nearest integer, can be handled as well, but we stick to this simple rounding function in order to simplify the discussion.

[4] See `http://daten.wahlen.stzh.ch/`.

system (4) are the same as for the real version (1). The solution values $x_{ij} = [a_{ij}\lambda_i\mu_j]$ are unique except for the presence of ties.[5] (The multipliers λ_i and μ_j can of course vary in a certain range where they don't modify the rounded values x_{ij}.)

Balinski and Demange [1] have given an algorithm that computes a solution in $O(n^3h)$ time (see also [2]).[6]

Gaffke and Pukelsheim [5] have investigated the relation between the primal and dual version of these problems, (6–9) and (4), respectively, (with generalized rounding functions) which is expressed in Theorem 1 below.

Our Contribution. We observe that the formulation (6–9) for the *integer* matrix scaling problem can in fact be modeled as a standard minimum-cost network flow problem. This opens the way to use any of numerous alternative methods for network flow problems, leading to better runtimes (Theorems 2 and 3 in Sections 2 and 3).

We then apply our result as an approximation algorithm for the *real-number* matrix scaling problem in a straightforward way, leading to a simpler and faster algorithm (Theorem 4 in Section 4).

2 A Network Flow Model for Matrix Scaling

The following constraints (7–9) define a network flow problem on the bipartite graph $G(A)$ with n source vertices R_i and n sink vertices S_j, see Figure 1a.

$$\text{minimize} \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{t=1}^{x_{ij}} \log \frac{t}{a_{ij}} \tag{6}$$

$$\text{subject to} \sum_{j=1}^{n} x_{ij} = r_i \text{ for } i = 1,\ldots,n \tag{7}$$

$$\sum_{i=1}^{n} x_{ij} = s_j \text{ for } j = 1,\ldots,n \tag{8}$$

$$x_{ij} \geq 0,\ x_{ij} \text{ integer},\ x_{ij} = 0 \text{ if } a_{ij} = 0 \tag{9}$$

[5] Some occasional ambiguity as in (5) is inherent in the model, for otherwise there might be no solution: Consider the 2×2 problem where all entries of A are identical, and the row and column sums are 1. The two possible solutions are $X = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ and $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, but no "consistent" way of rounding fractions will produce these matrices, whatever the choice of λ_i and μ_j might be.

[6] With hindsight, this algorithm can be interpreted as a special instance of a standard primal-dual algorithm for our network flow model (6–9).

The variable x_{ij} is the flow value in arc (R_i, S_j). If $a_{ij} = 0$, there is no arc (R_i, S_j). The cost function for an arc

$$f_{ij}(x) := \sum_{t=1}^{x} \log \frac{t}{a_{ij}}$$

is piecewise linear and convex, as illustrated in Figure 1b.

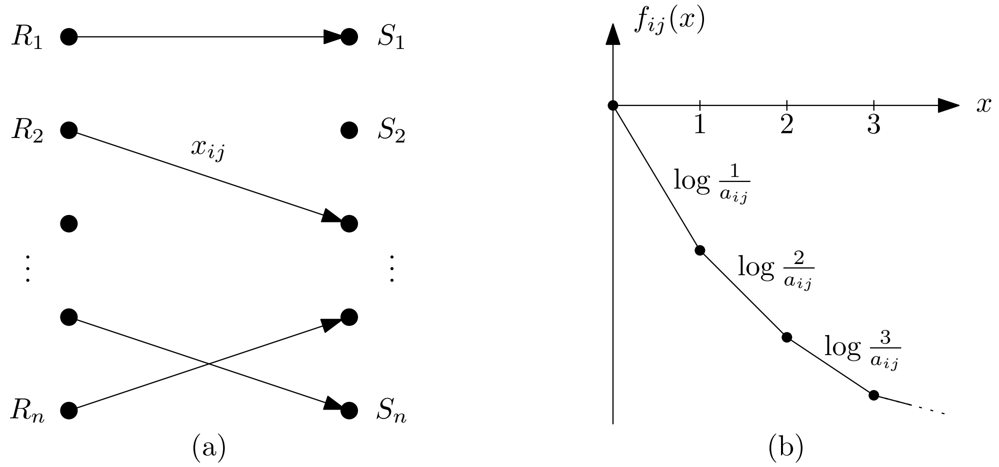

Figure 1: (a) The graph $G(A)$. (Only a few arcs are shown.) (b) The piecewise linear cost function $f_{ij}(x)$. Between successive breakpoints, the function makes vertical steps of size $\log \frac{t}{a_{ij}}$, for $t = 1, 2, 3, \ldots$.

