COMPUTING THE DIMENSION OF A BIPARTITION MATRIX

DAWSON FREEMAN AND RONALD UMBLE

Abstract. The dimension of a bipartition matrix (BPM) is the sum of the dimensions of its indecomposable factors. The dimension of an indecomposable BPM is the sum of its row, column, and entry dimensions. To compute these dimensions, we apply four routines of independent interest: (1) Factor a bipartition as a product of indecomposables; (2) recover a bipartition from its indecomposable factorization; (3) factor a BPM as a product of indecomposables; and (4) compute the “transpose-rotation” (the column dimension of a BPM is the row dimension of its transpose-rotation).

1. Introduction

The permutahedron $P_n$ is an $(n - 1)$-dimensional contractible polytope whose vertices are identified with the $n!$ permutations of $\underline{n} = \{1, 2, \ldots, n\}$. In low dimensions, $P_1$ is a point, $P_2$ is a unit interval, $P_3$ is a hexagon, and $P_4$ is a truncated octahedron. While $P_4$ was known to Johannes Kepler in 1619 [2], the family $P = \{P_n\}_{n \geq 1}$ of permutahedra was first studied by the Dutch mathematician Pieter Hendrik Schoute in 1911 [7].

The facets of $P_n$ are identified with the partitions of $\underline{n}$ obtained by removing a single bar, called a divider, from each permutation $A_1 \cdots A_n$. For example, from the vertex $3|1|2$ of $P_3$ we obtain the edges $13|2$ and $3|12$ with common vertex $3|12$ (see Figure 1). In general, removing $n - k$ dividers from a permutation of $\underline{n}$ produces a partition of length $k$ identified with an $(n - k)$-dimensional face of $P_n$.

There is a combinatorial bijection between the set of partitions of $\underline{n}$ and the set of faces of $P_n$; the dimension of a partition of $\underline{n}$ is the geometric dimension of the corresponding face. Furthermore, there is a combinatorial bijection between
the set of faces of $P_n$ and the set of down-rooted planar leveled trees (PLTs) with $n + 1$ leaves. Given a PLT $T$ with $n + 1$ leaves and $k$ levels, number the leaves from left-to-right and assign the label $\ell$ to the vertex of $T$ at which the branch containing leaf $\ell$ meets the branch containing leaf $\ell + 1$ (a vertex may have multiple labels). With level 1 as the top level, let $A_i$ denote the set of vertex labels in level $i$; then $T$ corresponds to the $(n - k)$-dimensional face $A_1|\cdots|A_k$ of $P_n$. For example, the PLT corresponding to the 2-dimensional face $5|13|24$ of $P_5$ appears in Figure 2.

![Figure 2. The PLT corresponding to 5|13|24.](image)

There is a related family of contractible polytopes $K = \{ K_n \}_{n \geq 2}$ called associahedra, constructed by J. Stasheff in 1963 \cite{6}, whose faces are indexed by planar rooted trees (PRTs) with $n$ leaves (without levels), and there is a natural projection $\vartheta : P_{n-1} \to K_n$ due to A. Tonks \cite{8} given by forgetting levels. Let $\vartheta_n$ denote the $n$-leaf corolla with a single vertex. Forgetting levels, the PRT in Figure 2 can be constructed by grafting a copy of $\vartheta_2$ onto each leaf of $\vartheta_3$.

Denote the identity operator by $1$, and think of $\vartheta_n$ as a multilinear operation with $n$ inputs and one output. Define the $\vartheta_i$-composition

$$\vartheta_n \circ_i \vartheta_m := \vartheta_n (1^{\otimes i-1} \otimes \vartheta_m \otimes 1^{\otimes n-i}) ;$$

then the grafting process can be thought of as a sequence of $\vartheta_i$-composition. For example, beginning with the root vertex and working upward, the PRT in Figure 2 can be represented (non-uniquely) as $((\vartheta_3 \circ_1 \vartheta_2) \circ_3 \vartheta_2) \circ_5 \vartheta_2$.

A non-$\Sigma$ operad $(\mathcal{P}, \circ_i)$ in the category of sets consists of a sequence of sets $\mathcal{P} = \{ \mathcal{P}(n) \}_{n \geq 1}$, a unit map $1 \in \mathcal{P}(1)$, and products $\circ_i : \mathcal{P}(m) \times \mathcal{P}(n) \to \mathcal{P}(m + n - 1)$ for $m, n \geq 1$ and $1 \leq i \leq m$ interacting associatively \cite{3}. Examples include (1) a collection of multilinear operations $\{ \vartheta_n : \vartheta_1 = 1 \}$ together with $\circ_i$-compositions, and (2) the top-dimensional cells of the associahedra $K$ identified with multilinear operations $\{ \vartheta_n \}$ together with faces of $K$ identified with $\circ_i$-compositions.

The levels in a PLT encode grafting order. For example, the PLT in Figure 2 with one vertex in level 1, two vertices in level 2, and one vertex in level 3 is uniquely represented by the composition $(\vartheta_3 (\vartheta_2 \otimes \vartheta_2 \otimes 1)) (1^{\otimes 4} \otimes \vartheta_2)$ in which two copies of $\vartheta_2$ are simultaneously grafted onto the first and second leaves of $\vartheta_3$ followed by a copy of $\vartheta_2$ grafted onto the fifth leaf of $\vartheta_3 (\vartheta_2 \otimes \vartheta_2 \otimes 1)$. Equality as operators is an equivalence relation of the set of PLTs. Thus $(\vartheta_3 (\vartheta_2 \otimes \vartheta_2 \otimes 1)) (1^{\otimes 4} \otimes \vartheta_2)$ and $\vartheta_3 (\vartheta_2 \otimes \vartheta_2 \otimes \vartheta_2)$ are equivalent in $P_5$, and their images under $\vartheta$ lie in the same equivalence class in $K_6$.

Stasheff defined the notion of an $A_{\infty}$-algebra by appealing to the operadic structure of the associahedra $K$ in the following way: Denote the top-dimensional cell
of $K_n$ by $\theta_n$ and the cellular chain complex of $K$ by $(CC_* (K), \partial)$. Given a commutative ring $R$ with unity and a DG $R$-module $(A, d)$, denote the tensor module of $A$ by $TA$ and let $\nabla$ denote the differential on $Hom^* (TA, A)$ induced by $d$, i.e., for $f \in Hom(A^\otimes n, A)$,

$$\nabla f := \sum_{i=0}^{n-1} f (1^\otimes i \otimes d \otimes 1^\otimes n-i-1) - (-1)^{|f|} df.$$

**Definition 1.** A DG $R$-module $(A, d)$ with differential $d$ of degree +1 together with a family of multilinear operations $\{\omega_n \in Hom^{2-n} (A^\otimes n, A) : n \geq 2\}$ is an $A_\infty$-algebra if there exists a chain map of non-$\Sigma$ operads $\alpha : (CC_* (K), \partial) \rightarrow (Hom^* (TA, A), \nabla)$ such that $\alpha (\theta_n) = \omega_n$ for each $n$.

