INSTITUT DE FRANCE
Académie des sciences

Comptes Rendus
Mathématique

Edward Chlebus and Viswatej Kasapu

An Entropy Optimizing *RAS*-Equivalent Algorithm for Iterative Matrix Balancing

Volume 361 (2023), p. 737-746

Published online: 11 May 2023

https://doi.org/10.5802/crmath.398

CENTRE MERSENNE

Les Comptes Rendus. Mathématique *sont membres du*
Centre Mersenne pour l'édition scientifique ouverte
www.centre-mersenne.org
e-ISSN : 1778-3569

Comptes Rendus
Mathématique
2023, Vol. 361, p. 737-746
https://doi.org/10.5802/crmath.398

Algebra, Control theory / *Algèbre, Théorie du contrôle*

An Entropy Optimizing *RAS*-Equivalent Algorithm for Iterative Matrix Balancing

Edward Chlebus *, [a] **and Viswatej Kasapu** [a]

[a] Department of Computer Science, Illinois Institute of Technology, 10 W. 31st St., Chicago, Illinois 60616, USA

E-mails: chlebus@iit.edu (Edward Chlebus), vkasapu@hawk.iit.edu (Viswatej Kasapu)

Abstract. We have developed a new simple iterative algorithm to determine entries of a normalized matrix given its marginal probabilities. Our method has been successfully used to obtain two different solutions by maximizing the entropy of a desired matrix and by minimizing its Kullback–Leibler divergence from the initial probability distribution. The latter is fully equivalent to the well-known RAS balancing algorithm. The presented method has been evaluated using a traffic matrix of the GÉANT pan-European network and randomly generated matrices of various sparsities. It turns out to be computationally faster than RAS. We have shown that our approach is suitable for efficient balancing both dense and sparse matrices.

2020 Mathematics Subject Classification. 65F35, 65F50, 65K05, 90B20, 94A17.

Manuscript received 28 September 2021, revised 3 February 2022 and 2 June 2022, accepted 12 June 2022.

1. Introduction

We would like to find all elements a_{ij} of the nonnegative $m \times n$ matrix $\mathbf{A}$ whose row and column sums $r_i = \sum_{j=1}^{n} a_{ij}, 1 \le i \le m$ and $c_j = \sum_{i=1}^{m} a_{ij}, 1 \le j \le n$, respectively, are given. This problem known as matrix balancing occurs in economics, demography, statistics and stochastic modeling. Matrix balancing is applicable to input–output analysis, supply and use tables, social accounting matrices, urban and transportation planning, classifying items in contingency tables, determining transition probabilities, estimating interregional migration, and modeling origin–destination flows [15]. Assuming each $a_{ij} > 0$ the trivial solution to this problem is given by $a_{ij} = p_{r_i} p_{c_j} T$, where $p_{r_i} = r_i / T$ and $p_{c_j} = c_j / T$ are the row and column sums normalized with respect to the total sum of all the matrix elements $T = \sum_{i=1}^{m} r_i = \sum_{j=1}^{n} c_j$. A normalized matrix $\mathbf{P} = \mathbf{A}/T$ with entries $p_{ij} = p_{r_i} p_{c_j}$ can be interpreted as a certain probability distribution. Unfortunately this solution doesn't work if $\mathbf{A}$ (and consequently $\mathbf{P}$) has zero elements which is common in some practical applications. For example, $p_{ij} = a_{ij} = 0$ if the original matrix $\mathbf{A}$ represents network traffic and there is no flow from node i to node j.

* Corresponding author.

2. The RAS algorithm

In such a case, in order to find **P** whose structure (i.e. positions of all zero and nonzero elements) is known, we can use the well-known RAS algorithm (see, e.g. [1,4,7,8,10,11,13–15] and references therein). It iteratively employs proportional rescaling of all rows and columns of an initial matrix **Q** with entries q_{ij} until all the marginal probabilities p_{r_i} and p_{c_j} are satisfactorily matched. Here are the subsequent steps of this algorithm [15]:

Algorithm 1 (RAS).

(**Step 1**) *Set the initial conditions* $k = 0$ *and* $\mathbf{Q}^{(0)} = \mathbf{Q}$.

(**Step 2**) *Increase the iteration index* $k \leftarrow k+1$.

(**Step 3**) *For the* i^{th} *row,* $1 \le i \le m$*, find* $\alpha_i = p_{r_i} / \sum_{j=1}^{n} q_{ij}^{(k-1)}$ *and rescale each element of this row*

$$q_{ij}^{(k-1)} \leftarrow \alpha_i q_{ij}^{(k-1)}, 1 \le j \le n.$$

(**Step 4**) *For the* j^{th} *column,* $1 \le j \le n$*, find* $\beta_j = p_{c_j} / \sum_{i=1}^{m} q_{ij}^{(k-1)}$ *and rescale each element of this column*

$$q_{ij}^{(k)} = \beta_j q_{ij}^{(k-1)}, 1 \le i \le m.$$

(**Step 5**) *If the convergence criterion*

$$\left| \frac{p_{r_i} - \sum_{j=1}^{n} q_{ij}^{(k)}}{p_{r_i}} \right| < \epsilon, 1 \le i \le m,$$

*is met then go to **Step 6**, else go to **Step 2**.*

(**Step 6**) $\mathbf{P} = \mathbf{Q}^{(k)}$.