A piecewise linear convex cost function can be modeled by linear cost functions on a sequence of parallel arcs: between A_i and B_j, we introduce parallel arcs with capacity 1 and cost coefficients $c_{ij}^t = \log \frac{t}{a_{ij}}$, for $t = 1, 2, \ldots$. Clearly, a flow of some integral value x from R_i to S_j will fill the x cheapest arcs, and hence the total cost equals $f_{ij}(x)$.

This is then a classical minimum-cost network flow problem, and the standard theory can be applied. In particular, we know that there is an integral optimum solution if there is a solution at all.

The following connection to the matrix scaling problem was proved in [5, Theorem 8.1], in a more general setting.

Theorem 1. *An optimum solution $X = (x_{ij})$ of the flow problem* (6–9) *is a solution $x_{ij} = [a_{ij}\lambda_i\mu_j]$ of the integer matrix scaling problem* (4).

Proof. For an optimum solution (x_{ij}), there are dual variables α_i, β_j for which all reduced costs $c_{ij}^t - \alpha_i - \beta_j$ of arcs in the residual network are nonnegative. In particular, looking at the forward arcs (R_i, S_j), this means that

$$\log \frac{t}{a_{ij}} - \alpha_i - \beta_j \geq 0, \text{ for all } t \geq x_{ij} + 1,$$

which is equivalent to

$$\log \frac{x_{ij}+1}{a_{ij}} - \alpha_i - \beta_j \geq 0$$

or

$$x_{ij} + 1 \geq a_{ij} \cdot e^{\alpha_i} e^{\beta_j}.$$

Similarly, from the backward arcs (S_j, R_i), we have $-c_{ij}^t + \alpha_i + \beta_j \geq 0$, or

$$x_{ij} \leq a_{ij} \cdot e^{\alpha_i} e^{\beta_j}.$$

With $\lambda_i := e^{\alpha_i}$ and $\nu_j := e^{\beta_j}$ this means that x_{ij} is the correctly rounded value of $a_{ij}\lambda_i\mu_j$:

$$a_{ij}\lambda_i\mu_j - 1 \leq x_{ij} \leq a_{ij}\lambda_i\mu_j \tag{10}$$

□

Thus, we can solve the integer matrix scaling problem by solving a minimum-cost network flow problem with any of the standard algorithms. A simple algorithm is the least-cost augmenting path method, cf. [3, Section 4.3]: start with the zero flow and successively augment the flow along minimum-cost augmenting paths, using the labels from the shortest path computation to update dual variables α_i and β_j.

In order not to distract from the algorithm, we assume in this section that all calculations (including logarithms and exponentials) are computed exactly. We will analyze the necessary precision in Section 5.

For the row sums and column sums of a matrix $X = (x_{ij})$ we use the notations $x_{i.} := \sum_{j=1}^n x_{ij}$ and $x_{.j} := \sum_{i=1}^n x_{ij}$. We define

$$\Delta(X) := \sum_{i=1}^{n} |x_{i.} - r_i| + \sum_{j=1}^{n} |x_{.j} - s_j|$$

as the total discrepancy from the desired row and column sums.

Lemma 1. *Suppose we have a set of values α_i, β_j and a set of integer values x_{ij} with*

$$\log a_{ij} + \alpha_i + \beta_j - 1 \leq \log x_{ij} \leq \log a_{ij} + \alpha_i + \beta_j \tag{11}$$

and $\Delta(X) > 0$. Then, in $O(n^2)$ time, we can find a new solution where Δ is decreased by 2, or we establish that no such solution exists, by solving a single-source shortest path problem. With the new solution, we also find new values α_i, β_j such that (11) is maintained.

Proof. (cf. [3, Lemma 4.16]) Suppose w.l.o.g. that there is a row i in which $x_{i.} < r_i$ holds. (The case $x_{i.} > r_i$ is similar.) We compute a shortest path in the residual network, starting from the row vertex R_i and ending at any column vertex S_j whose in-flow is smaller than the demand s_j, using the non-negative reduced costs $\bar{c}^t_{ij} = c^t_{ij} - \alpha_i - \beta_j$ and $\bar{c}^t_{ji} = -c^t_{ij} + \alpha_i + \beta_j$. Such a shortest path can be computed in $O(n^2)$ time by Dijkstra's algorithm. The shortest distance labels can be used to update the dual variables α_i, β_j such that (11) is maintained. If no path is found, it means that there is no possibility to increase $x_{i.}$, and therefore the problem is infeasible. $\square$

The total desired flow is h. Incrementing the flow in steps of 1 unit leads to h shortest path computations. Thus we get:

Theorem 2. *The integer matrix scaling problem can be solved in $O(n^2h)$ steps.* $\square$

