Rounding of Sequences and Matrices, with Applications

Benjamin Doerr, Tobias Friedrich, Christian Klein, and Ralf Osbild

Max-Planck-Institut für Informatik, Saarbrücken, Germany

Abstract. We show that any real matrix can be rounded to an integer matrix in such a way that the rounding errors of all row sums are less than one, and the rounding errors of all column sums as well as all sums of consecutive row entries are less than two. Such roundings can be computed in linear time. This extends and improves previous results on rounding sequences and matrices in several directions. It has particular applications in just-in-time scheduling, where balanced schedules on machines with negligible switch over costs are sought after. Here we extend existing results to multiple machines and non-constant production rates.

1 Introduction

In this paper, we analyze a rounding problem with connections to different areas in discrete mathematics, computer science, and operations research. Roughly speaking, we show that any real matrix can be rounded to an integer one in such a way that the rounding errors of all row and column sums are less than one, and the rounding errors of all sums of consecutive row entries are less than two.

Let m, n be positive integers. For some set S, we write $S^{m\times n}$ to denote the set of $m \times n$ matrices with entries in S. For real numbers a, b let $[a..b] := \{z \in \mathbb{Z} | a \leq z \leq b\}$.

Theorem 1. *Let $X \in \mathbb{R}^{m\times n}$ having integral column sums. Then there is a $Y \in \mathbb{Z}^{m\times n}$ such that*

$$\forall j \in [1..n] : \sum_{i=1}^{m}(x_{ij} - y_{ij}) = 0,$$

$$\forall b \in [1..n], i \in [1..m] : \Big|\sum_{j=1}^{b}(x_{ij} - y_{ij})\Big| < 1.$$

Such a matrix Y can be computed in time $O(mn)$.

It is easy to see that the second condition implies that for all $a, b \in [1..n]$ and $i \in [1..m]$ we have $|\sum_{j=a}^{b}(x_{ij} - y_{ij})| < 2$. Also, the theorem can easily be extended to matrices having arbitrary column sums. See Section 3 for the details.

Theorem 1 extends and improves a number of results from different applications.

1.1 Rounding of Sequences

One of the most basic rounding results states that any sequence $x_1, \ldots, x_n$ of numbers can be rounded to an integer one $y_1, \ldots, y_n$ in such a way that the rounding errors $|\sum_{j=a}^{b}(x_j - y_j)|$ are less than one for all $a, b \in [1..n]$. Such roundings can be computed efficiently in linear time by a one-pass algorithm resembling Kadane's scanning algorithm (described in Bentley's Programming Pearls [4]). Extensions in different directions have been obtained in [9, 10, 13, 16, 18]. This rounding problem has found a number of applications, among others in image processing [1, 17].

Theorem 1 yields a multi-sequence analogue of this result. Assume that we have m sequences $x_1^{(i)}, \ldots, x_n^{(i)}$, $i \in [1..m]$, such that for all $k \in [1..n]$, the k-th terms sum up to at most one (that is, $\sum_{i=1}^{m} x_k^{(i)} \leq 1$). Then we may simultaneously round the sequences such that (i) all errors $|\sum_{j=a}^{b}(x_j^{(i)} - y_j^{(i)})|$ are less than two and (ii) no two sequences have a 1 in the same position, that is, $y_j^{(i_1)} = y_j^{(i_2)} = 1$ implies $i_1 = i_2$.

While we solve this problem in linear time, one has to be more careful than in the one-dimensional case. A simple greedy algorithm may produce a rounding error of $\Omega(\log m)$ as shown in Section 5.1.

1.2 Linear Discrepancy in More Than Two Colors

Let $k \in \mathbb{N}_{\geq 2}$. Denote by E_k the set of the k unit vectors in $\mathbb{R}^k$ and by $\overline{E}_k$ the convex hull of E_k. In other words, $\overline{E}_k = \{v \in [0,1]^k \mid \|v\|_1 = 1\}$. Let $\mathcal{H} = (X, \mathcal{E})$ be a hypergraph. The linear discrepancy problem of $\mathcal{H}$ in k colors is to find for given mixed coloring $p : X \to \overline{E}_k$ a pure coloring $q : X \to E_k$ such that

$$\operatorname{lindisc}(\mathcal{H}, p, q) := \max_{E \in \mathcal{E}} \left\| \sum_{x \in E} (p(x) - q(x)) \right\|_\infty$$

is small. The linear discrepancy of $\mathcal{H}$ in k colors is $\operatorname{lindisc}(\mathcal{H}, k) := \max_p \min_q \operatorname{lindisc}(\mathcal{H}, p, q)$. This notion introduced in [11] extends the classical linear discrepancy notion (see e.g. Beck and Sós [3]), which refers to two colors only.

Let $\mathcal{H}_n$ be the hypergraph of intervals in $[n]$, that is, $\mathcal{H}_n = ([n], \{[a..b] \mid a, b \in [n]\})$. Then Theorem 2, a slight variant of Theorem 1, shows $\operatorname{lindisc}(\mathcal{H}_n, k) < 2$ for all n and k. Theorem 4 shows that for all $k \geq 3$ and all n, $\operatorname{lindisc}(\mathcal{H}_n, k) \geq 1.5 - 6n^{-1/2}$. The lower bound shows that the bound $\operatorname{lindisc}(\mathcal{H}_n, k) < 1$ only holds for $k = 2$. Note that $\mathcal{H}_n$ is a unimodular hypergraph, and that we have $\operatorname{lindisc}(\mathcal{H}, 2) < 1$ for all unimodular hypergraphs.

