Internship report M2

Greedy algorithms for computing the Birkhoff decomposition

Damien Lesens

Supervisors
Jérémy E. Cohen – CREATIS,
Bora Uçar – LIP

Location
ROMA team, LIP
ENS de Lyon

February–July 2024

Contents

1 Introduction

A square matrix is called doubly stochastic if it has nonnegative entries, and the sum of entries in each row and in each column is one. Birkhoff Theorem states that any doubly stochastic matrix can be written as a convex combination of permutation matrices [4]. That is, for an $n \times n$ doubly stochastic matrix $\mathbf{A}$, there exist coefficients $\alpha_1, \alpha_2, \ldots, \alpha_k \in (0, 1]$ with $\sum_{i=1}^{k} \alpha_i = 1$ and $n \times n$ permutation matrices $\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_k$ such that

$$\mathbf{A} = \alpha_1 \mathbf{P}_1 + \alpha_2 \mathbf{P}_2 + \cdots + \alpha_k \mathbf{P}_k \,. \tag{1}$$

This representation is called the Birkhoff-von Neumann (BvN) decomposition. A given doubly stochastic matrix in general has multiple BvN decompositions, and there are various applications in which a decomposition with a small number of permutation matrices is required (discussed in Section 2.3). Dufossé and Uçar [12] show that the problem of finding a BvN decomposition of a given matrix with the minimum number of permutation matrices is NP-complete.

In consequence, heuristic approaches to obtaining BvN decompositions with small number of permutation matrices have been proposed and analysed [11, 12]. These heuristics perform well in general on matrices arising in various applications. However, it has been proven that there are optimal BvN decompositions that these heuristics can not reach. The main work of my internship has been to propose, study and implement a new family of algorithms named GompBvN based on methods used to solve the sparse coding problem (defined in Section 2.4). GompBvN can solve optimally instances that

were out of reach for previously known algorithms, while showing similar performance in other cases.

I have also studied a variant of the Birkhoff theorem which decomposes a symmetric doubly stochastic matrix into a convex combination of symmetric permutation matrices. I proposed a new algorithm for the problem, which improves on the asymptotic cost of the only known algorithm while being easier to implement. To support this claim I implemented my algorithm, making it is the first working implementation for this problem. I will briefly present this part of my work in the last section of this report.

2 Background

This section will first introduce useful notation and concepts. I will then present previous works and applications for the problem we are interested in.

2.1 Notation

2.1.1 Matrices

We use bold upper-case letters to refer to matrices, and bold lower-case letters to refer to vectors. We use $\text{vec}(\cdot)$ to convert a matrix to a vector by stacking the columns, e.g., for an $n \times n$ matrix $\mathbf{A}$, the vector $\mathbf{a} = \text{vec}(\mathbf{A})$ is of size n^2.

A *permutation matrix* is a square matrix consisting of zeros and ones, with exactly a single one in each column and in each row. Let $\mathbf{A}$ be an $n \times n$ matrix and $\mathbf{P}$ be an $n \times n$ permutation matrix. We use the notation $\mathbf{P} \subseteq \mathbf{A}$ to denote that the entries of $\mathbf{A}$ at the positions corresponding to the nonzero entries of $\mathbf{P}$ are also nonzero. That is, the nonzero pattern of $\mathbf{P}$ is included in the nonzero pattern of $\mathbf{A}$. We use $\mathbf{P} \odot \mathbf{A}$ to denote the entry-wise product of $\mathbf{P}$ and $\mathbf{A}$, which selects the entries of $\mathbf{A}$ at the positions corresponding to the nonzero entries of $\mathbf{P}$. We also use $\min\{\mathbf{P} \odot \mathbf{A}\}$ to denote the minimum entry of $\mathbf{A}$ at the nonzero positions of $\mathbf{P}$ and $\sum \mathbf{P} \odot \mathbf{A}$ is the sum of the entries of $\mathbf{A}$ at those nonzero positions.

A matrix is *doubly stochastic* if it has nonnegative entries, and the sum of each row and in each column is one. Finally, a square matrix $\mathbf{A}$ is called *fully indecomposable*, when $\mathbf{AQ}$ has nonzeros on the diagonal for a permutation matrix $\mathbf{Q}$, and no permutation matrix $\mathbf{P}$ exists such that $\mathbf{PAQP}^T$ is block upper diagonal [3, Ch.2]. This later definition will appear multiple times in this report, but it is not necessary to understand it.

2.1.2 Graphs

A *matching* in a graph is a set of edges no two of which share a common vertex. A vertex is said to be matched if there is an edge in the matching incident on the vertex. In a *perfect matching*, all vertices are matched.

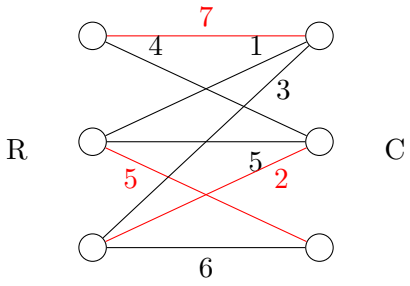
Let $\mathbf{A}$ be an $n \times n$ doubly stochastic matrix. The standard weighted bipartite graph $G = (R \cup C, E)$ associated with $\mathbf{A}$ has n vertices in the set R and another n vertices in the set C such that each $a_{i,j} \neq 0$ uniquely defines an edge $(r_i, c_j) \in E$ in G with weight $a_{i,j}$ (R stands for rows and C for columns). The other way around, any weighted bipartite graph G can be associated with a doubly stochastic matrix $\mathbf{A}$ if the weight of each edge is positive and for each node the sum of the weight of adjacent edges is 1.

This correspondence between matrices and bipartite graph is central to the understanding of the BvN decomposition. Indeed, for a given matrix $\mathbf{A}$ a permutation matrix

$$
\begin{array}{c} \\ R \end{array}
\begin{array}{c} C \\ \begin{pmatrix} 7 & 4 & 0 \\ 1 & 5 & 5 \\ 3 & 2 & 6 \end{pmatrix} \end{array}
$$

$$
= 2 \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} + \begin{pmatrix} 5 & 4 & 0 \\ 1 & 5 & 3 \\ 3 & 0 & 6 \end{pmatrix}
$$

(a) Matrix representation



(b) Graph representation

Figure 1: Example of the equivalence between the matrix and graph representation. Rows form the set of nodes R on the left and columns the set C on the right. The entries of the matrix and the edge labels should be divided by 11 for the matrix to be doubly-stochastic. After subtracting a permutation matrices, an entry of the matrix is put to zero and now all rows and columns sum to 9. The corresponding perfect matching in represented in red in the graph

$\mathbf{P}$ with $\mathbf{P} \subseteq \mathbf{A}$ corresponds to a unique perfect matching in the bipartite graph of $\mathbf{A}$. An example with a 3 by 3 matrix is represented in Figure 1. The *weight* of a perfect matching is $\sum \mathbf{P} \odot \mathbf{A}$ where $\mathbf{P}$ is the permutation associated with this matching.

In the following, we will use $m = |E|$ to denote the number of edges of a graph $G = (V, E)$.

2.1.3 Polytopes

A *polyhedra* P in dimension d is the intersection of a finite number of half spaces of $\mathbb{R}^d$, i.e.,

$$P = \{\mathbf{x} \in \mathbb{R}^d | \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$$

with $\mathbf{A} \in \mathbb{R}^{k \times d}$ and $\mathbf{b} \in \mathbb{R}^k$. The sign $\leq$ used for vectors means that every entry of the vector is less than the other. Each line of $\mathbf{A}$ gives the coefficients for the linear constraints $\mathbf{x}$ has to verify to be in the polyhedra.

A *polytope* is the convex hull of a finite number of points in $\mathbb{R}^d$, i.e.

$$P = \{\mathbf{x} \in \mathbb{R}^d | \mathbf{x} = \sum_i \lambda_i \mathbf{y}_i, \sum_i \lambda_i = 1, \lambda_i \geq 0\}$$

with $\mathbf{y}_i \in \mathbb{R}^d$ for all i. The points $\mathbf{y}_i$ are called the *vertices* of the polytope. The Minkowski-Weyl Theorem establishes that any bounded polyhedra is a polytope. In this case, the linear constraints defining its boundaries are called the *faces* of the polytope and intersection of faces are *edges*.