3 Scaling the Data

The pseudo-polynomial complexity of Theorem 2 can be improved to a polynomial complexity, by solving a sequence of more and more refined discretizations. For an integer $k \geq 0$ we define the modified problem P^k with $\hat{r}_i := \lfloor r_i/2^k \rfloor$ and $\hat{s}_j := \lfloor c_j/2^k \rfloor$. Since $\sum \hat{r}_i = \sum \hat{s}_j$ is no longer guaranteed, we have to modify the concept of a solution for such a problem instance.[7] A *solution* x_{ij} with $x_{ij} = [a_{ij}\lambda_i\mu_j]$ must satisfy

$$\sum_{j=1}^{n} x_{ij} \leq r_i \quad \text{and} \quad \sum_{i=1}^{n} x_{ij} \leq s_j, \tag{12}$$

while achieving the maximum value of $\sum_{i=1}^n \sum_{j=1}^n x_{ij}$ (i.e., the minimum discrepancy) under the constraints (12).

Lemma 2. *Suppose that the original problem P^0 is feasible. Let λ_i, μ_j and $x_{ij} = [a_{ij}\lambda_i\mu_j]$ be a solution to a problem instance P^k ($k \geq 1$). Then the values $x'_{ij} := [2a_{ij}\lambda_i\mu_j]$ satisfy the row and column sum requirements for the problem instance P^{k-1} with a discrepancy $\Delta(X') \leq 6n^2$.*

Proof. First we show that $\Delta(X) \leq 2n^2$. Let (x^0_{ij}) be a solution to the original problem. Then $\hat{x}_{ij} := \lfloor x^0_{ij}/2^k \rfloor$ satisfies (12) for problem P^k, and an easy calculation shows that $\Delta(\hat{X}) \leq 2n^2$. Therefore, the solution X has $\Delta(X) \leq 2n^2$.

[7] Even if $\sum \hat{r}_i = \sum \hat{s}_j$, a solution of P^k in the sense of (3) might not exist, if A contains zeros.

It is easy to check that $x'_{ij} = 2x_{ij}$ or $x'_{ij} = 2x_{ij} + 1$. (If there is a tie, we can choose x_{ij} appropriately.) The new row sum requirements are $\hat{r}'_i = 2\hat{r}_i$ or $\hat{r}'_i = 2\hat{r}_i + 1$, and similarly for the column sums. Thus, the discrepancy in each row is twice the old discrepancy plus at most n. For all rows and columns, this adds up to $\Delta(X') \le 2\Delta(X) + 2n^2 \le 6n^2$. $\square$

We thus solve the sequence P^k, P^{k-1}, ... of more and more refined problems until the original problem P^0 is solved. We multiply the input matrix A by a scalar such that the largest entry is $\frac{1}{2}$, without changing the problem. Then, for $k := 1 + \lceil \log_2 h \rceil$, we have $\hat{r}_i = \hat{s}_j = 0$, and with $\lambda_i = \mu_j = 1$ (or $\alpha_i = \beta_j = 0$), we already have $x_{ij} = 0$ as a valid solution for P^k[8] Going from P^k to P^{k-1} incurs $O(n^2)$ shortest path computations, by Lemma 2: We first have to ensure that no row or column sum exceeds the desired value (12). This might involve at most $6n^2$ flow changes according to Lemma 1, and increment the discrepancy to $18n^2$. Then with at most $9n^2$ additional applications of Lemma 1, $\Delta(X)$ is decreased to the minimum. (If this minimum is bigger than $2n^2$, we can abort since, as proved in the proof of Lemma 2, there is no feasible solution for the original problem.) Thus, the transition from P^k to P^{k-1} takes $O(n^4)$ time, and we have:

Theorem 3. *The integer matrix scaling problem can be solved in $O(n^4 \log h)$ steps.* $\square$

4 Scaling of Real Matrices

The application to the approximate scaling of real matrices (2) is straightforward. We multiply all row and column sums r_i and s_j by the constant $F := \frac{n}{\varepsilon}$ and round them down to integers $\hat{r}_i$ and $\hat{s}_j$, respectively. We get the total sum $H \le hn/\varepsilon$. We solve the integer scaling problem for these data, using Theorem 2 or 3, in $O(n^2 H)$ or $O(n^4 \log H)$ time, respectively. In the end, we have to use a modification of Lemma 1 to get a solution in the sense of (12) in which the *maximum* discrepancy $\Delta_{\max}(X) := \max_{i,j}\{\, \hat{r}_i - x_{i.}, \hat{s}_j - x_{.j} \,\}$ is as small as possible. Taking the real-valued solution of (1), scaling it by F, and rounding everything down, we see that an integer solution with $\Delta_{\max} \le n - 1$ exists. Let us now estimate the error in (1), looking for example at row i:

$$\sum_{j=1}^{n} a_{ij}\lambda_i\mu_j - Fr_i = \left(\sum_{j=1}^{n} [a_{ij}\lambda_i\mu_j] - \hat{r}_i\right) + \sum_{j=1}^{n} \big(a_{ij}\lambda_i\mu_j - [a_{ij}\lambda_i\mu_j]\big) + (\hat{r}_i - Fr_i)$$

