Algorithms for symmetric Birkhoff-von Neumann decomposition of symmetric doubly stochastic matrices

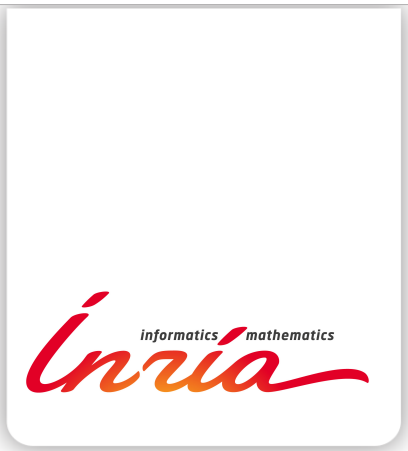
Damien Lesens, Jérémy E. Cohen, Bora Uçar

Algorithms for symmetric Birkhoff–von Neumann decomposition of symmetric doubly stochastic matrices

Damien Lesens, Jérémy E. Cohen, Bora Uçar

RESEARCH REPORT
N° 9566
Jan 2025
Project-Team ROMA

ISSN 0249-6399 ISRN INRIA/RR--9566--FR+ENG



Algorithms for symmetric Birkhoff–von Neumann decomposition of symmetric doubly stochastic matrices

Damien Lesens*, Jérémy E. Cohen†, Bora Uçar‡

Project-Team ROMA

Research Report n° 9566 — Jan 2025 — 30 pages

Abstract: The classical Birkhoff–von Neumann (BvN) decomposition writes a given doubly stochastic matrix as a convex combination of permutation matrices. We investigate the BvN decomposition of symmetric doubly stochastic matrices where the permutation matrices in the decomposition are also symmetric, called symBvN decomposition. This decomposition is not always possible. Two pioneering theoretical works establish the conditions under which such a decomposition is possible for the adjacency matrices of a special class of weighted undirected graphs. Building on these two works, we derive a practical algorithm for the same class of matrices. We also discuss a transformation which converts a given symmetric doubly stochastic matrix, with possibly nonzero diagonal elements, to be the adjacency matrix of an undirected graph. The adjacency matrix of the resulting graph admits a symBvN decomposition if and only if the given matrix does so. This transformation and the practicality of the derived algorithm allow us to propose the first-ever, to the best of our knowledge, implementation of algorithms for symmetric BvN decomposition of symmetric doubly stochastic matrices. We present a set of experiments on decomposing symmetric doubly stochastic matrices as a convex combination of symmetric permutation matrices. Our experiments suggest that the proposed algorithm is as effective as the state-of-the-art algorithms for the classical BvN decomposition.

Key-words: symmetric matrices, Birkhoff–von Neumann decomposition, perfect matchings in graphs

* ENS de Lyon and LIP (UMR5668 Université de Lyon - ENS de Lyon - UCBL - CNRS - Inria), 46 allée d'Italie, ENS Lyon, Lyon F-69364, France.

† CNRS and Creatis, France

‡ CNRS and LIP (UMR5668 Université de Lyon - ENS de Lyon - UCBL - CNRS - Inria), 46 allée d'Italie, ENS Lyon, Lyon F-69364, France.

RESEARCH CENTRE
Centre Inria de Lyon

Bâtiment CEI-2, Campus La Doua
56, Boulevard Niels Bohr - CS 52132
69603 Villeurbanne

Algorithmes pour la décomposition symétrique de Birkhoff–von Neumann des matrices bistochastiques symétriques

Résumé : La décomposition de Birkhoff–von Neumann (BvN) écrit une matrice bistochastique comme une combinaison convexe de matrices de permutation. Nous étudions la décomposition BvN pour des matrices symétriques bistochastiques, en imposant que les matrices de permutation de la décomposition soient aussi symétriques. Cette décomposition est appelée symBvNh et n'est pas toujours possible. Deux travaux théoriques précurseurs ont établis les conditions sous lesquelles la décomposition symBvNh est possible pour les matrices d'adjacence d'une classe spéciale de graphes pondérés non-orientés. En s'appuyant sur ces deux travaux, nous proposons un algorithme pratique pour la même classe de matrices. Nous présentons aussi une transformation qui convertit une matrice symétrique bistochastique donnée, ayant possiblement des entrées diagonales non nulles, en la matrice d'adjacence d'un graphe non-orienté. Cette matrice d'adjacence du graphe obtenu admet une décomposition symBvNh si et seulement si la matrice originale en admet une aussi. Cette transformation et l'aspect pratique de notre algorithme nous permet de proposer la première implémentation d'un algorithme pour la décomposition BvN symétrique de matrices bistochastiques. Munis de cette implémentation, nous expérimentons la décomposition de matrices symétriques bistochastiques en combinaison convexe de matrices de permutation symétriques. Nos expériences indiquent que l'algorithme proposé est aussi efficace que les meilleurs algorithmes connus pour la décomposition BvN classique.

Mots-clés : matrices symmetriques, décomposition de Birkhoff–von Neumann, couplages parfaits dans des graphes

Contents

1 Introduction

A matrix with nonnegative entries is doubly stochastic if all row sums and column sums are one. The well-known Birkhoff–von Neumann (BvN) decomposition of doubly stochastic matrices writes a given doubly stochastic matrix as a convex combination of permutation matrices [3]. Formally, an $n \times n$ doubly stochastic matrix $\mathbf{A}$ can be written as $\mathbf{A} = \sum_{i=1}^{r} \alpha_i \mathbf{P}_i$ for some r, where $\sum_{i=1}^{r} \alpha_i = 1$, $\alpha_i > 0$ and each $\mathbf{P}_i$ is an $n \times n$ permutation matrix. This decomposition is not unique and minimizing the number r of permutation matrices is NP-complete [8].

When $\mathbf{A}$ is a symmetric doubly stochastic matrix, one wonders if a BvN decomposition of $\mathbf{A}$ using only symmetric permutation matrices is possible. This is not in general possible, for example when n is odd and the diagonal is zero. Apart from using the classical BvN decomposition formulation, there are two other ways to adapt the BvN decomposition to symmetric doubly stochastic matrices. First, one can write $\mathbf{A} = \sum_i \alpha_i (\mathbf{P}_i + \mathbf{P}_i^T)$ for any symmetric doubly stochastic matrix $\mathbf{A}$, without the symmetry requirement on the permutation matrices $\mathbf{P}_i$. Here, each individual term $\alpha_i(\mathbf{P}_i + \mathbf{P}_i^T)$ in the decomposition is symmetric. Second, one can characterize a class of symmetric doubly stochastic matrices for which there is a BvN decomposition using only symmetric permutation matrices, and then develop algorithms for this class of matrices. We call such a BvN decomposition in which all permutation matrices are symmetric a *symmetric BvN decomposition* (symBvN decomposition for short).

A class of symmetric doubly stochastic matrices for which there is a symBvN decomposition can be described in terms of graphs. A fractional perfect matching in a general undirected graph $G = (V, E)$ is a vector $x \in \mathbb{R}^E_{\geq 0}$ having an entry x_e for each edge $e \in E$, where $0 \leq x_e \leq 1$ and $\sum_{v \in e} x_e = 1$ for each vertex $v \in V$. The adjacency matrix of a fractional perfect matching is symmetric and doubly stochastic. A perfect matching M in a graph is a subset of edges in which each vertex is included in exactly one edge of M. A *decomposable fractional perfect matching* (decomposable FPM) in a graph G is a fractional perfect matching which can be written as convex combination of perfect matchings in G. The adjacency matrix of a decomposable FPM is therefore symmetric, doubly stochastic, and by definition can be written as a convex combination of symmetric permutation matrices.

There are two known algorithms for decomposing a given decomposable FPM as a convex combination of perfect matchings; we refer to such algorithms as FPMD algorithms (for fractional perfect matching decomposition). Padberg and Wolsey [20] propose an algorithmic framework for some graph decomposition problems. The framework is general enough for deriving an FPMD algorithm. Vazirani [25] independently proposes another FPMD algorithm, which can be viewed as a specialization of the framework described by Padberg and Wolsey. The FPMD algorithms can be used to compute a symBvN decomposition of the adjacency matrices of decomposable fractional perfect matchings.

We observe that the adjacency matrix of a decomposable FPM (i) is a symmetric doubly stochastic matrix; (ii) has an even number of rows/columns; (iii) has zeros in the diagonal; and (iv) is a convex combination of symmetric permutation matrices. We describe a simple transformation which transforms any symmetric doubly stochastic matrix, possibly with nonzero diagonal entries and an odd number of rows/columns, into the adjacency matrix of a graph. The original matrix admits a symBvN decomposition if and only if the transformed matrix is the adjacency matrix of a decomposable FPM. When the transformed matrix is the adjacency matrix of a decomposable FPM, algorithms for FPMD can be used to obtain the symBvN decomposition of the original matrix. Motivated by this broadened application of FPMD algorithms, we continue exploring them along the lines of Padberg and Wolsey [20] and Vazirani [25]. We propose an FPMD algorithm by following Vazirani's approach closely, while simplifing it to obtain a practical algorithm. Our simplifications make the proposed algorithm a more faithful specialization of

the framework of Padberg and Wolsey. Furthermore, the practicality of the presented algorithm allows us to implement it and present a tool for computing a symBvN decomposition of symmetric doubly stochastic matrices, whenever this is doable.

A tool for symBvN decomposition can be used in many different contexts. For example, symmetric doubly stochastic matrices arise naturally in matching markets [21, 23]. Furthermore, matrix scaling algorithms that retain the symmetry of the given matrix [14, 15] can convert a nonnegative symmetric matrix to be doubly stochastic under mild conditions. For a matrix having negative and positive entries, absolute values of the entries can be taken to obtain a doubly stochastic matrix with matrix scaling algorithms. A symBvN decomposition of such a matrix can be used to decompose the original matrix with signed symmetric permutation matrices, in which entries are $\{-1, 0, 1\}$, with a single nonzero in each row and column [2, Section 3.5].

This paper is organized as follows. In the next section, we first give the notation used in the rest of the paper. We then provide some background material related to the classical Birkhoff–von Neumann decomposition, the characterization of decomposable fractional perfect matchings, and the key ingredients to recognize them. In the same section, we also describe the matrix transformation mentioned before. We close that section by briefly describing the relationship between the two existing FPMD algorithms. Section 3 contains the proposed FPMD algorithm, whose implementation details are given in Section 4 along with experiments. We give a summary in Section 4. The appendix contains further explanation of Vazirani's approach and a variant of the proposed algorithm which is closer to that of Vazirani.

2 Background and preliminaries

In the following, we present some background for the symBvN decomposition and the FPMD algorithms.

2.1 Basic graph notation

We start by recalling some common notation and definitions. More specific definitions are given later, when needed.

For a graph $G = (V, E)$, we use $n = |V|$ and $m = |E|$ to refer to the number of vertices and edges, respectively. In a bipartite graph, the vertex set V has two disjoint sets R and C, where all edges are between a vertex from R and another one from C. Undirected graphs are without self-loops. 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 $\delta(S) = \{e : e \in E, e = (s, v) \text{ with } s \in S, \text{ and } v \notin S\}$.

For a vertex set $S \subset V$, the cut $\delta(S)$ is the set of edges having exactly one end point in S. A set $S \subset V$, where $|S|$ is odd, is called odd set. A vertex subset S and its complement $\bar{S}$ defines the cut $\delta(S)$. When $|V|$ is even, an odd set S is sometimes called an odd cut, as both S and $\bar{S}$ are odd.