1.3 Baranyai's Rounding Lemma and Applications in Statistics

Baranyai [2] used a similar rounding result to obtain his famous results on coloring and partitioning complete uniform hypergraphs. He showed that any matrix can be rounded in a way that the errors in all rows, all columns and the whole

matrix are less than one. He used a formulation as flow problem to prove this statement.

Independently, this result was obtained by Causey, Cox and Ernst [6]. In statistics, there are two applications for such rounding results [8]. Note first that instead of rounding to integers, our results also applies to rounding to multiples of any other base (e.g., whole multiples of one percent). This can be used in statistic to improve the readability of data tables. A second reason to apply such rounding procedures is confidentiality. Frequency counts that directly or indirectly disclose small counts may permit the identification of individual respondents. In this case, rounding to multiples of e.g. 10 can prevent such risks. However, in both applications one would like to have that rounding errors in columns and rows are small. This allows to use the rounded matrix to obtain information on the row and column totals.

Our result allows to retrieve further reliable information from the rounded matrix, namely also on the sums of consecutive elements in rows. Such queries make sense if there is a linear ordering on statistical attributes. Here is an example. Let x_{ij} be the number of people in country i that are j years old. Say Y is such that $\frac{1}{1000}Y$ is a rounding of $\frac{1}{1000}X$ as in Theorem 1. Now $\sum_{j=20}^{40} y_{ij}$ is the number of people in country i that are between 20 to 40 years old, apart from an error of less than 2000. Note that such guarantees are not provided by the results of Baranyai and Causey, Cox and Ernst.

Also, our result is algorithmically highly efficient. Both Baranyai, who was not interested in algorithmic issues, and Causey, Cox and Ernst used a reduction of the rounding problem to a flow or transportation problem. Though such problems can be solved relatively efficiently, our linear time solution clearly beats their runtimes.

1.4 Flexible Transfer Line Scheduling

Surprisingly, our matrix rounding problem remains non-trivial if all columns are equal. This problem occurs as a scheduling problem. In the *flexible transfer line scheduling problem* we try to produce m different goods on a single machine in a balanced manner. We know the demands $d_i \in \mathbb{N}$, $i \in [1..m]$, for each good in advance. We assume that our machine (typically a mixed-model assembly line) can produce any good in one unit of time. Furthermore, there are no switch-over costs, that is, we may change from one product to another at no cost.

Our goal is to design a production schedule for $n = \sum_{i=1}^{m} d_i$ time steps such that exactly d_i units of product i are produced. Moreover, at any time and for any product we want our production rate to be close to the average rate $r_i = d_i/n$: After j time steps, we hope to have produced jr_i units of product i. Such production lines are a central part of many just-in-time systems, see e.g. Monden's work [14, 15] on Toyota's production system.

Denote by p_{ij} the number of units of product i produced up to time step j. In the *maximum deviation just-in-time scheduling problem* (MDJIT), our aim is to keep the maximum deviation of these production numbers from the aimed

at values jr_i small. In other words, we are looking for a schedule minimizing $\max\{|p_{ij} - jr_i| \mid i \in [1..m], j \in [1..n]\}$.

For this problem, Steiner and Yeomans [19] as well as Brauner and Crama [5] give a number of interesting results. In particular, they show that the MDJIT can be solved with maximum error less than one. Via Theorem 1, we extend this result to significantly more general settings. (i) We allow non-constant production rates. Instead of only prescribing that the total production of d_i units of product i ideally should be obtained by producing r_i units in each time step, we allow arbitrary aimed at production rates r_{ij} for each product i and time step j. Of course, $\sum_{i=1}^{m} r_{ij}$ should be one for each time step since we assumed that we may produce a single item each time. This generalized setting makes sense if we know or expect changing demands over a period of time.

(ii) We also allow the use of more than one machine. If we have k machines, we may simply use larger rates satisfying $\sum_{i=1}^{m} r_{ij} = k$. In fact, we are quite flexible in this respect. We may use a different number of machines each time step, that is, have $\sum_{i=1}^{m} r_{ij} = k_j$ with different k_j. We may also have non-integral k_j and in this case use between $\lfloor k_j \rfloor - 1$ and $\lceil k_j \rceil$ machines.

1.5 Lower Bounds

We also present a non-trivial lower bound for the error in arbitrary intervals. Earlier works only regarded errors in initial intervals $[1..t]$. From the view-point of balanced schedules approximating average expected demands, it also makes sense to investigate errors in arbitrary intervals. For upper bounds, the simple triangle inequality argument of Lemma 5 extends any upper bound for initial intervals to twice this bound for arbitrary intervals. For lower bounds, things are more complicated. In particular, the example of Brauner and Crama [5] showing a lower bound of $1 - 1/m$ for initial intervals yields no better bound for arbitrary intervals. We present a three product instance (in the simple model with constant rates and one machine) such that any schedule produces an error of at least $1.5 - \varepsilon$. Note that this also yields an error of $0.75 - \varepsilon$ for initial intervals, that cannot be derived from existing works.

