Quadratic Convergence for Scaling of Matrices*

Martin Fürer†

Abstract

Matrix scaling is an operation on nonnegative matrices with nonzero permanent. It multiplies the rows and columns of a matrix with positive factors such that the resulting matrix is (approximately) doubly stochastic. Scaling is useful at a preprocessing stage to make certain numerical computations more stable. Linial, Samorodnitsky and Wigderson have developed a strongly polynomial time algorithm for scaling. Furthermore, these authors have proposed to use this algorithm to approximate permanents in deterministic polynomial time. They have noticed an intriguing possibility to attack the notorious parallel matching problem. If scaling could be done efficiently in parallel, then it would approximate the permanent sufficiently well to solve the bipartite matching problem. As a first step towards this goal, we propose a scaling algorithm that is conjectured to run much faster than any previous scaling algorithm. It is shown that this algorithm converges quadratically for strictly scalable matrices. We interpret this as a hint that the algorithm might always be fast. All previously known approaches to matrix scaling can result in linear convergence at best.

1 Introduction

The permanent of an $n \times n$ matrix A with entries a_{ij} $(i, j = 1, \ldots, n)$ is defined by

$$\operatorname{per}(A) = \sum_{\pi} \prod_{i=1}^{n} a_{i\,\pi(i)}$$

where the sum is taken over all permutations π of $\{1, \ldots, n\}$. In strong contrast to determinants, permanents are #P-complete [18] and therefore considered infeasible. The best known deterministic algorithm due to Ryser [16] runs in time $O(2^n n)$ for $n \times n$ matrices. It is based on the inclusion-exclusion principle.

As an exact computation is difficult, much attention has been given to approximation algorithms for permanents, culminating in the approximation scheme of Jerrum, Sinclair and Vigoda [9].

A different approach has been initiated by Linial, Samorodnitsky and Wigderson [13]. These authors use matrix scaling and the Theorem of D.I. Falikman [5] and G.P. Egorichev [4] (previously known as the van der Waerden conjecture) to show that a deterministic polynomial time algorithm can approximate permanents of 0–1 matrices up to an exponential factor. Given the fact that such matrices have permanents anywhere in the range of integers from 0 to $n!$, this is already a good approximation.

Matrix scaling has long been investigated [7, 17, 12, 6, 15, 11, 10, 2, 13, 14]. In its basic form, matrix scaling tries to transform a nonnegative quadratic matrix into a doubly stochastic matrix by multiplying rows and columns with positive reals. In its general form, the number of rows can be different from the number of columns, and the desired row and column sums are not necessarily all 1, but arbitrarily given. For simplicity, we restrict attention to the basic form for quadratic $n \times n$ matrices.

When scaling is possible, then the scaled matrix is uniquely determined even though the factors are not. Most authors call a matrix still *scalable* if it is *approximately scalable* in the following sense: There is a sequence of scaling transformations such that the sequence of transformed matrices converges to a doubly stochastic matrix.

Example. The matrix

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

is scaled by the row factors $(k, 1/k)$ and the column factors $(1/k, k)$ into the matrix

$$\begin{pmatrix} 1 & 0 \\ 1/k^2 & 1 \end{pmatrix}$$

For $k = 1, 2, \ldots,$ the sequence of matrices converges to the doubly stochastic unit matrix, while obviously no single scaling operation can transform the given matrix into a doubly stochastic one.

Every 0–1 matrix A can be viewed as the bipartite adjacency matrix of a bipartite graph G. Every a_{ij} with value 1 represents an edge between vertex i in the left part and vertex j in the right part. The permanent

*Research supported in part by NSF Grant CCR-0209099

†Department of Computer Science and Engineering, Pennsylvania State University

per (A) is the number of perfect matchings in G. With every nonnegative matrix A, we associate a bipartite adjacency matrix by replacing every positive entry a_{ij} by 1.

It can be shown that a nonnegative matrix is approximately scalable if and only if the associated bipartite graph contains a perfect matching. Furthermore, such a matrix is exactly scalable if every edge $\{i, j\}$ (corresponding to a positive entry a_{ij}) participates in some perfect matching.

It has been shown [13] that a nonnegative matrix is approximately scalable if and only if it can be scaled into a matrix in which all row sums are 1 and all column sums deviate from 1 by less than $1/n$. Therefore any provably fast converging iterative scaling algorithm can be used to decide whether a bipartite graph has a perfect matching.