One could reformulate Birkhoff's Theorem using the terminology of polytopes in the following way: the polyhedra on $\mathbb{R}^{n \times n}$ defined by the constraints:

$$\sum_{i=1}^{n} a_{ij} = 1 \quad \text{for all } j \in \{1, \ldots, n\} \tag{2a}$$

$$\sum_{j=1}^{n} a_{ij} = 1 \quad \text{for all } i \in \{1, \ldots, n\} \tag{2b}$$

$$a_{ij} \geq 0 \quad \text{for all } (i,j) \in \{1,\dots,n\}^2 \tag{2c}$$

is exactly the polytope whose vertices are the permutation matrices.

2.1.4 Optimization problems

In this report we will use optimisation problems as sub-routines for our algorithms.

Linear programs (LP) are problems that can be expressed in a standard form as

$$\begin{array}{ll} \text{Find a vector } \mathbf{x} & \text{maximizing } \mathbf{c}^T\mathbf{x} \\ \text{subject to} & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ \text{and} & \mathbf{x} \geq 0 \end{array} \tag{LP}$$

with $\mathbf{A} \in \mathbb{R}^{k\times d}$, $\mathbf{b} \in \mathbb{R}^k$ and $\mathbf{c} \in \mathbb{R}^d$ in other words the problem is to maximize a linear function over a polytope. The problem is in $\mathcal{P}$ and there are many efficient algorithms to solve it.

A *quadratic program* (QP) can be expressed as

$$\begin{array}{ll} \text{Find a vector } \mathbf{x} & \text{minimizing } \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{c}^T\mathbf{x} \\ \text{subject to} & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ \text{and} & \mathbf{x} \geq 0 \end{array} \tag{QP}$$

with $\mathbf{Q} \in \mathbb{R}^{d\times d}$. In this general formulation this problem is $\mathcal{NP}$-hard, but if the objective has the form $||\mathbf{B}\mathbf{x} - \mathbf{r}||_2^2$, with $||\mathbf{v}||_2 = \mathbf{v}^T\mathbf{v}$ then it is in $\mathcal{P}$ (more precisely it is in $\mathcal{P}$ if the objective is convex). It is equivalent to minimizing a quadratic function on a polytope.

Dropping the linear constraints, we get a *least squares* (LS) problem of the form

$$\text{Find a vector } \mathbf{x} \quad \text{minimizing } ||\mathbf{B}\mathbf{x} - \mathbf{r}||_2 \,. \tag{LS}$$

2.2 Previous works

In this subsection we give an overview of the known heuristics to get an optimal BvN decomposition.

2.2.1 Birkhoff's theorem

Birkhoff's original proof [4] of the existence of a BvN decomposition of the form (1) is constructive.

First, find a permutation matrix $\mathbf{P}_0 \subseteq \mathbf{A}$. To prove the existence of such a permutation Birkhoff uses the "graph view" of $\mathbf{A}$. Indeed, as we observed earlier, finding a permutation $\mathbf{P}_0 \subseteq \mathbf{A}$ is equivalent to finding a perfect matching in the weighted bipartite graph associated with $\mathbf{A}$. I will not give the details here, but Birkhoff's proof makes use of Hall's marriage theorem [16] to show that such perfect matching always exists in a bipartite graph with positive weights such that for each node the sum of the weight of adjacent edges is equal.

Then, set $\alpha_0 = \min\{\mathbf{P}_0 \odot \mathbf{A}\}$, i.e., take α_0 to be the smallest edge weight among the edges of the perfect matching. The central observation now is that if we define $\mathbf{A}^{(1)} = \mathbf{A} - \alpha_0\mathbf{P}_0$, then $\frac{1}{1-\alpha_0}\mathbf{A}^{(1)}$ is doubly stochastic. Indeed, we removed exactly α_0 from each line and each column so the sum of each line and each column is now $1 - \alpha_0$. We can thus repeat the process with $\mathbf{A}^{(1)}$: find $\mathbf{P}_1 \subseteq \mathbf{A}^{(1)}$, subtract it and repeat. A step of this algorithm is represented in Figure 1.

This process will end as at each iteration we set to zero at least one entry of $\mathbf{A}$. Finding a perfect matching in a bipartite graph can be done in $O(\sqrt{n}m)$ time [17], and we repeat the process at most m times.

Algorithm 1 is a pseudo-code for Birkhoff's constructive proof.

Algorithm 1 Birkhoff's original algorithm

1: **Input**: $\mathbf{A}$: a doubly stochastic matrix
2: **Output**: $\alpha_1, \ldots, \alpha_k$ and $\mathbf{P}_1, \ldots, \mathbf{P}_k$, where $\mathbf{A} = \sum_{i=1}^{k} \alpha_i \mathbf{P}_i$
3: Let $\mathbf{A}^{(0)} \leftarrow \mathbf{A}$ and $i \leftarrow 1$
4: **while** $\mathbf{A}^{(i-1)} \neq 0$ **do**
5: find a permutation matrix $\mathbf{P}_i \subseteq \mathbf{A}^{(i-1)}$
6: $\alpha_i \leftarrow \min\{\mathbf{P}_i \odot \mathbf{A}^{(i-1)}\}$
7: $\mathbf{A}^{(i)} \leftarrow \mathbf{A}^{(i-1)} - \alpha_i \mathbf{P}_i$
8: $i \leftarrow i + 1$
9: **end while**
10: $k \leftarrow i - 1$

Based on the observation that at line 7 we annihilate at least one entry, Markus-Ree theorem [26] states that for a dense matrix, Birkhoff's algorithm will obtain at most $k \leq n^2 - 2n + 2$ permutation matrices. Brualdi and Gibson [7] and Brualdi [6] show that for a sparse, fully indecomposable doubly stochastic matrix with τ nonzeros, the relation $k \leq \tau - 2n + 2$ holds.

We can also obtain a lower bound on k (the number of permutation matrices) by observing that at each iteration we set to zero at most one entry per line and per column. Hence $k \geq d_{\max}$ where $d_{\max}$ is the maximum number of nonzeros in a row or a column. It is also the maximum degree in the graph G associated with $\mathbf{A}$, hence the name $d_{\max}$.

In his original proof, Birkhoff selects at line 5 a permutation which touches a minimum nonzero entry in $\mathbf{A}$. One can select any permutation inside $\mathbf{A}^{(i-1)}$, and that is why we can say that Birkhoff actually gives a family of heuristics, differentiating from each other by the way a permutation matrix is selected at each step. Picking a well chosen permutation matrix can greatly reduce the number of components produced by the algorithm, as we will see next.

2.2.2 BvN greedy

The coefficient picked for a permutation $\mathbf{P}_i$ at line 6 is $\alpha_i = \min\{\mathbf{P}_i \odot \mathbf{A}^{(i-1)}\}$. Also, another stopping criterion for the while loop would be $\sum_{i=1}^{j-1} \alpha_i = 1$. In consequence, if one wants to make the most progress in the algorithm, it makes sense to pick a permutation that maximizes $\min\{\mathbf{P}_i \odot \mathbf{A}^{(i-1)}\}$. This idea is proposed and studied by Dufossé and Uçar [12]. The resulting algorithm is called BvNG (for BvN greedy).

At each step, BvNG chooses a perfect matching M in $G = (R \cup C, E)$ which maximizes $\min_{r_k, c_j \in M} \mathbf{A}_{k,j}^{(i-1)}$. The problem of finding a perfect matching whose minimum edge weight is maximum amongst all perfect matchings is called the *bottleneck matching problem* (BPM). For the example of Figure 1, the identity matrix with bottleneck value 5 is such a perfect matching. One way to solve it is to find the value b among edge weights such that there is a perfect matching in $G = (R \cup C, E \cap \{(i,j) | a_{i,j} \geq b\}$, but not in $G = (R \cup C, E \cap \{(i,j) | a_{i,j} > b\}$. This value can be found by binary search on the set of edge weights.

Panagiotas et al. [31] use a different method to solve this problem which I will not explain here. They show that their method is faster and more stable in practice, and provide a C implementation. This is the state of the art algorithm, which I use in Section 4 for the experiments.

Dufossé and Uçar show that this heuristic is very efficient in general compared to Birkhoff's original algorithm, obtaining a number of permutation more than 40 times smaller on some real life instances (see Section 4).