[8] We could have defined the scaled problems by rounding upward: $\hat{r}_i := \lceil r_i/2^k \rceil$, etc. Then the first problem $P^{1+\lceil \log_2 h \rceil}$ has all row and column sums $\hat{r}_i$ and $\hat{s}_j$ equal to 1. Incidentally, this problem is precisely equivalent to the problem that is solved as the preprocessing step in the first algorithm of Linial et al. [7, Theorem 3.1].

The first term in parentheses is between $-n+1$ and 0, by the bound on $\Delta_{\max}$ for the solution $x_{ij} = [a_{ij}\lambda_i\mu_j]$; the second sum is between 0 and n, and the last term is between -1 and 0. This accumulates to a total error between $-n$ and $+n$ in each row sum, and similarly in each column sum. In terms of the original (unscaled) data, this is an absolute error of at most ε.

Theorem 4. *The (real-valued) matrix scaling problem can be solved to an accuracy ε in $O(n^3\frac{h}{\varepsilon})$ or in $O(n^4(\log n + \log\frac{h}{\varepsilon}))$ time.* $\square$

The algorithms are simple, and they beat the best previous algorithms of Linial et al. [7] by several orders of magnitude.[9] Only for the special case of doubly stochastic scaling (where $h = n$) and for moderate error bounds ε, the $O(n^3 \log n/\varepsilon^2)$ algorithm in [7], which is just Sinkhorn scaling with a special preprocessing step, is faster. It might be interesting to investigate hybrid algorithms that combine different approaches in various phases.

5 Numerical Accuracy

For the quantities related to the costs, we use fixed-point arithmetic with fixed precision, i. e. all numbers $\log a_{ij}$, $\log x_{ij}$, α_i and β_j are approximately represented as integer multiples of some small value δ, with an absolute error of $O(\delta)$. This means that, after approximating the "input data", shortest paths are computed exactly, as they involve only additions and comparisons.

A nice feature of the flow problem (6–9) is that we can use Theorem 1 without propagating errors. We can take *arbitrary* values α_i and β_j and define x_{ij} by $x_{ij} := \lfloor a_{ij} \cdot e^{\alpha_i} e^{\beta_j} \rfloor$ or more precisely, by the relation (11), These values can be used as a starting solution for Lemma 1.

Let us analyze what happens if (11) is only satisfied approximately. It is better to rewrite (11) as

$$\log x_{ij} \le \log a_{ij} + \alpha_i + \beta_j \le \log(x_{ij}+1) \tag{13}$$

to make it clear that $\log(x_{ij}+1)$ is a different (approximate) calculation from $\log x_{ij}$ (but all computations of the same number $\log x$ yield the same approximate result). Conceptually we locate the value $\log a_{ij} + \alpha_i + \beta_j$ in the sorted list of approximate values $\log 1$, $\log 2$, $\log 3$, ... To be specific, let us use natural logarithms in this section unless otherwise stated. Suppose that all values x_{ij} that occur in the computation are bounded by some bound M. Then, if we calculate $\log x_{ij}$ with an absolute error $\le 0.1/M$ we

[9] It must be conceded, however, that Linial et al. [7] were apparently satisfied with achieving, for the first time, a strongly polynomial running time for the problem, and did not make a great effort to optimize the running times for their algorithms.

may incur an error of approximately ± 0.1 in terms of x_{ij}. In other words, (11) holds for some *approximation* $\tilde{x}_{ij}$ instead of the integer value x_{ij}, with $|\tilde{x}_{ij} - x_{ij}| \le e^{0.1} < 0.11$. This has the effect that, in the "optimum" solution that we compute, (11) will hold with an additional error term of ± 0.11 on both sides. If $\log a_{ij}$ is also computed with an error of at most ± 0.1, we get that (10) is satisfied with an error term of ± 0.23 on both sides. This is good enough for achieving an ε-approximation of the real-valued matrix scaling problem. The scaling factor F in the proof of Theorem 4 must be adjusted a little, but the asymptotic runtime is unaffected.