S. Saneblidze and the second author extended the families of permutahedra and associahedra in [4] and [5]. The biassociahedra $KK = \{KK_n\}_{n \geq 2}$ is a family of $(m + n - 1)$-dimensional contractible polytopes, of which $KK_0^m = PP_0^m \cong P_m$. The biassociahedra $KK = \{KK_n\}_{n \geq 2}$ is a family of $(m + n - 3)$-dimensional contractible polytopes of which $KK_1^m \cong KK_1^n \cong K_n$ (a picture of the heptagon $KK_3^3$ appears in Figure 3). There is a natural projection $\vartheta_\nabla : PP_m^m \rightarrow KK_{m+1}^n$, which agrees with $\vartheta$ when $mn = 0$, and the notion of a matrad [4], which generalizes the definition of a non-$\Sigma$ operad. Denote the top-dimensional cell of $KK_n^m$ by $\theta_n^m$.

**Definition 2.** A DG $R$-module $(A, d)$ with differential $d$ of degree +1 together with a family of multilinear operations $\{\omega_n^m \in Hom^{3-m-n} (A^\otimes m, A^\otimes n) : m + n \geq 3\}$ is an $A_\infty$-bialgebra if there exists a chain map of matrads $\alpha : (CC_* (KK), \partial) \rightarrow (Hom^* (TA, TA), \nabla)$ such that $\alpha (\theta_n^m) = \omega_n^m$ for each $m$ and $n$.

The definition of a matrad rests heavily on the notion of a bipartition matrix (BPM). A bipartition is an ordered pair of partitions of the same length. A BPM factors as a formal product of indecomposable BPMs, and the dimension of a BPM is the sum of the dimensions of its indecomposable factors. The dimension of an indecomposable BPM has three independent components: row dimension, column dimension, and entry dimension.

The faces of $PP_m^m$ are identified with certain products of “coherent generalized BPMs” (see [3] for details). Just as the dimension of a partition of $n$ is the geometric dimension of the corresponding face of $P_n$, the dimension of a matrix product corresponding to a face of $PP_m^m$ is the geometric dimension of the face (see Figure 3). Thus computing the dimension of generalized BPMs (and BPMs in particular) is fundamentally important.

The dimension of a BPM is the sum of the dimensions of its indecomposable factors, and the dimension of an indecomposable BPM is the sum of its row, column, and entry dimensions. The dimension of a BPM is defined recursively and is well-suited for machine computation. This article presents a computer program that computes the dimension of a BPM by applying four routines of independent interest: (1) a routine that factors a bipartition as a product of indecomposable BPMs, (2) a routine that computes the inverse and recovers the bipartition, (3) a routine that factors a BPM as a product of indecomposables, and (4) a routine that calculates the “transpose-rotation” (the column dimension of a BPM is the row dimension of its transpose-rotation).
2. Bipartitions and Bipartition Matrices

2.1. Partitions and Bipartitions. An ordered set is either the empty set or a strictly increasing set of positive integers.

Definition 3. A partition of length \( r \), denoted \( A_1|A_2|\cdots|A_r \), is an ordered set of (possibly empty) disjoint subsets of a finite set \( A \). Each subset \( A_i \) is called a block; two adjacent blocks are separated by a divider. An empty block is denoted by 0.

Definition 4 differs from the standard definition of a partition in which all blocks are non-empty. This difference is important—the algorithm for computing the dimension of a BPM essentially reduces to counting empty blocks. Dividers will play a pivotal role in the definition of an “embedding partition” (see Definition 6).

Example 1. The partition \( p := 120\{0\}0 \) has length 3 and contains three disjoint subsets of \( A = \{1,2\} \). The first block contains the elements 1 and 2, and as a stylistic choice we will not separate them by commas. The second and third blocks are empty. In the context of the computer program in Appendix A, the partition \( p \) is represented as

\[
[[1, 2], [], []].
\]

Definition 4. A bipartition of length \( r \) is an ordered pair of partitions \( b := (A_1|A_2|\cdots|A_r, B_1|B_2|\cdots|B_r) \) displayed as the fraction

\[
b = \frac{B_1|B_2|\cdots|B_r}{A_1|A_2|\cdots|A_r}.
\]

The input set of \( b \) is denoted and defined by \( IS(b) := A_1 \cup A_2 \cup \cdots \cup A_r \); the output set of \( b \) is denoted and defined by \( OS(b) := B_1 \cup B_2 \cup \cdots \cup B_r \). An elementary bipartition is a bipartition of length 1; a null bipartition has the form \( \frac{\emptyset_0}{\emptyset_0} \).

Note that the partitions \( A_1|A_2|\cdots|A_r \) and \( B_1|B_2|\cdots|B_r \) in Definition 4 may partition different sets; the only requirement is that they have the same length.

Example 2. The bipartition \( b := \frac{560010}{1424} \) has length 3, \( IS(b) = \{1,2,4\} \), and \( OS(b) = \{5,6\} \). In the context of the computer program in Appendix A, the bipartition \( b \) is represented as

\[
[[[1], [2], [4]], [[5, 6], [], []]].
\]

In the development that follows, a bipartition \( q/p \) is thought of as a multivariable operator whose denominator \( p \) is a sequence of inputs and whose numerator \( q \) is a sequence of outputs.

2.2. Bipartition Matrices.

Definition 5. An \( m \times n \) BPM \( C = (c_{ij}) \) has the form

\[
C = \begin{pmatrix}
B_1^{m_1}|B_2^{m_1}|\cdots|B_{r_{m_1}}^{m_1} & \cdots & B_1^{m_n}|B_2^{m_n}|\cdots|B_{r_{m_n}}^{m_n} \\
A_1^{n_1}|A_2^{n_1}|\cdots|A_{r_{n_1}}^{n_1} & \cdots & A_1^{n_n}|A_2^{n_n}|\cdots|A_{r_{n_n}}^{n_n}
\end{pmatrix},
\]
where for $1 \leq i \leq m$ and $1 \leq j \leq n$, the following conditions hold:

1. All bipartitions in a given column have equal input sets.
2. All bipartitions in a given row have equal output sets.
3. Either $\text{IS}(c_{ij}) = \emptyset$ or $\min \text{IS}(c_{ij}) > \max \text{IS}(c_{kj})$ for all $k < i$.
4. Either $\text{OS}(c_{ij}) = \emptyset$ or $\min \text{OS}(c_{ij}) > \max \text{OS}(c_{ik})$ for all $k < j$.

A null BPM has null bipartition entries.

Items (3) and (4) in Definition 5 indicate that the elements of the output sets in a given column increase with the rows and the elements of the input sets in a given row increase with the columns.

**Example 3.** The bipermutahedron $PP_2^1$ and the biassociahedron $KK_3^2$ are identical heptagons (see Figure 3). Faces are identified with products of “coherent” bipartition matrices whose dimension is the dimension of the corresponding face.

![Figure 3. $PP_2^1 = KK_3^2$.](image-url)

**Remark 1.** The computational routines applied in this article are accessible at:

https://dawsonfreeman.pythonanywhere.com/

**Example 4.** In the context of the computer program in Appendix A, the BPM

$$C = \begin{pmatrix} 4/2 & 12/4 \\ 5/10 & 5/167/8 \\ 5/0/0 & 5/0/4 \end{pmatrix}$$

is represented as
3. Embedding Partitions

Let $A = \{a_1, a_2, \ldots, a_m\}$ and $B = \{b_1, b_2, \ldots, b_n\}$ be ordered sets with $A \subseteq B$. The precise way in which $A$ embeds in $B$ is encoded by the “embedding partition.”