However, Dufossé et al. [11] also show that all heuristics in the family of Algorithm 1 are flawed since they put to zero an entry of $\mathbf{A}$ at each step. In particular, no algorithm from this family, nor similar algorithms making sequential greedy decisions about coefficients without revising them, can always find optimal solutions, even if they test all available permutations. To show this they construct a class of matrices for which Birkhoff type algorithms will never find the optimal solution. They start by associating each letter from a to j with 2^p for $p = 0, \ldots, 9$ in the lexicographic order. Then, the matrix

$$\mathbf{A} = \frac{1}{1023} \begin{pmatrix} a+b & d+i & c+h & e+j & f+g \\ e+g & a+c & b+i & d+f & h+j \\ f+j & e+h & d+g & b+c & a+i \\ d+h & b+f & a+j & g+i & c+e \\ c+i & g+j & e+f & a+h & b+d \end{pmatrix}$$

is doubly stochastic, as is

$$\begin{pmatrix} \mathbf{I}_{n\times n} & 0_{n\times 5} \\ 0_{5\times n} & \mathbf{A} \end{pmatrix}, \tag{3}$$

for any $n \geq 0$, where $\mathbf{I}_{n\times n}$ is the $n \times n$ identity matrix and $0_{n\times 5}$ and $0_{n\times 5}$ are matrices of zeros. An optimal decomposition of (3) requires ten permutation matrices corresponding to the ten letters, and Dufossé et al. show that any algorithm annihilating an entry in the first iteration without revising the coefficients cannot find it.

This is a strong motivation for finding another family of heuristics that could overcome this flaw inherent to Birkhoff type algorithms. Furthermore, we will see later that this type of difficult instances can be extended to a more general class of matrices and that the proposed family of heuristics can find optimal solutions on those cases.

2.3 Applications

Before diving into the main contribution of this report, let me introduce some of the applications of the sparse Birkhoff-von Neumann decomposition problem.

2.3.1 Matrix decomposition

The BvN decomposition can be used to decompose a large class of matrices into a sum of simple atoms which can be utilized in some numerical applications. Getting a sparse decomposition allows for more efficiency in those applications.

At first sight, the set up in which the BvN decomposition applies seems to be restrictive, as we need a square positive doubly-stochastic matrix. However, given any fully indecomposable square matrix $\mathbf{M}$, one can transform it efficiently into a doubly stochastic matrix $\mathbf{A}$ and make use of the BvN decomposition. Indeed, one can first take the absolute value of its entries and then find two diagonal factor matrices $\mathbf{D_R}$ and $\mathbf{D_C}$ such that $\mathbf{A} \approx \mathbf{D_R M D_C}$.

These factor matrices can be found by scaling alternatively the rows and the columns of $\mathbf{M}$, i.e. for each $i \in \{1, \ldots, n\}$, divide the ith row by $\sum_{j=1}^{n} m_{i,j}$, then for each $j \in \{1 \ldots, n\}$, divide the jth column by $\sum_{i=1}^{n} m_{i,j}$. Sinkhorn and Knopp [21] showed in 1967 that repeating this process for a positive fully indecomposable matrix will converge to a doubly-stochastic matrix. Later, Knight and Ruiz [19] provided a more complex algorithm converging faster than the one of Sinkhorn and Knoop.

Once a BvN decomposition of $\mathbf{A}$ is obtained, one can decompose $\mathbf{M}$ in a similar fashion. By remembering the negative entries in $\mathbf{M}$, one can then "reverse" the absolute value applied at the beginning by setting to -1 the entries of the permutation matrices obtained corresponding to negative entries in $\mathbf{M}$. In other words, we can write any fully-indecomposable matrix as a convex combination of scaled and signed permutation matrices. Many matrices appearing in real life applications are fully-indecomposable, thus this decomposition applies to a wide range of matrices. Obtaining a decomposition with a small number of terms is of interest as it allows for a smaller representation in memory and faster manipulations for methods making use of it.

This decomposition is notably used by Benzi and Uçar [2] to solve challenging linear systems. The smaller the number of terms in the decomposition, the faster their algorithm is.

We will also see in Section 5 that there are hopes that this decomposition would preserve the symmetry of the original matrix.

2.3.2 Data center scheduling

Another application of the BvN decomposition appears in scheduling for data centers. High-speed circuit switches are increasingly replacing traditional electronic packet switches in those infrastructures. Circuit switches provide higher data rates at a lower energy cost but are less flexible, meaning that they have a longer reconfiguration time. The problem of computing switching configurations for circuit switches boils down to the BvN decomposition. Indeed, we can model a circuit switch by a doubly-stochastic matrix, with rows representing data senders and columns data receivers. The entries of the matrix represent the percentage of its data each sender wants to send to a given receiver. In this context, the permutation matrices of the BvN decomposition will each represent a switch configuration and coefficients give the amount of time each configuration should be used. Getting a BvN decomposition with a minimal number of terms thus optimizes the reconfiguration time of circuit switches.

This remains a simplistic modelling and many papers present more complex and realistic models deviating from our framework (see e.g., [24, 5, 32]). Results on the classical BvN decomposition remain of use for those more accurate models.

2.4 Sparse coding

The main algorithm proposed in this report for the BvN decomposition takes inspiration from the field of *sparse coding*. The general problem tackled in this field is the following: we are given an input data $\mathbf{x} \in \mathbb{R}^d$ and some atoms $(\mathbf{d}_i)_i \in \mathbb{R}^d$ composing a dictionary $\mathbf{D} = (\mathbf{d}_1|\mathbf{d}_2|...|\mathbf{d}_n) \in \mathbb{R}^{d \times n}$. The goal is to find a representation vector $\mathbf{r} \in \mathbb{R}^n$ such that $\mathbf{x} \approx \mathbf{D}\mathbf{r}$ and $\mathbf{r}$ is the sparsest, i.e., it has as few non-zero entries as possible. The dictionary $\mathbf{D}$ is assumed to have more columns than rows ($d < n$) and to have full column rank. The vector $\mathbf{x}$ can be seen as an observed signal from which we want to recover the sources that produced it, i.e., the atoms $\mathbf{d}_i$ and their associated coefficients r_i. The motivation for

sparsity in $\mathbf{r}$ is to provide the simplest possible explanation of $\mathbf{x}$ as a linear combination of as few atom as possible.

This problem is NP-hard in general [28], but there are some approximation algorithms [14].

2.4.1 BvN decomposition as a sparse coding problem

Dufossé et al. [11] formulate the BvN decomposition of a given doubly stochastic matrix $\mathbf{A}$ as a linear system of equations, thus making the link between sparse coding and the BvN decomposition. This link was not explored before my internship.

Let Ω_n be the set of all $n \times n$ permutation matrices. The $n!$ matrices in Ω_n are, without loss of generality, ordered and referred to as $\mathbf{P}_1, \ldots, \mathbf{P}_{n!}$. Dufossé et al. define a matrix $\mathbf{M}$ of size $n^2 \times n!$ as

$$\mathbf{M} = (\mathrm{vec}(\mathbf{P}_1) | \, \mathrm{vec}(\mathbf{P}_2) | \cdots | \, \mathrm{vec}(\mathbf{P}_{n!}))$$

Matrix $\mathbf{M}$ is therefore a dictionary of permutations. Consequently, a BvN decomposition with the smallest number of permutation matrices is the following sparse coding problem

$$\min_{\mathbf{x} \in \mathbb{R}_+^{n!}} \|\mathbf{x}\|_0 \text{ such that } \mathbf{M}\mathbf{x} = \mathbf{a} \, , \tag{4}$$

where $\mathbf{a} = \mathrm{vec}(\mathbf{A})$, $\mathbf{x}$ is a vector of $n!$ nonnegative elements, x_j corresponding to the permutation matrix $\mathbf{P}_j$ and $||\mathbf{x}||_0$ is the number of nonzeros in $\mathbf{x}$. The dictionary is dramatically large even for tiny values of n, making this sparse coding problem very challenging:

- in theory: from a linear algebra standing point, there are a lot of atoms which are linearly dependant and thus there is a lot of redundancy in the dictionary. This implies that a single signal can have a lot of different representations, and finding the sparsest one is very hard
- in practice: the size of the dictionary prohibits storing or manipulating it.

However, we are going to see in the next sections that the structure of the dictionary can be exploited to still use some algorithms from the sparse coding literature.

2.4.2 Matching pursuit

The generalized Birkhoff family is closely related to the Matching Pursuit (MP) algorithm from the sparse coding literature [25]. MP is a greedy iterative algorithm that gradually finds the locations of non-zeros in $\mathbf{x}$ one at a time. Everything presented in this subsection is valid for any dictionary $\mathbf{M}$ and observation vector $\mathbf{a}$. We will specialize the algorithms in Section 3 to handle the dictionary of permutations we are interested in.