2 The Algorithm

In this section, we present an algorithm solving the matrix rounding problem of Theorem 1. For a region $R \subseteq [1..m] \times [1..n]$, the rounding error in R is $|\sum_{(i,j)\in R}(x_{ij} - y_{ij})|$. Our aim is to achieve low rounding errors in all columns and in all intervals of rows. Note that by subtracting integer part, we may always assume that $X \in [0, 1)^{m\times n}$.

We denote by X_i and X^j the i-th row and j-th column of X, respectively. We define the partial sums $s_{ij} := \sum_{\ell=1}^{j} x_{i\ell}$ for all $i \in [1..m]$ and $j \in [1..n]$.

2.1 Basic Algorithm

Here we consider the *restricted* problem with uniform column sums $\|X^j\|_1 = 1$ for all $j \in [1..n]$. Note that in this case each column of the rounded matrix Y contains just a single 1. The solution to this special problem is later on called *basic algorithm*.

First we give a motivation for the solution. By Lemma 5, it suffices to keep the errors

$$\left| \sum_{j=1}^{b} (x_{ij} - y_{ij}) \right|, \quad \forall i \in [1..m],\ \forall b \in [1..n], \tag{1}$$

small in all initial intervals. For the moment, consider a single row $i \in [1..m]$. The idea is to place 1s into Y_i between the row indices where the partial sums of row X_i exceed the next integral values at that time. Formally, we require to place the k-th 1 in row i onto position y_{ij}, where j is some column index in the range $I_i^k := [a_i^k..b_i^k]$ with limits $a_i^k := \min\{j \in [1..n] \mid k - 1 < s_{ij}\}$ and $b_i^k := \max\{j \in [1..n] \mid s_{ij} < k \vee (s_{ij} = k \wedge x_{ij} \neq 0)\}$. We call I_i^k the *k-th index interval of row i*. One particularity of this definition is, that no 1 is placed onto a 0 (say $x_{ij} = 0$), if the row sum s_{ij} is integral. This way, all errors in Equation (1) are less than 1.

The algorithm works as follows. The columns of Y are computed successively, Y^j at time $j \in [1..n]$, that is, we have to place a single 1 into Y^j. To select an appropriate position in column Y^j, we regard the set C^j of all index intervals that contain j and whose corresponding entries in Y are still zeros, i.e., $I_i^k \in C^j$, if and only if $j \in I_i^k$ and $y_{ih} = 0$ for all $h \in I_i^k$, $h < j$. Now, C^j contains implicitly all the positions where the 1 could be placed. From those we choose the position ℓ that belongs to the earliest ending interval $[a_\ell..b_\ell]$ of C^j. (In case of a tie we choose the uppermost row.) Then we set column Y^j to the ℓ-th canonical unit column vector, i.e., $y_{\ell j} = 1$. Then we proceed with Y^{j+1} in the same way.

The index intervals I_i^k can be computed as follows. The *initial step* of the algorithm is to determine the limits a_i^1 and b_i^1 of the intervals I_i^1 for all rows X_i, $i \in [1..m]$. For that purpose, each partial row sum is computed up to the first entry where the sum is no longer smaller than 1 or until we reach the end of the row. (The latter case is indicated by any index larger than n.) The values $a_i := \min(I_i^1)$, $b_i := \max(I_i^1)$ and $s_i := s_{i,b_i}$ are stored in three arrays of length m each. With this information we compute the first column Y^1. Each time after we have placed a 1 in Y, an *update step* is necessary, because then the demand of a current index interval for a 1 is just satisfied. Hence we replace this interval by its succeeding interval $I_i^{\lceil s_i \rceil + 1}$. This can be done similar to the initial step. We continue computing the partial row sum of X_i up to the first entry where the sum is no longer smaller than the next integral value (which is $\lceil s_i \rceil + 1$) or until we reach the end of the row. As before the current values of the interval limits and the sum so far are stored in the arrays.

```
ComputeRounding(X ∈ [0,1]^{m×n})
    ▷ Initialization
    for i ← 1 to m
          do s[i] ← 0
             b[i] ← 0
             (a[i], b[i], s[i]) ← GetNextInterval(i)
    ▷ Main Loop
    for j ← 1 to n
          do C ← {i ∈ [1..m] | j ∈ [a[i]..b[i]]}
             ℓ ← argmin_{i∈C} b[i]
             Y^j ← ℓ-th unit column vector
             (a[ℓ], b[ℓ], s[ℓ]) ← GetNextInterval(ℓ)
    return Y ∈ {0,1}^{m×n}

GetNextInterval(i)
    j ← b[i] + 1
    while j ≤ n and x_ij = 0
          do j ← j + 1
    if j > n
       then return (n + 2, n + 2, s[i])
    a[i] ← j
    k ← ⌈s[i]⌉ + 1
    while s[i] + x_ij ≤ k
          do s[i] ← s[i] + x_ij
             if s[i] = k
                then return (a[i], j, s[i])
             j ← j + 1
             if j > n
                then return (a[i], j, s[i])
    return (a[i], j − 1, s[i])
```

2.2 Time and Space Complexity

For the time being we ignore the calls of GetNextInterval in the analysis of the runtime. Then the initialization loop has runtime $\Theta(m)$ and the main loop, which is executed exactly n times, needs $\Theta(m)$ time for each of the three non-trivial assignments. Together that takes $\Theta(mn)$ time.