Let $G = (V, E)$ be a graph with edge weights $x \in \mathbb{R}^E_{\geq 0}$. We use $G_x = (V, E_x)$ to denote an edge induced subgraph of G defined by edges e for which $x_e > 0$. For an edge $e = (i, j)$ we use x_e or x_{ij} to refer to the weight of e. We note $x(\delta(S))$ the value of the cut of S with respect to x, i.e., $x(\delta(S)) = \sum_{e \in \delta(S)} x_e$.

A matching in graph is a set of edges no two of which share a common vertex. A given matching is perfect if all vertices are in an edge of the matching. A perfect matching in a graph G corresponds to a symmetric permutation matrix in the adjacency matrix of G. 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 $V \setminus S$.

2.2 The classical BvN decomposition

Let $\mathbf{A}$ be an $n \times n$ doubly stochastic matrix. The standard bipartite graph $G = (R \cup C, E)$ associated with $\mathbf{A}$ has n vertices in the set R, each corresponding to a unique row of $\mathbf{A}$, and another n vertices in the set C, each corresponding to a unique column of $\mathbf{A}$, such that each $a_{ij} \neq 0$ defines an edge $(r_i, c_j) \in E$ in G. Each permutation matrix in any BvN decomposition of $\mathbf{A}$ corresponds to a perfect matching in G. 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 the matrix $\mathbf{P}$ are also nonzero.

Birkhoff's original proof [3] of the existence of a BvN decomposition is constructive, and leads to the following family of heuristics. Let $\mathbf{A}^{(0)} = \mathbf{A}$. At every step $i \geq 1$, find a permutation matrix $\mathbf{P}_i \subseteq \mathbf{A}^{(i-1)}$, use the minimum nonzero entry of $\mathbf{A}^{(i-1)}$ at the positions identified by $\mathbf{P}_i$ as α_i, set $\mathbf{A}^{(i)} = \mathbf{A}^{(i-1)} - \alpha_i \mathbf{P}_i$, and repeat the computations in the next step until the iteration matrix $\mathbf{A}^{(i)}$ contains only zeros.

2.3 Polytope of fractional perfect matchings

For a bipartite graph $G = (R \cup C, E)$, the Birkhoff theorem states that the polytope

$$\sum_{e \in \delta(v)} x_e = 1 \quad \text{for all } v \in V = R \cup C$$
$$x_e \geq 0 \quad \text{for all } e \in E$$

is exactly the polytope whose vertices are the perfect matchings in G.

Edmonds [9] showed that the convex hull of all perfect matchings in a general graph $G = (V, E)$, with an even number of vertices, is defined by the polytope

$$\sum_{e \in \delta(v)} x_e = 1 \quad \text{for all } v \in V \tag{1a}$$
$$\sum_{e \in \delta(S)} x_e \geq 1 \quad \text{for any odd set } S \subset V \tag{1b}$$
$$x_e \geq 0 \quad \text{for all } e \in E. \tag{1c}$$

We use $\mathcal{P}(G, 1)$ to refer to this polytope. The inequalities (1b) are called the odd set constraints.

For a given graph $G = (V, E)$, let $x \in \mathbb{R}^E_{\geq 0}$ be a vector. If x satisfies (1a), then it is a fractional perfect matching. If in addition x satisfies the odd set constrains (1b), then x is called a decomposable fractional perfect matching (decomposable FPM) and is in $\mathcal{P}(G, 1)$. For example, the edge weighs in the graph $G(\mathbf{A}_1)$ of Figure 1a define a fractional perfect matching, however is not decomposable as the odd set constraint is not satisfied for $S = \{1, 2, 3\}$.

Given an $x \in \mathcal{P}(G, 1)$, a fractional perfect matching decomposition (FPMD) algorithm computes a decomposition of x as a convex combination of perfect matchings. Hence, it can be seen as the extension of Birkhoff's algorithm to general graphs. Observe that such an algorithm computes a symBvN of the adjacency matrix of G_x.

For $1 \geq \alpha \geq 0$, the *α-fractional perfect matching polytope* $\mathcal{P}(G, \alpha)$ of a graph G is obtained by replacing 1's with α's in the right hand side of the polytope constraints (1). We observe that if a vector x is in $\mathcal{P}(G, \alpha)$, then $\frac{1}{\alpha} x$ is in $\mathcal{P}(G, 1)$.

2.4 Minimum odd cuts

Let $G = (V, E)$ be an undirected graph with nonnegative edge weights whose vertices are labelled as odd and even, where there are an even number of odd-labelled vertices. The minimum odd

cut problem asks for a partition V_1, V_2 of V such that both V_1 and V_2 contain an odd number of odd-labelled vertices, and the edge cut $\delta(V_1) = \delta(V_2)$ is minimized. We are interested in a specialization of this problem, in which $|V|$ is even and all vertices are labelled as odd. In this case, the problem asks for an odd set $S \subset V$ such that $\delta(S)$ is the minimum. The minimum odd cut problem is solvable in polynomial time, as shown by Padberg and Rao [19].

Let $G = (V, E)$ be a graph and $x \in \mathbb{R}^E_{\geq 0}$. We can use algorithms for the minimum odd cut problem to see if x is in $\mathcal{P}(G, 1)$. After verifying that the vertex constraints (1a) are satisfied, one needs to test the odd set constraints (1b). For this purpose, one can find a minimum odd cut and check whether the cut is at least 1. If so, all other odd cuts are too.

Some minimum odd sets defined below are central in FPMD algorithms, as we shall see later in Section 3.1.

Definition 2.1. *For a given graph $G = (V, E)$, let x be a point in the α-fractional perfect matching polytope $\mathcal{P}(G, \alpha)$. An odd set S with $|S| \geq 3$ for which the odd set constraint* (1b) *is tight, that is $\sum_{e \in \delta(S)} x_e = \alpha$, is called a* ***tight odd α-cut****. The family of all tight odd α-cuts is denoted by $\mathcal{F}_\alpha = \{S : S$ is a tight odd α-cut$\}$.*

We will be interested in odd cuts with cut value equal to α and crossed multiple times by a perfect matching M. Because singleton cuts have always value α, but are always crossed exactly once by a perfect matching, they are excluded from the definition of tight odd α-cuts.

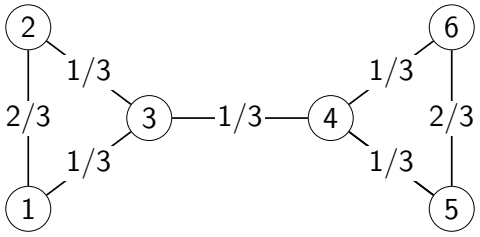
2.5 Decomposing symmetric doubly stochastic matrices

Given a symmetric doubly stochastic matrix, one can decompose it with an algorithm for the classical BvN decomposition, without symmetric terms. One can also use the formulation $\mathbf{A} = \sum_i \alpha_i(\mathbf{P}_i + \mathbf{P}_i^T)$ inside the iterative algorithms summarized in Section 2.2. For this second alternative, in each iteration i, one finds a permutation matrix $\mathbf{P}_i \subseteq \mathbf{A}^{(i-1)}$, which implies $(\mathbf{P}_i + \mathbf{P}_i^T) \subseteq \mathbf{A}^{(i-1)}$. Then, by using $\alpha_i = \max_\beta\{\mathbf{A}^{(i-1)} - \beta(\mathbf{P}_i + \mathbf{P}_i^T) \geq 0\}$, and by updating the iteration matrix symmetrically $\mathbf{A}^{(i)} = \mathbf{A}^{(i-1)} - \alpha_i(\mathbf{P}_i + \mathbf{P}_i^T)$ one obtains a decomposition of the required form. Both of these two alternative decompositions are based on perfect matchings in bipartite graphs. For a symBvN decomposition, an interpretation of the symmetric matrices in terms of general undirected graph is more appropriate. In the rest of the paper we explore this third approach where the given input symmetric doubly stochastic matrix is decomposed using symmetric permutation matrices, assuming such a decomposition is possible.

Not all symmetric doubly stochastic matrices have symBvN decomposition. When a matrix has an odd number of rows and a zero diagonal, for example, such a decomposition is not possible—consider for example the 3×3 matrix, with zeros on the diagonal and $1/2$ elsewhere. Being symmetric doubly stochastic with even number of rows is not in general enough. The 6×6 matrix $\mathbf{A}_1$ shown in Figure 1a does not admit a symBvN decomposition. On the other hand, the adjacency matrix of a decomposable FPM defined on a graph has a symBvN decomposition. Let us look more closely at the adjacency matrix $\mathbf{A}$ of a decomposable FPM defined on a graph. It is a symmetric doubly stochastic matrix and respects the additional constraints stated in (1b): for any subset S of $\{1, \ldots, n\}$ with $|S|$ odd, the sum of the entries a_{ij} for $i \in S$ and $j \notin S$ should be no smaller than one. Such a matrix has a zero diagonal and an even number of rows/columns as it needs to be the adjacency matrix of a graph without self-loops. The matrix $\mathbf{A}_1$ in Figure 1 does not satisfy the odd set constraints (1b) for $S = \{1, 2, 3\}$, and cannot be symmetrically decomposed. On the other hand, $\mathbf{A}_2$ in Figure 1b satisfies all polytope constraints (1) and can thus be decomposed symmetrically.

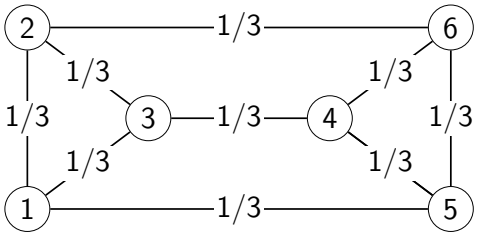
The class of matrices corresponding to fractional perfect matchings is restricted as it contains doubly stochastic matrices with with even size and a zero diagonal. In the next section we show

$$\mathbf{A}_1 = \begin{bmatrix} & 2/3 & 1/3 & & & \\ 2/3 & & 1/3 & & & \\ 1/3 & 1/3 & & 1/3 & & \\ & & 1/3 & & 1/3 & 1/3 \\ & & & 1/3 & & 2/3 \\ & & & 1/3 & 2/3 & \end{bmatrix}$$



(a) A symmetric doubly stochastic matrix $\mathbf{A}_1$ and its graph $G(\mathbf{A}_1)$. The edge weights on $G(\mathbf{A}_1)$ define a fractional perfect matching, but not a decomposable one.

$$\mathbf{A}_2 = \begin{bmatrix} & 1/3 & 1/3 & & 1/3 & \\ 1/3 & & 1/3 & & & 1/3 \\ 1/3 & 1/3 & & 1/3 & & \\ & & 1/3 & & 1/3 & 1/3 \\ 1/3 & & & 1/3 & & 1/3 \\ & 1/3 & & 1/3 & 1/3 & \end{bmatrix}$$



(b) A symmetric doubly stochastic matrix $\mathbf{A}_2$ and its graph $G(\mathbf{A}_2)$. The edge weights on $G(\mathbf{A}_2)$ define a decomposable fractional perfect matching.

Figure 1: Two symmetric doubly stochastic matrices $\mathbf{A}_1$ and $\mathbf{A}_2$, and their corresponding weighted graphs $G(\mathbf{A}_1)$ and $G(\mathbf{A}_2)$.

how to decompose any symmetric doubly stochastic matrix which admits a symmetric BvN decomposition.