The MP algorithm finds the column (atom) in $\mathbf{M}$ that is the most correlated with the current residual (initialized to $\mathbf{a}$). It then removes this atom from the residual and repeats the process. Finding the most correlated atom is done by solving the selection problem

$$\arg\max_{j \leq n!} |\mathbf{M}_j^T \mathbf{r}^{(i)}|$$

where $\mathbf{r}^{(i)}$ is the residual at step i. Updating the residual to obtain $\mathbf{r}^{(i+1)}$ can be done by projecting the previous residual $\mathbf{r}^{(i)}$ on the hyperplane orthogonal to the atom chosen at step i, in simpler words we remove the atom from the residual with a coefficient $\frac{1}{\|\mathbf{M}_j\|_2^2} \mathbf{M}_j^T \mathbf{r}^{(i)}$.

Algorithm 2 Matching pursuit

Input: dictionary $\mathbf{M}$ and vector $\mathbf{a}$
Output: the vector $\mathbf{x}$, a sparse approximate solution to $\mathbf{Mx} = \mathbf{a}$
Let $\mathbf{r}^{(0)} \leftarrow \mathbf{a}$ and $\mathbf{x} \leftarrow 0$;
$i \leftarrow 0$
while not converged **do**
    $j \leftarrow \arg\max_{j \leq n!} \{|\mathbf{M}_j^T \mathbf{r}^{(i)}|\}$
    compute coefficient α_j for $\mathbf{M}_j$
    $\mathbf{r}^{(i+1)} \leftarrow \mathbf{r}^{(i)} - \alpha_j \mathbf{M}_j$
    $i \leftarrow i + 1$
end while

MP is therefore similar to the Birkhoff heuristic, as it finds at each step an atom present in $\mathbf{a}$ and removes it. However, the update rule for the coefficients in the MP algorithm may not guarantee that the residuals $\mathbf{a} - \mathbf{Mx}^{(i)}$ are nonnegative. When this happens, the matrix in the next iteration cannot be a (scaled) doubly stochastic matrix and hence the decomposition cannot correspond to (1). We elaborate on this to develop novel algorithms in the next section.

A similar observation on dictionary learning is used by Valls et al. [32], who connect the Frank-Wolfe algorithm [18] to the Birkhoff's heuristics. In this context, Frank-Wolfe has the same selection rule as MP, but a different coefficient update method that guarantees that the solution stays in the space of doubly stochastic matrices at all times throughout the iterative algorithm.

2.4.3 Orthogonal Matching pursuit

A significant issue with generalized Birkhoff heuristics, MP or Frank-Wolfe is that they always estimate permutation weights sequentially, leading to suboptimality on instances (3). A solution to this issue, also quickly explored in [32], is to improve upon the coefficients update rules. To this end, we will use the framework of Orthogonal Matching Pursuit (OMP). Algorithm 3 is the standard OMP algorithm from the literature [14, p. 65], but in the next Section we will part from the usual formulation to derive a class of generalized OMP heuristics for the BvN decomposition.

Algorithm 3 Orthogonal matching pursuit

Input: a given matrix $\mathbf{M}$ and a given vector $\mathbf{a}$
Output: the vector $\mathbf{x}$, a sparse approximate solution to $\mathbf{Mx} = \mathbf{a}$
Let $S^{(0)} \leftarrow \emptyset$ and $\mathbf{x}^{(0)} \leftarrow 0$
$i \leftarrow 0$
while not converged **do**
    $S^{(i+1)} \leftarrow S^{(i)} \cup \{j\}$ where $j = \arg\max_{j \notin S^{(i)}} \{|\mathbf{M}_j^T(\mathbf{a} - \mathbf{Mx}^{(i)})|\}$ ▷ OMP_1
    $\mathbf{x}^{(i+1)} \leftarrow \arg\min_{\mathbf{z}} \{\|\mathbf{a} - \mathbf{Mz}\|_2 \text{ where } \text{supp}(\mathbf{z}) \subseteq S^{(i+1)}\}$ ▷ OMP_2
    $i \leftarrow i + 1$
end while

The notation $\text{supp}(\mathbf{z})$ denotes the support of $\mathbf{z}$, i.e., the set of indices of the nonzero entries of $\mathbf{z}$. In Step 6 of Algorithm 3, similarly to MP heuristics, we choose the index of the largest absolute value in the vector $\mathbf{M}^T(\mathbf{a} - \mathbf{Mx}^{(i)})$, and add it to the solution support S. However in Step 7, we now find the best solution vector whose nonzero indices are in

the current set S, therefore updating all coefficients at each iteration.

It is worthwhile to note that variants of OMP have been studied rigorously in the literature. Of particular relevance is the Nonnegative OMP heuristic [29], which imposes nonnegativity in Step 7 for the coefficients and has similar performances and guarantees as the standard OMP.

3 OMP based algorithm for BvN decomposition

3.1 Algorithm

Here, I explain the proposed OMP-based methods to solve (4) in order to obtain effective heuristics for the BvN decomposition. The principal traits of the proposed approach are as follows: (i) it is based on the solid OMP-framework to obtain sparse solutions; (ii) it generalizes and improves the BvNG heuristic [12] in multiple aspects, including obtaining optimum solutions that cannot be computed by any generalized Birkhoff heuristic.

Let us start by examining (4) and Steps 6 and 7 of Algorithm 3. As the matrix $\mathbf{M}$ has $n!$ columns, it cannot be stored when carrying out the Step 6. We will exploit the special structure of $\mathbf{M}$ for this step. As highlighted before, one needs to guarantee that the matrix entries are nonnegative to obtain a BvN decomposition (1). We will present approaches to do so in the OMP framework for Step 7, to obtain a suitable heuristic for sparse BvN decompositions.

Step 6 of Algorithm 3 finds the column of $\mathbf{M}$ having the largest inner product with the residual $\mathbf{a} - \mathbf{M}\mathbf{x}^{(i)}$. We first realize that computing the residual translates to the matrix operations $\mathbf{A}^{(i)} \leftarrow \mathbf{A} - \sum_{j \in S^{(i)}} x_j \mathbf{P}_j$, therefore it can be computed efficiently, as we do not need to form $\mathbf{M}$ explicitly. We then recognize that $\mathbf{M}^T$ contains n nonzeros in each row, which are all one, and that those nonzeros in a row define a permutation matrix. Therefore $\mathbf{M}^T(\mathbf{a} - \mathbf{M}\mathbf{x}^{(i)})$ translates to computing the value $\sum \mathbf{P} \odot \mathbf{A}^{(i)}$ for all permutation matrices $\mathbf{P}$ and storing them in a vector. Obviously, the arg max in Step 6 selects the maximum of this vector.

As we have seen in Section 2.1, $\sum \mathbf{P} \odot \mathbf{A}^{(i)}$ is the weight of the perfect matching associated with $\mathbf{P}$ in the bipartite graph of $\mathbf{A}^{(i)}$. Therefore, Step 6 of Algorithm 3 can be solved by finding a maximum weight perfect matching in bipartite graph of $\mathbf{A}^{(i)}$. A minimum weight perfect matching (which we will abbreviate MWPM) can be obtained using for instance the Hungarian algorithm [22] in time $O(n^2 m)$.

Step 7 of Algorithm 3 solves a least squares (LS) problem which is efficiently solvable by off-the-shelf methods. Since a solution to an LS problem can contain both negative and positive components, and the BvN decomposition asks for positive only, we should solve a nonnegative LS problem here, as in nonnnegative OMP [29]. Furthermore, one needs to ensure that the residual is nonnegative, that is $\mathbf{a} - \mathbf{M}\mathbf{x}^{(i)} \geq 0$, at the ith iteration. This is necessary for continuing with finding perfect matchings, and also for making sure that the remaining matrix when multiplied with $\frac{1}{\sum_{j=1}^{i} \alpha_j}$ is doubly stochastic. Therefore, Step 7 is reformulated as

$$\underset{\mathbf{z} \geq 0,\ \mathbf{a} \geq \mathbf{M}\mathbf{z}}{\arg\min} \left\{ \|\mathbf{a} - \mathbf{M}\mathbf{z}\|_2 \text{ where } \operatorname{supp}(\mathbf{z}) \subseteq S^{(i+1)} \right\}. \tag{QP}$$

With this formulation in Step 7, we have thus a heuristic based on the OMP methodology to find a BvN decomposition of a given doubly stochastic matrix. Note that (QP) can be

solved via quadratic program solvers.