Nathan Linial and Alex Samorodnitsky and Avi Wigderson [13] write on this subject: "The goal of this short subsection is to emphasize the following curious and potentially promising aspect of this work. Our new scaling algorithm may be used to decide whether a given bipartite graph has a perfect matching. The approach we use is conceptually different from known methods." These authors notice that each iteration of their algorithm can be carried out in NC. Hence, the parallel decision problem *Bipartite Matching* would be solved if one could find a similar algorithm that requires only a *polylogarithmic* rather than a polynomial number of iterations.

An approximately doubly stochastic matrix represents an approximate fractional matching in the associated bipartite graph. It is known [8] how to convert such a fractional matching into a proper perfect matching in polylogarithmic time (for bipartite graphs).

This paper fits in the line of research proposed by Linial and Samorodnitsky and Wigderson [13]. It might bring us closer to a solution of this most challenging problem of parallel computing. First, we show that the Linial-Samorodnitsky-Wigderson algorithm (LSW algorithm) indeed requires a polynomial rather than a polylogarithmic number of iterations, even though it is much faster than any previously known algorithm. The reason for the large number of iterations is that in some sense the algorithm operates locally on a matrix. We present a new global algorithm, indicating for the first time that scaling can often be done fast. We are not (yet) able to prove or disprove the conjecture that our algorithm always runs in NC. What we can show is quadratic convergence, which can be interpreted as a hint that this algorithm might run much faster than any other known scaling algorithm. All previously known scaling algorithms converge at most linearly.

The prominence of *Bipartite Matching* among the open problems in parallel computing justifies the study of such a convergence result, even though it is at best a step towards solving the parallel bipartite matching problem.

2 Sinkhorn's Algorithm and the LSW Algorithm

Sinkhorn's algorithm [17] uses the most natural approach. It alternates between dividing all rows by their respective row sums and all columns by their respective column sums. Proving convergence of this algorithm has not been an easy task. Sinkhorn [17] has shown convergence for the special case with all matrix entries strictly positive. Much later, Bapat and Raghavan [1] have solved the general case. Franklin and Lorenz [6] have used Hilbert's projective metric to show that for strictly positive matrix entries, convergence is indeed quite good, namely linear (also known as geometric convergence).

It is easy to see that Sinkhorn's algorithm requires at least a linear number of iterations to achieve a deviation of $O(1/n)$ from 1 in every row sum.

Example. Let A be the following $n \times n$ matrix with $\frac{1}{2}$ in two diagonals and zeros everywhere else.

$$A = \frac{1}{2}\begin{pmatrix} 0 & & & & 1 \\ & & & 1 & 1 \\ & & \iddots & \iddots & \\ & 1 & 1 & & \\ 1 & 1 & & & 0 \end{pmatrix}$$

Sinkhorn's algorithm first only affects the ends of the diagonals and works its way towards the center over the next $n/2$ iterations. With appropriate choices of factors for the rows and columns, it is not hard to show that A can be transformed into any of the following matrices B_k.

$$B_k = \begin{pmatrix} 0 & & & & 1 \\ & & & 1 & 1/k \\ & & \iddots & \iddots & \\ & 1 & 1/k & & \\ 1 & 1/k & & & 0 \end{pmatrix}$$

Thus the matrix A is approximately scalable (into a matrix $\hat{A}$, with entries 1 in the top-right to bottom-left diagonal), as the sequence $B_1, B_2, \ldots$ converges to $\hat{A}$.

The LSW algorithm is much more sophisticated in order to achieve strongly polynomial time, i.e.,the

bound on the number of arithmetic operations depends (polynomially)on n and $1/\log\epsilon$, but not on the matrix entries. Linial, Samorodnitsky and Wigderson [13] actually provide two algorithms, the second being a modification of Sinkhorn's algorithm. Here, we disregard this second algorithm, because it clearly needs at least a linear number of iterations for 0–1 matrices. For arbitrary nonnegative matrices it even uses a matching algorithm in its preprocessing stage, making it useless for our special purposes.