2.6 Decompose all matrices that admit a symBvN decomposition

We start by observing that any symmetric matrix $\mathbf{A}$, with possibly nonzeros on the main diagonal, can be transformed into a symmetric matrix with only zeros on the main diagonal and an even number of rows. This can be achieved by the transformation

$$t : \mathbf{A} \longmapsto \begin{pmatrix} \mathbf{A} - \mathrm{diag}(\mathbf{A}) & \mathrm{diag}(\mathbf{A}) \\ \mathrm{diag}(\mathbf{A}) & \mathbf{A} - \mathrm{diag}(\mathbf{A}) \end{pmatrix},$$

where $\mathrm{diag}(\mathbf{A})$ denotes the matrix defined by $\mathrm{diag}(\mathbf{A})_{ii} = \mathbf{A}_{ii}$ and $\mathrm{diag}(\mathbf{A})_{ij} = 0$ if $i \neq j$. If $\mathbf{A}$ is a symmetric doubly stochastic matrix, then $t(\mathbf{A})$ is also, as all entries of $t(\mathbf{A})$ are nonnegative and row/column sums are again one.

Let $\mathbf{A}$ be a symmetric matrix and G be its undirected graph representation, with self-loops corresponding to nonzero diagonal entries of $\mathbf{A}$. The graph G' of $t(\mathbf{A})$ has no self-loops, has an even number of vertices. The transformation thus can be described in terms of graphs G and G' as follows. For an undirected graph G with potentially self-loops, let G' contain two copies $G^{(1)}$ and $G^{(2)}$ of G by ignoring the self loops, and an edge between a vertex in $G^{(1)}$ and its copy in $G^{(2)}$ whenever there is a self-loop in G.

If the vector x corresponding to the nonzeros of $t(\mathbf{A})$ is in $\mathcal{P}(G', 1)$, then x can be decomposed as a convex combinational of perfect matchings in G'. Observe that an edge of a perfect matching in G' is either contained in $G^{(1)}$, or is contained in $G^{(2)}$, or links a vertex in $G^{(1)}$ to its copy in $G^{(2)}$. We can obtain a symBvN decomposition of $\mathbf{A}$ by using the perfect matchings in the decomposition

of x as follows. For one such perfect matching, we use the edges inside $G^{(1)}$ to create the corresponding (two off-diagonal) entries in the permutation matrix, and use the matching edges linking vertices and their copies as diagonal entries in the permutation matrix, and ignore the other matching edges. These observations thus help us characterize which symmetric doubly stochastic matrices admit a symBvN decomposition. We state this formally in the following lemma.

Lemma 2.2. *A symmetric doubly stochastic matrix* $\mathbf{A}$ *is decomposable as a convex combination of symmetric permutation matrices if and only if the weighted graph whose adjacency matrix is* $t(\mathbf{A})$ *satisfy the constraints of the fractional perfect matching polytope.*

Proof. If $\mathbf{A}$ has a symBvN decomposition $\mathbf{A} = \sum_i \alpha_i \mathbf{P}_i$, the vector of edge weights of the graph of $t(\mathbf{A})$ is in the fractional perfect matching polytope, as $t(\mathbf{P}_i)$ is a perfect matching for the graph of $t(\mathbf{A})$, and $t(\mathbf{A}) = \sum_i \alpha_i t(\mathbf{P}_i)$. On the other hand, from a decomposition of the graph G' of $t(\mathbf{A})$, we can read out symmetric permutation matrices for $\mathbf{A}$ as described above. □

2.7 Existing FPMD algorithms

A fractional forest is a convex combination of a set of forests in a given graph. Padberg and Wolsey [20] propose a framework which decomposes a fractional forest into a convex combination of forests. Padberg and Wolsey state that the framework is general enough to solve some other fractional decomposition problems, including fractional matchings. While details of the framework are given for decomposing fractional forests in the paper, the fractional matching case is mentioned only briefly. It has been observed recently by Panageas et al.[21] that Padberg and Wolsey's framework can be applied to the problem of finding a decomposition of a given decomposable FPM in a graph. The key steps in applying the framework to obtain an FPMD algorithm are not given in these works.

Vazirani [25] proposes an FPMD algorithm. Vazirani's algorithm fits into the framework of Padberg and Wolsey, but is described independently from it. This independence allows a modern and accessible presentation. However, Vazirani's approach also includes some additional combinatorial techniques, which renders the algorithm very costly and impractical. It is thus more of a proof that one can decompose a decomposable FPM in polynomial time.

We derive an FPMD algorithm by following that of Vazirani, and by simplifying it with the ideas from the original framework of Padberg and Wolsey. The resulting algorithm is a faithful specialization of Padberg and Wolsey's framework for decomposing a given decomposable FPM in a graph. It is also practical.

3 Algorithm for fractional perfect matchings decomposition

We present the proposed algorithm following Vazirani's presentation and formalism. We need to reproduce some of Vazirani's original lemmas and proofs to make the key ingredients understandable. We also adapt certain proofs by Padberg and Wolsey to show run time complexity bounds.

Similarly to the bipartite Birkhoff algorithm, the decomposition algorithm starts with a given fractional perfect matching x in $\mathcal{P}(G, 1)$. It finds a perfect matching M and a scalar $\beta > 0$, such that the updated vector $x - \beta \cdot M$ is in $\mathcal{P}(G, 1-\beta)$, reports $\beta \cdot M$ as a term in the decomposition, and repeats these steps until all edge weights are reduced to zero. One cannot choose any perfect matching in general. Vazirani demonstrates that for the graph $G(\mathbf{A}_2)$ whose adjacency matrix

is $\mathbf{A}_2$ of Figure 1, one cannot use the perfect matching $(1,5),(2,6),(3,4)$ in the decomposition. This is so, as no matter which coefficient $\beta > 0$ one uses, the odd set $\{1,2,3\}$ will not satisfy the odd-set constraints for $1-\beta$ in the reduced graph $G(\mathbf{A}_2') = G(\mathbf{A}_2) - \beta \cdot M$. Any other perfect matching, for example $(1,2),(3,4),(5,6)$, with the coefficient $1/3$ can be used in a decomposition.

That the coefficient of the sample perfect matching $G(\mathbf{A}_2)$ given above is equal to the minimum entry in the selected positions is accidental, and cannot hold in general. Vazirani uses the Petersen graph shown in Figure 2 to demonstrate this. A suitable decomposition uses each edge twice and has the coefficient $1/6$ in each term, as shown in Figure 2.

After making the insightful observation that one needs to find a suitable perfect matching and then a suitable coefficient for it, Vazirani describes how to accomplish these two tasks. In the next two subsections we describe how we do these by adapting Vazirani's approach. We also give a proof (in Section 3.2.1), on the run time of a subroutine by Vazirani, which was not available in the original work. We then combine ingredients from Padberg and Wolsey [20] to put everything together in an algorithm (in Section 3.3). The worst-case run time complexity of the algorithm is analyzed (in Section 3.4).

3.1 Characterizing suitable perfect matching

Recall that an odd set S with $|S| \geq 3$ for which $\sum_{e\in\delta(S)} x_e = \alpha$ is called a tight odd α-cut. The tight odd α-cuts are key to choose a right perfect matching and a right coefficient.

We start by characterizing the right perfect matchings in the following lemma using tight odd α-cuts.

Lemma 3.1 (Vazirani [25], Lemma 3)**.** *Let $\beta \cdot M$ be a term in some decomposition of a given $x \in \mathcal{P}(G,\alpha)$. Then, M must cross every tight odd α-cut exactly once.*

Proof. This proof is from Vazirani [25, Lemma 3]. By definition, M is a perfect matching in G_x, and hence must cross every odd set, including those that are tight odd α-cuts, at least once. Let $x' = x - \beta \cdot M$. Since $\beta \cdot M$ is a term in a valid decomposition of x, x' must be in $\mathcal{P}(G,\alpha-\beta)$. Suppose for the sake of contradiction that M crosses a tight odd α-cut S a number $r > 1$ times. Then $\sum_{e\in\delta(S)} x'_e = \alpha - r\beta$, therefore S violates the odd set constraint (1b), and hence $x' \notin \mathcal{P}(G,\alpha-\beta)$, which is a contradiction. Therefore, M must cross every tight odd α-cut exactly once. □

The lemma also informs us about the tight odd cuts. We observe from the proof that if S is a tight odd α-cut for an $x \in \mathcal{P}(G,\alpha)$, and M is a term in a decomposition of x with the coefficient β, then S stays a tight odd cut for $x' = x - \beta M$ with $x' \in \mathcal{P}(G,\alpha-\beta)$.

In order to find perfect matchings characterized in Lemma 3.1, Vazirani defines a weight function on the edges of G_x, and show that a minimum weight perfect matching under this weight function helps to select a suitable perfect matching. We define this weight function and then use it in the following lemma.

Definition 3.2. *For a given family $\mathcal{H}$ of tight odd α-cuts on a graph G_x with $x \in \mathcal{P}(G,\alpha)$, the weight function $w_{\mathcal{H}}(\cdot)$ assigns a weight to each edge e of G_x by setting $w_{\mathcal{H}}(e) = |\{S \in \mathcal{H} : e \in \delta(S)\}|$.*

Lemma 3.3. *Let $\mathcal{H}$ be a family of tight odd α-cuts, and $w_{\mathcal{H}}(e)$ be the weights from Definition 3.2. Let M be a minimum weight perfect matching in graph G_x under the weight function $w_{\mathcal{H}}(\cdot)$. Then the weight of M, where $w_{\mathcal{H}}(M) = \sum_{e\in M} w_{\mathcal{H}}(e)$, is $|\mathcal{H}|$, and M crosses each tight odd α-cut in $\mathcal{H}$ exactly once.*

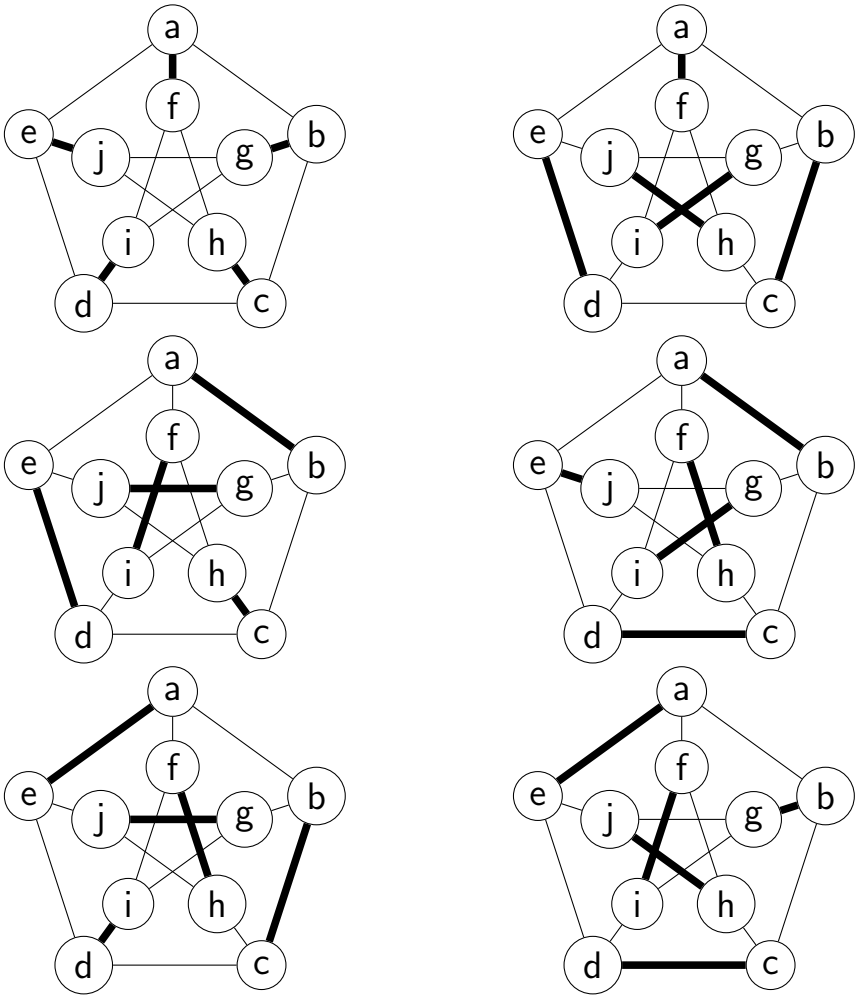


Figure 2: The Petersen graph with edge weights 1/3 and its decomposition into six perfect matchings. The matching edges are shown with thick lines. Each edge is used in two perfect matchings in the decomposition, and the coefficient of each term in the decomposition is 1/6.