In Lemma 2, the discrepancy after using the solution of P^k as the initial solution for P^{k-1} may be larger than $6n^2$ but it is still $O(n^2)$. Again, this does not affect the asymptotic running time.

The values x_{ij} that occur during the course of the computation are bounded by $M := H + O(n^2) = hn/\varepsilon + O(n^2)$. We set $\delta = 0.1/M$, and this means that we need $\log_2 \frac{h}{\varepsilon} + 2\log_2 n + O(1)$ bits for the fraction.

To estimate the maximum size of the numbers, one has to observe that, during the algorithm, the new values of α_i and β_j are always determined by setting some of the inequalities (13) to equations (corresponding to arcs of zero reduced cost in the residual network). Each value α_i or β_j is connected to a "free" value α_i or β_j by such a chain of equations, going over at most $2n$ arcs. If we take care that the "free" value is the initial value of 0, we can ensure that the maximum absolute value of the numbers α_i and β_j is bounded by $2n \cdot (\log M + \log \max_{a_{i,j} \neq 0} |\log a_{ij}|)$. As in the proof leading to Theorem 3, we can scale the input matrix A such that the largest entry is $\frac{1}{2}$. Thus we need at most $1 + \log_2 n + \log_2 \log M + \log_2 \log(2L(A))$ bits to represent the integer part of the numbers α_i and β_j, where $L(A)$ is the ratio between the largest and the smallest positive entry in A. Thus, we have:

Theorem 5. *For achieving Theorem 4, arithmetic operations on numbers with $O(\log n + \log\log L(A) + \log \frac{h}{\varepsilon})$ bits are sufficient, where $L(A)$ is the ratio between the largest and the smallest positive entry in the input matrix.* $\square$

For example, if $L(A) \le 10^{10}$, $\varepsilon = 10^{-6}h$, and $n \le 1000$, the algorithm can work with IEEE double-precision floating point numbers.[10]

An obvious alternative is to eliminate the logarithms altogether, to use directly the exponentiated variables λ_i and μ_j instead of α_i and β_j, and to work with a multiplicative version of Dijkstra's algorithm. However, this would incur a propagation of errors in the shortest path calculations, which needs to be analyzed.

[10] The reader may wonder why only $O(\log\log L(A))$ bits are needed when one must, of course, *look* at the $\log L(A)$ bits of the input numbers a_{ij} in the first place. But since we are interested in an approximate solution, excessive trailing bits of very long input numbers can be ignored.

The approximate calculations discussed in this section are of course not appropriate for carrying out an election (Theorems 2 and 3), where the tiniest fraction can be important to decide about one seat. We are planning to investigate the necessary precision to compute an exact result in a future work.

References

[1] Michel Balinski and Gabrielle Demange. Algorithms for proportional matrices in reals and integers. *Math. Programming*, 45:193–210, 1989.

[2] Michel Balinski and Svetlozar T. Rachev. Rounding proportions: methods of rounding. *Math. Scientist*, 22:1–26, 1997.

[3] William R. Cook, William H. Cunningham, William R. Pulleyblank, and Alexander Schrijver. *Combinatorial Optimization*. Wiley, 1998.

[4] Martin Fürer. Quadratic convergence for scaling of matrices. In Lars Arge, Giuseppe F. Italiano, and Robert Sedgewick, editors, *Proc. ALENEX/ANALC 2004, 6th Workshop on Algorithm Engineering and Experiments and 1st Workshop on Analytic Algorithmics and Combinatorics, New Orleans*, pages 216–223. SIAM, January 2004.

[5] Norbert Gaffke and Friedrich Pukelsheim. Divisor methods for proportional representation systems: an optimization approach to vector and matrix problems. Technical Report Preprint 06-18, Universität Magdeburg, Fakultät für Mathematik, March 2006. http://www.math.uni-magdeburg.de/preprints/shadows/06-18report.html.

[6] B. Kalantari and L. Khachian. On the complexity of nonnegative matrix scaling. *Linear Algebra Appl.*, 240:87–104, 1996.

[7] Nathan Linial, Alex Samorodnitsky, and Avi Wigderson. A deterministic strongly polynomial algorithm for matrix scaling and approximate permanents. *Combinatorica*, 20(4):545–568, 2000.

[8] U. Rothblum and H. Schneider. Scaling of matrices which have prespecified row sums and column sums via optimization. *Linear Algebra Appl.*, 114–115:737–764, 1989.

[9] Richard Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *Ann. Math. Statist.*, 35:876–879, 1964.

[10] George W. Soules. The rate of convergence of Sinkhorn balancing. *Linear Algebra Appl.*, 150:3–40, 1991.