RAS is applicable to balancing not only dense matrices but sparse ones with zero elements as well. This algorithm is very intuitive and easy to implement but it is often used blindly. It simply is not obvious that RAS minimizes the Kullback–Leibler divergence (see the proof in [2,3,9])

$$D(\mathbf{P}||\mathbf{Q}) = \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} \log \frac{p_{ij}}{q_{ij}}, \tag{1}$$

hence most papers reporting successful RAS application completely disregard this important property. Previous attempts to relate RAS to entropy optimization were made in, e.g. [10,12–14]. As a consequence of minimizing (1), RAS results in **P** which is the least distinguishable from **Q** and satisfies the aforementioned constraints p_{r_i} and p_{c_j}. If this is our goal, RAS provides a perfect solution but what if there is no reason to favor any specific initial probability distribution **Q**?

3. Maximum entropy solution

Under such circumstances the uniform distribution $q_{ij} = \mathbf{1}(a_{ij} > 0)/k$, where $k = \sum_{i=1}^{m} \sum_{j=1}^{n} \mathbf{1}(a_{ij} > 0)$ is the number of positive entries in matrix **A** counted with the indicator function $\mathbf{1}(\circ)$, seems a natural and reasonable choice for **Q**. This simplifies formula (1) to

$$D(\mathbf{P}||\mathbf{Q}) = \log k + \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} \log p_{ij}. \tag{2}$$

Since k is a constant, minimizing the Kullback–Leibler divergence (2) is equivalent to maximizing the entropy of **P**

$$H(\mathbf{P}) = -\sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} \log p_{ij}. \tag{3}$$

Let's introduce a vector of Lagrange multipliers $\boldsymbol{\lambda}$ and set partial derivatives of the Lagrangian function

$$L(\mathbf{P},\boldsymbol{\lambda}) = -\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}\log p_{ij} + \sum_{i=1}^{m}\lambda_{r_i}\left(\sum_{j=1}^{n} p_{ij} - p_{r_i}\right) + \sum_{j=1}^{n}\lambda_{c_j}\left(\sum_{i=1}^{m} p_{ij} - p_{c_j}\right) \tag{4}$$

to zeros for all positive entries $p_{ij} > 0$, the first of which in the i^{th} row is denoted by p_{ii_c}, $1 \le i \le m$

$$\frac{\partial L(\mathbf{P},\boldsymbol{\lambda})}{\partial p_{ii_c}} = -\log p_{ii_c} - 1 + \lambda_{r_i} + \lambda_{c_{i_c}} = 0 \tag{5}$$

$$\frac{\partial L(\mathbf{P},\boldsymbol{\lambda})}{\partial p_{ij}} = -\log p_{ij} - 1 + \lambda_{r_i} + \lambda_{c_j} = 0, \qquad i_c < j \le n. \tag{6}$$

Subtracting (6) from (5) and using the natural log with base e, we get $p_{ij} = p_{ii_c}\exp(\lambda_{c_j} - \lambda_{c_{i_c}})$ which along with the marginal value $\sum_{j=i_c}^{n} p_{ij} = p_{r_i}$ gives a formula for nonzero entries of $\mathbf{P}$

$$p_{ij} = p_{r_i}\frac{\exp\left(\lambda_{c_j}\right)}{\sum\limits_{p_{il}\neq 0,\, 1\le l\le n}\exp\left(\lambda_{c_l}\right)}. \tag{7}$$

It follows immediately from the requirement $\sum_{i=1}^{m} p_{ij} = p_{c_j}$ imposed on all the probabilities of column j that

$$\exp\left(\lambda_{c_j}\right) = \frac{p_{c_j}}{\sum\limits_{p_{ij}\neq 0,\, 1\le i\le m}\dfrac{p_{r_i}}{\sum\limits_{p_{il}\neq 0,\, 1\le l\le n}\exp\left(\lambda_{c_l}\right)}}. \tag{8}$$

We can obtain an analogous solution if we follow a columnwise approach instead of a rowwise one ((5)–(6)).

4. Iterative algorithm to determine the maximum entropy matrix P

It's obvious from (8) that variables $\exp(\lambda_{c_j})$, $1 \le j \le n$, can be determined as a function of themselves, hence the following iterative algorithm to maximize the entropy $H(\mathbf{P})$ (3) is straightforward:

Algorithm 2.