Proof. A perfect matching crosses a tight odd α-cut at least once, so the weight of any perfect matching is at least $|\mathcal{H}|$. This lower bound is achieved by perfect matchings crossing each tight odd α-cut exactly once. Those perfect matching exist, as we can read them out from any valid decomposition of G_x. □

Vazirani [25, Lemma 4] uses a more structured subset of tight odd α-cuts, hence our lemma is slightly more applicable.

If we were to use the set $\mathcal{F}_\alpha$ of all tight odd α-cuts from Definition 2.1 in Lemma 3.3 as the family of tight odd α-cuts, a minimum weight perfect matching M in graph G_x under the weight function $w_{\mathcal{F}}(\cdot)$ would cross each tight odd α-cut once, and hence by Lemma 3.1 would be in the output. The number of tight odd α-cuts in $\mathcal{F}$ is exponential in general and hence the weight function $w_{\mathcal{F}_\alpha}(\cdot)$ cannot be computed in polynomial time. We will discuss in later sections how to avoid computing $w_{\mathcal{F}_\alpha}(\cdot)$.

3.2 Finding a suitable coefficient

Suppose we have a perfect matching M crossing all tight odd α-cuts exactly once. We have to find a coefficient for M in the decomposition. As said earlier, we cannot choose coefficient $\beta = \min_{e\in M} x_e$ for M in general. We should choose a coefficient b such that for any odd set S, it must hold that

$$x(\delta(S)) - b \cdot |\delta(S) \cap M| \geq \alpha - b \,.$$

If $|\delta(S) \cap M| = 1$ this constraint is already verified, as $x(\delta(S)) \geq \alpha$ by assumption. Since the equations above need to be verified for all odd sets S with $|\delta(S) \cap M| > 1$, we need to take as coefficient

$$\gamma = \min_{S\in\mathcal{S}} \frac{x(\delta(S)) - \alpha}{|\delta(S) \cap M| - 1} \,, \tag{2}$$

where $\mathcal{S} = \{S : S \text{ is an odd set } |\delta(S) \cap M| > 1\}$, if taking coefficient $\beta = \min_{e\in M} x_e$ yields a minimum odd cut of value strictly less than $\alpha - \beta$. The set S giving the minimum here will be a tight odd set for $\alpha - \gamma$, as we have by definition

$$x(\delta(S)) - \gamma \cdot |\delta(S) \cap M| = \alpha - \gamma \,.$$

Both Padberg-Wolsey and Vazirani proposed two algorithms to compute this coefficient γ. The first is a binary search approach. Indeed, γ has the following equivalent definition

$$\gamma = \arg\max_b \{(x - b \cdot M) \text{ is in } \mathcal{P}(G, \alpha - \beta)\} \,.$$

Both papers prove that a binary search on the values of b will have a polynomial time run time bound. However, both bounds assume rational numbers for the entries of x. They depend on the largest denominator of those weights, which we will note d. Vazirani argues that the common denominator of edge weights during his algorithm is bounded by $O(d^m n^{n/2})$. This implies that the binary search for γ has a worst case run time complexity of $O(\log(d^m n^{n/2}))$ and is thus polynomial. The two papers also propose another algorithm, which we will call LinearSearch to compute γ. This algorithm is more interesting to us, as it is more numerically stable and its complexity does not depend on d.

3.2.1 The subroutine LinearSearch

Vazirani presents the algorithm LinearSearch, shown in Algorithm 1 to compute γ of (2). The idea of the algorithm is the following. Assume we are given a perfect matching M and a minimum odd cut S, for which the odd set constraint is violated if we were to use $\beta = \min_{e \in M} y_e$ as the coefficient. We compute $\gamma = \frac{y(\delta(S))-\alpha}{|S \cap M|-1}$ associated with S, and verify that the coefficient is right for all odd cuts (by examining a minimum odd cut). If not, we repeat the process with another minimum odd cut not respecting the odd set constraints if γ were used as a coefficient.

Algorithm 1 Linear search [25] to compute a coefficient $\gamma = \mathsf{LinearSearch}(G, y, \alpha, M, S)$

Input: α: the value of a fractional perfect matching
y: a point in $\mathcal{P}(G, \alpha)$
M: A perfect matching
S: A minimum odd cut of y with $|\delta(S) \cap M| > 1$
Output: $\gamma = \min_{T \in \mathcal{S}} \frac{y(\delta(T))-\alpha}{|T \cap M|-1}$
1: $T \leftarrow S$
2: **repeat**
3: $k \leftarrow |\delta(T) \cap M|$
4: $\gamma \leftarrow \frac{y(\delta(T))-\alpha}{k-1}$
5: $y' \leftarrow y - \gamma \cdot M$
6: $T \leftarrow$ minimum odd cut in graph $G_{y'}$ with edge weights y'
7: **until** $y'(\delta(T)) \geq \alpha - \gamma$
8: **return** γ

Padberg and Wolsey prove in their general framework that Algorithm 1 finds the correct γ, the γ values seen in the iterations form a decreasing sequence, and the algorithm terminates in at most $n/2$ iterations [20, Theorems 5 and 7]. Vazirani proves the first two points [25, Lemma 10], but the arguments in the proof do not give a bound on the number of iterations. We prove next that LinearSearch takes at most $n/2$ iterations using Vazirani's notations for completeness.

3.2.2 Worst case run time analysis of LinearSearch

We state Vazirani's lemma concerning the γ values seen in the repeat-until loop of Algorithm 1 below, whose proof is clear in the original paper [25, Lemma 10]. We will use this lemma in our run time complexity analysis of LinearSearch.

Lemma 3.4 (Vazirani [25] Lemma 10, Padberg and Wolsey [20] Theorem 7)**.** *The γ values computed at Line 4 of Algorithm 1 strictly decrease from one iteration to the next of the repeat-until loop.*

The lemma focuses on the values of γ, which does not lead to an upper bound on the number of iterations.

We focus on the k values computed at Line 3 as shown in the following theorem. We note that at the beginning of LinearSearch, $k = |\delta(S) \cap M| > 1$ holds; otherwise $\beta = \min_{e \in M} y_e$ would be a valid coefficient. Then, during LinearSearch all k values are larger than one; otherwise the"until" condition will be satisfied.

Theorem 3.5. *In Algorithm 1, the k values obtained at Line 3 do not repeat.*

Proof. Let T_1 and T_2 be two minimum odd cuts seen in the body of the repeat-until loop. Without loss of generality, assume that T_1 is computed before T_2. Let γ_1 and γ_2 be the associated values computed at Line 4, just before finding T_1 and T_2 at Line 6. We have

$$\gamma_2 < \gamma_1 \tag{3}$$

by Lemma 3.4. Consider the moment where T_1 is selected at Line 6, instead of T_2. Since T_1 gives the minimum odd cut, we have

$$y'(\delta(T_1)) \leq y'(\delta(T_2)) \tag{4}$$

where $y' = y - \gamma' \cdot M$ and γ' is the coefficient computed at Line 4 before T_1 is found.

Now suppose for the sake of contradiction that $|\delta(T_1) \cap M| = |\delta(T_2) \cap M| = k$. By plugging in the right hand side formula from Line 4 for γ_1 and γ_2 in (3), we obtain

$$\frac{y(\delta(T_2)) - \alpha}{k-1} < \frac{y(\delta(T_1)) - \alpha}{k-1} \, .$$

Canceling the common terms of both sides we obtain

$$y(\delta(T_2)) < y(\delta(T_1)) \, . \tag{5}$$

Just before selecting T_1 at Line 6, we have

$$y'(\delta(T_1)) = y(\delta(T_1)) - k\gamma' \quad \text{and} \quad y'(\delta(T_2)) = y(\delta(T_2)) - k\gamma'$$

as $|\delta(T_1) \cap M| = |\delta(T_2) \cap M| = k$. Combining with (5), we obtain

$$y'(\delta(T_2)) + \gamma' k < y'(\delta(T_1)) + \gamma' k \, ,$$

which contradicts (4). Hence $|\delta(T_1) \cap M| \neq |\delta(T_2) \cap M|$, for any two minium odd cuts T_1 and T_2 selected by Algorithm 1 in the repeat-until loop. □

We bound the number of iterations of the repeat-until loop of Algorithm 1 using Theorem 3.5.

Corollary 3.6. *In Algorithm 1, the repeat-until loop runs for at most $n/2 - 1$ iterations.*

Proof. For any cut S, odd or even, $|\delta(S) \cap M|$ is at most $|M| = n/2$. We will not iterate on an odd cut S for which $|\delta(S) \cap M| = 1$ as those sets already respect the odd set constraints. Since k can be at most $n/2$ and at least 2, and it does not repeat, we have at most $n/2 - 1$ iterations. □

3.3 Putting it all together

Based on our observation in Lemma 3.3 and guided by the framework of Padberg and Wolsey [20], we propose the FPMD algorithm shown in Algorithm 2. It is a simplification of Vazirani's approach, which is summarized in Appendix A.1.

The algorithm takes as input $x \in \mathcal{P}(G, 1)$, sets $y = x$ at the beginning, and updates y iteratively such that $y \in \mathcal{P}(G, \alpha)$ for nonincreasing α, and stops when y is all zeros. It keeps a set $\mathcal{H}$ of tight odd α-cuts, which is empty at the beginning. At every iteration, it either adds a term to the decomposition, or adds a new tight odd cut to $\mathcal{H}$, or does both. The simplicity of the proposed algorithm is thanks to starting with an empty $\mathcal{H}$ and adding new tight odd α-cuts one by one as the need arises.

At Line 5 of Algorithm 2, a minimum weighted perfect matching M is computed with respect to edge weights $w_{\mathcal{H}}(\cdot)$ of Definition 3.2. In the first iteration, M can be any perfect matching

in G, as $\mathcal{H}$ is empty. We then compute $\beta = \min_{e \in M} y_e$ and test whether adding M to the decomposition with coefficient β is possible at Line 9, in which case the minimum odd cut in $y' = y - \beta \cdot M$ has value $(\alpha - \beta)$. If $\beta \cdot M$ is a term in the decomposition, we can proceed with the next iteration. If not, $\beta \cdot M$ is not a term in a decomposition for possibly two reasons: either M crosses all tight odd α-cuts exactly once but β is not the right coefficient, or M crosses some tight odd α-cut more than once. We need to find the right coefficient in the first case. In the second case, we must update $\mathcal{H}$ so that we can choose a perfect matching crossing more tight odd α-cuts exactly once. We handle both cases using Algorithm 1 with a slight modification.