It remains to add the time spend in GetNextInterval. Be aware that the row index i never changes within this procedure. Hence its total runtime can be estimated by multiplying the maximal time spend in a single row X_i by m. Each of the commands in GetNextInterval can be executed in constant time except the while loop. Since this loop successively increases j – which is swapped to $b[i]$ when the procedure returns – $\Theta(n)$ time is needed for each row X_i. It follows that the runtime of the entire algorithm is $\Theta(mn)$.

The algorithm only needs to keep track of the m current intervals and the m accumulated row sums. So $\Theta(m)$ space suffices in addition to the space needed for input and output.

2.3 Correctness

To show that our algorithm returns a valid solution, we have to show that (i) each column vector Y^j contains *exactly* one 1 and (ii) each index interval gets assigned *exactly* one column with 1. For this it will be convenient to assume integrality of the row sums, i.e., $\sum_{j=1}^{n} x_{ij} \in \mathbb{N}$ for all i. This can by achieved by adding additional columns at the end. If the algorithm returns a valid solution even for these columns, it is also correct for the original matrix. Note that it is not necessary to actually compute these additional columns, i.e., they are only needed for the analysis. The following lemma gives the main property of the algorithm. It shows that at each step there are enough unsatisfied intervals to choose from.

Lemma 1. *Let k_{ij} be the number of intervals which have started until column j in the i-th row. Then $\sum_{i=1}^{m} k_{ij} \geq j$ for all $j \in [1..n]$.*

Proof (by induction on j). For $j = 1$ at least one interval has to start due to the norm condition $\sum_{i=1}^{m} x_{i1} = 1$ for the first column. Now assume the lemma has been established until column j. If there are already *more* than j intervals, there is nothing to prove for $j + 1$. So let us assume that there are exactly j intervals so far, that is to say, $\sum_{i=1}^{m} k_{ij} = j$. Since $\sum_{i=1}^{m} s_{ij} = \sum_{i=1}^{m} \sum_{\ell=1}^{j} x_{i\ell} = \sum_{\ell=1}^{j} \sum_{i=1}^{m} x_{i\ell} = \sum_{\ell=1}^{j} 1 = j$, we get $\sum_{i=1}^{m} k_{ij} = \sum_{i=1}^{m} s_{ij}$. With $0 \leq s_{ij} \leq k_{ij}$ and $k_{ij} \in \mathbb{N}$ for all $i \in [1..m]$, it follows that $S^j = K^j$ and hence $S^j \in \mathbb{N}^m$. This means that all intervals have ended until column j. So at least one interval has to start at position $j + 1$, analogously to the start of the induction base. So $\sum_{i=1}^{m} k_{(i+1),j} \geq \sum_{i=1}^{m} k_{ij} + 1 \geq j + 1$. □

That there is ($\mathrm{i}_{\leq 1}$) *no column with more than one* 1 is guaranteed by the algorithm as it chooses the uppermost 1 in the case that there are two closest ending intervals at one time. Due to Lemma 1 the algorithm has passed at least j intervals till the j-th column and has by construction satisfied only $j - 1$ of them. Therefore the algorithm can always satisfy at least one interval and will ($\mathrm{i}_{\geq 1}$) *not return any empty column.*

Also ($\mathrm{ii}_{\leq 1}$) *no interval will get more than one* 1, because a 1 is only assigned to unsatisfied intervals. We furthermore know $\|X^j\|_1 = 1$ for all columns j and hence $\sum_{j=1}^{n} \sum_{i=1}^{m} x_{ij} = n$. The integrality assumption of the row sums gives that we have exactly n intervals overall. Since each column contains exactly one 1, we have assigned n 1s to intervals. Due to the pigeonhole principle there is ($\mathrm{ii}_{\geq 1}$) *no interval with no assigned* 1 because there is no interval with more than one 1.

2.4 Error Bounds

Lemma 2. $\left|\sum_{\ell=1}^{j}(x_{i\ell}-y_{i\ell})\right|<1$ *for all* $i\in[1..m]$ *and* $j\in[1..n]$.

Proof. x_{ij} belongs to the k_{ij}-th interval in the i-th row, that is, to $I_i^{k_{ij}}$. The algorithm assigns to each interval exactly one 1 (cf. Section 2.3). So depending on whether the 1 that corresponds to $I_i^{k_{ij}}$ is in some column at most j or later, $\sum_{\ell=1}^{j} y_{i\ell}$ is either $k_{ij}-1$ or k_{ij}, respectively. Hence we have $k_{ij}-1<\sum_{\ell=1}^{j} x_{i\ell}\leq k_{ij}$ as well as $k_{ij}-1\leq\sum_{\ell=1}^{j} y_{i\ell}\leq k_{ij}$, where the second sum equals k_{ij} if the first sum does. This shows $\left|\sum_{\ell=1}^{j} x_{i\ell}-\sum_{\ell=1}^{j} y_{i\ell}\right|<1$. □

Lemma 3. $\left|\sum_{j=a}^{b}(x_{ij}-y_{ij})\right|<2$ *for all* $1\leq a\leq b\leq n$ *and* $i\in[1..m]$.

Proof. This follows immediately from Lemma 2 using Lemma 5. □

The results of the basic algorithm can be subsumed as follows.