**Definition 6.** The embedding partition of $A$ in $B$, denoted by $\mathcal{E}P_B A$, is the partition of $A$ with the following properties:

1. The elements of $\mathcal{E}P_B A$ appear in the same order as in $A$.
2. The number of dividers in $\mathcal{E}P_B A$
   - between consecutive elements $a_i$ and $a_{i+1}$ equals the number of elements between $a_i$ and $a_{i+1}$ in $B$.
   - preceding $a_1$ equals the number of elements preceding $a_1$ in $B$.
   - succeeding $a_m$ equals the number of elements succeeding $a_m$ in $B$.
   - when $A = \emptyset$ equals the cardinality $\#A$.

Note that the number of dividers in $\mathcal{E}P_B A$ is $\#B - \#A$.

**Example 5.** Let $A = \{3, 5, 8, 9\}$ and $B = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$; then

$$\mathcal{E}P_B A = [\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset].$$

and has $\#B - \#A = 5$ dividers. To compute $\mathcal{E}P_B A$ using the code in Appendix A, input

```plaintext
print ( augmentedConsecutivePartition ([1, 2, 3, 4, 5, 6, 7, 8, 9], [3, 5, 8, 9]) );
```

the program returns

$$[[], [], [3], [5], [], [8, 9]].$$

4. Factoring Bipartitions

When calculating BPM dimension, we begin by factoring the BPM as a formal product of indecomposables. Algorithm 1 (below) factors a $1 \times 1$ BPM $C$. Although $C$ does not equal its bipartition entry, we abuse notation and represent $C$ as a bipartition and interpret the expression “factoring a bipartition” to mean “factoring a $1 \times 1$ BPM.”

**Algorithm 1.** Let $c = B_1 | B_2 | \ldots | B_r$ be a bipartition with $r > 1$.

For $k = 1$ to $r$

1. Let $a_1^k | a_2^k | \ldots | a_p^k := \mathcal{E}P_{A_1 \cup \ldots \cup A_k} A_k$.
2. Let $b_1^k | b_2^k | \ldots | b_q^k := \mathcal{E}P_{B_k \cup \ldots \cup B_k} B_k$.
3. Form the matrix

$$C_k = \begin{pmatrix}
\begin{array}{ccc}
\vdots & \cdots & \vdots \\
a_1^k & \cdots & a_p^k \\
\vdots & \ddots & \vdots \\
b_1^k & \cdots & b_q^k \\
a_1^k & \cdots & a_p^k
\end{array}
\end{pmatrix}.$$
The **indecomposable factorization** of \( c \) is the formal matrix product
\[
c = C_1 \cdots C_r.
\]
Clearly, the formal matrix product given by Algorithm 1 is not the standard matrix product in linear algebra.

**Example 6.** The bipartition \( \begin{bmatrix} 0 & 34 \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \end{bmatrix} \) factors as a product of three BPMs whose entries are determined by
\[
a_1^1 = 1 \quad b_1^2|b_1^2|b_1^4 = 0|0|0
\]
\[
a_2^1|a_2^2 = 0|0 \quad b_2^2|b_2^4 = 34|0
\]
\[
a_3^1|a_3^3 = 0|2 \quad b_3^3 = 5.
\]
Thus
\[
\begin{bmatrix} 0 & 34 \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \end{bmatrix} = \begin{pmatrix}
0 & 1 \\
0 & 0
\end{pmatrix} \begin{pmatrix}
34 & 0 \\
0 & 0
\end{pmatrix} \begin{pmatrix}
\frac{5}{2} & 0 \\
0 & \frac{5}{2}
\end{pmatrix}.
\]

To compute this factorization using the code in Appendix A, input
\[
\text{print (indecomposableFactorization ([[1], [], [2]], [[], [3], [4], [5]]))}.
\]
The program returns the following list of bipartition matrices:
\[
\begin{bmatrix}
[[[[[1]], []]], [[[1]], [[]]], [[[1]], []]], [[[[1]], []]], [[[1]], []]], [[[[1]], []]], [[[1]], []]], [[[[1]], []]], [[[1]], []]]
\end{bmatrix}.
\]

**5. Equalizers**

Equalizers enable us to factor a BPM as a formal product of indecomposables and play a pivotal role in the calculation of its dimension.

**5.1. Row Equalizers.** A “row equalizer” transforms a BPM into a BPM with equal numerators in each row.

**Definition 7.** Let \( C = (c_{ij}) \) be a \( q \times p \) BPM, where
\[
c_{ij} := \frac{B_{ij}^1|B_{ij}^2|\cdots|B_{ij}^r}{A_{ij}^1|A_{ij}^2|\cdots|A_{ij}^r}.
\]

A **row equalizer** of \( C \) is a \( q \times p \) matrix of ordered sets \( E_{ij}^{\text{req}} = (e_{ij}^{\text{req}} = \{e_{ij}^{1}, \ldots, e_{ij}^{n_{ij}}\}) \) such that, when the blocks of the corresponding bipartitions are combined in the following way, the resulting output partitions in each row are equal:
\[
c_{ij}^{\text{req}} = \frac{(B_{ij}^1 \cup \cdots \cup B_{ij}^{e_{ij}^{1}})(B_{ij}^{e_{ij}^{1}+1} \cup \cdots \cup B_{ij}^{e_{ij}^{2}}) \cdots (B_{ij}^{e_{ij}^{n_{ij}}+1} \cup \cdots \cup B_{ij}^{r})}{(A_{ij}^1 \cup \cdots \cup A_{ij}^{e_{ij}^{1}})(A_{ij}^{e_{ij}^{1}+1} \cup \cdots \cup A_{ij}^{e_{ij}^{2}}) \cdots (A_{ij}^{e_{ij}^{n_{ij}}+1} \cup \cdots \cup A_{ij}^{r})}.
\]

The matrix \( C_{ij}^{\text{req}} = (c_{ij}^{\text{req}}) \) is the corresponding row equalization of \( C \).
A row equalizer combines blocks in numerators and denominators so that all numerators in the same row are equal. For example, a row equalizer entry $e_{req}^{ij} = \{2, 5\}$ indicates that blocks 3, 4, and 5 in the numerator and denominator of $c_{ij}$ are to be combined.