(**Step 1**) *Set the initial conditions* $k = 0$ *and* $\exp(\lambda_{c_j}^{(0)})$, $1 \le j \le n$.

(**Step 2**) *Increase the iteration index* $k \leftarrow k+1$ *and use formula* (8) *to determine new numerical values of* $\exp(\lambda_{c_j}^{(k)})$, $1 \le j \le n$.

(**Step 3**) *If the convergence criterion*

$$\left|\frac{\exp\left(\lambda_{c_j}^{(k)}\right) - \exp\left(\lambda_{c_j}^{(k-1)}\right)}{\exp\left(\lambda_{c_j}^{(k)}\right)}\right| < \epsilon, 1 \le j \le n,$$

is met then go to ***Step 4****, else go to* ***Step 2****.*

(**Step 4**) *Use formula* (7) *to find desired numerical values of the nonzero matrix entries* p_{ij}, $1 \le i \le m$, $1 \le j \le n$.

5. A RAS-equivalent algorithm

Let's systematically analyze the RAS algorithm. As we already know it minimizes the Kullback–Leibler divergence (1) between the resulting matrix $\mathbf{P}$ and the initial one $\mathbf{Q}$. The corresponding Lagrangian has the form

$$L(\mathbf{P},\boldsymbol{\lambda}) = -\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}\log\frac{p_{ij}}{q_{ij}} + \sum_{i=1}^{m}\lambda_{r_i}\left(\sum_{j=1}^{n} p_{ij} - p_{r_i}\right) + \sum_{j=1}^{n}\lambda_{c_j}\left(\sum_{i=1}^{m} p_{ij} - p_{c_j}\right). \tag{9}$$

Using the approach identical to that presented in Section 3 we get formulas

$$p_{ij} = p_{r_i} \frac{q_{ij} \exp\left(\lambda_{c_j}\right)}{\sum\limits_{p_{il} \neq 0, 1 \leq l \leq n} q_{il} \exp\left(\lambda_{c_l}\right)} \tag{10}$$

and

$$\exp\left(\lambda_{c_j}\right) = \frac{p_{c_j}}{\sum\limits_{p_{ij} \neq 0, 1 \leq i \leq m} \frac{p_{r_i} q_{ij}}{\sum\limits_{p_{il} \neq 0, 1 \leq l \leq n} q_{il} \exp\left(\lambda_{c_l}\right)}} \tag{11}$$

analogous to (7) and (8), respectively, but dependent on the initial probability distribution **Q**. Hence, *Algorithm 2* with ((10)–(11)) replacing ((7)–(8)) is fully equivalent to RAS.

It is clear from equations ((7)–(8)) and ((10)–(11)) that in order to implement and execute *Algorithm 2* we need the initial values of $\exp(\lambda_{c_j}^{(0)})$, $1 \leq j \leq n$, successfully set to p_{c_j} in **Step 1** of all our computations, and positions of all zero (if any) and nonzero elements of the matrix. We assume that its structure is known. Thus, the introduced *Algorithm 2* is easy to implement, and applicable to balancing both dense and sparse matrices.

6. Performance comparison of the matrix balancing algorithms

First we compared the performance of our new algorithm and RAS using a traffic matrix of the GÉANT network [5] interconnecting Europe's national research and education networks with a high-speed backbone. GÉANT, cofunded by the European Commision, is a result of a collaboration of over 40 partners. Currently it serves 50 million users in over 10,000 organizations.

Topology of the GÉANT pan-European high-bandwidth backbone network evolves over time. It consisted of 31 nodes interconnected with 96 links over a two-year period from June 1st, 2018 to May 31st, 2020 when we collected traffic measurements. The corresponding 31 × 31 sparse matrix of 961 entries with only 96 nonzero link loads was investigated. We gathered link load statistics accessible online with the public Cacti tool [6].

For numerical experiments reported in this paper we selected GÉANT's traffic of January 15$^{\text{th}}$, 2020. The traffic measurements recorded for subsequent 5-minute intervals determine 288 daily traffic matrices $\mathbf{M}_k$, $1 \leq k \leq 288$. We tried to balance each of them given the marginal probabilities p_{r_i} and p_{c_j}, $1 \leq i, j \leq 31$, obtained by normalizing the measurements r_i and c_j of total original and transit traffic outgoing from node i and incoming to node j, respectively.

We started the performance analysis by comparing RAS (*Algorithm 1*) with our new *Algorithm 2*, both implemented in Python. As we pointed out in Sections 2 and 3, these algorithms optimize different entropy-based criteria but if our primary objective is to balance a matrix, such a comparison makes sense. We used the modified gravity model [16]

$$b_{ij} = r_i \frac{c_j}{\sum\limits_{p_{il} \neq 0, 1 \leq l \leq 31} c_l}, \qquad 1 \leq i, j \leq 31, p_{ij} \neq 0 \tag{12}$$

to determine nonzero elements $q_{ij} = b_{ij}/T$ of the initial matrix **Q** required in **Step 1** of *Algorithm 1* (RAS). They were normalized with respect to $T = \sum_{i=1}^{31} r_i = \sum_{j=1}^{31} c_j$. We examined running times of both the investigated methods for all daily traffic matrices $\mathbf{M}_k$, $1 \leq k \leq 288$. Our proposed *Algorithm 2* was on average 14.4495 times (standard deviation $\sigma = 0.9732$) faster than RAS.