While the above development follows the OMP-framework, we need to part from this strict interpretation. This is so, as the additional conditions in Step 7 breaks the original design of OMP. Indeed, the problems solved at Step 6 and Step 7 were motivated by the following inequality on successive residuals [14, Lemma 3.3]:

$$\|\mathbf{a} - \mathbf{M}\mathbf{x}^{(i+1)}\|_2^2 \leq \|\mathbf{a} - \mathbf{M}\mathbf{x}^{(i)}\|_2^2 - \frac{1}{n}|(\mathbf{M}_j^T(\mathbf{a} - \mathbf{M}\mathbf{x}^{(i)})|^2 \tag{5}$$

where j is the index chosen at Step 6. It gives a guaranty on the progress made by the algorithm at each step toward approximating the target vector. Introducing the non-negative residual constraint breaks this result, which is key to the performance of the original OMP algorithm. We therefore propose to minimize another loss function in Step 7 for which we can get an inequality similar to (5) while verifying our additional constraints.

Let $\mathbf{z}$ be a feasible vector, that is $\mathbf{z} \geq 0$ and $\mathbf{a} \geq \mathbf{M}\mathbf{z}$. Then, $\|\mathbf{z}\|_1 = \sum z_i$, and $\|\mathbf{a} - \mathbf{M}\mathbf{z}\|_1 = n(1 - \sum z_i)$. This is so, as each row/column of $\mathbf{A}$ adds up to 1 and $\mathbf{a} - \mathbf{M}\mathbf{z}$ translates to subtracting a total of $\sum z_i$ from each row/column using the permutation matrices identified by the support of $\mathbf{z}$. Minimizing the quantity $1 - \sum z_i$, or maximizing $\sum_i z_i$, in Step 7 will therefore minimize an upper bound on $\|\mathbf{a} - \mathbf{M}\mathbf{z}\|_2$, since $\|\mathbf{v}\|_1 \geq \|\mathbf{v}\|_2$ for any vector $\mathbf{v}$. Once we choose $\sum z_i$ as the objective function, we should modify Step 6 to be relevant to this objective function. Let $\mathbf{x}^{(i)}$ be the current solution where $\text{supp}(\mathbf{x}^{(i)}) \subseteq S^{(i)}$, and j be the permutation matrix selected in Step 6. Since the vector $\mathbf{z}' = [\mathbf{x}^{(i)}, z'_j]$ for $0 \leq z'_j \leq \min\{\text{vec}(\mathbf{P}_j) \odot (\mathbf{a} - \mathbf{M}\mathbf{x}^{(i)})\}$ is supported by $S^{(i)} \cup \{j\}$, and it holds that $\mathbf{z}' \geq 0$, and $\mathbf{a} - \mathbf{M}\mathbf{z}' \geq 0$, choosing j which maximizes $\min\{\text{vec}(\mathbf{P}_j) \odot (\mathbf{a} - \mathbf{M}\mathbf{x}^{(i)})\}$ guarantees an improvement of at least $\min\{\text{vec}(\mathbf{P}_j) \odot (\mathbf{a} - \mathbf{M}\mathbf{x}^{(i)})\}$ in the objective function $\sum z_i$. We can thus use:

$$\underset{\mathbf{z} \geq 0,\ \mathbf{a} \geq \mathbf{M}\mathbf{z}}{\arg\max} \{\|\mathbf{z}\|_1 \text{ where } \text{supp}(\mathbf{z}) \subseteq S^{(i+1)}\} \tag{LP}$$

as the objective function in Step 7, with a BPM solver for Step 6. As its name indicate, (LP) is a linear program for which very efficient solvers exist. With this choice of subroutines we get an inequality similar to (5):

$$\|\mathbf{a} - \mathbf{M}\mathbf{x}^{(i+1)}\|_1 \leq \|\mathbf{a} - \mathbf{M}\mathbf{x}^{(i)}\|_1 - n(\min\{\text{vec}(\mathbf{P}_j) \odot (\mathbf{a} - \mathbf{M}\mathbf{x}^{(i)})\}). \tag{6}$$

I will use GompBvN(BPM,LP) to denote the proposed OMP-based solver, where the first parameter designates the problem solved in Step 6, and the second one designates the problem solved in Step 7. The strict OMP-inspired method is similarly referred to as GompBvN(MWPM,QP). Obviously, the other combinations GompBvN(BPM,QP) and GompBvN(MWPM,LP) are possible.

As we have made multiple modifications to the original OMP algorithm, let us give at Algorithm 4 the pseudo-code for GompBvN(BPM,LP) as an illustration.

3.2 Properties

The standard OMP algorithm has been extensively studied in the literature and many properties on its performance have been proven (see [14] for some of them). However, we part from the exact formulation of the original algorithm and our dictionary is very specific so properties like Equation (5) for example are lost. Notably, properties guarantying that we will find the sparsest solution do not apply in our case.

Algorithm 4 GompBvN(BPM,LP)

1: **Input**: $\mathbf{A}$: a doubly stochastic matrix
2: **Output**: $\alpha_1, \ldots, \alpha_k$ and $\mathbf{P}_1, \ldots, \mathbf{P}_k$, where $\mathbf{A} = \sum_{i=1}^{k} \alpha_i \mathbf{P}_i$
3: Let $i \leftarrow 1$, $S \leftarrow \emptyset$, $\mathbf{x} \leftarrow 0$
4: **while** $\sum_{j=1}^{i} x_j < 1$ **do**
5: $\quad \mathbf{A}^{(i)} \leftarrow \mathbf{A} - \sum_{\mathbf{P}_j \in S} x_j^{(i)} \mathbf{P}_j$
6: $\quad$ find a bottleneck perfect matching $\mathbf{P}_i \subseteq \mathbf{A}^{(i)}$
7: $\quad S \leftarrow S \cup \{\mathbf{P}_i\}$
8: $\quad \mathbf{M}_i \leftarrow (\mathrm{vec}(\mathbf{P}_j))_{\mathbf{P}_j \in S}$
9: $\quad \mathbf{x}^{(i+1)} \leftarrow \arg\max_{\mathbf{z} \geq 0, \mathbf{a} \geq \mathbf{M}_i \mathbf{z}} \sum_j z_j$ (LP)
10: $\quad i \leftarrow i + 1$
11: **end while**
12: $k \leftarrow i - 1$

Nevertheless, this subsection highlights some properties of the algorithm GompBvN(BPM,LP). The proof of the results stated are in Appendix A.

First, in the original OMP algorithm at each step a new atom is chosen. This is not enforced in Algorithm 4 and because coefficients are recomputed we might imagine that it could happen. However, we have the following lemma.

Lemma 3.1. *In Algorithm 4 at line 6 we never pick a permutation that is already in S.*

Another property I showed describes the evolution of the number of zero entries in $\mathbf{B}$ at line 5:

Lemma 3.2. *In Algorithm 4, if S contains k permutations then $\mathbf{B}$ has at least k zero entries at line 5.*

This properly shows the convergence of GompBvN(BPM,LP) and allows to get the upper-bounds on the number of components mentioned in Section 2.2.1 for this algorithm.

Another remark allows us to speed up the computation of step 9. Indeed, we can avoid solving the linear program from scratch each time and use the result of the previous step. The simplex algorithm [8] solves a linear program by moving from vertices to vertices on the constraint polytope, starting from an arbitrary point contained in it (e.g., 0 here) and following edges which augment the objective function. We can observe that $\mathbf{x}$ at step i is included in the polytope of the step $i + 1$ and thus we can do a warm-start by taking the previous solution $(\mathbf{x}, 0)$ as a starting point for the next run of the simplex algorithm. This hot-start trick can also be used for the GompBvN($\cdot$,QP) variants. Indeed, the active set algorithm [27] is a method for solving quadratic programming which, similarly to the simplex algorithm, moves from vertices to vertices while augmenting the quadratic objective function. We can thus also do a warm-start in this case.

4 Experiments

In this section I present a selection of experimental results to compare the proposed GompBvN heuristics with the state of the art.

4.1 Code and experimental set-up

I have implemented all algorithms in Python, interfaced throught Cython [1] with some C and Fortran to make use of existing codes for graph matchings. I have used Gurobi [15] solvers for the linear and quadratic programming problems. The code for the MWPM problem is from MC64 [10] (in Fortran) and I used Bottled [31] for BPM (in C). My code is available at `https://gitlab.inria.fr/dlesens/omp_based_algo_bvn_decomp`. Previous implementations of heuristics for the sparse BvN decomposition (mainly [12]) were in Matlab. We chose Python for this project as it is free and more popular.

To test the algorithms, I followed the experimental set-up of [11] and [12]. They use matrices from the SuiteSparse Matrix collection [9] that arise in diverse applications. Those matrices are stored in a sparse format, meaning that only the position of their nonzero entries (as well as their values) are stored. Sparse matrices can be handled in Python using Scipy [34] and its `scipy.sparse` library. All the external codes/libraries mentioned in the first paragraph are designed for sparse matrices.