**Example 7.** Consider the matrix in Example 4:
\[
C = \begin{pmatrix}
12 & 3 & 12 \\
5678 & 3000 & 5678 \\
3000 & 5678 & 0004
\end{pmatrix}.
\]

To equalize numerators in the first row, combine all blocks in the numerator and denominator of the first entry; then the first row becomes
\[
\begin{array}{c}
12 \\
3 \\
4
\end{array}
\]
\[
\begin{array}{c}
12 \\
5678 \\
3000
\end{array}
\begin{array}{c}
5678 \\
0004
\end{array}
\]

To equalize numerators in the second row, combine second and third blocks of the numerator and denominator of the first entry; then the second row becomes
\[
\begin{array}{c}
5 \\
3 \\
4
\end{array}
\begin{array}{c}
67 \\
0 \\
0
\end{array}
\begin{array}{c}
8 \\
5 \\
67 \\
0
\end{array}
\begin{array}{c}
7 \\
0 \\
4
\end{array}
\]

A row equalizer for $C$ is $E_{req} = (\emptyset \emptyset \{1, 3\} \{1, 2\})$ and the corresponding row equalization is
\[
C_{req} = \begin{pmatrix}
12 & 12 \\
5678 & 5678 \\
3000 & 0004
\end{pmatrix}.
\]

Note that an empty set in an equalizer indicates that all blocks in the numerator and denominator of the corresponding entry are to be combined. As a general rule, if a bipartition entry $c_{ij}$ has length $r_{ij}$ and $e_{req}^{ij} \neq \emptyset$, then $\max e_{req}^{ij} < r_{ij}$.

5.2. Column Equalizers. A column equalizer transforms a BPM into a BPM with equal denominators in each column.

**Definition 8.** Let $C = (c_{ij})$ be a $q \times p$ BPM, where
\[
c_{ij} = \frac{B_{1j}^i B_{2j}^i \cdots B_{r_{1j}}^i}{A_{1j}^i | A_{2j}^i | \cdots | A_{r_{1j}}^i}.
\]

A column equalizer of $C$ is a $q \times p$ matrix of ordered sets $E_{ceq} = (e_{ceq}^{ij} = \{e_{ij}^1, \ldots, e_{ij}^{n_{ij}}\})$ such that, when the blocks of the corresponding bipartition are combined in the following way, the resulting input partitions in each column are equal:
\[
\begin{align*}
C_{ceq}^{ij} &= \frac{(B_{1j}^{e_{ij}^1} \cup \cdots \cup B_{e_{ij}^1}^{e_{ij}^1})(B_{e_{ij}^1+1,j}^{e_{ij}^1} \cup \cdots \cup B_{r_{ij}^1}^{e_{ij}^1})}{(A_{1j}^{e_{ij}^1} \cup \cdots \cup A_{e_{ij}^1,j}^{e_{ij}^1})(A_{e_{ij}^1+1,j}^{e_{ij}^1} \cup \cdots \cup A_{r_{ij}^1,j}^{e_{ij}^1})}.
\end{align*}
\]

The matrix $C_{ceq} = (C_{ceq}^{ij})$ is the column equalization of $C$.

**Example 8.** Consider the matrix in Example 4
\[
C = \begin{pmatrix}
12 & 3 & 12 \\
5678 & 3000 & 5678 \\
3000 & 5678 & 0004
\end{pmatrix}.
\]
To equalize denominators in the first column, combine second, third, and fourth blocks in the numerator and denominator of the entry in the second row; then the first column becomes
\[
\begin{pmatrix}
1 & 2 \\
3 & 0 \\
5 & 678 \\
3 & 0 \\
6 & 78
\end{pmatrix}^T.
\]

To equalize denominators in the second column, combine all blocks in the numerator and denominator of the entry in the second row; then the second column becomes
\[
\begin{pmatrix}
12 \\
4 \\
5 & 678 \\
4 \\
5 & 678
\end{pmatrix}^T.
\]

A column equalizer for the matrix \( C \) is
\[
E_{\text{ceq}} = \begin{pmatrix}
\{1\} \\
\{1\}
\end{pmatrix}
\]
and the corresponding equalization is
\[
C_{\text{ceq}} = \begin{pmatrix}
12 & 12 \\
3 & 4 \\
5678 & 5678 \\
3 & 4
\end{pmatrix}.
\]

5.3. Equalizers.

**Definition 9.** An equalizer of a \( q \times p \) BPM \( C \) is a \( q \times p \) matrix of ordered sets \( E \) that is simultaneously a row equalizer and a column equalizer of \( C \).

The program in Appendix A generates a list of all row equalizers of \( C \) and a list of all column equalizers of \( C \). Matrices in both lists are equalizers of \( C \).

**Example 9.** Consider the BPM \( C \) in Example 4:
\[
C = \begin{pmatrix}
1 & 2 \\
3 & 0 \\
5 & 678 \\
3 & 0 \\
6 & 78
\end{pmatrix}.
\]
The only way to equalize numerators in each row and denominators in each column is to combine all blocks. The equalized matrix is the elementary
\[
C_{\text{eq}} = \begin{pmatrix}
12 \\
4 \\
5678 \\
4
\end{pmatrix}.
\]
Its equalizer is the null equalizer
\[
E_{\emptyset} = \begin{pmatrix}
\{1\} \\
\{1\}
\end{pmatrix}.
\]

One can always trivially equalize a BPM by combining all blocks as in Example 9.

**Example 10.** A non-trivial equalization of the BPM
\[
C = \begin{pmatrix}
12 & 12 \\
3 & 4 \\
5678 & 5678 \\
3 & 4
\end{pmatrix}.
\]
is
\[
C_{\text{eq}} = \begin{pmatrix}
12 & 12 \\
3 & 4 \\
6789 & 6789 \\
3 & 4
\end{pmatrix}.
\]
with equalizer

\[
E = \left( \begin{array}{ll}
\{1\} & \{1\} \\
\{2\} & \{2\}
\end{array} \right).
\]

**Definition 10.** Let \( C \) be a \( q \times p \) BPM. Let \( \mathcal{EP} \) denote the set of all equalizers of \( C \) with the following property: The cardinality of each entry is greater than or equal to the cardinalities of the corresponding entries in all equalizers of \( C \). The **maximal equalizer** of \( C \) is the matrix in \( \mathcal{EP} \) whose entries are less than the corresponding entries in all other matrices in \( \mathcal{EP} \) with respect to lexicographic order.

**Example 11.** To compute the maximal equalizer of the matrix in Example 4 using the code in Appendix A, input

\[
\text{print (getMaximalEqualizer ([[[[3], [1]], [[1], [2]]], [[[4], [1, 2]]], [[[3], [1], [1], [1]], [[5], [6], [7], [8]]], [[[1], [1], [4]], [[5], [6, 7], [8]]], []).}
\]

The program returns the maximal equalizer

\[
[[[1], [1]], [[1], [1]], [[1], [1]], [[1], [1]]].
\]

**Example 12.** Suppose a \( 2 \times 2 \) BPM \( C \) has the following equalizers:

\[
E_1 = \left( \begin{array}{ll} 
\{1,2\} & \{2,3\} \\
\{1,3\} & \{1,2\}
\end{array} \right); \quad E_2 = \left( \begin{array}{ll} 
\{1\} & \{2\} \\
\{2\} & \{1\}
\end{array} \right);
\]

\[
E_3 = \left( \begin{array}{ll} 
\{1,2\} & \{1,3\} \\
\{1,3\} & \{1,2\}
\end{array} \right); \quad E_4 = \left( \begin{array}{ll} 
\{\} & \{\} \\
\{\} & \{\}
\end{array} \right).
\]

The entries of \( E_1 \) and \( E_3 \) have cardinality 2, the entries of \( E_2 \) have cardinality 1, and the entries of \( E_4 \) have cardinality 0. Since \( E_3 \) precedes \( E_1 \) in a least-to-greatest lexicographically sorted list, \( E_3 \) is the maximal equalizer of \( C \).