When the nonzero elements of the initial matrix **Q** of *Algorithm 1* (RAS) were determined by the uniform distribution which (as we know from Sections 3 and 4) makes RAS and *Algorithm 2* fully equivalent with respect to the maximized optimization criterion (3), the execution time of RAS was shortened on average 14.4136 times (standard deviation $\sigma = 1.7069$).

Our top objective was a comprehensive performance evaluation of two different implementations minimizing the Kullback–Leibler divergence (1). Therefore, RAS (*Algorithm 1*) was compared with a modified version of *Algorithm 2* using equations ((10)–(11)) instead of ((7)–(8)). The initial matrix $\mathbf{Q}_k$, $1 \le k \le 288$, identical for both algorithms, was randomly generated for each of the 288 daily traffic matrices $\mathbf{M}_k$. RAS and *Algorithm 2* were implemented in three programming languages: C++, Java and Python, and run 100 times for each pair of matrices $(\mathbf{Q}_k, \mathbf{M}_k)$ to average an impact of background processes on the measured execution times t_{RAS} and t_{A2} required by both the examined methods to produce the balanced matrix $\mathbf{P}_k$. Statistics of $288 \times 100 = 28800$ samples of t_{RAS} and t_{A2} collected for each of three numerical values 10^{-6}, 10^{-9} and 10^{-12} of the convergence criterion ϵ (cf. **Step 5** of *Algorithm 1* and **Step 3** of *Algorithm 2*) are given in Tables 1–3.

Java and Python provide the fastest and slowest of all the observed executions of the examined algorithms taking on average 64.179 μs and 569.35 ms, respectively. The measured running times differ by 2–3 orders of magnitude for each algorithm implemented in both these programming languages. Performance of C++ is also inferior to that of Java but the difference between them isn't that significant. It's obvious that decreasing the convergence criterion ϵ increases the running times t_{RAS} and t_{A2}. The mean ratio t_{RAS}/t_{A2} of their corresponding measurements ranges from around 5 for Java to around 10 for C++ and Python, as is clearly seen from Table 3. The introduced *Algorithm 2* is always faster than RAS and consistently beats it.

Both the investigated algorithms produce exact (i.e. not approximate) results but they employ iteration so numerical values of the convergence criterion ϵ affect computational accuracy of obtained 96 nonzero probabilities $p_{ij}^{(k)}$ (10), $1 \le i, j \le 31$, of the balanced matrix $\mathbf{P}_k$ corresponding to each pair $(\mathbf{Q}_k, \mathbf{M}_k)$, $1 \le k \le 288$. To comparatively evaluate this accuracy we found out the relative percentage difference

$$d_{ij}^{(k)} = \frac{\left| p_{ij}^{(k,RAS)} - p_{ij}^{(k,A2)} \right|}{\frac{p_{ij}^{(k,RAS)} + p_{ij}^{(k,A2)}}{2}} \times 100\% \qquad 1 \le i, j \le 31, 1 \le k \le 288, p_{ij}^{(k)} \ne 0 \tag{13}$$

between elements of each pair of corresponding nonzero probabilities $(p_{ij}^{(k,RAS)}, p_{ij}^{(k,A2)})$ obtained by applying RAS and *Algorithm 2*, respectively. Our C++, Java and Python programs performed all the calculations with double precision.

Table 4 shows statistics of d (13) based on $288 \times 96 = 27648$ available samples. All the relative percentage differences reported for *Algorithm 2* and RAS are absolutely negligible. The highest numerical value of $d = 0.0963\%$ was obtained for C++ and $\epsilon = 10^{-6}$. The more stringent convergence criterion ϵ, the lower the values of d for all the programming implementations of the algorithms, as expected. Setting $\epsilon = 10^{-6}$ seems absolutely satisfactory in practice but this parameter can be easily adjusted if the results of super high numerical accuracy are required.