Theorem 2. *Let* $X\in[0,1]^{m\times n}$ *with* $\|X^j\|_1=1$ *for all* $j\in[1..n]$. *Then there is a* $Y\in\{0,1\}^{m\times n}$ *such that* $\|Y^j\|_1=1$ *and*

$$\forall b\in[1..n], i\in[1..m]:\left|\sum_{j=1}^{b}(x_{ij}-y_{ij})\right|<1.$$

Such a matrix Y *can be computed in time* $O(mn)$.

The following example shows that the above error bound is tight for our algorithm, i.e. that it may indeed generate errors arbitrarily close to two. To see this let $\varepsilon\in(0,1/2)$ and

$$X_\varepsilon:=\begin{pmatrix}\varepsilon & 1-\varepsilon/2 & 1-2\varepsilon & \varepsilon/2 & \varepsilon\\ (1-\varepsilon)/2 & \varepsilon/4 & \varepsilon & 1/2-\varepsilon/4 & (1-\varepsilon)/2\\ (1-\varepsilon)/2 & \varepsilon/4 & \varepsilon & 1/2-\varepsilon/4 & (1-\varepsilon)/2\end{pmatrix}.$$

This yields the index intervals [1..1] and [2..5] for the first row, and [1..3] and [4..5] for the second and third row. Hence the algorithm puts the first 1 into row one, followed by 1s into row two and three. This yields an error of $(1-\varepsilon/2)+(1-2\varepsilon)=2-5\varepsilon/2$ in the interval [2..3] in the first row.

2.5 Naïve Generalization

We now show that the basic algorithm of Section 2.1 can be utilized for input matrices with arbitrary column sums $\|X^j\|_1=c_j\in\mathbb{N}$ for $j\in[1..n]$. In this case, the output matrix $Y\in\mathbb{N}^{m\times n}$ has to satisfy $\|Y^j\|_1=c_j$. The error on arbitrary intervals is still at most two. First we show how to reduce this generalization to

the unitary problem and solve it with the basic algorithm in $\Theta(m^2 n)$ time. We then modify the algorithm in such a way that it can handle the general problem directly in time $\Theta(mn)$. Note that we can still assume $x_{ij} \in [0, 1)$ (and hence $c_j < m$) for all $j \in [1..n], i \in [1..m]$ as discussed in Section 2.

A simple way to solve the general problem is to preprocess the input by expanding each vector X^j into c_j identical vectors $\tilde{X}^{\ell_j}, \ldots, \tilde{X}^{\ell_j + c_j - 1}$ each of the form $(x_{1j}/c_j, \ldots, x_{mj}/c_j)^T$. With this preprocessing we obtain a new matrix $\tilde{X}$ having $\sum_{j=1}^{n} c_j$ columns, each having sum one. The basic algorithm applied to $\tilde{X}$ yields a matrix $\tilde{Y}$ with errors at most two on arbitrary intervals.

In a postprocessing step we then condense for each $j \in [1..n]$ the c_j output vectors $\tilde{Y}^{\ell_j}, \ldots, \tilde{Y}^{\ell_j + c_j - 1}$ to one vector Y^j (having column sum c_j) by summing them up. This yields a solution Y to the original problem. Since all intervals $[a..b] \subseteq [1..n]$ of the general problem correspond to an interval $[\ell_a..(\ell_b + c_b - 1)]$ of the expanded problem, Y satisfies the properties of Theorem 1.

Observe that this approach may produce entries of value two in the solution. This can happen if an unsatisfied interval ends in the expansion of an input vector and the following index interval ends "close enough" after this expansion. The behavior of the expanding algorithm and the solution it computes can be characterized as follows.

Lemma 4. *Let $\hat{c}_j, j \in [1..n]$, be the number of index intervals that end in (or directly after) the expansion of X^j and are not satisfied before the expansion.*

(a) No index interval is fully contained in the expansion.
(b) $\hat{c}_j \leq c_j$.
(c) The basic algorithm applied to the expanded matrix will first satisfy the $\hat{c}_j$ unsatisfied intervals ending in the expansion. If $\hat{c}_j < c_j$ it will then satisfy the $c_j - \hat{c}_j$ first ending unsatisfied intervals (all of them ending after the expansion).

Proof. The first claim follows since all entries are smaller than one, the second claim follows directly from the correctness of the basic algorithm.

For the third claim observe that there are two types of unsatisfied intervals in the expansion: those ending in (or directly after) it and those continuing afterward. As argued for the second claim, the unsatisfied intervals ending in the expansion are satisfied by the algorithm. Furthermore, all other crossing intervals end after the expansion and hence later than these $\hat{c}_j$ intervals. Thus the algorithm will distribute the remaining 1s to these intervals. □

2.6 Linear Time Generalization

Since expanding X and running the basic algorithm worsens the runtime, we now give an algorithm that simulates this approach and needs nothing more than the basic algorithm of Section 2.1. To achieve this the algorithm has to satisfy c_j intervals instead of just a single one in each step $j \in [1..n]$. According to Lemma 4(c), this can be done in two distribution steps: First identify the $\hat{c}_j$

unsatisfied index intervals ending in the expansion of X^j and assign them a 1. Then satisfy the remaining $c_j - \hat{c}_j$ earliest ending index intervals in the data structure. According to Lemma 4(a) it is not necessary to update and search the data structure after each assigned 1. Instead this can be postponed until the end of each distribution step.