The (first) LSW algorithm plans a block-wise treatment of columns together with a Sinkhorn-like treatment of every row. There are just two blocks, based on the largest gap between column sums. The treatment of the two blocks of columns depends on a parameter δ. In the block with small column sums, all columns are multiplied by $1+\delta$, while the other columns are not modified. Only after the rows are individually multiplied to obtain row sums 1, is the parameter δ chosen to optimize the combined effect.

The argument used to show that Sinkhorn's algorithm is slow in the previous example, applies to the LSW algorithm as well. Initially, all but the first and last column have the same sum (after row normalization). Thus all these columns belong to the same block. Every iteration cannot chip away more than 2 columns from this block. Therefore, at least a linear number of iterations is required before the "center" of the matrix is affected at all.

Indeed, it seems that with some effort, an $\Omega(n^3)$ lower bound on the number of iterations is provable.

Example. If the matrix A of the previous Example is modified to A'' writing a small nonzero element k^{-n} in the top left corner, then A'' is strictly scalable. Indeed the scaled matrix $\widehat{A''}$ is obtained from B_k of that Example by writing $1/k$ in the top left position and multiplying the whole matrix with $1/(1+1/k)$. Nevertheless, Sinkhorn and LSW are still as slow as before, while the new algorithm converges first linearly for $O(\log k)$ steps. Then the approximation is already close to the scaled matrix and converges quadratically.

As a drawback, note that the running time of the new algorithm depends not only on the dimension of the matrix. It is slowed down by tiny positive entries.

3 The New Algorithm

Without loss of generality, we assume the given $n \times n$ matrix A is symmetric and the same factor x_i is used for row i and column i. If the given matrix A is not symmetric, we would just consider the $2n \times 2n$ matrix A' defined as follows (where A^T is A transposed).

$$A' = \begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix}$$

We actually consider a whole class of algorithms parameterized by a continuous nondecreasing function g with the properties $g(-s)+g(s)=0$ and $g'(0)=1$. In this proceedings version, we only consider $g(s)=s$. Possibly better choices are $g(s)=\ln(1+s)$ (for $s \geq 1$) or $g(s)=\frac{1}{2}\ln(1+2s)$ (for $s \geq 0$), because of a smaller tendency to over-reaction.

Main Conjecture *Algorithm $\mathcal{A}$ runs in time polylogarithmic in n and polynomial in $1/\epsilon$ for all 0–1 matrices with positive permanent (if a fast parallel Gaussian elimination procedure is used).*

Based on this conjecture, Algorithm $\mathcal{A}$ should be amended to stop with the conjecture "$\operatorname{per} A = 0$" whenever the running time is too high or when progress is stalling (based on an extended conjecture). Alternatively, one might dare to conjecture that Algorithm $\mathcal{A}$ (as it is) already detects the case "$\operatorname{per} A = 0$" fast.

The idea of Algorithm $\mathcal{A}$ is quite simple and natural. Assume, we are given a vector $\mathbf{s} \in \mathbb{R}^n$ (an n-vector). It determines the vector $\mathbf{x} \in \mathbb{R}^n$ by $x_i = e^{g(s_i)}$ for $i = 1, \ldots, n$. The vector $\mathbf{x}$ is used to scale the matrix A. Multiplying every row i of A by x_i and every column j of A by x_j, we obtain a scaled matrix B. Now let $\mathbf{r} \in \mathbb{R}^n$ be the n-vector of row sums of B (which are equal to the column sums). Finally, we subtract $\mathbf{1}$ from $\mathbf{r}$ where $\mathbf{1}$ is the n-vector with all components being 1. Note that $\mathbf{1}$ is the desired row sum vector.

So far, we have described a function $f : \mathbb{R} \to \mathbb{R}$ mapping $\mathbf{s}$ to $\mathbf{r}-\mathbf{1}$. Obviously, we are interested in an n-vector $\mathbf{s}$ which is a zero of f. Such a zero would scale A immediately into a doubly stochastic matrix $B = \hat{A}$. As we have no means to find such an $\mathbf{s}$ directly, we employ a Newton method to approximate such a zero.

We have a current value of $\mathbf{s} = \mathbf{0}$, resulting in the current matrix A (by scaling it with $\mathbf{x} = \mathbf{1}$). The next step of Newton iteration finds a better $\mathbf{s}$ as follows. The function f is replaced by its linear approximation $\tilde{f}$ at $\mathbf{0}$ (defined with the help of its Jacobian matrix L). Computing a zero of this linear approximation means solving a system of linear equations, i.e., doing Gaussian elimination. If there are multiple solutions, then Gaussian elimination is able to pick the one with minimal L_2-norm. If there is no zero, then it turns out that also the scaling problem has no solution, and the algorithm can stop right away.