The studied 31-node GÉANT pan-European backbone network has only 96 links, which is why the sparsity of the corresponding 31×31 traffic matrix $s = (961 - 96)/961 = 90.01\%$ is very high. In order to show the versatility of our iterative algorithm we have applied the investigation methodology, similar to that successfully used to produce the results reported in Tables 1–4, to less sparse and fully dense matrices as well. For each of the five selected numerical values of the matrix sparsity $s \in \{0\%, 10\%, 40\%, 70\%, 90\%\}$ we have randomly generated 100 pairs of 100×100 matrices $(\mathbf{Q}_k(s), \mathbf{M}_k(s))$, $1 \le k \le 100$, each of which has $s \times 100 \times 100$ zero elements. $s = 0$ implies a fully dense matrix with no zero elements. The corresponding statistics of the running times t_{RAS} and t_{A2}, their ratios t_{RAS}/t_{A2}, and the relative percentage differences d determined by (13) (with modified ranges of the indices $1 \le i, j \le 100$ and $1 \le k \le 100$) are presented in Tables 5–8. Each resulting matrix $\mathbf{P}_k(s)$ determined by $(\mathbf{Q}_k(s), \mathbf{M}_k(s))$, $1 \le k \le 100$, was balanced 100 times to get a statistically significant sample of 10000 execution times t_{RAS} and t_{A2} for the given numerical value of matrix sparsity s.

Table 7 clearly shows that RAS is comparable with *Algorithm 2* in terms of execution time for small values of the matrix sparsity s only. As s increases and ϵ decreases the superiority of *Algorithm 2* is more and more visible which means that our approach is especially suitable for balancing very sparse matrices with high requiremens for numerical accuracy of the results. The statistics of d reported in Table 8 reinforce the conclusions we have already drawn above from the analysis of the analogous results given in Table 4 for the GÉANT network.

7. Conclusions

In this paper we revisited the well-known RAS matrix balancing algorithm. We reexamined its implicit and usually completely omitted property, i.e. the entropy-based minimization criterion (1) which we optimized by using the Lagrange multipliers. This systematic approach results in a surprisingly simple iterative scheme for the variables $\exp(\lambda_{c_j})$ (or $\exp(\lambda_{r_i})$) whose computed numerical values can be used to easily determine the desired balanced matrix $\mathbf{P}$ which satisfies the given marginal constraints. We developed two versions of this iterative procedure, i.e. insensitive and sensitive to the initial matrix $\mathbf{Q}$. The latter is fully equivalent to RAS but superior to it in terms of analytical justification, mathematical formalism of derivations, and performance.

Execution times and numerical accuracy of the proposed algorithm were examined for 288 daily traffic patterns of the extremely sparse 31×31 traffic matrix (with $s = 90.01\%$) of the GÉANT pan-European backbone network, and 500 randomly generated 100×100 matrices of various sparsities $0\% \le s \le 90\%$. Our method is simple to implement regardless of the programming language used, and the size and sparsity of a matrix. We didn't experience any issues with setting the initial conditions or the convergence of the proposed algorithm. It works well for all the investigated matrices, both actual ones reflecting the existing network configuration and the measured traffic volume of GÉANT, and randomly generated ones for given input values of s. Our new iterative scheme outperformes RAS, as is seen from the outcomes of all the examined programming implementations, reducing its running time even by one order of magnitude for extremely sparse matrices without compromising numerical accuracy of the obtained results.

Table 1. *The GÉANT traffic matrix:* statistics of the execution times t_{RAS} (in seconds) for different programming implementations of RAS with various numerical values of the convergence criterion ϵ.

Language	ϵ	Range		Mean	Standard Deviation
		Min	Max		
C++	10^{-12}	1.6483×10^{-3}	8.2756×10^{-3}	2.4139×10^{-3}	8.6565×10^{-4}
	10^{-9}	1.1674×10^{-3}	5.9977×10^{-3}	1.7309×10^{-3}	6.3468×10^{-4}
	10^{-6}	5.5820×10^{-4}	3.7181×10^{-3}	1.0478×10^{-3}	4.0706×10^{-4}
Java	10^{-12}	4.6904×10^{-4}	2.4578×10^{-3}	7.5155×10^{-4}	3.0896×10^{-4}
	10^{-9}	3.3710×10^{-4}	1.7765×10^{-3}	5.3908×10^{-4}	2.2526×10^{-4}
	10^{-6}	1.6987×10^{-4}	1.1034×10^{-3}	3.2666×10^{-4}	1.4263×10^{-4}
Python	10^{-12}	3.8974×10^{-1}	1.9744	5.6935×10^{-1}	2.0856×10^{-1}
	10^{-9}	2.7951×10^{-1}	1.4271	4.0842×10^{-1}	1.5262×10^{-1}
	10^{-6}	1.3426×10^{-1}	8.8661×10^{-1}	2.4802×10^{-1}	9.7824×10^{-2}

Table 2. *The GÉANT traffic matrix:* statistics of the execution times t_{A2} (in seconds) for different programming implementations of *Algorithm 2* with various numerical values of the convergence criterion ϵ.