The first distribution step can be done in time $\Theta(m)$ by scanning the data structure once and extracting the $\hat{c}_j$ just ending intervals. Then 1 is added to the entries in Y^j corresponding to those index intervals and their consecutive index intervals are added to the data structure.

For the second distribution step we first extract the $(c_j - \hat{c}_j)$-th earliest ending interval. This too is possible using $\Theta(m)$ time (see e.g. Chapter 10, Medians and Order Statistics, in Cormen et al. [7]). Knowing this interval, the algorithm can locate the other $(c_j - \hat{c}_j) - 1$ earliest ending intervals by just doing a pass over the data structure, again taking $\Theta(m)$ time. Finally, as after the first step, we add 1 to each entry in Y^j corresponding to those index intervals and update the data structure by adding their consecutive index intervals.

Since each update of the data structure takes constant time, the generalized algorithm still needs time $\Theta(mn)$.

The only detail still missing is how to detect if an interval would end inside the expansion of a column X^j and how to compare the endpoints of index intervals ending in the same expansion. For this, first consider the unexpanded input. Let $x_{i,j-1}$ be the last entry belonging to the k-th interval. Then $s_{i,j-1} \leq k < s_{i,j}$ holds. But in the expanded input, the interval would still have a value of $0 \leq k - s_{i,j-1} < x_{i,j} < 1$ left to cover vectors in $\tilde{X}^{\ell_j}, \ldots, \tilde{X}^{\ell_j + c_j - 1}$ of X^j. Since the expansion of X^j has entries x_{ij}/c_j in the i-th row, the interval would continue for

$$\ell := \left\lfloor \frac{k - s_{i,j-1}}{x_{ij}/c_j} \right\rfloor$$

entries into the expansion of X^j.

Hence the end of each index interval is represented by a tuple (j, ℓ) instead of just by the number j as in the basic algorithm. Interval endpoints can then be compared lexicographically.

All in all we can conclude that Theorem 1 holds.

3 Extensions

In this section, we provide two easy extensions of Theorem 1 that are useful in some of the applications described in the introduction. First, it is easy to see that we immediately obtain rounding errors of less than two in arbitrary intervals in rows. This is supplied by the following lemma.

Lemma 5. *Let Y be a rounding of X such that the errors $|\sum_{j=1}^{b}(x_{ij} - y_{ij})|$ in all initial intervals of rows are at most d. Then the errors in arbitrary intervals*

of rows are at most $2d$*, that is, for all* $i \in [1..m]$ *and all* $1 \le a \le b \le n$*,*

$$\left|\sum_{j=a}^{b}(x_{ij} - y_{ij})\right| \le 2d.$$

Proof. Let $i \in [1..m]$ and $1 \le a \le b \le n$. Then

$$\begin{aligned}\left|\sum_{j=a}^{b}(x_{ij} - y_{ij})\right| &= \left|\sum_{j=1}^{b}(x_{ij} - y_{ij}) - \sum_{j=1}^{a-1}(x_{ij} - y_{ij})\right| \\ &\le \left|\sum_{j=1}^{b}(x_{ij} - y_{ij})\right| + \left|\sum_{j=1}^{a-1}(x_{ij} - y_{ij})\right| \le 2d.\end{aligned}$$

□

Second, we may extend Theorem 1 to include matrices having non-integral column sums.

Theorem 3. *Let* $X \in \mathbb{R}^{m\times n}$*. Then there is a* $Y \in \mathbb{Z}^{m\times n}$ *such that*

$$\forall j \in [1..n] : \left|\sum_{i=1}^{m}(x_{ij} - y_{ij})\right| < 2,$$

$$\forall b \in [1..n], i \in [1..m] : \left|\sum_{j=1}^{b}(x_{ij} - y_{ij})\right| < 1.$$

Such a matrix Y *can be computed in time* $O(mn)$*.*

Proof. For an arbitrary matrix X, we add an extra row taking what is missing towards integral column sums: Let $\tilde{X} \in [0,1)^{(m+1)\times n}$ be such that $\tilde{x}_{ij} = x_{ij}$ for all $i \in [1..m]$, $j \in [1..n]$, and $\tilde{x}_{m+1,j} = \lceil\sum_{i=1}^{m} x_{ij}\rceil - \sum_{i=1}^{m} x_{ij}$ for all j.

Clearly $\tilde{X}$ has integral column sums. Using Theorem 1, we can compute a rounding $\tilde{Y} \in \{0,1\}^{(m+1)\times n}$ of $\tilde{X}$ as described in Theorem 1. Note that there are no rounding errors in the columns, i.e., we have $\sum_{i=1}^{m+1} \tilde{y}_{ij} = \sum_{i=1}^{m+1} \tilde{x}_{ij}$ for all $j \in [1..n]$.

Define $Y \in \{0,1\}^{m\times n}$ by $y_{ij} = \tilde{y}_{ij}$ for all $i \in [1..m]$, $j \in [1..n]$. Now the errors in the columns are $|\sum_{i=1}^{m}(x_{ij} - y_{ij})| = |\tilde{x}_{m+1,j} - \tilde{y}_{m+1,j}|$. By Lemma 5, all single entry rounding errors $|x_{ij} - y_{ij}|$ are less than two, proving the first set of inequalities.