Before we describe our algorithm in more detail, we introduce some notation. For every n-vector $\mathbf{v}$, let $\operatorname{diag}(\mathbf{v})$ be the $n \times n$ matrix with $\mathbf{v}$ in its diagonal. We

also use the notation $S = \mathrm{diag}(\mathbf{s})$ and $X = \mathrm{diag}(\mathbf{x})$. Furthermore, we denote by $e^{\mathbf{v}}$ the n-vector with ith component

$$(e^{\mathbf{v}})_i = e^{v_i}$$

Then

$$f(\mathbf{s}) = \mathrm{diag}(e^{\mathbf{s}})\, A\, \mathrm{diag}(e^{\mathbf{s}})\, \mathbf{1} - \mathbf{1}$$

Note that taking row sums is expressed by a matrix multiplication with the $n \times 1$ matrix $\mathbf{1}$.

We use a tilde to denote linear approximations. Thus $\tilde{f}$ is the linear approximation of the function f. Our current matrix A is transformed by the linear approximation (to the first scaling operation) into a matrix $\tilde{A}$. The actual first scaling operation transforms A into B. The goal of scaling is to obtain a doubly stochastic matrix, i.e., a matrix with nonnegative entries with row sums and column sums equal to 1. If the matrix A can be transformed into a doubly stochastic matrix, then we denote this unique matrix by $\hat{A}$. The matrices obtained by any convergent scaling algorithm approach $\hat{A}$ in the limit.

Let us determine the linear approximation to f at $\mathbf{0}$.

$$\begin{aligned}
\tilde{f}(\mathbf{s}) &= \text{lin. approx. to } \mathrm{diag}(\widetilde{e^{g(\mathbf{s})}})\, A\, \mathrm{diag}(\widetilde{e^{g(\mathbf{s})}})\, \mathbf{1} - \mathbf{1} \\
&= \text{lin. approx. to } \mathrm{diag}(\widetilde{e^{\mathbf{s}}})\, A\, \mathrm{diag}(\widetilde{e^{\mathbf{s}}})\, \mathbf{1} - \mathbf{1} \\
&= \text{lin. approx. to } \mathrm{diag}(1 + \mathbf{s})\, A\, \mathrm{diag}(1 + \mathbf{s}) \\
&= \text{lin. approx. to } (A + SA + AS + SAS)\, \mathbf{1} - \mathbf{1} \\
&= (A + SA + AS)\, \mathbf{1} - \mathbf{1} \\
&= A\, \mathbf{1} + SA\, \mathbf{1} + A\, \mathbf{s} - \mathbf{1} \\
&= A\, \mathbf{1} + \mathrm{diag}(A\, \mathbf{1})\, \mathbf{s} + A\, \mathbf{s} - \mathbf{1} \\
&= (A + \mathrm{diag}(A\, \mathbf{1}))\, \mathbf{s} + A\, \mathbf{1} - \mathbf{1}
\end{aligned}$$

Thus the Jacobian matrix of the function f is

$$L = A + \mathrm{diag}(A\mathbf{1})$$

We now describe the scaling algorithm $\mathcal{A}$ (see Figure 1) associated with g in somewhat more detail. The conditions on g ensure that the linear approximation of $g(s_i)$ at $\mathbf{0}$ is just s_i, and therefore at $\mathbf{0}$, the linear approximation of $e^{g(s_i)+g(s_j)}$ is the same as the linear approximation of $e^{s_i+s_j}$, namely $1 + s_i + s_j$.

Hence, the linear approximation $\tilde{A}$ to the scaled matrix B is

$$\tilde{A} = A + SA + AS$$

With Gauss elimination, we determine an $\mathbf{s}$ such that this matrix $\tilde{A}$ has all row sums 1.