Language	ϵ	Range		Mean	Standard Deviation
		Min	Max		
C++	10^{-12}	1.5740×10^{-4}	7.3414×10^{-4}	2.2351×10^{-4}	7.0959×10^{-5}
	10^{-9}	1.1809×10^{-4}	5.4038×10^{-4}	1.6733×10^{-4}	5.2127×10^{-5}
	10^{-6}	7.6277×10^{-5}	3.4324×10^{-4}	1.1125×10^{-4}	3.3370×10^{-5}
Java	10^{-12}	6.9929×10^{-5}	4.8373×10^{-4}	1.3970×10^{-4}	5.5130×10^{-5}
	10^{-9}	5.1776×10^{-5}	3.5363×10^{-4}	1.0190×10^{-4}	4.0597×10^{-5}
	10^{-6}	3.3284×10^{-5}	2.2504×10^{-4}	6.4179×10^{-5}	2.6400×10^{-5}
Python	10^{-12}	3.6308×10^{-2}	1.8120×10^{-1}	5.4158×10^{-2}	1.8964×10^{-2}
	10^{-9}	2.6078×10^{-2}	1.3250×10^{-1}	3.9406×10^{-2}	1.3931×10^{-2}
	10^{-6}	1.4812×10^{-2}	8.2768×10^{-2}	2.4724×10^{-2}	8.9330×10^{-3}

Table 3. *The GÉANT traffic matrix:* statistics of the ratios of the execution times t_{RAS}/t_{A2} for different programming implementations of RAS and *Algorithm 2* with various numerical values of the convergence criterion ϵ.

Language	ϵ	Range		Mean	Standard Deviation
		Min	Max		
C++	10^{-12}	9.1789	12.8413	10.7190	0.5459
	10^{-9}	8.1509	13.0099	10.2450	0.6804
	10^{-6}	6.0654	12.3235	9.2831	0.9308
Java	10^{-12}	3.0040	22.2358	5.4858	1.5778
	10^{-9}	2.8759	22.0625	5.3923	1.5542
	10^{-6}	2.5178	21.3290	5.1934	1.5432
Python	10^{-12}	8.6863	13.1169	10.4935	0.4984
	10^{-9}	7.8254	14.2760	10.3420	0.6362
	10^{-6}	6.7682	13.8256	9.9964	0.9072

Table 4. *The GÉANT traffic matrix:* statistics of the relative percentage differences d (13) for different programming implementations of RAS and *Algorithm 2* with various numerical values of the convergence criterion ϵ.

Language	ϵ	Range		Mean	Standard Deviation
		Min	Max		
C++	10^{-12}	0	0	0	0
	10^{-9}	0	0	0	0
	10^{-6}	0	9.63×10^{-2}	6.74×10^{-5}	1.58×10^{-3}
Java	10^{-12}	0	3.83×10^{-9}	7.84×10^{-11}	2.39×10^{-10}
	10^{-9}	0	3.84×10^{-6}	7.88×10^{-8}	2.39×10^{-7}
	10^{-6}	3.42×10^{-14}	3.84×10^{-3}	7.81×10^{-5}	2.30×10^{-4}
Python	10^{-12}	0	3.83×10^{-9}	7.85×10^{-11}	2.39×10^{-10}
	10^{-9}	0	3.84×10^{-6}	7.88×10^{-8}	2.39×10^{-7}
	10^{-6}	5.12×10^{-14}	3.84×10^{-3}	7.81×10^{-5}	2.30×10^{-4}

Table 5. *The randomly generated* 100 × 100 *matrices of different sparsities:* statistics of the sample of 10000 execution times t_{RAS} (in seconds) for the Python programming implementation of RAS with various numerical values of the convergence criterion ϵ.

Sparsity s	ϵ	Range		Mean	Standard Deviation
		Min	Max		
0%	10^{-12}	1.1263×10^{-1}	1.2662×10^{-1}	1.2085×10^{-1}	4.6746×10^{-3}
	10^{-9}	8.9145×10^{-2}	1.0591×10^{-1}	9.6370×10^{-2}	4.6578×10^{-3}
	10^{-6}	5.9771×10^{-2}	7.9746×10^{-2}	6.7061×10^{-2}	5.6219×10^{-3}
10%	10^{-12}	1.2549×10^{-1}	1.4411×10^{-1}	1.3392×10^{-1}	4.6912×10^{-3}
	10^{-9}	1.0113×10^{-1}	1.2134×10^{-1}	1.0687×10^{-1}	4.8764×10^{-3}
	10^{-6}	7.4044×10^{-2}	8.1293×10^{-2}	7.6455×10^{-2}	1.2706×10^{-3}
40%	10^{-12}	1.7242×10^{-1}	1.9315×10^{-1}	1.8136×10^{-1}	5.0508×10^{-3}
	10^{-9}	1.3174×10^{-1}	1.6175×10^{-1}	1.4432×10^{-1}	5.9942×10^{-3}
	10^{-6}	8.6096×10^{-2}	1.0540×10^{-1}	9.8039×10^{-2}	5.6630×10^{-3}
70%	10^{-12}	2.4456×10^{-1}	3.1806×10^{-1}	2.6372×10^{-1}	9.7593×10^{-3}
	10^{-9}	1.8327×10^{-1}	2.0532×10^{-1}	1.9343×10^{-1}	5.7941×10^{-3}
	10^{-6}	1.2376×10^{-1}	1.4914×10^{-1}	1.3550×10^{-1}	6.5494×10^{-3}
90%	10^{-12}	5.2929×10^{-1}	9.6860×10^{-1}	6.1602×10^{-1}	5.2761×10^{-2}
	10^{-9}	3.8629×10^{-1}	6.9003×10^{-1}	4.5483×10^{-1}	3.7816×10^{-2}
	10^{-6}	2.5027×10^{-1}	4.8563×10^{-1}	3.0601×10^{-1}	2.8711×10^{-2}