Algorithm 2 Fractional perfect matching decomposition algorithm $\mathsf{fpmda}(G, x)$

Input: $G = (V, E)$: An undirected graph
x: a decomposable fractional perfect matching in G, that is $x \in \mathcal{P}(G, 1)$
Output: a decomposition of x as a sum of perfect matchings in G

1: $\alpha \leftarrow 1$
2: $y \leftarrow x$
3: $\mathcal{H} \leftarrow \emptyset$
4: **while** $y \neq 0$ **do**
5: $M \leftarrow$ a minimum weight perfect matching in G_y wrt. weights $w_{\mathcal{H}}(\cdot)$
6: $\beta \leftarrow \min_{e \in M} y_e$
7: $y' \leftarrow y - \beta \cdot M$
8: $S \leftarrow$ a minimum odd cut in $G_{y'}$ with edge weights y'
9: **if** $y'(\delta(S)) \geq \alpha - \beta$ **then**
10: **Output** $\beta \cdot M$
11: $\alpha \leftarrow \alpha - \beta$
12: $y \leftarrow y'$
13: **else**
14: $\langle \gamma, S' \rangle \leftarrow \mathsf{LinearSearchCut}(G, \alpha, y, M, S)$
15: **if** $\gamma > 0$ **then**
16: **Output** $\gamma \cdot M$ ▶ M is a valid matching and $\gamma > 0$ is its coefficient
17: $y \leftarrow y - \gamma \cdot M$
18: $\alpha \leftarrow \alpha - \gamma$
19: **end if**
20: $\mathcal{H} \leftarrow \mathcal{H} \cup \{S'\}$ ▶ S' is a tight odd $(\alpha - \gamma)$-cut for $\gamma > 0$, otherwise a tight odd α-cut.
21: update $w_{\mathcal{H}}(\cdot)$
22: **end if**
23: **end while**

A close examination of Algorithm 1 reveals that it can return an odd cut which attains the minimum in γ's definition (2). The modified search procedure is presented explicitly in Algorithm 3. The odd cut which corresponds to the γ value returned is the penultimate one, T_{prev} in Algorithm 3, not the last minimum odd cut computed using y'. If the currently chosen M crosses some tight odd α-cut more than once (because that tight odd α-cut was not in $\mathcal{H}$), then Algorithm 3 returns $\gamma = 0$ and a tight odd α-cut S'. This returned tight odd α-cut S' can then be added to $\mathcal{H}$, which will enforce that the minimum weight perfect matchings with respect to the updated $w_{\mathcal{H}}(\cdot)$ will cross S' exactly once. If $\gamma > 0$, then M crosses all tight odd α-cuts once, and a term of the decomposition is produced. In this case, the tight odd α-cut S' returned by $\mathsf{LinearSearchCut}$ is a tight odd $(\alpha - \gamma)$-cut and can be added to $\mathcal{H}$, which is now a set of tight odd $(\alpha - \gamma)$-cuts.

In the if-case at Line 9 of Algorithm 2, we set at least one edge to zero, so we know that this

Algorithm 3 Linear search returning the cut $\langle \gamma, T_{prev} \rangle = \mathsf{LinearSearchCut}(G, \alpha, y, M, S)$

Input: α: the value of a fractional perfect matching
y: a point in $\mathcal{P}(G, \alpha)$
M: A perfect matching
S: A minimum odd cut of y with $|\delta(S) \cap M| > 1$
Output: γ: which gives $\gamma = \min_{T \in \mathcal{S}} \frac{y(\delta(T)) - \alpha}{|T \cap M| - 1}$
T_{prev}: an odd cut achieving the minimum γ

1: $T \leftarrow S$
2: **repeat**
3: $T_{prev} \leftarrow T$
4: $k \leftarrow |\delta(T) \cap M|$
5: $\gamma \leftarrow \frac{y(\delta(T)) - \alpha}{k-1}$ ▶ $\gamma = 0$ if T is a tight odd α-cut
6: $y' \leftarrow y - \gamma \cdot M$
7: $T \leftarrow$ minimum odd cut in the graph $G_{y'}$ with edge weights y'
8: **until** $y'(\delta(T)) \geq \alpha - \gamma$
9: **return** $\langle \gamma, T_{prev} \rangle$

case will happen at most m times. The bound on the number of else cases happening is more involved to show. Below we bound the number of iterations of the while loop by following the framework of Padberg and Wolsey, treating the if- and the else-cases together.

Observe that Algorithm 2 starts with a $y = x$ with m components. Since for each vertex v one of the components of y in $\delta(v)$ can be determined from others, the decomposition should take care of at most $m - n/2$ free components of y. At each iteration, the algorithm either sets at least one component of y to zero (Line 12), or adds a tight odd α-cut S' at Line 20. Since the odd set inequality $y(\delta(S'))$ is tight for S', one of the components of y in $\delta(S')$ is determined by the others. That is, each iteration of the while-loop adds a constraint among the free components of the input x. If all these constraints are linearly independent, then Algorithm 2 should terminate in at most $m - n/2 + 1$ iterations. Below, we show that all added constraints are linearly independent.

Let us note $\chi(F) \in \{0, 1\}^{|E|}$ the indicator vector of an edge set $F \subseteq E$, and denote the inner product of two vectors $\mathbf{v}$ and $\mathbf{w}$ by $\langle \mathbf{v}, \mathbf{w} \rangle$. The algorithm enforces a constraint of the form $\langle \chi(\{e\}), y \rangle = 0$, when $\beta = \min_{e \in M} y_e$ is chosen as coefficient, or $\langle \chi(\delta(S)), y \rangle = \alpha - \gamma$, when γ is the coefficient. Let us note $\mathcal{C}$ the set of vectors encoding the constraints imposed on y by the algorithm. The set $\mathcal{C}$ is of the form $\mathcal{C} = \{\chi(\delta(S)) | S \in \mathcal{H}\} \cup \{\chi(\{e\}) |$ for some $e \in E\}$. Observe that the number of elements of $\mathcal{C}$ is the number of iterations of the algorithm. We will show that $\mathcal{C}$ is a linearly independent family of vectors in $\mathbb{R}^{|E|}$.

Theorem 3.7 (Adaptation of Theorem 6 from Padberg and Wolsey [20])**.** *In all iterations of Algorithm 2 until the last one, the family of vectors $\mathcal{C}$ is linearly independent.*

Proof. Let us index the tight odd sets in $\mathcal{H}$ by i and the edges set to 0 by j. We show that a new constraint added is linearly independent of the elements of $\mathcal{C}$ if it is not the last iteration.

Let $\chi(F)$ be a new element added to $\mathcal{C}$. Assume for the sake of contradiction that there exist coefficients λ_i and μ_j not all 0 such that

$$\chi(F) = \sum_i \lambda_i \chi(\delta(S_i)) + \sum_j \mu_j \chi(\{e_j\}) \,. \tag{6}$$

Let M be the perfect matching computed before $\chi(F)$ is added to $\mathcal{C}$.

If we compute the inner product of the vectors on both sides of (6) with the vector $y - b \cdot M$ for any $b \in \mathbb{R}$ we get

$$\langle \chi(F), y - b \cdot M \rangle = \sum_i \lambda_i (\alpha - b) \ . \tag{7}$$

Indeed $\langle \chi(\{e_j\}), y - b \cdot M \rangle = 0$ for all j as the edges e_j are already set to 0, and $\langle \chi(\delta(S_i)), y - b \cdot M \rangle = \alpha - b$ as tight odd α-cuts in $\mathcal{H}$ are crossed exactly once by M.

We have two cases depending on the type of constraint $\chi(F)$ imposes. If $F = \{e\}$ with e achieving the minimum in $\beta = \min_{e \in M} y_e$ we get

$$\beta - b = \sum_i \lambda_i (\alpha - b) \ .$$

We have two affine functions of b, and hence the coefficients should be the same; therefore $\beta = \sum_i \lambda_i \alpha$ and $\sum_i \lambda_i = 1$ and hence $\beta = \alpha$. Since $\beta = \alpha$, y is the only perfect matching and we must be at the last iteration.

The second case is $F = \delta(S)$ for an odd set S. As this happens in the else-case at Line 13, we must have

$$|\delta(S) \cap M| > 1 \ .$$

For $F = \delta(S)$, the equality (7) reads as

$$y(\delta(S)) - |\delta(S) \cap M| \cdot b = \sum_i \lambda_i (\alpha - b) \ . \tag{8}$$

We have two affine functions in b which are equal for all $b \in \mathbb{R}$, hence

$$|\delta(S) \cap M| = \sum_i \lambda_i \tag{9}$$

should hold for any b. The algorithm chooses $b = \gamma$ in which case the equality (8) reads

$$\alpha - \gamma = \sum_i \lambda_i (\alpha - \gamma) \ .$$

This implies that $\sum_i \lambda_i = 1$, but this equality combined with (9) contradict the fact that $|\delta(S) \cap M| > 1$. Hence, the coefficients λ_i and μ_j satisfying (6) do not exist. □

Observe that given a graph $G = (V, E)$ and an $x \in \mathcal{P}(G, 1)$, Algorithm 2 decomposes the graph $G_x = (V, E_x)$. Letting $\mathbf{A}$ be the adjacency matrix of G_x, where $a_{ij} = a_{ji} = x_{ij}$, we can collect the output β and M pairs as terms of the decomposition of $\mathbf{A}$.

3.4 Analysis of the worst-case run time complexity

The FPMD algorithm uses solvers for the minimum weighted perfect matching problem and the minimum odd cut problem (based on the Padberg-Rao algorithm). We will analyse the cost of the algorithm principally in terms of those solvers, denoted, respectively, as MWPM and PR. We do so as both problems can be solved using many different algorithms, for which there are theoretical algorithms and practical, state-of-the-art implementations with varying run time complexity.

Let r denote the number of terms produced by the algorithm. We saw in the last section that r is at most $m - n/2 + 1$, or $O(m)$. At each iteration, a call to the minimum weight perfect matching algorithm is made. Then Padberg-Rao algorithm is called to test whether $\beta \cdot M$ is a

valid term. If it is, we proceed to the next iteration. If not, LinearSearchCut is called and costs $O(n)$ Padberg-Rao calls.

All other operations including computing β, updating the graph, updating $w_{\mathcal{H}}(\cdot)$ have cost $O(n+m)$ per iteration and are negligible. The total cost of the proposed FPMD algorithm is thus $O(m(\text{MWPM} + n\text{PR}))$, which is much lower than the cost $O(n^2m^2\text{PR})$ of Vazirani's algorithm [25, Theorem 16].

The variation of the proposed FPMD algorithm presented in the Appendix A.2 has an asymptotic worst case run time complexity of $O(m(\text{MWPM}+\text{PR})+n^2\text{PR})$, which is smaller than that given above for Algorithm 2. That variant keeps and updates a more involved family of tight odd α-cuts to reduce the worst-case run time upper bound, without tangible impact in the performance. On the other hand, the presented approach Algorithm 2 is easier, and lends itself to better instrumentation for handling floating point values in an implementation.

The Padberg-Rao algorithm costs $O(n^2+\text{GH})$ where GH is the cost of computing a Gomory-Hu tree [12]. The currently best theoretical complexity for computing a Gomory-Hu tree [1] is $O(m^{1+o(1)})$, matching the best run time complexity for the max-flow problem. The state of the art implementation for computing a Gomory-Hu tree is provided by Kolmogorov [17], and compared to other implementations in the computational study [16].

The minimum weighted perfect matching problem is equivalent to the maximum weighted perfect matching problem by replacing edge weight $w_{\mathcal{L}}(e)$ by $N - w_{\mathcal{L}}(e)$ where N is the maximum edge value in the graph. The current best algorithm for integer weights by [6] has complexity $O(m\sqrt{n}\log(nN))$. The LEMON library proposes an implementation with complexity $O(nm\log(n))$.

4 Implementation and experiments

We now come back to decomposing symmetric doubly stochastic matrices, by transforming them into fractional perfect matchings as explained in Section 2.6 and using the algorithm introduced in Section 3.3. The implementation of the proposed fpmda method, shown in Algorithm 2, for decomposing matrices is called symBvNh in the rest of this section.

We present our C++ implementation of symBvNh as well as some experiments. The code is available at `https://gitlab.inria.fr/dlesens/symbvn`. Experiments were conducted on an AMD Ryzen 7 6800H 3.2Ghz CPU, and the code was compiled with the optimization flag `-O4`.