Once we have determined such a vector $\mathbf{s} \in \mathbb{R}^n$, we define the n-vectors $\mathbf{x}$ (by $x_i = e^{g(s_i)}$), the matrix B, and its row sum vector $\mathbf{r}$. The matrix B is obtained from A by one scaling step.

$$B = XAX$$

and $\mathbf{r}$ is defined by

$$\mathbf{r} = XAX\mathbf{1} = XA\,\mathbf{x}$$

Algorithm $\mathcal{A}$:

1. W.l.o.g., assume all matrix entries are between 0 and 1 and all row sums are at least 1. A is symmetric.
2. $L = A + \mathrm{diag}(A\,\mathbf{1})$ is the Jacobian matrix associated with the function f. The matrix L is symmetric.
3. Compute $\mathbf{s}$ (solving $L\,\mathbf{s} + A\,\mathbf{1} - \mathbf{1} = \mathbf{0}$, i.e., $\tilde{f}(\mathbf{s}) = \mathbf{0}$), and let $S = \mathrm{diag}(\mathbf{s})$, i.e., S is the diagonal matrix with the vector $\mathbf{s}$ in its diagonal.
   Select the solution $\mathbf{s}$ of minimal norm if there are multiple solutions.
   If there is no solution, write "$\mathrm{per}\,(A) = 0$" and stop.
4. Find the connected components in the bipartite graph of the linear approximation $\tilde{A} = A + AS + SA$ to the scaled matrix B of A.
5. If in the bipartite graph of $A+AS+SA$, the large side of an unbalanced connected component has no edge to any other component of the graph of A, then write "$\mathrm{per}\,(A) = 0$" and stop.
6. If all entries of $A+AS+SA$ are nonnegative then write "$\mathrm{per}\,(A) > 0$."
7. While the current correction would not improve the worst row sum do
   for all i do $s_i := s_i/2$
8. $b_{ij} := e^{g(s_i+s_j)} a_{ij}$ (e.g., $g(x) = x$)
9. If $\forall i \left|\sum_{j=1}^{n} b_{ij} - 1\right| < \epsilon$ then stop
   else $A := B$ goto 1.

Figure 1: Algorithm $\mathcal{A}$

or equivalently

$$r_i = x_i \sum_{j=1}^{n} a_{ij} x_j$$

As our goal is to obtain a matrix $\hat{A}$ with all row sums (and therefore all column sums) equal to 1, we define the deviation function f by

$$f(\mathbf{s}) = \mathbf{r} - \mathbf{1}$$

or equivalently

$$(f(\mathbf{s}))_i = e^{s_i} \sum_{j=1}^{n} a_{ij}\, e^{s_j} - 1$$

In Step 3, the linear equation $\tilde{f}(\mathbf{s}) = \mathbf{0}$, i.e., $L\,\mathbf{s} + A\,\mathbf{1} - \mathbf{1} = \mathbf{0}$ is solved in polylogarithmic time by parallel Gaussian elimination [3].

The system of linear equations is unsolvable in the trivial case, where the bipartite graph associated with A has connected components with unequal numbers of vertices in the two parts.

Many non-trivial cases with permanent 0 are discovered in Step 5, because some entries of $\tilde{A}$ might be 0, even though the corresponding entries in A are not. In fact, it is fairly difficult to hand-pick a matrix A with permanent 0 that is not caught in Step 5 during the first round (or the first few rounds).

If all entries of $\tilde{A} = A + AS + SA$ are nonnegative, then the bipartite graph associated with $\tilde{A}$, and thus also the bipartite graph associated with A, have a fractional matching and therefore also a perfect matching.

Decreasing the amount of a Newton correction in Step 7 is certainly required for some matrices with tiny positive entries that have to be scaled to big entries. In this case, Algorithm $\mathcal{A}$ is slow. We are not really interested in such matrices, as we conjecture that they don't occur when we start from a 0–1 matrix.

We have omitted the treatment of singular Jacobian matrices from this proceedings version. If the rank deficiency is only 1, than this is relatively easy to handle.

The linear approximation associated with the n-vector $\mathbf{s}$ produces a symmetric matrix L with

$$\ell_{ij} = a_{ij}(1 + s_i + s_j)$$

with all row sums equal to 1. It can be shown that this equation is always solvable when the matrix A is approximately scalable (and in some other cases).