Table 6. *The randomly generated* 100 × 100 *matrices of different sparsities:* statistics of the sample of 10000 execution times t_{A2} (in seconds) for the Python programming implementation of *Algorithm 2* with various numerical values of the convergence criterion ϵ.

Sparsity s	ϵ	Range		Mean	Standard Deviation
		Min	Max		
0%	10^{-12}	8.9289×10^{-2}	9.7341×10^{-2}	9.0225×10^{-2}	8.5198×10^{-4}
	10^{-9}	7.4168×10^{-2}	9.2356×10^{-2}	7.9497×10^{-2}	5.0402×10^{-3}
	10^{-6}	6.5505×10^{-2}	7.8184×10^{-2}	7.0388×10^{-2}	1.8639×10^{-3}
10%	10^{-12}	8.1484×10^{-2}	9.8452×10^{-2}	8.8399×10^{-2}	4.5759×10^{-3}
	10^{-9}	7.8466×10^{-2}	9.2174×10^{-2}	8.2270×10^{-2}	2.1960×10^{-3}
	10^{-6}	6.2700×10^{-2}	7.0551×10^{-2}	6.5223×10^{-2}	1.4246×10^{-3}
40%	10^{-12}	7.1191×10^{-2}	8.1459×10^{-2}	7.6576×10^{-2}	2.7609×10^{-3}
	10^{-9}	6.0643×10^{-2}	7.5507×10^{-2}	6.7175×10^{-2}	2.8427×10^{-3}
	10^{-6}	5.2840×10^{-2}	6.1184×10^{-2}	5.5397×10^{-2}	1.3084×10^{-3}
70%	10^{-12}	5.5872×10^{-2}	7.3954×10^{-2}	5.9421×10^{-2}	2.4112×10^{-3}
	10^{-9}	4.7069×10^{-2}	5.1978×10^{-2}	4.8864×10^{-2}	1.1949×10^{-3}
	10^{-6}	4.0591×10^{-2}	4.5647×10^{-2}	4.2245×10^{-2}	8.8261×10^{-4}
90%	10^{-12}	4.8315×10^{-2}	6.5324×10^{-2}	5.2995×10^{-2}	2.4413×10^{-3}
	10^{-9}	4.1187×10^{-2}	5.0542×10^{-2}	4.3975×10^{-2}	1.7201×10^{-3}
	10^{-6}	3.3335×10^{-2}	3.9874×10^{-2}	3.6013×10^{-2}	1.4768×10^{-3}

Table 7. *The randomly generated* 100 × 100 *matrices of different sparsities:* statistics of the sample of 10000 ratios t_{RAS}/t_{A2} of the execution times for the Python programming implementations of RAS and *Algorithm 2* with various numerical values of the convergence criterion ϵ.

Sparsity s	ϵ	Range Min	Range Max	Mean	Standard Deviation
0%	10^{-12}	1.2486	1.3929	1.3394	0.0502
	10^{-9}	1.0465	1.3350	1.2157	0.0790
	10^{-6}	0.8429	1.1012	0.9532	0.0818
10%	10^{-12}	1.3191	1.6605	1.5178	0.0725
	10^{-9}	1.1827	1.4837	1.2992	0.0518
	10^{-6}	1.0947	1.2856	1.1727	0.0294
40%	10^{-12}	2.1724	2.6072	2.3709	0.0948
	10^{-9}	1.9409	2.3692	2.1497	0.0713
	10^{-6}	1.5385	1.9341	1.7708	0.1112
70%	10^{-12}	4.0306	4.8991	4.4414	0.1583
	10^{-9}	3.6270	4.3375	3.9604	0.1417
	10^{-6}	2.8492	3.5824	3.2079	0.1507
90%	10^{-12}	9.5022	14.8275	11.6199	0.7110
	10^{-9}	8.7802	13.6526	10.3378	0.6456
	10^{-6}	7.2530	12.3962	8.4959	0.6776

Table 8. *The randomly generated* 100 × 100 *matrices of different sparsities:* statistics of the sample of $100 \times (1-s) \times 10000$ relative percentage differences d for the Python programming implementations of RAS and *Algorithm 2* with various numerical values of the convergence criterion ϵ.