Because the SuiteSparse collection has many matrices, a small sample was selected in a process similar to [11, 12], keeping only matrices having the following properties: square, between 500 and 1000 rows, fully indecomposable, and with at most 50 nonzeros per row. This selection yields 58 matrices from 11 different groups. At most two matrices per group are retained to remove any bias that might be arising from the group. This resulted in 18 matrices given in Table 1. In this table, the column n lists the number of rows, and the column τ gives the number of nonzeros for each matrix. The matrices are preprocessed as explained in Section 2.3 by taking the absolute values of their nonzeros and scaling them to be doubly stochastic by the method by Knight and Ruiz [19], until the maximum deviation of the row and column sums from 1 were less than 10^{-6}. The different algorithms are run until the coefficients obtained add up to at least 0.999.

4.2 Experimental results

The results comparing the heuristics presented in this report can be found in Table 1. As I mentionned in Section 2.2.2, BvNG produces a significantly smaller number of terms than Birkhoff's original algorithm. For the sake of completude and to illustrate this claim in this report, I first reproduced for our set of matrices the experiments from [12] comparing those two algorithms. For Birkhoff, I stopped the algorithm at 1000 iterations and the sum of coefficients in the first subcolumn gives an indication of the progress made before the interruption. BvNG always outperforms Birkhoff by a significant margin.

Table 1 does not contain any results from GompBvN(MWPM, QP) as in many instances it obtained a larger number of permutation matrices with much more computing time than GompBvN(BPM, LP). Therefore, we give only a few results with GompBvN(MWPM, QP) for the sake of completeness. For example on matrices 662_bus and EX1, GompBvN(MWPM, QP) obtained 116 and 90 permutation matrices, which are larger than other numbers given in Table 1. On some matrices, for example on bcsstk34, the run time was large. On this matrix, GompBvN(BPM, LP) runs in less than 20 seconds on a laptop with 2,5 GHz Intel Core i7 with 16GB memory and obtains 118 permutation matrices, but GompBvN(MWPM, LP) does not deliver a result within 10 minutes (we have stopped it at its 81st iteration).

We observe from Table 1 that in cases where BvNG is effective so is GompBvN(BPM, $\cdot$). This is of course not a surprise, as at a given iteration the solution space in Step 7 of GompBvN(BPM, $\cdot$) includes the coefficients found by BvNG, should they

Table 1: Performance of heuristics on a set of matrices from the SuiteSparse matrix collection. We give the number of component found by each algorithm, which are run until the sum of coefficients is greater than 0.999. In the case of Birkhoff, we stop the iteration at $k = 1000$ and give the sum of coefficients as an indication of the progress made.

Matrix			Birkhoff			GompBvN	
Name	n	τ	sum coeff	k	BvNG	(BPM,LP)	(BPM,QP)
EX1	560	8736	0.467	1000	56	56	64
EX2	560	8736	0.396	1000	67	68	70
cdde1	961	4681	0.999	209	22	19	19
cdde2	961	4681	0.999	239	23	16	18
bcsstk34	588	21418	0.388	1000	120	118	119
bcsstm34	588	24270	0.198	1000	174	174	170
ex21	656	18964	0.171	1000	166	165	168
Trefethen_500	500	8478	0.988	1000	69	69	69
ex22	839	22460	0.102	1000	173	171	177
L	956	3640	0.734	1000	59	58	58
ch5-5-b3	600	2400	1.000	4	4	4	4
dynamicSoaringProblem_1	647	5367	0.093	1000	305	316	312
685_bus	685	3249	0.782	1000	41	40	40
662_bus	662	2474	0.999	973	35	34	34
spaceShuttleEntry_1	560	6891	0.022	1000	274	280	259
Trefethen_700	700	12654	0.989	1000	73	73	73
netz4504_dual	615	2342	0.925	1000	30	28	28
Si2	769	17801	0.580	1000	87	86	87

both contain the same set of permutation matrices. The near identical solution quality between BvNG and GompBvN(BPM, $\cdot$) thus not only confirms that GompBvN(BPM, $\cdot$) is effective but also sheds light on the good performance of BvNG.

However, all four variants of GompBvN solve the special $(n+5) \times (n+5)$ case (3) optimally with 10 permutation matrices. This outcome clearly separates the GompBvN solvers from the generalized Birkhoff heuristics, in particular from BvNG which finds 12 permutation matrices. As BvNG and GompBvN(BPM, $\cdot$) apply the same algorithm for selecting the permutation matrices, their first selection is the same. Step 7 then helps GompBvN(BPM, $\cdot$) to revise the coefficients and to deliver optimal results.

Moreover, one can construct a new family of matrices to give further insights into the performance of GompBvN. This set of matrices has two parameters n and k. First, an $n \times n$ permutation matrix $\mathbf{P}$ is created. Then, another k permutation matrices are created, each of which has distinct n/k common elements with $\mathbf{P}$, while having random permutations for the remaining $n - n/k$ elements. $\mathbf{P}$ is multiplied by 2^k, and other permutation matrices with 2^p, for $p = \{0, \ldots, k-2\}$ in arbitrary order, and all $k+1$ scaled permutation matrices are added to produce a doubly stochastic matrix after a straightforward normalization. By construction, a matrix from the (n, k)-family can be decomposed with $k+1$ permutation matrices. BvNG will find $\mathbf{P}$ in the first step and will use the minimum weight $2^k + 1$ (before normalization) and thus cannot find the said decomposition. The same is true of a variant of BvNG which uses MWPM for selecting permutations. We compare BvNG and the four variants of GompBvN in Table 2 for three (n, k) pairs. As seen from this table, GompBvN variants find decompositions with $k+1$ permutation matrices, but BvNG obtains nearly twice larger number of permutation matrices.

In this family of matrices, we take coefficients to be powers of 2 so that algorithms

Table 2: Performance of heuristics on a set of constructed matrices

		GompBvN		GompBvN	
(n,k)	BvNG	(BPM,LP)	(BPM,QP)	(MWPM,LP)	(MWPM,QP)
(100,10)	19	11	11	11	11
(200,15)	29	16	16	16	16
(500,20)	39	21	21	21	21

will pick permutations in decreasing order of coefficients. Both Birkhoff like algorithms and GompBvN will pick the same permutations but GompBvN will recompute at some point the coefficients, thus correcting the error of taking 2^k+1 as first coefficient. Powers of 2 are interesting because if we note $u_j = 2^j$ then for all j, $\sum_{i=0}^{j-1} u_i < u_j$. This implies that even if there are a lot of collisions between permutations, at step i the permutation with biggest bottleneck value will always be the one with coefficient 2^{n-i+1}, thus leading the algorithm to pick permutations in decreasing order of coefficient. All permutations colliding is very unlikely, so actually taking for coefficients any sequence u_j verifying $u_{j-2} + u_{j-1} < u_j$ is enough to observe the behavior of Table 2. This last remark further extends the class of matrices on which GompBvN performs better than BvNG.

Remark: All the results presented before this remark led to the publication of an article untitled "Orthogonal matching pursuit-based algorithms for the Birkhoff-von Neumann decomposition" [23] accepted at EUSIPCO 2024. EUSIPCO is an international conference on signal processing taking place in Lyon this year. I gave in this report more details on my work than in the published paper.

5 Generalised Birkhoff Von Neumann decomposition

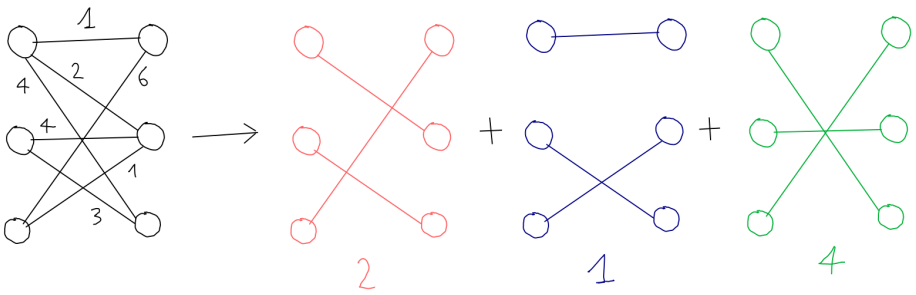
In this section I will present my work on a generalisation of the BvN decomposition. As stated in Section 2.1.3, a way to present Birkhoff's theorem is to say that the convex hull of perfect matchings of an unweighted bipartite graph $G = (R \cup C, E)$ is exactly the set of doubly-stochastic weightings on G. A generalisation would be to consider not only bipartite graphs but also general graphs, as shown in Figure 2. Vazirani [33] proved that in this more general case similarly to Birkhoff's algorithm, there is a polynomial time algorithm which, given a point on the convex hull of perfect matchings, returns a valid decomposition.