**Example 13.** Refer to Example 10 and consider the equalizers

\[
E_\emptyset = \left( \begin{array}{ll} 
\{\} & \{\} \\
\{\} & \{\}
\end{array} \right) \quad \text{and} \quad E = \left( \begin{array}{ll} 
\{1\} & \{1\} \\
\{2\} & \{2\}
\end{array} \right).
\]

The maximal equalizer is

\[
\left( \begin{array}{ll} 
\{1\} & \{1\} \\
\{2\} & \{2\}
\end{array} \right).
\]

6. **The Transverse Decomposition**

As mentioned above, the first step in computing BPM dimension is to factor the BPM as a product of indecomposables. The “transverse decomposition” of a decomposable BPM \( C \) is a factorization of the form \( C = AB \), where \( A \) is an indecomposable column matrix and \( B \) is a (possibly decomposable) row matrix.

**Algorithm 2.** Let \( C = (c_{ij}) \) be an \( m \times n \) BPM with non-null maximal equalizer \( E = (e_{ij}) \).

For each \( c_{ij} = \frac{B_{1i}^1B_{2i}^2\cdots B_{ki}^k}{A_{1j}^1A_{2j}^2\cdots A_{kj}^k} \) in \( C \):

Let \( e \) be the first element of \( e_{ij} \).
Let \( a_{ij} := \mathcal{E} \mathcal{P}_{1S(c_{ij})} (A_{c+1}^{ij} \cup \cdots \cup A_{ij}^{ij}) \).

For \( k \) from 1 to \( p \):
- Let \( a_{k,c+1}^{ij} := (A_{c+1}^{ij} \cap a_{k}^{ij}) \cdots |(A_{ij}^{ij} \cap a_{k}^{ij}) \).

Let \( b_{ij}^{ij} := \mathcal{E} \mathcal{P}_{OS(c_{ij})} (B_{1}^{ij} \cup \cdots \cup B_{c}^{ij}) \).

For \( k \) from 1 to \( q \):
- Let \( b_{k,c}^{ij} := (B_{1}^{ij} \cap b_{k}^{ij}) \cdots |(B_{c}^{ij} \cap b_{k}^{ij}) \).

Define the formal matrix decomposition

\[
C_{1}^{ij} C_{2}^{ij} = \begin{pmatrix}
A_{1}^{ij} & \cdots & A_{ij}^{ij} \\
b_{ij}^{ij} & \cdots & b_{ij}^{ij}
\end{pmatrix}
\begin{pmatrix}
B_{1}^{ij} & \cdots & B_{ij}^{ij} \\
B_{ij}^{ij} & \cdots & B_{ij}^{ij}
\end{pmatrix}
\begin{pmatrix}
C_{1}^{ij} & \cdots & C_{ij}^{ij} \\
C_{ij}^{ij} & \cdots & C_{ij}^{ij}
\end{pmatrix}
\begin{pmatrix}
A_{1}^{ij} & \cdots & A_{ij}^{ij} \\
b_{ij}^{ij} & \cdots & b_{ij}^{ij}
\end{pmatrix}
\]

The transverse decomposition of \( C \) with respect to \( E \) is the formal matrix factorization \( C := C_{1} C_{2} \) where

\[
C_{1} = \left( C_{1}^{ij} \right) = \begin{pmatrix}
A_{1}^{ij} & \cdots & A_{ij}^{ij} \\
b_{ij}^{ij} & \cdots & b_{ij}^{ij}
\end{pmatrix}
\begin{pmatrix}
B_{1}^{ij} & \cdots & B_{ij}^{ij} \\
B_{ij}^{ij} & \cdots & B_{ij}^{ij}
\end{pmatrix}
\begin{pmatrix}
C_{1}^{ij} & \cdots & C_{ij}^{ij} \\
C_{ij}^{ij} & \cdots & C_{ij}^{ij}
\end{pmatrix}
\begin{pmatrix}
A_{1}^{ij} & \cdots & A_{ij}^{ij} \\
b_{ij}^{ij} & \cdots & b_{ij}^{ij}
\end{pmatrix}
\]

and \( C_{2} = \left( C_{2}^{ij} \right) = \begin{pmatrix}
A_{1}^{ij} & \cdots & A_{ij}^{ij} \\
b_{ij}^{ij} & \cdots & b_{ij}^{ij}
\end{pmatrix}
\begin{pmatrix}
B_{1}^{ij} & \cdots & B_{ij}^{ij} \\
B_{ij}^{ij} & \cdots & B_{ij}^{ij}
\end{pmatrix}
\begin{pmatrix}
C_{1}^{ij} & \cdots & C_{ij}^{ij} \\
C_{ij}^{ij} & \cdots & C_{ij}^{ij}
\end{pmatrix}
\begin{pmatrix}
A_{1}^{ij} & \cdots & A_{ij}^{ij} \\
b_{ij}^{ij} & \cdots & b_{ij}^{ij}
\end{pmatrix}
\]

The entries \( C_{1}^{ij} \) and \( C_{2}^{ij} \) have the following properties:

1. \( C_{1}^{ij} \) is indecomposable.
2. \( 1 + \# IS(C_{1}^{ij}) \) equals the number of entries in \( C_{2}^{ij} \).
3. \( 1 + \# OS(C_{2}^{ij}) \) equals the number of entries in \( C_{1}^{ij} \).

**Example 14.** To factor the (indecomposable) matrix in Example 9 using the code in Appendix A, input
The program returns the indecomposable factorization $C$ in the following way: If the maximal equalizer $C$ of the indecomposable factorization is $\text{null}$, then $C = C_1$ is the indecomposable factorization. Otherwise, compute the transverse decomposition $C = C_1C_2$ with respect to $E_1$. If the maximal equalizer $E_2$ of $C_2$ is null, then $C = C_1C_2$ is the indecomposable factorization. Otherwise, compute the transverse decomposition $C_2 := C_2C_3$ with respect to $E_2$. If the maximal equalizer of $C_3$ is null, then the indecomposable factorization is $C = C_1C_2C_3$. Otherwise, continue this process for $s-1$ steps until the maximal equalizer $E_s$ of $C_s$ is null. Then $C = C_1C_2 \cdots C'_{s-1}C_s$ is the indecomposable factorization.

The tools we need to compute the dimension of a BPM $C$ are now in place.
7. The Dimension Algorithm

Given a BPM $C$, let $\pi(C)$ denote the matrix obtained from $C$ by discarding empty biblocks 0/0 in non-null entries and by reducing null entries to 0/0.

**Definition 11.** Let $C$ be a $q \times p$ BPM. Denote the dimension of $C$ by $|C|$. If $C$ is null, define $|C| := 0$. Otherwise, define

$$
|C| := |C|^{\text{row}} + |C|^{\text{col}} + |C|^{\text{ent}},
$$

where the **row dimension** $|C|^{\text{row}}$, **column dimension** $|C|^{\text{col}}$, and **entry dimension** $|C|^{\text{ent}}$ are independent and given by the following recursive algorithms:

**Row Dimension Algorithm.** Let $C$ be a $q \times p$ BPM with indecomposable factorization $C = C_1 \cdots C_r$.

If $r > 1$, define $|C|^{\text{row}} := \sum_{k \in \mathbb{Z}} |C_k|^{\text{row}}$, where $|C_k|^{\text{row}}$ is given by setting $C = C_k$ and continuing recursively.