The C++ code uses the LEMON library [5] for its graph templates, the maximum weighted perfect matching algorithm, and the perfect matching algorithm. As explained in Section 3.4, a minimum odd cut algorithm is invoked at least once per iteration of fpmda and hence its efficiency is critical for the overall performance. We have carefully implemented the minimum odd cut algorithm of Padberg-Rao using the Gomory-Hu tree algorithm [12]. In a recent study, Kolmogorov [16] compares a number of Gomory-Hu algorithms (including LEMON's implementation) and identifies an algorithm called OrderedCut [17] as the state of the art. We use OrderedCut in our codes.

Updating the weights $w_{\mathcal{H}}$ when a new cut S is added to $\mathcal{H}$ is done by subtracting one from the weight of all edges in $\delta(S)$, (or by incrementing by one for the minimum weighted perfect matching case).

4.1 Handling floating point values

We pay special attention to floating point values in our implementation so that our codes can be used with doubly stochastic matrices, not necessarily having integer weights divided by a common constant. In such cases, having all row sums and column sums exactly equal to one is

rare. A matrix is nearly doubly stochastic if the maximum deviation of a row sum or a column sum from one is bounded. This is typically the case when a doubly stochastic matrix is obtained from a nonnegative one by an iterative algorithm [14, 15, 24]. Let τ be the maximum deviation of a row sum or a column sum from one. This accuracy will translate to the corresponding fractional perfect matching, where τ will be the maximum deviation from 1 of the sum of edge weights incident on a given vertex. We consider that a fractional perfect matching built from a nearly doubly-stochastic matrix is decomposable if its minimum odd cut has value at least $1-\tau$. We note α_{init} the value of the minimum odd cut before starting the algorithm. Let us define ε to be the desired accuracy in the symBvN decomposition, that is $\varepsilon = 1 - \sum_i \alpha_i$. We stop the decomposition when the accumulated sum of coefficients is more than $1-\varepsilon$. One cannot obtain a decomposition in which $\varepsilon < \tau$. This difference then can be used to handle precision errors due to using floating values.

One way to avoid precision errors is to remove, if possible, comparisons of floating point values at the critical places in the algorithm. The if-statement at Line 9 in Algorithm 2 is one of them. We can avoid floating point value comparison there, by replacing the condition "$y'(\delta(S)) \geq \alpha - \beta$" with "$|\delta(S) \cap M| = 1$". The latter condition implies the former, as a tight odd cut crossed once satisfies the former condition by equality. The reverse implication is not true: we might at some point go in the "else-statement" despite the minimum cut value being $\alpha - \beta$. In this case, S is a tight odd $(\alpha-\beta)$-cut so the algorithm will add an element to $\mathcal{H}$, after reporting a term in the decomposition. Hence, we can use the test "$|\delta(S) \cap M| = 1$" at Line 9 in Algorithm 2.

There are two other critical places at which special care is required. One of them is the update $y' \leftarrow y - \beta \cdot M$, where at least one edge should be set to 0. The other is inside LinearSearchCut, in which we exit the loop if $y'(\delta(T)) = \alpha - \gamma$. To handle those cases carefully, let `myZero`$= \frac{\varepsilon-\tau}{2m}$ be a threshold value, which we use to test equality in the implementation. We set an edge to 0 if its value is less than `myZero`, and we exit the loop of LinearSearchCut, when $y'(\delta(T)) > \alpha - \gamma -$ `myZero`. By using those approximations, it may happen that the value α of the minimum odd cut in the graph at iteration t is less than $\alpha_{\text{init}} - \sum_{i=1}^{t} \alpha_i$. We show next that this does not impact the quality of the decomposition.

Let us define $\alpha_{\text{res}} = \alpha_{\text{init}} - \sum_{i=1}^{t} \alpha_i$, i.e., the value that the minimum odd cut should have, knowing the terms of the decomposition we have computed until the step t. Let the value of a vertex $v \in V$ be $\sum_{e \in \delta(v)} x_e$ and the value of an odd cut S be $\sum_{e \in \delta(S)} x_e$. At the beginning of the algorithm, we have the guaranty that $\alpha_{\text{init}} \geq 1 - \tau$. The goal is then to show that the number of times we use `myZero` to reduce the value of a vertex or an odd cut is less than $2m$. This way, we know that at the beginning we have an x with $\sum_{e \in \delta(v)} x_e \geq 1 - \tau$ for all v, and the accumulated error is less than $\varepsilon - \tau$, so the decomposition accumulates a total of $\sum \alpha_i \geq (1 - \varepsilon)$.

The value of a vertex can only be reduced by setting an edge incident on it to 0, if that edge's weight is less than `myZero`. This can happen at most n times. Hence, discarding weights smaller than `myZero` will not impact the value of vertices.

In the LinearSearchCut algorithm, the change from $y'(\delta(T)) = \alpha - \gamma$ to $y'(\delta(T)) > \alpha - \gamma -$ `myZero` implies that each time we use the procedure, the value of a minimum odd cut α loses at most `myZero` compared to α_{res}. We use LinearSearchCut at most m times, so the accumulated error for an odd cut is at most $\frac{\varepsilon-\tau}{2}$.

The value of a cut can also be impacted by setting edges to 0 when updating the graph. Because there are m edges in the graph, the value of a cut can be reduced by at most $\frac{\varepsilon-\tau}{2}$. In total the value of any odd cut can only be reduced by $\varepsilon - \tau$, which is exactly the margin we have.

In conclusion, we have the guaranty that at the beginning of our algorithm $\alpha \geq 1 - \tau$, and the value of a minimum odd cut can be reduced by `myZero` at most $2m$ times. Hence, at the end of the algorithm, the remaining weight is positive, and the accumulated sum of coefficient is

more than $1 - \varepsilon$.

4.2 Bottleneck matching

As in earlier work on practical heuristics for the classical BvN decomposition [8], it is preferable to reduce the number of permutation matrices used in a symBvN decomposition. This is so, as applications prefer a smaller number of terms, and the run time of the decomposition heuristics is proportional to the number of terms. Dufossé and Uçar [8] present a heuristic which obtains small number of terms by maximizing the coefficient chosen at each step of the decomposition. Maximizing the coefficient at each step translates to finding a perfect matching M in a bipartite graph, where the minimum edge weight in M is the maximum. This problem is known as the bottleneck perfect matching problem in bipartite graphs, for which efficient solvers are available [22].

As discussed before, in Algorithm 2, we do not always associate $\min_{e \in M} y_e$ to a perfect matching M, but sometimes use a coefficient γ in $[0, \min_{e \in M} y_e]$. Because it seems hard to find a perfect matching M which will maximize the corresponding γ coefficient, we choose to maximize $\min_{e \in M} y_e$ when picking a perfect matching. This heuristic approach provides the LinearSearchCut algorithm with a large upper limit and thus will likely yield a large γ coefficient. Recall also that we cannot use any perfect matching; we need to select a perfect matching which has weight $|\mathcal{H}|$ under the weight function $w_{\mathcal{H}}(\cdot)$ defined in Definition 3.2. In order to take these constraints into account, we propose a binary search based algorithm to find a bottleneck perfect matching with respect to weights y, while having the weight $|\mathcal{H}|$ according to the weight function $w_{\mathcal{H}}(\cdot)$. The proposed algorithm thus finds the largest threshold h such that for edges in $E_h = \{e \in E | y_e \geq h\}$ the graph $G^{(h)} = (V, E_h)$ has a perfect matching, and the weight of the perfect matching under the weight function $w_{\mathcal{H}}(\cdot)$ is $|\mathcal{H}|$.

We assess the effects of the bottleneck matching on a set of constructed problem instances built as follows. For an even positive integer n and a positive integer $r^\star$, we pick uniformly at random $r^\star$ perfect matchings in $\{1, \ldots, n\}$ and $r^\star$ coefficients in $\{1, \ldots, 10\}$. We then add them up to form the edges of an undirected weighted graph, which is a decomposamble α-fractional perfect matching, with α equal to the sum of coefficients. We compare the number of permutation matrices in the decomposition of these problem instances using the proposed bottleneck matching algorithm and choosing an arbitrary minimum weight perfect matching with respect to $w_{\mathcal{H}}(\cdot)$. The arbitrary matchings are obtained by calling the minimum weighted perfect matching algorithm from LEMON when $|\mathcal{H}| > 0$ and the perfect matching algorithm when $|\mathcal{H}| = 0$. The bottleneck algorithm uses the same perfect matching functions inside a binary search method to find the threshold h. Results are presented in Table 1, where the number r of terms in a decomposition are reported, as the average of 20 random instances. As seen from this table, using a bottleneck perfect matching reduces the number of terms in the decomposition by a factor more than 3. The set $\mathcal{H}$ contains only a few tight odd cuts, thus $w_{\mathcal{H}}(\cdot)$ does not create stringent contraints in selecting perfect matchings. Furthermore, the number r of terms with the bottleneck approach are close to the number $r^\star$ used in constructing the instances. Given these results, our codes use the bottleneck perfect matching by default in all remaining experiments.

4.3 Comparing different ways of decomposing a symmetric doubly-stochastic matrix

Recall from Section 2.5 that one can decompose a symmetric doubly stochastic matrix using the classical BvN decomposition, the decomposition with $\mathbf{P} + \mathbf{P}^T$ terms, and the symBvN decom-

Table 1: The number r of terms in the decomposition of a set of constructed instances, where a decomposition with $r^\star$ terms is possible, with and without the bottleneck objective, and the median value of $|\mathcal{H}|$ over the 20 random experiments.

		Arbitrary		Bottleneck	
n	$r^\star$	$\lvert\mathcal{H}\rvert$	r	$\lvert\mathcal{H}\rvert$	r
100	30	3	170	2	46
200	40	3	220	3	60
400	50	4	280	3	80

position. We now apply these three decompositions to a set of matrices to see how much they differ.

Among all fully indecomposable symmetric matrices with size between 500 and 1000 rows from the SuiteSparse Matrix Collection [4], we have chosen the first two according to ids from each family to remove any bias that might be arising from the family. This resulted in 14 matrices. One of the matrices (netz4504_dual) has an odd number of rows and a zero diagonal, so it does not admit a symBvN decomposition. We thus present results with 13 matrices given in Table 2. We have processed these matrices following the common methodology in the earlier computational studies using BvN decompositions [2, 7, 8]. On a given matrix $\mathbf{A}$, we (i) first take absolute values of its entries; (ii) then scale the resulting matrix with a symmetry preserving scaling algorithm [15] so that the maximum deviation of row (and column) sums from 1, that is τ, is less than 10^{-4}; (iii) form the graph associated with $\mathbf{A}$, using the transformation of Section 2.6 if necessary; (iv) verify that the minimum odd cut in this weighted graph has value more than $1-\tau$.

We have used the implementation of the greedy heuristic [8] in Python by Lesens et al.[18] for the classical BvN decomposition (denoted BvN below). This code is slightly modified for obtaining the decomposition with the terms of the form $\mathbf{P}+\mathbf{P}^T$ (denoted BvN_{P+P^T} below). The proposed symBvNh heuristic applies the transformation described in Section 2.6, and returns a decomposition of the original (scaled) matrix. Matrices with a star "*" next to their name, namely EX1, EX2 and L have a zero diagonal and even size, so we do not need to apply the graph transformation and can consider them as adjacency matrices directly. The other matrices in the list have full diagonal, except for dynamicSoaringProblem_1 which has 363 nonzeros on the diagonal. The first two decompositions based on bipartite graphs use the state of the art bottleneck matching method [22]. Hence, all three decompositions use a bottleneck approach to choose a perfect matching at each step. We decompose matrices until the sum of coefficients is at least $0.999 = 1-\varepsilon$.