If A is a random nonnegative matrix with nonzero permanent from a wide variety of probability distributions, then it is conjectured that the matrix L exists and all its entries are nonnegative with high probability. If this happens, we can already conclude that the permanent of A is not zero, i.e., a perfect matching exists in the corresponding bipartite graph. In other words, not only does our algorithm converge fast after many iterations, it is also very difficult to find examples where the algorithm actually requires more than a few iterations (when the goal is just to decide whether a perfect matching exists).

In any case, one iteration of our algorithm consists of finding $\mathbf{s}$ (and thus $\mathbf{x}$ and X) as indicated above, and computing $B = XAX$ from A. The next iteration continues with B instead of A.

The main problem to show effective bounds on the onset of quadratic convergence is to prove that once all the row sums are close to 1, the matrix entries cannot change much anymore. Finally, we have found a very short and elegant proof of this difficult result.

LEMMA 3.1. *If all row (and column) sums of the matrix A differ from 1 by at most δ, then every matrix entry changes by at most $n\delta$, when going from A to the linear approximation L, i.e., $|a_{ij} - \ell_{ij}| \le n\delta$.*

Proof. Given any pair i', j', select $t \in \mathbb{R}$ such that either

- $s_{i'} \ge t$ and $s_{j'} \ge -t$ (if $s_{i'} + s_{j'} \ge 0$)
- $s_{i'} \le t$ and $s_{j'} \le -t$ (if $s_{i'} + s_{j'} < 0$)

Define

- I^+ as the set of rows i with $s_i \ge t$ and I^- as $\{1, \ldots, n\} - I^+$
- J^+ as the set of columns j with $s_j \ge t$ and J^- as $\{1, \ldots, n\} - J^+$

We assume every row sum of A differs from 1 by at most δ. The correction towards the linear approximation, is given by

$$\ell_{ij} - a_{ij} = a_{ij}(s_i + s_j)$$

using the assumption on the row sums of A and the fact that the row sums of L are exactly 1, we obtain the inequalities

$$\sum_{i=1}^{n} a_{ij}(s_i + s_j) \le \delta \quad \text{for all } j \tag{3.1}$$

and

$$-\sum_{j=1}^{n} a_{ij}(s_i + s_j) \le \delta \quad \text{for all } i \tag{3.2}$$

Summing Inequality (3.1) over all $j \in I^+$ and Inequality (3.2) over all $i \in I^-$, we obtain

$$\sum_{j\in I^+}\sum_{i=1}^{n} a_{ij}(s_i+s_j) - \sum_{i\in I^-}\sum_{j=1}^{n} a_{ij}(s_i+s_j) \leq (|I^+|+|I^-|)\delta \tag{3.3}$$

or equivalently

$$\sum_{i\in I^+}\sum_{j\in I^+} a_{ij}|s_i+s_j| + \sum_{i\in I^-}\sum_{j\in I^-} a_{ij}|s_i+s_j| \leq n\delta \tag{3.4}$$

As each additive term on the left hand side of Inequality (3.4) is nonnegative, each of them fulfills the inequality

$$a_{ij}|s_i+s_j| \leq n\delta \tag{3.5}$$

Noting that $a_{i'j'}|s'_i+s'_j|$ for the given i', j' is also such a term, finishes the proof. □

We have shown that the linear approximation $\tilde{A} = A + AS + SA$ associated with A does not differ much from A as soon as all row sums of A are sufficiently close to 1. Similarly, we show now that in this situation, even the scaled matrix $\hat{A}$ does not differ much from A.

LEMMA 3.2. *If all row (and column) sums of matrix A differ from 1 by at most δ, then every matrix entry changes by at most $n\delta$, when going from A to its scaled matrix $\hat{A}$, i.e., $|a_{ij} - \widehat{a}_{ij}| \leq n\delta$.*

Proof. The argument is similar to that of Lemma 3.1 utilizing the fact that like the linear approximation $\tilde{A}$ also the fully scaled matrix $\hat{A}$ has row sums exactly 1. We omit the proof from this proceedings version.

For only approximately scalable matrices,the argument is slightly more complicated, as we have to work with an appropriate approximately scaled matrix as well as with the limit matrix. □

REMARK 3.1. *The bound on the absolute change of a_{ij} in Lemmas 3.1 and 3.2 cannot be replaced by any bound on its relative change. It can be shown that for $\delta \geq \frac{1}{n-1}$, the factor $1 + s_i + s_j$ can be arbitrarily high.*

From Lemmas 3.1 and 3.2, one can derive quadratic convergence for sufficiently small δ for every scalable matrix. This result does not hold for only approximately scalable matrices as convergence to 0 of some matrix elements is much slower.