The errors in initial intervals in row 1 to m naturally remain unchanged, proving the second set of inequalities. □

4 Lower Bounds

We present a new lower bound for the matrix rounding problem. Theorem 4 shows that there are $3 \times n$ matrices such that any rounding has an error of $1.5 - \varepsilon$

in *arbitrary intervals*. Via a triangle inequality argument similar to Lemma 5, this matrix also yields an error of $0.75 - \varepsilon$ in *initial intervals*. The latter is particularly interesting for the MDJIT problem (see Section 1.4), where Steiner and Yeomans [19] showed a lower bound of $1-1/m$ by means of an $m \times m$ matrix. So for the three-part type MDJIT problem we could raise the lower bound from 2/3 to 3/4.

Theorem 4 (Lower Bound). *For all $\varepsilon \in (0,1)$ there are problem instances $X \in [0,1]^{3\times n}$ such that for all solutions $Y \in \{0,1\}^{3\times n}$ there are $i \in [1..3]$ and $1 \le a \le b \le n$ with $\left|\sum_{j=a}^{b}(x_{ij} - y_{ij})\right| \ge 1.5 - \varepsilon$.*

Proof. Let $n > 1.5/\varepsilon^2$ and $X \in [0,1]^{3\times n}$ with

$$X := \begin{pmatrix} 1-\varepsilon & 1-\varepsilon & & 1-\varepsilon \\ \varepsilon-\varepsilon^2 & \varepsilon-\varepsilon^2 & \cdots & \varepsilon-\varepsilon^2 \\ \varepsilon^2 & \varepsilon^2 & & \varepsilon^2 \end{pmatrix}.$$

Assume that there is a valid solution Y with $\left|\sum_{j=a}^{b}(x_{ij} - y_{ij})\right| < 1.5 - 4\varepsilon$ for all $i \in [1..3]$ and $1 \le a \le b \le n$. By choice of n, there is at least one column j having a 1 in the third row. Let $p \ge 0$ and $q \ge 0$ be the number of consecutive columns equal to $(1,0,0)^T$ to the left and right of column j, respectively. Thus

$$Y = \begin{pmatrix} & 0 & 1\ldots1 & 0 & 1\ldots1 & 0 & \\ \cdots & ? & 0\ldots0 & 0 & 0\ldots0 & ? & \cdots \\ & ? & \underbrace{0\ldots0}_{p \text{ times}} & 1 & \underbrace{0\ldots0}_{q \text{ times}} & ? & \end{pmatrix}.$$

column j

The rounding error of the interval $[(j-p-1)\,..\,(j+q+1)]$ in the first row is $\left|\sum_{\ell=j-p-1}^{j+q+1}(x_{1,\ell} - y_{1,\ell})\right| = (p+q+3)\cdot(1-\varepsilon) - (p+q) = 3\cdot(1-\varepsilon) - \varepsilon\cdot(p+q)$. Since this is less than $1.5-4\varepsilon$, we have $p+q > (3\cdot(1-\varepsilon) - 1.5 + 4\varepsilon)/\varepsilon = 1.5/\varepsilon + 1$. The error of the interval $[j-p..j+q]$ in the second row now is $\left|\sum_{\ell=j-p}^{j+q}(x_{2,\ell} - y_{2,\ell})\right| = (p+q+1)\cdot(\varepsilon-\varepsilon^2) > (1.5/\varepsilon+2)\cdot(\varepsilon-\varepsilon^2) = 1.5 + 0.5\varepsilon - 2\varepsilon^2 > 1.5 - 4\varepsilon$. This contradicts our assumption. □

5 Alternative Approaches

5.1 Greedy Algorithm

A greedy algorithm traverses the matrix X column by column and sets the 1s in Y only based on the columns previously read. The 1 is assigned to a row i with the highest difference between the accumulated sum s_{ij} and the number of 1s in this row so far. That this may produce a rounding error of $\Omega(\log n)$ can be

shown by the following example:

$$X := \begin{pmatrix} \frac{1}{n} & 0 & 0 & \cdots & 0 & 0 \\ \frac{1}{n} & \frac{1}{n-1} & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{1}{n} & \frac{1}{n-1} & \frac{1}{n-2} & \cdots & 0 & 0 \\ \frac{1}{n} & \frac{1}{n-1} & \frac{1}{n-2} & \cdots & \frac{1}{2} & 0 \\ \frac{1}{n} & \frac{1}{n-1} & \frac{1}{n-2} & \cdots & \frac{1}{2} & 1 \end{pmatrix} \in [0,1]^{n \times n}$$

The greedy algorithm returns the identity matrix whereby the discrepancy of the interval [1,n-1] in the last row becomes $\left|\sum_{j=1}^{n-1}(x_{n,j} - y_{n,j})\right| = \sum_{j=2}^{n} 1/j = H_n - 1 > \log n - 1$ with H_n being the harmonic number of n.