Table 2 shows the number of terms obtained by the three decomposition heuristics. As seen from this table, BvN obtains the least number of terms in eight cases. In 685_bus, L, and dynamicSoaringProblem_1, BvN_{P+P^T} obtains the least number of terms. In the remaining two cases bcsstm34 and dendrimer, symBvNh results in the least number of terms. The symmetric adaptation BvN_{P+P^T} of BvN obtains higher number of terms than BvN, as it probably has to use coefficients less than the bottleneck value. Nonetheless the performance of BvN_{P+P^T} is close to that of BvN in most cases. Due to all the associated constraints, we have expected symBvNh to obtain larger number of permutation matrices than BvN and BvN_{P+P^T}. The fact that it is comparable in general and even tangibly better in two cases (bcsstm34 and dendrimer) motivates the use of symmetric permutations in decomposing symmetric matrices. Furthermore, representing a symmetric permutation matrix should use less space than representing a general permutation matrix (e.g., $n/2$ vs n in case all diagonal entries are zero). Hence, a symBvN decomposition can be preferable to a BvN decomposition, whenever there are no more than

Table 2: Comparison of decomposition algorithms for a set of symmetric doubly stochastic matrices with $500 \leq n \leq 1000$ taken from the SuiteSparse Matrix Collection [4]. The columns n and z give, respectively, the number of rows/columns, and the number of nonzeros. For each algorithm we give the number of terms obtained. The matrices marked with a "*", namely EX1, EX2 and L, the transformation of Section 2.6 was not used.

			The number r of terms		
Matrix name	n	z	BvN	BvN$_{P+P^T}$	symBvNh
662_bus	662	2474	34	34	40
685_bus	685	3249	42	40	86
bcsstk34	588	21418	117	145	117
bcsstm34	588	24270	171	204	159
Si2	769	17801	86	90	93
EX1*	560	8736	57	81	56
EX2*	560	8736	67	82	69
Trefethen_500	500	8478	69	72	70
Trefethen_700	700	12654	73	76	75
L*	956	3640	57	55	138
dynamicSoaringProblem_1	647	5367	319	286	302
spaceShuttleEntry_1	560	6891	258	258	292
dendrimer	730	63024	567	672	486

twice the number of terms in the latter one. Our symBvNh algorithm is thus standing as a reasonable option in terms of sparsity for decomposing a symmetric doubly stochastic matrix as a sum of symmetric permutation matrices.

We next investigate different decomposition methods for computing a given number of components. For this set of experiments we have selected a set of larger matrices and run the decompositions methods to obtain 10 terms. The matrices are square, numerically symmetric, have at least 5000 at most 10000 rows, and fully indecomposable. There were 74 matrices at the SuiteSparse collection. We have retained only the first two matrices from different families. This resulted in 34 matrices. The matrices and the results are shown in Table 3. For each matrix, the sum of the coefficients obtain by BvN is given, followed by the ratio of those obtained by BvN$_{P+P^T}$ and symBvNh to that of BvN. The last line gives the geometric mean of the ratios over all matrices. The results from this table are similar to those in Table 2. In most cases, the sum of coefficients accumulated by the nonsymmetric variants (BvN and BvN$_{P+P^T}$) and symBvNh are comparable; there are only six cases in which the ratio of the coefficients obtained by symBvNh is less than 90% of that of BvN. The performance of BvN$_{P+P^T}$ is again closer to that of BvN. Therefore, we conclude that symBvNh can also be an effective tool to select a few important terms from a decomposition.

4.4 The run time analysis of symBvNh

Table 4 gives a breakdown of the computing time of the proposed symBvNh heuristic along with other data. For each matrix we give the number r of terms in the decomposition (repeated from Table 2 for convenience), the cardinality of the set of odd cuts $\mathcal{H}$ at the end of the algorithm, the number of calls to and the time spent in the minimum odd cut algorithm, the number of calls to and the time spent in the perfect matching algorithms. For these last two data, we did not make a distinction between the perfect matching algorithm (used when $|\mathcal{H}| = 0$) and the minimum weighted perfect matching algorithm.

From Table 4, we see that more than 99% of the total time is spent in the subroutines for the minimum odd cut and the MWPM problems. This is so, as apart from these calls there is

Table 3: Comparison of decomposition algorithms for a set of symmetric doubly stochastic matrices with $5000 \leq n \leq 10000$ taken from the SuiteSparse Matrix Collection [4]. For the BvN algorithm we give the sum of the coefficients for the first 10 terms computed. For the other two algorithms, we give the ratio of the sum of the first 10 coefficients produced compared to the BvN algorithm. The last line give the geometric mean of this ratio for the BvN_{P+P^T} and symBvNh.

	Sum of the first 10 coefficients		
Matrix name	BvN	BvN_{P+P^T}	symBvNh
bcspwr10	0.7421	0.990	0.403
bcsstk33	0.1045	0.948	1.000
bcsstk38	0.2418	0.958	0.964
ex15	0.7305	0.912	0.953
meg4	1.0000	1.000	1.000
nemeth01	0.6575	1.004	0.913
nemeth02	0.9842	1.000	1.000
aft01	0.5487	1.003	1.022
barth	0.7310	0.945	0.757
barth4	0.7193	1.003	0.930
fv1	0.9443	0.955	0.975
fv2	0.9460	0.934	0.996
Alemdar	0.7441	0.978	0.823
nd3k	0.6922	0.982	0.845
exdata_1	0.0210	0.997	1.000
Kuu	0.4627	0.941	1.000
Muu	0.6597	0.960	1.011
benzene	0.8312	0.963	0.983
Na5	0.6998	0.967	1.018
fxm3_6	0.1500	0.963	0.995
net25	0.1408	1.000	1.000
flowmeter0	0.8534	0.989	0.799
rail_5177	0.7758	1.012	0.916
c-29	0.0867	0.999	1.000
c-30	0.0154	1.000	1.000
s1rmq4m1	0.5060	0.990	1.000
s2rmq4m1	0.4920	0.994	1.001
g3rmt3m3	0.2238	0.971	1.000
t520	0.0871	0.963	1.000
delaunay_n13	0.7472	0.942	0.381
TSC_OPF_1047	0.2227	1.000	0.970
TSC_OPF_300	0.4453	1.000	0.969
freeFlyingRobot_10	0.4539	0.983	1.000
freeFlyingRobot_11	0.4677	1.010	1.006
geomean		0.9778	0.9120

Table 4: The number r of terms obtained by the symBvNh algorithm, the size of $\mathcal{H}$, the number of calls to the minimum odd cut algorithm and the time spent, the number of calls to MWPM algorithms and the time spent for that, and finally the total time of the computation. The run time is given in seconds.

			min-odd-cut		MWPM		Total
Matrix name	r	$\lvert\mathcal{H}\rvert$	calls	time	calls	time	time
662_bus	40	8	48	1.38	429	0.22	1.61
685_bus	86	63	156	4.56	986	1.62	6.20
bcsstk34	117	3	120	3.47	1599	4.39	7.89
bcsstm34	159	1	160	5.18	2262	5.19	10.42
Si2	93	10	103	3.76	1274	3.76	7.54
EX1*	56	1	57	0.62	498	0.11	0.74
EX2*	69	1	70	0.59	733	0.44	1.04
Trefethen_500	70	0	70	1.27	862	0.23	1.51
Trefethen_700	75	0	75	2.48	964	0.37	2.86
L*	138	75	221	5.35	1492	1.01	6.37
dynamicSoaringProblem_1	302	0	302	17.70	3612	0.85	18.57
spaceShuttleEntry_1	292	17	309	13.58	3601	4.46	18.07
dendrimer	486	29	515	45.26	7634	87.94	133.55

only the management of the set of odd cuts and updates of the working graph. The matrices are small, and hence each individual call to subroutines for the minimum odd cut and the MWPM problems consumes very little time (geometric mean of the run time is 0.029s for the minimum odd cut, and 0.00089s for the MWPM). Accumulating over the repeated calls result in 129.99s for the matrix with the largest number of nonzeros and terms in the decomposition.

An interesting observation from Table 4 is that the number $r + |\mathcal{H}|$ is very close (if not equal) to the number of calls made to the minimum odd cut algorithm. This means that the LinearSearchCut procedure is efficient and performs very few iterations of the while loop on average.

Looking at the $|\mathcal{H}|$ column, we can explain the large number of terms produced by symBvNh compared to the other heuristics (see Table 2) for the matrices 685_bus and L. The family of tight odd α-cuts $\mathcal{H}$ grows when the algorithm goes into the "else" case at Line 13 and produces a coefficient γ via Equation (2). These coefficients are smaller than those in the "if" case, therefore the algorithm makes less progress and consequently produces more terms.

The size of $\mathcal{H}$ at the end of the algorithm happens to be very small and even zero in four out of 13 instances. This can be due to the algorithm being very lucky and never picking a perfect matching that crosses a tight odd α-cut more than once. This supports the approach of building $\mathcal{H}$ incrementally and not maintaining a maximality property like Vazirani's algorithm does (see Appendix A.1).

We conclude the experimental investigation by noting that the nonsymmetric variants are very fast compared to symBvNh. For example in dendrimer, BvN from Lesens et al. [18] takes 2.35 seconds, even though it obtains a higher number of terms in the decomposition (see Table 2). There are multiple reasons for this. First, the nonsymmetric variants find the suitable coefficient in $O(n)$ time, while symBvNh makes a call to LinearSearchCut which invokes a minimum odd cut subroutine $O(n)$ times. Second, the bottleneck matching for the nonsymmetric variants uses a specialized solver [22], while that in symBvNh uses a minimum weighted perfect matching algorithm in a black-box fashion within a binary search method.

5 Conclusion

We have investigated the symBvN decomposition of symmetric doubly stochastic matrices, in which the permutation matrices in the decomposition are also symmetric. Based on earlier work [20, 25] on decomposable fractional perfect matchings in graphs, we have showed when symBvN decomposition is possible, and have proposed an algorithm to carry out the desired decomposition. The proposed algorithm is derived by simplifying Vazirani's approach [25] using key ideas by Padberg and Wolsey [20]. We have implemented the proposed algorithm into a software (symBvNh), which, to the best of our knowledge, is the first tool for decomposing fractional perfect matchings and obtaining a symBvN decomposition for a given matrix. We have carried out experiments on constructed and real-life sparse matrices. The experiments have demonstrated the suitability of the symBvN decomposition in practical settings.

Much work remains to be done. First, new developments in Gomory-Hu tree construction algorithms [1], including the dynamic variants [13], can be used for speeding up the computations. Similarly, there is room for improvement in our bottleneck MWPM algorithm. Optimizations that are used in the bipartite case [22] could be adapted. Typically, the classical BvN decomposition is used in contexts where a few terms in a decomposition are used to round non-integral solutions of covering/assignment problems to solutions that are integral. The same use cases also exist for decomposable fractional perfect matchings. We want to explore the effects of our tool in those use cases.

Appendix A A variant of the proposed algorithm which enforces laminarity on the identified family of tight odd α-cuts

A.1 Vazirani's original algorithm

We first present briefly Vazirani's original algorithm [25]. Vazirani describes a way to find a suitable perfect matchings based on a narrower version of our Lemma 3.3. Recall that a *laminar* family $\mathcal{L}$ is a family of subsets of a ground set such that for all $S, T \in \mathcal{L}$, either $S \subset T$, or $T \subset S$, or $S \cap T = \emptyset$. Vazirani shows that if one has a maximal laminar family $\mathcal{L}$ of tight odd α-cuts, a minimum weighted perfect matching M in G_x under the weight function $w_{\mathcal{L}}(\cdot)$ has weight $|\mathcal{L}|$ and crosses every tight odd α-cut only once [25, Lemma 4]; we stress that M crosses every tight odd α-cut only once, not only those in $\mathcal{L}$. Vazirani describes algorithms for building a maximal laminar family $\mathcal{L}$ of tight odd 1-cuts for a given graph G and $x \in \mathcal{P}(G, 1)$. He also provides a method for updating $\mathcal{L}$ so that it is always a maximal laminar family of tight odd α-cuts when progress is made in decomposing x.