THEOREM 3.1. *Locally, with Algorithm $\mathcal{A}$, the row sums of any strictly scalable matrix A converge quadratically.*

Proof. Let the nonnegative symmetric matrix A be exactly scalable. Let $\lambda \leq 1$ be the minimum positive entry in the scaled matrix $\hat{A}$. Assume, A is already very close to being doubly stochastic, i.e.,

$$\left|\sum_{j=1}^{n} a_{ij} - 1\right| \leq \delta \quad (i = 1, \ldots, n)$$

for a sufficiently small $\delta > 0$.

We assume

$$\delta \leq \frac{\lambda}{2n}\ln 2$$

implying $\delta \leq \frac{\lambda}{2n}$ and $\delta \leq 1/2$.

By Lemma 3.2, we know that $|a_{ij} - \hat{a}_{ij}| \leq n\delta$. Because $\delta \leq \frac{\lambda}{2n}$, we know that all positive a_{ij} have a value of at least $\frac{\lambda}{2}$.

By Lemma 3.1, the difference between a matrix entry a_{ij} in the given matrix and the corresponding entry $\ell_{ij} = a_{ij}(1 + s_i + s_j)$ in the linear approximation fulfills

$$|a_{ij} - \ell_{ij}| \leq a_{ij}|s_i + s_j| \leq n\delta \quad (i, j = 1, \ldots, n)$$

This implies

$$|s_i + s_j| \leq \frac{n\delta}{a_{ij}} \leq \frac{2n\delta}{\lambda} \leq \ln 2$$

because $\delta \leq \frac{\lambda}{2n}\ln 2$.

We only consider the case $g(s) = s$. Thus the matrix B obtained from A in one step has entries

$$b_{ij} = a_{ij}\, e^{s_i+s_j}$$

We use the fact that the linear approximation has row sums exactly 1 (except for some rounding error that can easily be tolerated), thus

$$\sum_{j=1}^{n} a_{ij}(1 + s_i + s_j) = 1$$

Now we show that the matrix B obtained from A in one step of Algorithm $\mathcal{A}$ has significantly improved row sums.

$$\begin{aligned}
\left|\sum_{j=1}^{n} b_{ij} - 1\right| &= \left|\sum_{j=1}^{n} a_{ij} e^{s_i+s_j} - 1\right| \\
&= \left|\sum_{j=1}^{n} a_{ij}(e^{s_i+s_j} - (1 + s_i + s_j))\right| \\
&\leq \sum_{j=1}^{n} a_{ij} \left|e^{s_i+s_j} - (1 + s_i + s_j)\right|
\end{aligned}$$

$$\begin{aligned}
&\leq \sum_{j=1}^{n} a_{ij} \frac{e^{\xi_j}}{2!} |s_i + s_j|^2 \\
&\quad \text{for some } \xi_j \text{ with } 0 \leq \xi_j \leq |s_i + s_j| \leq \ln 2 \\
&\leq \sum_{j=1}^{n} a_{ij} \left(\frac{2n\delta}{\lambda}\right)^2 \\
&\leq (1+\delta) \left(\frac{2n\delta}{\lambda}\right)^2 \\
&\leq 6 \left(\frac{n\delta}{\lambda}\right)^2
\end{aligned}$$

□

REMARK 3.2. *Theorem 3.1 could also be obtained from general principles after restricting f to a subspace with regular Jacobian. The direct approach gives some insight in the start of quadratic convergence. From Lemma 3.2, we immediately obtain the following more interesting result.*

THEOREM 3.2. *Locally, with Algorithm $\mathcal{A}$, every element of a strictly scalable matrix A converges quadratically.* □

4 Conclusion

The main purpose of this paper was to present a new algorithm for scaling of matrices. It has been shown that this algorithm has some nice properties not shared by any previously known scaling algorithm. While previous scaling algorithms operate much more locally, our fairly natural algorithm that takes a global view. This makes it conceivable, for the first time, to perform scaling (with sufficient precision) in polylogarithmic time in parallel, while the previously known algorithms can easily be forced to run for at least a linear number of steps. Every step of the new algorithm runs in NC. If (as conjectured), the number of steps is polylogarithmic, then this would decide the perfect bipartite matching problem in NC as suggested by Linial, Samorodnitsky and Wigderson [13]. As we are unable to prove this conjecture, we still support it by the fact that once the new algorithm is sufficiently close to a doubly stochastic matrix, it converges quadratically, i.e., much faster than any previous scaling algorithm.