5.2 Row Intervals

If one accepts a quadratic runtime we can extend Theorem 2 in such a way that not only the initial row intervals, but also the initial column intervals are small:

Theorem 5. *Let $X \in [0,1)^{m \times n}$. Then there is a $Y \in \{0,1\}^{m \times n}$ such that*

$$\forall b \in [1..n], i \in [1..m] : \left|\sum_{j=1}^{b}(x_{ij} - y_{ij})\right| < 1,$$

$$\forall b \in [1..m], j \in [1..n] : \left|\sum_{i=1}^{b}(x_{ij} - y_{ij})\right| < 1.$$

Such a matrix Y can be computed in time $O(m^2n^2)$.

Proof. Knuth [13] showed how to round a sequence of n real numbers x_i to $y_i \in \{\lfloor x_i \rfloor, \lceil x_i \rceil\}$ such that for two given permutations σ_1 and σ_2, we have $\sum_{i=1}^{k}(x_{\sigma_1(i)} - y_{\sigma_1(i)}) < 1$ as well as $\sum_{i=1}^{k}(x_{\sigma_2(i)} - y_{\sigma_2(i)}) < 1$ for all k. To apply this to our problem of rounding a matrix $X \in \mathbb{R}^{m \times n}$, we first assume integrality of the row and column sums without loss of generality as detailed in Section 3. Consider all elements x_{ij} of the matrix X as the sequence to be rounded. With a permutation σ_1, which enumerates the x_{ij} row by row, Knuth's two-way rounding gives $\sum_{i=1}^{k}\sum_{j=1}^{n}(x_{ij} - y_{ij}) < 1$ for all $k \in [1..m]$. Note that the integrality of the row sums yields by induction $\sum_{i=1}^{k}\sum_{j=1}^{n}(x_{ij} - y_{ij}) = 0$ for all k, which in turn shows for the initial row intervals $\sum_{j=1}^{b}(x_{ij} - y_{ij}) < 1$ for all $b \in [1..n]$ and $i \in [1..m]$. For initial column intervals one can achieve $\sum_{i=1}^{b}(x_{ij} - y_{ij}) < 1$ for all $b \in [1..m]$ and $j \in [1..n]$ in an analogous manner by choosing a permutation σ_2, which enumerates the x_{ij} column by column. His proof employs integer flows in a certain network [12]. On account of this he only achieves a runtime of $O(m^2n^2)$. □

Note that both inequalities in Theorem 5 are actually $\left|\sum(x_{ij} - y_{ij})\right| \leq mn/(mn+1)$.

6 Acknowledgments

The authors wish to thank Pavol Hell for pointing out the relation to controlled rounding.

References

1. T. Asano. Digital halftoning: Algorithm engineering challenges. *IEICE Trans. on Inf. and Syst.*, E86-D:159–178, 2003.
2. Zs. Baranyai. On the factorization of the complete uniform hypergraph. In *Infinite and finite sets (Colloq., Keszthely, 1973; dedicated to P. Erdős on his 60th birthday), Vol. I*, pages 91–108. Colloq. Math. Soc. Jánōs Bolyai, Vol. 10. North-Holland, Amsterdam, 1975.
3. J. Beck and V. T. Sós. Discrepancy theory. In R. Graham, M. Grötschel, and L. Lovász, editors, *Handbook of Combinatorics*, pages 1405–1446. Elsevier, 1995.
4. J. L. Bentley. Algorithm design techniques. *Commun. ACM*, 27:865–871, 1984.
5. N. Brauner and Y. Crama. The maximum deviation just-in-time scheduling problem. *Discrete Appl. Math.*, 134:25–50, 2004.
6. B. D. Causey, L. H. Cox, and L. R. Ernst. Applications of transportation theory to statistical problems. *Journal of the American Statistical Association*.
7. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. MIT Press, Cambridge, MA, 1990.
8. L. H. Cox and L. R. Ernst. Controlled rounding. *Informes*, 20(4):423–432, 1982.
9. B. Doerr. Lattice approximation and linear discrepancy of totally unimodular matrices. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 119–125, 2001.
10. B. Doerr. Global roundings of sequences. *Information Processing Letters*, 92:113–116, 2004.
11. B. Doerr and A. Srivastav. Multicolour discrepancies. *Combinatorics, Probability and Computing*, 12:365–399, 2003.
12. L. R. Ford, Jr., and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.
13. D. E. Knuth. Two-way rounding. *SIAM J. Discrete Math.*, 8:281–290, 1995.
14. Y. Monden. What makes the Toyota production system really tick? *Industrial Eng.*, 13:36–46, 1981.
15. Y. Monden. *Toyota Production System*. Industrial Engineering and Management Press, Norcross, GA, 1983.
16. K. Sadakane, N. Takki-Chebihi, and T. Tokuyama. Combinatorics and algorithms on low-discrepancy roundings of a real sequence. In *ICALP 2001*, volume 2076 of *Lecture Notes in Computer Science*, pages 166–177, Berlin Heidelberg, 2001. Springer-Verlag.
17. K. Sadakane, N. Takki-Chebihi, and T. Tokuyama. Discrepancy-based digital halftoning: Automatic evaluation and optimization. In *Geometry, Morphology, and Computational Imaging*, volume 2616 of *Lecture Notes in Computer Science*, pages 301–319, Berlin Heidelberg, 2003. Springer-Verlag.
18. J. Spencer. *Ten lectures on the probabilistic method*, volume 64 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1994.
19. G. Steiner and S. Yeomans. Level schedules for mixed-model, just-in-time processes. *Management Science*, 39:728–735, 1993.