First, I will briefly present the problem and the additional difficulties it poses compared to the bipartite case. Due to lack of space I will then only give the complexity improvement achieved by my algorithm on this problem.

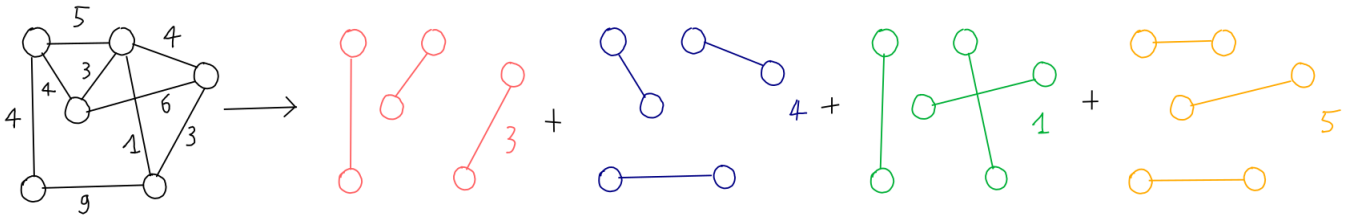
5.1 Overview of the difficulties

We are given an undirected graph $G = (V, E)$, with $|V| = n$ vertices, $|E| = m$ edges, and n is even. For a vertex v, we use $\delta(v)$ to denote the set of edges incident on v, which is extended to a set of vertices S as $\delta(S) = \{e : e \in E, e = (s, v) \text{ with } s \in S, \text{ and } v \notin S\}$ and called the *cut of S*.

In the following, we will consider weightings on $G = (V, E)$. For a weighting x, x_e is the weight of the edge $e \in E$. Edmonds [13] shows that the convex hull of all perfect matchings in G, is defined by the following constraints:



(a) Bipartite graph (Bikhoff theorem)



(b) General graph

Figure 2: Example of decomposition of a fractional perfect matching into a convex combination of perfect matchings. Edge weights do not sum to 1 on a neighborhood, but this can be done by a straightforward scaling.

$$\sum_{e \in \delta(v)} x_e = 1 \quad \text{for all } v \in V \tag{7a}$$

$$\sum_{e \in \delta(S)} x_e \geq 1 \quad \text{for all odd set } S \tag{7b}$$

$$x_e \geq 0 \quad \text{for all } e \in E \tag{7c}$$

A point x of this polytope is a *fractional perfect matching* in G.

An α*-fractional perfect matching* for $1 \geq \alpha \geq 0$ is a point of the polytope obtained by replacing 1's with α's in the right hand side of the polytope specified in (7). That is,

$$\sum_{e \in \delta(v)} x_e = \alpha \quad \text{for all } v \in V \tag{8a}$$

$$\sum_{e \in \delta(S)} x_e \geq \alpha \quad \text{for all odd set } S \tag{8b}$$

$$x_e \geq 0 \quad \text{for all } e \in E \tag{8c}$$

If x is a α*-fractional perfect matching*, then $\frac{1}{\alpha}x$ is a *fractional perfect matching*.

Given a matching M and a set of vertices S, we say that M *crosses* the cut S if $M \cap \delta(S) \neq \emptyset$. In other words, a matching M crosses a cut S, if M matches at least one vertex of S to a vertex in $V \setminus S$. If M is a perfect matching, then M must cross every odd set S (else a vertex remains unmatched in the set). This is captured by the inequality (7b).

We will note $G_x = (V, E_x)$ the graph G with edge weights x where E_x contains only edges e for which $x_e > 0$. A minimum odd cut in G_x is an odd set S with $|S| \geq 3$, where $\sum_{e \in \delta(S)} x_e$ is minimum among all odd sets.

Odd sets S for which constraint (8b) is tight are called tight odd α-cuts.

Remark. *Looking at the polytope (7), a first question one might ask is whether it is in P to test if a point x belongs to the polytope. Indeed, constraints (7a) are easy to test, but there are exponentially many constraints (7b). However, Padberg and Rao show [30] that it is in P to find a minimum odd cut in a weighted graph. Using this algorithm, one can test if x is in Edmond's polytope by checking whether the minimum odd cut has value greater or less than 1.*

Similarly to the bipartite Birkhoff algorithm, we are going to start with a 1-*fractional perfect matching* x, subtract a scalar $b > 0$ times a perfect matching M such that $x - b \cdot M$ is a $(1-b)$-*fractional perfect matching*, and repeat until we have reduced all edge weights to zero (0-fractional perfect matching).

Like in the bipartite case, any perfect matching will preserve the conditions (8a). However, in the general case we have odd set constraints which prevent us from choosing any perfect matching.

Lemma 5.1 (Vazirani [33], Lemma 3)**.** *Let x be an α-fractional perfect matching and let $b \cdot M$ be a term in some decomposition of x, where M is a perfect matching in G_x. Then, M must cross every tight odd α-cut exactly once.*

Proof. This proof is from Vazirani [33, Lemma 3]. M must cross every tight odd α-cut at least once because they are odd sets. Let $x' = x - b \cdot M$. Since $b \cdot M$ is a term in a valid decomposition of x, x' must be a $(\alpha - b)$-fractional perfect matching. If M crosses a tight odd α-cut S a number $r > 1$ times, then x' would not be an $(\alpha - b)$-fractional matching, which is a contradiction: it holds that $\sum_{e \in \delta(S)} x'_e = \alpha - rb$, which violates the constraint in (8b). □

This adds potentially exponentially many constraints to satisfy for a perfect matching M to be in a valid decomposition. In order to verify all those conditions in polynomial time, one needs to maintain a specific data structure on tight odd α-cuts that I will not detail here. This additional cost is what I greatly reduced in my algorithm compared to Vazirani's version, as we will see in the next section.

Finally, there is an additional difficulty regarding the choice of the coefficient which cannot always be $b = \min_{e \in M}(x_e)$, but I will not get into those details.

5.2 Complexity improvement

Vazirani proposes in [33] a polynomial time algorithm to solve this problem. He states that his algorithm has complexity $O(knF + n^3m^2F)$ where k is the number of components produced ($k \leq m$) and F is the cost of a max-flow min-cut computation in a weighted graph, $O(m^{1+o(1)})$ in theory but only implemented in $O(n^2\sqrt{m})$. The term $O(knF)$ is due to the fact that computing a component induces a fixed cost of $O(nF)$ and the second term $O(n^3m^2F)$ comes from additional computations necessary to overcome the difficulties explained in the last section.

His algorithm is very costly and not really implementable as it makes use of very small epsilon for computations which would necessitate a multiprecision framework for manipulating edge weights. Furthermore, the algorithm turns out to be incorrect, and a

Table 3: Performance of heuristics on a set of constructed matrices

n	k	"Any" perf. matching	Bottleneck
50	10	55.4	23.0
100	30	167.8	65.6
200	40	223.4	80.4

correct version would run in $O(knF + n^2m^3F)$ instead.

I proposed an algorithm which is at the same time correct, easier to implement and faster. My algorithm has complexity $O(knF + (n^3 \log(n) + n^2m)F)$ thus maintaining the fixed cost per component but greatly reducing the additional cost. Furthermore, in the best case scenario, my algorithm has only complexity $O(knF)$ whereas Vazirani's algorithm has always at least cost $O(knF + m^3F)$.

5.3 Experiments

A python implementation of my algorithm available at `https://gitlab.inria.fr/dlesens/bvng_algo` supports the claim that it is easy to implement. Furthermore, I implemented an equivalent of BvNG for this generalisation, i.e. a version in which at each step amongst matching crossing all tight odd α-cuts, we pick one with a minimal edge that is maximal.

I compared this bottleneck version with the original one on some artificial instances, built the following way: given n even and k the number of components, pick uniformly at random k perfect matchings in $[\![1, n]\!]$ as well as k coefficients in $[\![1, 10]\!]$ and add them up to form a weighted graph. Such graph is an α-fractional perfect matching, with α equal to the sum of coefficients. I had to restrict myself to such artificial instances as my code is only at a proof of concept phase and can only handle integer edge values. Future versions will be able to handle floating points values like in the bipartite case.

Results are presented in Table 3. For each set of (n, k) I give the average number of components for 5 runs of the algorithm. Similarly to the bipartite case, picking a bottleneck matching greatly reduces the number of components produced, by a factor more than 2.