Even though this is a Newton-like approach, quadratic convergence of the matrices is not obvious. It is based on the fact that a bound δ on the deviation of the row sums from 1, implies a (tight) bound of $n\delta$ on the change in every matrix element during the scaling process.

We conjecture this algorithm to run in time polylogarithmic in n and polynomial in $1/\epsilon$ for all with positive permanent. This would provide a polylogarithmic bipartite matching algorithm. So far, this conjecture is supported mainly by a lack of counterexamples, while the result can be shown for many classes of matrices for which all other known algorithms are slow.

References

[1] R. B. BAPAT AND T. E. S. RAGHAVAN, *An extension of a theorem of Darroch and Ratcliff in loglinear models and its application to scaling multidimensional matrices*, Linear Algebra Appl., 114/115 (1989), pp. 705–715.

[2] A. BOROBIA AND R. CANTÓ, *Matrix scaling: A geometric proof of Sinkhorn's theorem*, Linear Algebra and its Applications, 268 (1998), pp. 1–8.

[3] L. CSANKY, *Fast parallel matrix inversion algorithms*, SIAM J. Comput., 5 (1977), pp. 618–623.

[4] G. EGORICHEV, *The solution of the van der Waerden problem for permanents*, Dokl. Akad. Nauk SSSR, 258 (1981), pp. 1041–1044.

[5] D. FALIKMAN, *A proof of van der Waerden's conjecture on the permanents of a doubly stochastic matrix*, Mat. Zametki, 29 (1981), pp. 931–938.

[6] J. FRANKLIN AND J. LORENZ, *On the scaling of multidimensional matrices*, Linear Algebra and its Applications, 114/115 (1989), pp. 717–735.

[7] D. R. FULKERSON AND P. WOLFE, *An algorithm for scaling matrices*, SIAM Review, 1962 (1962), pp. 142–146.

[8] A. V. GOLDBERG, S. A. PLOTKIN, D. B. SHMOYS, AND É. TARDOS, *Using interior-point methods for fast parallel algorithms for bipartite matching and related problems*, SIAM Journal on Computing, 21 (1992), pp. 140–150.

[9] JERRUM, SINCLAIR, AND VIGODA, *A polynomial-time approximation algorithm for the permanent of a matrix with non-negative entries*, in STOC: ACM Symposium on Theory of Computing (STOC), 2001.

[10] B. KALANTARI AND L. KHACHIYAN, *On the complexity of nonnegative-matrix scaling*, Linear Algebra and its Applications, 240 (1996), pp. 87–103.

[11] L. Khachiyan, *Diagonal matrix scaling is NP-hard*, Linear Algebra and its Applications, 234 (1996), pp. 173–179.

[12] R. R. Klimpel, *Matrix scaling by integer programming*, Communications of the ACM, 12 (1969), pp. 212–213.

[13] N. Linial, A. Samorodnitsky, and A. Wigderson, *A deterministic strongly polynomial algorithm for matrix scaling and approximate permanents*, in ACM Symposium on Theory of Computing (STOC), 1998, pp. 644–652.

[14] A. Nemirovski and U. Rothblum, *On complexity of matrix scaling*, Linear Algebra and its Applications, 302-303 (1999), pp. 435–460.

[15] U. Rothblum and H. Schneider, *Scalings of matrices which have prespecified row sums and column sums via optimization*, Linear Algebra Appl., 114/115 (1989), pp. 737–764.

[16] H. J. Ryser, *Combinatorial mathematics*, Published by The Mathematical Association of America, 1963. The Carus Mathematical Monographs, No. 14.

[17] R. Sinkhorn, *A relationship between arbitrary positive matrices and doubly stochastic matrices*, Annals of Mathematical Statistics, 35 (1964), pp. 876–879.

[18] L. G. Valiant, *The complexity of computing the permanent*, Theoretical Computer Science, 8 (1979), pp. 189–201.