If $r = 1$ and $q > 1$, define $|C|^{\text{row}} := \sum_{i \in \mathbb{Q}} |C_{i*}|^{\text{row}}$, where $|C_{i*}|^{\text{row}}$ is given by setting $C = C_{i*}$ and continuing recursively.

Otherwise, $C$ is a row matrix $(c_1 \cdots c_p)$. Set $C = \pi(C)$.

If $C$ is an elementary matrix and $\text{OS}(C) \neq \emptyset$, define $|C|^{\text{row}} := 0$.

If $C = \begin{pmatrix} a_1 & \cdots & a_q \end{pmatrix}$, define $|C|^{\text{row}} := \begin{cases} 0, & C \text{ is null} \\ \#\text{IS}(C) - 1, & \text{otherwise} \end{cases}$.

Otherwise, define $|C|^{\text{row}} := \sum_{j \in \mathbb{P}} |c_j|^{\text{row}}$, where $|c_j|^{\text{row}}$ is given by setting $C = c_j$ and continuing recursively.

**Column Dimension Algorithm.** Let $C$ be a $q \times p$ BPM with indecomposable factorization $C = C_1 \cdots C_r$.

If $r > 1$, define $|C|^{\text{col}} := \sum_{k \in \mathbb{Z}} |C_k|^{\text{col}}$, where $|C_k|^{\text{col}}$ is given by setting $C = C_k$ and continuing recursively.

If $r = 1$ and $p > 1$, define $|C|^{\text{col}} := \sum_{j \in \mathbb{P}} |C_{*j}|^{\text{col}}$, where $|C_{*j}|^{\text{col}}$ is given by setting $C = C_{*j}$ and continuing recursively.

Otherwise, $C$ is a column matrix $(c_1 \cdots c_q)^T$. Set $C = \pi(C)$.

If $C$ is an elementary matrix and $\text{IS}(C) \neq \emptyset$, define $|C|^{\text{col}} := 0$.

If $C = \begin{pmatrix} b_1 & \cdots & b_q \end{pmatrix}^T$, define $|C|^{\text{col}} := \begin{cases} 0, & C \text{ is null} \\ \#\text{OS}(C) - 1, & \text{otherwise} \end{cases}$.

Otherwise, define $|C|^{\text{col}} := \sum_{i \in \mathbb{Q}} |c_i|^{\text{col}}$, where $|c_i|^{\text{col}}$ is given by setting $C = c_i$ and continuing recursively.

**Entry Dimension Algorithm.** Let $C = (c_{ij})$ be a $q \times p$ BPM with indecomposable factorization $C = C_1 \cdots C_r$.

If $r > 1$, define $|C|^{\text{ent}} := \sum_{k \in \mathbb{Z}} |C_k|^{\text{ent}}$, where $|C_k|^{\text{ent}}$ is given by setting $C = C_k$ and continuing recursively.

Otherwise, define $|C|^{\text{ent}} := \sum_{(i,j) \in \mathbb{Q} \times \mathbb{P}} |c_{ij}|^{\text{ent}}$, where $|c_{ij}|^{\text{ent}}$ is given by setting $C = c_{ij}$ and continuing recursively unless $c_{ij} = \frac{b_j}{a_j}$, in which case define

$$
|c_{ij}|^{\text{ent}} := \begin{cases} \#a_j + \#b_j - 1, & a_j, b_j \neq \emptyset \\ 0, & \text{otherwise} \end{cases}
$$
Example 16. Consider the matrix in Example 7:

\[
C = \begin{pmatrix}
\frac{1}{2} & \frac{12}{4} \\
\frac{5}{6} & \frac{7}{8} \\
\frac{3}{10} & \frac{6}{7}
\end{pmatrix}.
\]

Using the program in Appendix A, the row dimension is 0, the column dimension is 1, and the entry dimension is 5. Therefore the dimension of \(C\) is 6. A similar example with output given by the program in Appendix A appears in Appendix B.

Example 17. Consider the matrix

\[
\begin{pmatrix}
1/2 & 1/2 & 1/2 & 1/2 \\
0/12 & 0/12 & 0/12 & 0/12 \\
3/4 & 3/4 & 3/4 & 3/4 \\
0/12 & 0/12 & 0/12 & 0/12 \\
5/6 & 5/6 & 5/6 & 5/6 \\
0/12 & 0/12 & 0/12 & 0/12 \\
9/8 & 9/8 & 9/8 & 9/8 \\
0/12 & 0/12 & 0/12 & 0/12
\end{pmatrix}.
\]

Using the program in Appendix A, the row dimension is 4, the column dimension is 10, and the entry dimension is 23. Therefore the dimension of \(C\) is 37.

Example 18. Consider the matrix

\[
\begin{pmatrix}
1/2 & 1/2 & 1/2 & 1/2 \\
0/12 & 0/12 & 0/12 & 0/12 \\
3/4 & 3/4 & 3/4 & 3/4 \\
0/12 & 0/12 & 0/12 & 0/12 \\
5/6 & 5/6 & 5/6 & 5/6 \\
0/12 & 0/12 & 0/12 & 0/12 \\
9/8 & 9/8 & 9/8 & 9/8 \\
0/12 & 0/12 & 0/12 & 0/12
\end{pmatrix}.
\]

This matrix has one more row and one more column than the matrix in Example 7. The row dimension of this matrix is 6, the column dimension is 12, and the entry dimension is 28. Therefore, its dimension is 46.

Example 19. Consider the matrix

\[
\begin{pmatrix}
1/2 & 1/2 & 1/2 & 1/2 \\
0/12 & 0/12 & 0/12 & 0/12 \\
3/4 & 3/4 & 3/4 & 3/4 \\
1/2 & 1/2 & 1/2 & 1/2 \\
5/6 & 5/6 & 5/6 & 5/6 \\
0/12 & 0/12 & 0/12 & 0/12 \\
9/8 & 9/8 & 9/8 & 9/8 \\
0/12 & 0/12 & 0/12 & 0/12
\end{pmatrix}.
\]

This matrix is almost identical to the matrix in Example 7, except that the bipartition in the first column and second row is altered so that the maximal equalizer is null and the BPM is indecomposable. The row dimension of this matrix is 6, the column dimension is 7, and the entry dimension is 23. Therefore, the dimension is 36.
In Sections 4 we discussed the indecomposable factorization of a bipartition. There is an inverse algorithm that recovers the bipartition from its indecomposable factorization.

**Algorithm 3.** Let $c = \frac{B_1|B_2|...|B_r}{A_1|A_2|...|A_r}$ be a bipartition and let $c = C_1 \cdots C_r$ be its indecomposable factorization.

For $k$ from 1 to $r$:

Let $n$ be the number of columns in $C_k$ and let $C_{ij}^k$ be the $(i, j)^{th}$ entry of $C_k$.