Sparsity s	ϵ	Range Min	Range Max	Mean	Standard Deviation
0%	10^{-12}	0	1.12×10^{-10}	1.07×10^{-11}	1.47×10^{-11}
	10^{-9}	1.51×10^{-14}	1.11×10^{-7}	1.37×10^{-8}	1.68×10^{-8}
	10^{-6}	6.75×10^{-12}	1.12×10^{-4}	1.75×10^{-5}	1.87×10^{-5}
10%	10^{-12}	0	1.14×10^{-10}	1.18×10^{-11}	1.53×10^{-11}
	10^{-9}	0	1.08×10^{-7}	1.52×10^{-8}	1.53×10^{-8}
	10^{-6}	0	7.61×10^{-5}	8.42×10^{-6}	7.76×10^{-6}
40%	10^{-12}	0	1.23×10^{-10}	9.28×10^{-12}	1.37×10^{-11}
	10^{-9}	0	1.12×10^{-7}	8.70×10^{-9}	1.22×10^{-8}
	10^{-6}	0	1.21×10^{-4}	8.15×10^{-6}	1.34×10^{-5}
70%	10^{-12}	0	1.41×10^{-10}	5.52×10^{-12}	1.22×10^{-11}
	10^{-9}	0	1.38×10^{-7}	5.86×10^{-9}	1.28×10^{-8}
	10^{-6}	0	1.31×10^{-4}	5.78×10^{-6}	1.26×10^{-5}
90%	10^{-12}	0	2.46×10^{-10}	2.36×10^{-12}	1.04×10^{-11}
	10^{-9}	0	2.40×10^{-7}	2.27×10^{-9}	9.86×10^{-9}
	10^{-6}	0	2.29×10^{-4}	2.26×10^{-6}	9.71×10^{-6}

References

[1] Z. Allen-Zhu, Y. Li, R. Oliveira, A. Wigderson, "Much faster algorithms for matrix scaling", in *Proc. 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS)* ((Berkeley, CA), Oct. 15-17), 2017, p. 890-901.

[2] M. Bacharach, *Biproportional Matrices and Input-Output Change*, University of Cambridge, Department of Applied Economics Monographs, no. 16, Cambridge University Press, 1970.

[3] D. F. Batten, *Spatial Analysis of Interacting Economies: The Role of Entropy and Information Theory in Spatial Input-Output Modeling*, Springer, 1983.

[4] M. B. Cohen, A. Madry, D. Tsipras, A. Vladu, "Matrix scaling and balancing via box constrained Newton's method and interior point methods", in *Proc. 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS)* ((Berkeley, CA), Oct. 15-17), 2017, p. 902-913.

[5] GÉANT, "About GÉANT", http://about.geant.org/about/.

[6] ———, "GÉANT Tools Portal", http://tools.geant.org/portal/.

[7] D. A. Gilchrist, L. V. St Louis, "Completing input–output tables using partial information, with an application to Canadian data", *Economic Systems Research* **11** (1999), no. 2, p. 185-194.

[8] G. Günlük-Şenesen, J. M. Bates, "Some experiments with methods of adjusting unbalanced data matrices", *Journal of the Royal Statistical Society: Series A (Statistics in Society)* **151** (1988), no. 3, p. 473-490.

[9] C. T. Ireland, S. Kullback, "Contingency tables with given marginals", *Biometrika* **55** (1968), no. 1, p. 179-188.

[10] B. Kalantari, I. Lari, F. Ricca, B. Simeone, "On the complexity of general matrix scaling and entropy minimization via the RAS algorithm", *Mathematical Programming* **112** (2008), no. 2, p. 371-401.

[11] M. L. Lahr, L. de Mesnard, "Biproportional techniques in input-output analysis: table updating and structural analysis", *Economic Systems Research* **16** (2004), no. 2, p. 115-134.

[12] A. Lemelin, "A GRAS variant solving for minimum information loss", *Economic Systems Research* **21** (2009), no. 4, p. 399-408.

[13] M. Lenzen, B. Gallego, R. Wood, "Matrix balancing under conflicting information", *Economic Systems Research* **21** (2009), no. 1, p. 23-44.

[14] R. A. McDougall, "Entropy theory and RAS are friends", *GTAP Working Papers* (1999), no. 6, Department of Agricultural Economics, Purdue University.

[15] M. H. Schneider, S. A. Zenios, "A comparative study of algorithms for matrix balancing", *Operations Research* **38** (1990), no. 3, p. 439-455.

[16] Y. Zhang, M. Roughan, N. Duffield, A. Greenberg, "Fast accurate computation of large-scale IP traffic matrices from link loads", *ACM SIGMETRICS Performance Evaluation Review* **31** (2003), no. 1, p. 206-217.