5.4 Applications

We can represent a graph by its adjacency matrix, defined by $A_{i,j} = w(i, j)$. Because we deal with undirected graphs, this matrix is symmetric. If it is doubly stochastic, with n even and zeros on the diagonal (no self-loop allowed) then the above algorithm decomposes it in a sum of symmetric permutation matrices. We could get a decomposition similar to the one explained in Section 2.3 but which takes as input a symmetric matrix and outputs symmetric components. This could apply to a wide range of matrix as Knight, Ruiz and Uçar [20] present a scaling algorithm preserving symmetry. Work remains to be done in order to handle odd n and nonzeros on the diagonal.

6 Conclusion

In this report, I presented a new family of heuristics for the sparse BvN decomposition of doubly-stochastic matrices. The proposed heuristics are based on the well-known greedy orthogonal matching pursuit (OMP) algorithm, and advances the state of the art in the BvN decomposition problem. Experimental results show that the proposed heuristics

overcome the innate limitation of the state of the art approaches in the literature and are competitive with them on general instances. Interesting future directions would be to find limitations for these heuristics and to explore other options/combinations for the two steps of the algorithm.

I also presented an amelioration of the only known algorithm for the generalisation of the BvN decomposition to non-bipartite graphs. I asymptotically improved the worst case run time complexity and produced the first implementation for this problem. Some work remains to be done on this algorithm to decompose real world symmetric matrices.

References

[1] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. Seljebotn, and K. Smith. Cython: The best of both worlds. *Computing in Science Engineering*, 13(2):31 –39, 2011.

[2] M. Benzi and B. Uçar. Preconditioning Techniques Based on the Birkhoff–von Neumann Decomposition. *Computational Methods in Applied Mathematics*, Dec. 2016.

[3] A. Berman and R. J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences*. SIAM, Philadelphia, PA, USA, 1994.

[4] G. Birkhoff. Tres observaciones sobre el algebra lineal. *Univ. Nac. Tucumán Rev. Ser. A*, (5):147–150, 1946.

[5] S. Bojja Venkatakrishnan, M. Alizadeh, and P. Viswanath. Costly circuits, submodular schedules and approximate carathéodory theorems. *Performance Evaluation Review*, 44(1):75–88, June 2016. Publisher Copyright: © 2016 ACM.

[6] R. A. Brualdi. Notes on the Birkhoff algorithm for doubly stochastic matrices. *Can. Math. Bulletin*, 25(2):191–199, 1982.

[7] R. A. Brualdi and P. M. Gibson. Convex polyhedra of doubly stochastic matrices: I. Applications of the permanent function. *Journal of Combinatorial Theory, Series A*, 22(2):194–230, 1977.

[8] G. B. Dantzig. Linear programming and extensions. In *Linear programming and extensions*. Princeton university press, 2016.

[9] T. A. Davis and Y. Hu. The University of Florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, 2011.

[10] I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM Journal on Matrix Analysis and Applications*, 22:973–996, 2001.

[11] F. Dufossé, K. Kaya, I. Panagiotas, and B. Uçar. Further notes on Birkhoff–von Neumann decomposition of doubly stochastic matrices. *Linear Algebra and its Applications*, 554:68–78, 2018.

[12] F. Dufossé and B. Uçar. Notes on Birkhoff–von Neumann decomposition of doubly stochastic matrices. *Linear Algebra and its Applications*, 497:108–115, 2016.

[13] J. Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of Research of the National Bureau of Standards B*, 69:125–130, 1965.

[14] S. Foucart and H. Rauhut. *A Mathematical Introduction to Compressive Sensing*. Birkhäuser Basel, 2013.

[15] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023.

[16] P. Hall. On representatives of subsets. *Journal of the London Mathematical Society*, s1-10(37):26–30, 1935.

[17] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[18] M. Jaggi. Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization. In *30th ICML*, pages 427–435. PMLR, Feb. 2013.

[19] P. A. Knight and D. Ruiz. A fast algorithm for matrix balancing. *IMA Journal of Numerical Analysis*, 33(3):1029–1047, 2013.

[20] P. A. Knight, D. Ruiz, and B. Uçar. A symmetry preserving algorithm for matrix scaling. *SIAM Journal on Matrix Analysis and Applications*, 35(3):931–955, 2014.

[21] P. Knopp and R. Sinkhorn. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343 – 348, 1967.

[22] H. W. Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[23] D. Lesens, J. E. Cohen, and B. Uçar. Orthogonal matching pursuit-based algorithms for the Birkhoff–von Neumann decomposition. In *Proceedings of EUSIPCO 2024 (to appear)*, Lyon, France, 2024. IEEE.

[24] H. Liu, M. K. Mukerjee, C. Li, N. Feltman, G. Papen, S. Savage, S. Seshan, G. M. Voelker, D. G. Andersen, M. Kaminsky, G. Porter, and A. C. Snoeren. Scheduling techniques for hybrid circuit/packet networks. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '15, New York, NY, USA, 2015. Association for Computing Machinery.

[25] S. G. Mallat and Z. Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, 1993.

[26] M. Marcus and R. Ree. Diagonals of doubly stochastic matrices. *The Quarterly Journal of Mathematics*, 10(1):296–302, 1959.

[27] K. G. Murty and F.-T. Yu. *Linear complementarity, linear and nonlinear programming*, volume 3. Citeseer, 1988.

[28] B. K. Natarajan. Sparse approximate solutions to linear systems. *SIAM Journal on Computing*, 24(2):227–234, 1995.

[29] T. T. Nguyen, C. Soussen, J. Idier, and E.-H. Djermoune. Exact recovery analysis of non-negative orthogonal matching pursuit. In *SPARS*, 2019.

[30] M. W. Padberg and M. R. Rao. Odd minimum cut-sets and b-matchings. *Mathematics of Operations Research*, 7(1):67–80, 1982.

[31] I. Panagiotas, G. Pichon, S. Singh, and B. Uçar. Engineering fast algorithms for the bottleneck matching problem. In *The 31st Annual European Symposium on Algorithms*, Amsterdam, Netherlands, 2023.

[32] V. Valls, G. Iosifidis, and L. Tassiulas. Birkhoff's decomposition revisited: Sparse scheduling for high-speed circuit switches. *IEEE/ACM Transactions on Networking*, 29(6):2399–2412, 2021.

[33] V. V. Vazirani. An extension of the Birkhoff-von Neumann theorem to non-bipartite graphs, 2020.

[34] P. Virtanen et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

A Proofs of properties on GompBvN(BPM,LP)

Lemma A.1 (Lemma 3.1)**.** *In Algorithm 4 at line 6 we never pick a permutation that is already in S.*

Proof. We will show that when we pick a matching, there is a zero included in the range of each $\mathbf{P}$ already in S. This implies that we cannot pick a permutation matrix already in S; it would contain at least one zero.

Suppose by contradiction that there is a permutation $\mathbf{P}_k \in S$ such that there is no zero in the range of $\mathbf{P}_k$ in $\mathbf{B} = \mathbf{A} - \sum_{\mathbf{P}_j \in S} x_j \mathbf{P}_j$. Coefficients $\mathbf{x}$ were obtained by solving at the previous step by solving a problem of the form

$$\underset{\mathbf{z}\geq 0,\ \mathbf{a}\geq \mathbf{M}_i\mathbf{z}}{\arg\max} \|\mathbf{z}\|$$

where $\|\cdot\|$ is any norm. Let $\alpha = \min\{\mathbf{B} \odot \mathbf{P}_k\}$, then $\mathbf{x}' = \mathbf{x} + \alpha \mathbf{e}_k$ is still a valid set of coefficients and $\mathbf{x} < \mathbf{x}'$ so $\|\mathbf{x}\| < \|\mathbf{x}'\|$ and this contradicts the optimality of $\mathbf{x}$. □

Regarding the number of zero entries in $\mathbf{B}$:

Lemma A.2 (Lemma 3.2)**.** *In the algorithm* GompBvN*(BPM,LP), if S contains k permutations then $\mathbf{B}$ has at least k zero entries at line 5.*

Proof. This can be seen by looking at how the simplex algorithm solves the linear programming in order to recompute coefficients. I will assume in this proof that the reader knows about the terminology of the simplex algorithm. I'm going to show that a variable of the objective will enter the dictionary and then never go out of it. If there are k variables in the objective, i.e. k permutations in S, there will thus be k slack variables in the basis, i.e. k zero entries in $\mathbf{B}$. When adding a new permutation to S, we add a new variable to the objective. This variable can immediately be pivoted as $\mathbf{P} \subseteq \mathbf{B}$. Then it can never be pivoted back without strictly diminishing the objective (which is $\sum_{j=1}^{i} x_j$). □