Then $B_k = C_{11}^k$ and $A_k = C_{nn}^k$.

The efficiency of the algorithm for computing the dimension of a BPM in Appendix A can be improved by applying the “transpose-rotation.”

**Definition 12.** Let $c = \frac{B_1|B_2|...|B_r}{A_1|A_2|...|A_r}$ be a bipartition. The rotation of $c$ is the bipartition $c^{rot} := \frac{A_r|A_{r-1}|...|A_1}{B_r|B_{r-1}|...|B_1}$. The transpose-rotation of a BPM $C = (c_{ij})$ is the matrix $C^{T-R} := (c_{ji}^{rot})$.

The row and column dimensions of $C^{T-R}$ are the column and row dimensions of $C$, respectively. Thus we can use the same algorithm to compute the row and column dimensions of $C$. This computationally more efficient algorithm appears in Appendix A.

Finally, there are “generalized bipartition matrices,” whose entries are bipartitions, products of bipartition matrices, or products of generalized bipartition matrices. The dimension algorithm for generalized bipartition matrices is identical to the algorithm for bipartition matrices, but a computer implementation of the algorithm in the more general setting has yet to be developed.

## 9. Appendix A: The Python Code

```python
import itertools

def intersection(list1, list2):
    list3 = [v for v in list1 if v in list2]
    return list3

def subsets(set):
    powerSet = []
    for i in range(1 << len(set)):
        powerSet.append([set[j] for j in range(len(set)) if (i & (1 << j))])
    return powerSet

def setOfPowersets(r):
    p = []
    for s in r:
        p.append(subsets([i for i in range(1, len(s[1]))]))
    return p
```
# returns all possible equalizers

def possibleEqualizers(p):
    return list(itertools.product(*p))

# checks if a matrix is equalized

def equalized(e):
    for x in e:
        if x != e[0]:
            return 0
    return 1

def combineEqualizers(e, isRow):
    if isRow == 1:
        return [transpose(transpose(eq)) for eq in list(
            itertools.product(*e))]
    else:
        return [transpose(eq) for eq in list(itertools.
            product(*e))]

def transpose(m):
    return [[row[i] for row in m] for i in range(len(m[0]))]

def unionize(m, i, j):
    unionizedList = []
    for h in range(i, j):
        unionizedList = unionizedList + m[h]
    return unionizedList

# row equalizer for matrix of arbitrary size

def arbitraryRE(b):
    equalizers = [[] for row in b]
    j = 0
    for row in b:
        pows = setOfPowersets(row)
        peqr = possibleEqualizers(pows)
        for eqr in peqr:
            i = 0
            eq = []
            for bip in row:
                n = 0
eq.append([])
for m in eqr[i]:
    if m < len(bip[1]) + 1:
        eq[i].append(unionize(bip[1], n, m))
    n = m
    elif n != len(bip[1]):
        eq[i].append(unionize(bip[1], n, len(bip[1])))
for a in eq[i]:
a.sort()
i = i + 1
if equalized(eq):
equalizers[j].append(eqr)
j = j + 1
return equalizers

# column equalizer for matrix of arbitrary size

def arbitraryCE(b):
btt = transpose(b)
equalizers = [[] for col in btt]
j = 0
for col in btt:
pows = setOfPowersets(col)
peqr = possibleEqualizers(pows)
for eqr in peqr:
i = 0
eq = []
for bip in col:
n = 0
eq.append([])
for m in eqr[i]:
    if m < len(bip[0]) + 1:
        eq[i].append(unionize(bip[0], n, m))
    n = m
    elif n != len(bip[0]):
        eq[i].append(unionize(bip[0], n, len(bip[0])))
for a in eq[i]:
a.sort()
i = i + 1
if equalized(eq):
equalizers[j].append(eqr)
j = j + 1
return equalizers

# returns all actual equalizers for a matrix
def equalize(b):
equalizers = []
rowEqualizers = combineEqualizers(arbitraryRE(b), 1)
colEqualizers = combineEqualizers(arbitraryCE(b), 0)
for req in rowEqualizers:
    for ceq in colEqualizers:
        if req == ceq:
            equalizers.append(req)
return equalizers

# returns the maximal equalizer for a matrix

def getMaximalEqualizer(b):
e = equalize(b)
maxEq = e[0].copy()
for i in range(len(e)):
    for j in range(len(e[i])):
        jBreak = 0
        for k in range(len(e[i][j])):
            kBreak = 0
            if len(e[i][j][k]) > len(maxEq[j][k]):
                maxEq = e[i]
                jBreak = 1
                break
            elif len(e[i][j][k]) == len(maxEq[j][k]):
                for l in range(len(e[i][j][k])):
                    if e[i][j][k][l] < maxEq[j][k][l]:
                        maxEq = e[i]
                        jBreak = 1
                        kBreak = 1
                        break
                else:
                    break
            else:
                jBreak = 1
        if jBreak == 1:
            break
    if jBreak == 0:
        break
return maxEq
# returns the ACP of a set and a subset

def augmentedConsecutivePartition(B, A):
    ACP = [[]]
    j = 0
    for i in range(len(B)):
        if B[i] in A:
            ACP[j].append(B[i])
        else:
            ACP.append([])
        j = j + 1
    return ACP

# returns the indecomposable factorization of a bipartition

def indecomposableFactorization(b):
    if len(b[0]) == 1:
        return [[[b]]]
    c = [[] for i in range(len(b[0]))]
    for k in range(len(b[0])):
        A = (augmentedConsecutivePartition(unionize(b[0], 0, k + 1), b[0][k]))
        B = (augmentedConsecutivePartition(unionize(b[1], k, len(b[0])), b[1][k]))
        for j in range(len(B)):
            c[k].append([])
        for i in range(len(A)):
            c[k][j].append([[A[i]], [B[j]]])
    return c

# returns the indecomposable factorization of a BPM

def factorization(b):
    if getMaximalEqualizer(b)[0][0] == []:
        return [b]
    factoredMatrix = [[[ for col in b[0]] for row in b ]]
    for row in range(len(b)):
        for col in range(len(b[row])):
            lam = getMaximalEqualizer(b)[row][col][0]
            A,ACP = []
            B,ACP = []
            Ai = []
            Bj = []
setA = unionize(b[row][col][0], 0, len(b[row][col][0]))
setA.sort()
setB = unionize(b[row][col][1], 0, len(b[row][col][1]))
setB.sort()
AACP = (augmentedConsecutivePartition(setA,
  unionize(b[row][col][0], lam, len(b[row][col][0])))
for i in range(len(AACP)):
  Ai.append([intersection(AACP[i], b[row][col][0][1] for l in range(lam, len(b[row][col][0])))
for i in range(len(BACP)):
  Bj.append([intersection(BACP[i], b[row][col][1][1] for l in range(lam))
factoredMatrix[row][col] = [[[b[row][col][0][k]
  for k in range(lam)] for j in range(len(BACP))]
  for i in range(len(AACP))]
fac = [[[[] for innerRow in factoredMatrix[row][col][0][1]]
  for innerRow in range(len(factoredMatrix[row][0][1])) for row in range(len(b))]
for innerRow in range(len(factoredMatrix[row][0][0])):
  for col in range(len(b[row])):
    ml[row][0][innerRow] = ml[row][0][innerRow] +
    factoredMatrix[row][col][0][innerRow]
for innerRow in range(len(factoredMatrix[row][0][1])):
  for col in range(len(b[row])):
    ml[row][1][innerRow] = ml[row][1][innerRow] +
    factoredMatrix[row][col][1][innerRow]
for row in range(len(b)):
  for innerRow in range(len(ml[row][0])):
    m2[0] = m2[0] + [ml[row][0][innerRow]]
for innerRow in range(len(ml[row][1])):
  m2[1] = m2[1] + [ml[row][1][innerRow]]
return \([m2[0]] + \text{factorization}(m2[1])\)

# calculates the row dimension of a BPM

def rowDimension(b):
    dimension = 0
    for m in factorization(b):
        dimension = dimension + rowDimension2(m)
    return dimension

def rowDimension2(b):
    rowDim = 0
    for row in b:
        isElementary = 1
        nonElementary = 1
        allEmpty = 1
        for bipartition in row:
            if len(bipartition[1]) > 1:
                isElementary = 0
            if len(bipartition[1]) <= 1:
                nonElementary = 0
            if nonElementary == 1:
                if getMaximalEqualizer([row]) != [[[ for bip in row]]:
                    rowDim = rowDim + rowDimension([row])
                if isElementary == 0:
                    for bipartition in row:
                        if getMaximalEqualizer([row]) == [[[ for bip in row]]:
                            for m in indecomposableFactorization(bipartition):
                                rowDim = rowDim + rowDimension(m)
                            if isElementary == 1:
                                if len(row) >= 1:
                                    for bipartition in row:
                                        if bipartition[1] != [[]]:
                                            allEmpty = 0
                                            break
                                            if allEmpty == 1:
                                                isC = 0
                                                for bipartition in row:
                                                    for s in bipartition[0]:
                                                        isC = isC + len(s)
                                            if isC != 0:
                                                rowDim = rowDim + isC - 1
                                            return rowDim
# calculates the column dimension of a BPM

def colDimension(b):
    dimension = 0
    for m in factorization(b):
        dimension = dimension + colDimension2(m)
    return dimension

def colDimension2(b):
    colDim = 0
    bt = transpose(b)
    for col in bt:
        nonElementary = 1
        isElementary = 1
        allEmpty = 1
        for bipartition in col:
            if len(bipartition[1]) > 1:
                isElementary = 0
            if len(bipartition[1]) <= 1:
                nonElementary = 0
            if nonElementary == 1:
                if getMaximalEqualizer(transpose([col])) != [[[[]]]]
                    for bip in col:
                        colDim = colDim + colDimension(transpose([col]))
                for bipartition in col:
                    if len(bipartition[0]) > 1:
                        isElementary = 0
                    if getMaximalEqualizer(transpose([col])) == [[[[]]]]
                        for bip in col:
                            for m in indecomposableFactorization(bipartition):
                                colDim = colDim + colDimension(m)
                if isElementary == 1:
                    if len(col) >= 1:
                        for bipartition in col:
                            if bipartition[0] != [[]] and bipartition[0] != []:
                                allEmpty = 0
                                break
                        if allEmpty == 1:
                            osC = 0
                        for bipartition in col:
                            osC = osC + len(s)
                if osC != 0:
colDim = colDim + osC - 1
return colDim

# calculates the entry dimension of a BPM

def entDimension(b):
    dimension = 0
    for m in factorization(b):
        dimension = dimension + entDimension2(m)
    return dimension

def entDimension2(b):
    entDim = 0
    for row in b:
        for bipartition in row:
            if len(bipartition) == 2:
                if len(bipartition[0]) > 1:
                    for m in indecomposableFactorization(bipartition):
                        entDim = entDim + entDimension(m)
                elif len(bipartition[0]) == 1 and len(bipartition[1]) == 1 and len(bipartition[0][0]) != 0 and len(bipartition[1][0]) != 0:
                    entDim = entDim + len(bipartition[0][0]) + len(bipartition[1][0]) - 1
    return entDim

# calculates the dimension of a BPM

def dimension(b):
    return rowDimension(b) + colDimension(b) + entDimension(b)

# returns transpose-rotation of a matrix for the simplified dimension algorithm

def newAlgorithm(b):
    bt_rot = []
    bt = transpose(b)
    for row in range(len(bt)):
        bt_rot.append([[]])
    for bipartition in range(len(bt[row])):
        bt_rot[row].append([[], []])
    for i in range(len(bt[row][bipartition][0])):
bt_rot[row][bipartition][0].append(bt[row][bipartition][1][len(bt[row][bipartition][0]) - i - 1])
bt_rot[row][bipartition][1].append(bt[row][bipartition][0][len(bt[row][bipartition][0]) - i - 1])
return bt_rot

# returns the composition of a decomposition of a bipartition

def multiply(C):
c = [[] , []]
for i in range(len(C)):
j = len(C[i][0])
a = C[i][0][j - 1][0]
b = C[i][0][0][1]
c[0].append(a)
c[1].append(b)
return c

10. Appendix B: Sample Output

print(factorization([[[], [], []], [[1], [2]], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], []

print(dimension([[[], [], [], [1], [2]], [[1], [2], [3], [4], [5]], [[], [], [], [], []], [], []]))
return 8
Acknowledgment. We wish to thank Jim Stasheff for his helpful editorial suggestions and comments.

References
[1] Freeman, D.: “Computing the Dimension of a Bipartition Matrix,” Senior Honors Thesis, Millersville Univ., Millersville PA (2020).
[2] Kepler, J.: “Harmonices Mundi V.” Opera omnia, Linz (1619).
[3] Markl, M., Shnider, S., and Stasheff, J.: “Operads in Algebra, Topology, and Physics.” AMS Math. Surveys and Monographs 96 (2002).
[4] Šaneblidze, S. and Umble, R.: Matrads, biassociahedra, and $A_{\infty}$-bialgebras, J. Homotopy and Appl., 13 (1) (2011), 1-57.
[5] ————. Framed matrices and $A_{\infty}$-bialgebras. Adv. Studies: Euro-Tbilisi Math. J., 15 (4) (2022), 41-140.
[6] Stasheff, J.: Homotopy associativity of $H$-spaces I, II. Trans. Am. Math. Soc., 108 (1963), 275-312.
[7] Schoute, P. H.: “Analytical treatment of the polytopes regularly derived from the regular polytopes.” J. Muller publisher, Amsterdam (1911).
[8] Tonks, A.: Relating the associahedron and the permutohedron, In: “Operads: Proceedings of the Renaissance Conferences (Hartford CT / Luminy Fr 1995)”, Contemporary Math., 202 (1997), 33-36.

DEPARTMENT OF MATHEMATICS, LEHIGH UNIVERSITY, BETHLEHEM, PA 18015
Email address: djf321@lehigh.edu

DEPARTMENT OF MATHEMATICS, MILLERSVILLE UNIVERSITY OF PENNSYLVANIA, MILLERSVILLE, PA. 17551
Email address: ron.umble@millersville.edu