Vazirani's approach of finding a suitable perfect matchings is useful in two ways. First, it allows choosing a suitable perfect matching to be included in the output in all iterations. Second, the size of a maximal laminar family of tight odd α-cuts is at most $\frac{n}{2} - 1$, which helps bound the number of times the family needs to be updated [25, Lemma 13]. However, this comes at a cost. Indeed, updating the laminar family $\mathcal{L}$, so that it is always maximal, requires $O(n^2m^2)$ calls to the Padberg-Rao minium odd-cut algorithm and dominates the run time [25, Theorem 16]. Moreover, in order to make Padberg-Rao algorithm find minimum cuts of cardinality at least three, Vazirani [25, Section 4.1] modifies the edge weights with very small values in the order of $O\left(n^{-n}\right)$, potentially requiring precisions going beyond quadruple, or octuple for a suitable implementation. This makes the algorithm very costly and impractical.

Note that in Algorithm 2 if $\mathcal{H}$ were a maximal laminar family, we would have the same bound

$n/2-1$ on the number of iterations of the while loop, which of course would render the check on γ at Line 15 of Algorithm 2 useless. We can keep a laminar family of tight odd α-cuts, not necessarily maximal, in which case Algorithm 2 again works as is, where the else-case at Line 13 of Algorithm 2 is bounded by $n/2-1$. We show how to do this efficiently next.

A.2 A fast fractional perfect matching decomposition algorithm by enforcing laminarity

We now present the modification of Algorithm 2 that enforces $\mathcal{H}$ to be laminar and gives a complexity of $O(m(\text{MWPM}+\text{PR})+n^2\text{PR})$.

Let us first give some background on laminarity. We say that two sets S and T are intersecting if $S\cap T\neq\emptyset$, $S\backslash T\neq\emptyset$ and $T\backslash S\neq\emptyset$. A laminar family is thus a family of sets in which no two sets intersect.

The following lemma is known in the literature, and used by Vazirani.

Lemma A.1 (Uncrossing lemma)**.** *Let x be an α-fractional perfect matching and M a perfect matching in x. If S,T are odd sets, have cut value α and are intersecting, then exactly one of the following is true:*

i. $S\cap T$ and $S\cup T$ are odd, have cut value α and

$$|\delta(S)\cap M|+|\delta(T)\cap M|=|\delta(S\cup T)\cap M|+|\delta(S\cap T)\cap M|\ .$$

ii. $S\backslash T$ and $T\backslash S$ are odd, have cut value α and

$$|\delta(S)\cap M|+|\delta(T)\cap M|=|\delta(S\backslash T)\cap M|+|\delta(T\backslash S)\cap M|\ .$$

Proof. We first recall the classical results for the cut function $S\mapsto x(\delta(S))$. For any two sets A,$B\subset V$ we have:

$$x(\delta(A))+x(\delta(B))=x(\delta(A\cup B))+x(\delta(B\cap A))+2x(\delta(A\backslash B)\cap\delta(B\backslash A))\ , \tag{10}$$

which can be verified by counting the contribution of each edge on both sides of the equation.

We also have

$$x(\delta(A))+x(\delta(B))=x(\delta(A\backslash B))+x(\delta(B\backslash A))+2x(\delta(A\cap B)\cap\delta(\overline{A\cup B}))\ , \tag{11}$$

again by counting.

Let us look at the intersection $S\cap T$, which is either odd or even.

Case 1: If $S\cap T$ is odd, then $S\cup T$ is also odd and we have:

$$\begin{aligned}
x(\delta(S))+x(\delta(T)) &\geq x(\delta(S\cup T))+x(\delta(S\cap T))\\
&\qquad\text{(by applying (10) on } S \text{ and } T \text{ and } x \text{ is positive)}\\
&\geq 2\alpha\\
&= x(\delta(S))+x(\delta(T))\\
&\qquad\text{(as both } S\cup T \text{ and } S\cap T \text{ are odd,}\\
&\qquad\ \text{and } x \text{ is a fractional perfect matching).}
\end{aligned}$$

Therefore, we have equality throughout. Hence $S\cap T$ and $S\cup T$ have cut value α, and there are no edges between $S\backslash T$ and $T\backslash S$ as $x(\delta(S\backslash T)\cap\delta(T\backslash S))=0$. Thus $\chi(\delta(S))+\chi(\delta(T))=\chi(\delta(S\cup T))+\chi(\delta(S\cap T))$ and $|\delta(S)\cap M|+|\delta(T)\cap M|=|\delta(S\cup T)\cap M|+|\delta(S\cap T)\cap M|$.

Case 2: If $S \cap T$ is even, then $S \backslash T$ and $T \backslash S$ are odd. Similarly to the first case, we have:

$$\begin{aligned} x(\delta(S)) + x(\delta(T)) &\geq x(\delta(S \backslash T)) + x(\delta(T \backslash S)) \\ &\geq 2\alpha \\ &= x(\delta(S)) + x(\delta(T)) \ . \end{aligned}$$

From which we deduce that $S \backslash T$ and $T \backslash S$ have cut value α. Also, there are no edges between $S \cap T$ and $\overline{S \cup T}$, so $\chi(\delta(S)) + \chi(\delta(T)) = \chi(\delta(S \backslash T)) + \chi(\delta(T \backslash S))$ and $|\delta(S) \cap M| + |\delta(T) \cap M| = |\delta(S \backslash T) \cap M| + |\delta(T \backslash S) \cap M|$.

The two disjoint cases cover all possibilities and correspond to the two alternatives stated in the lemma. □

This proof of the uncrossing lemma using submodularity and the two cases are well known [10, 11]. This lemma tells us that if S and T are intersecting odd sets and tight, one can find another odd tight set S' which is not intersecting T. The following lemma expresses this explicitly and is central to the algorithm maintaining laminarity. It was used implicitly by Vazirani without stating it. We thus state the lemma and give its proof.

Lemma A.2. *Let x be an α-fractional perfect matching and M be a perfect matching in a given graph G_x. Let S and T be two intersecting tight odd α-cuts, with $|\delta(S) \cap M| > 1$. Then there exists a tight odd α-cut S' such that $|\delta(S') \cap M| > 1$ and S' and T are not intersecting. Moreover, S' is one of the four sets $S \cap T, S \cup T, S \backslash T$, and $T \backslash S$.*

Proof. By definition, none of the four sets $S \cap T$, $S \cup T$, $S \backslash T$ and $T \backslash S$ form an intersecting pair with T. By Lemma A.1, we have exactly one of two equalities

$$|\delta(S) \cap M| + |\delta(T) \cap M| = |\delta(S \cup T) \cap M| + |\delta(S \cap T) \cap M|$$

$$|\delta(S) \cap M| + |\delta(T) \cap M| = |\delta(S \backslash T) \cap M| + |\delta(T \backslash S) \cap M|$$

true. Hence, if $|\delta(S) \cap M| > 1$, then the right hand side in both alternatives is also larger than one. As S' appears in one of the sets in the right hand side, it holds that $|\delta(S') \cap M| > 1$, which implies that S' is not a singleton. Since the four sets have cut value α in either cases of Lemma A.1, the odd set S' is also a tight odd α-cut. □

Vazirani notes that the cardinality of a maximal laminar family of tight odd α-cuts is bounded by $\frac{n}{2} - 1$ (Lemma 13 in [25]). He then states the following lemma for a maximal laminar family of tight odd α-cuts, rather than the whole family $\mathcal{F}$ (Lemma 4).

Lemma A.3 (Vazirani [25], Lemma 4)**.** *Let M be a perfect matching and $\mathcal{L}$ be a maximal laminar family of tight odd α-cut. If M crosses every tight odd α-cut $C \in \mathcal{L}$ exactly once, then M crosses every tight odd α-cut exactly once.*

Before proving Lemma A.3, we note how to use a maximal laminar family $\mathcal{L}$ of tight odd α-cuts to find a perfect matching in a decomposition of x. As before, for each edge e of G_x, define $w_{\mathcal{L}}(e) = |\{S \in \mathcal{L} : e \in \delta(S)\}|$. By Lemma 3.3 and Lemma A.3, we know that a perfect matching of the minimum weight $|\mathcal{L}|$ using the weight function $w_{\mathcal{L}}(\cdot)$ will cross every $S \in \mathcal{L}$ exactly once, and hence by Lemma A.3, every tight odd α-cut exactly once. This proof was not fully detailed in Vazirani's paper. The one we give below is adapted from Goemans [11, Theorem 4].

Proof of Lemma A.3. Suppose for the sake of contradiction that M crosses some tight odd α-cut S more than once. If there are several such sets S, choose one that forms an intersecting pair with as few sets of $\mathcal{L}$ as possible. The set S must form an intersecting pair with some $T \in \mathcal{L}$,

otherwise S can be added to $\mathcal{L}$, contradicting the maximality of $\mathcal{L}$. From Lemma A.2 there must exist a tight odd α-cut S' such that $|\delta(S') \cap M| > 1$ which does not form an intersecting pair with T. Hence S' forms an intersecting pair with fewer sets of $\mathcal{L}$ than S does, contradicting the choice of S. □

Vazirani's algorithm builds and maintains a maximal laminar family of tight odd α-cuts $\mathcal{L}$ in order to find a suitable perfect matching to be included as a term in the resulting decomposition. Maintaining maximality is too costly. Instead we can make a minor modification to Algorithm 2 to ensure that $\mathcal{H}$ is laminar. Before adding S' to $\mathcal{H}$ at Line 20, we can transform it by using Algorithm 4. This algorithm ensures that the set returned does not form an intersecting pair with any set in $\mathcal{L}$. For all $T \in \mathcal{L}$, if S and T form an intersecting pair, it uses Lemma A.2 to transform S into a set S' not intersecting T. Adding S' to $\mathcal{H}$ will thus maintain laminarity of $\mathcal{H}$. This way we get a bound of $n/2 - 1$ on the number of times the else case happens in Algorithm 2 and the cost of the algorithm is now $O(m(\mathrm{MWPM} + \mathrm{PR}) + n^2\mathrm{PR})$.

Algorithm 4 The procedure $S = \mathsf{Uncrossing}(M, S', \mathcal{L})$

Input: M: a matching
S': a tight odd α-cut with $|\delta(S) \cap M| > 1$
$\mathcal{L}$: a laminar family of tight odd α-cuts
Output: S: a tight odd α-cut to be added to $\mathcal{L}$ while maintaining laminarity

```
S ← S'
for T ∈ L do
  if T and S are intersecting then          ▶ i.e., T ∩ S ≠ ∅, T\S ≠ ∅ and S\T ≠ ∅
    if |T ∩ S| is odd then                  ▶ Uncross to either T ∩ S or T ∪ S
      if |δ(T ∩ S) ∩ M| > 1 then            ▶ T ∩ S is not a singleton
        S ← T ∩ S
      else                                  ▶ T ∪ S is not a singleton
        S ← T ∪ S
      end if
    else                                    ▶ Uncross to either T\S or S\T
      if |δ(T\S) ∩ M| > 1 then              ▶ T\S is not a singleton
        S ← T\S
      else                                  ▶ S\T is not a singleton
        S ← S\T
      end if
    end if
  end if
end for
return S
```

References

[1] A. Abboud, J. Li, D. Panigrahi, and T. Saranurak, *All-pairs max-flow is no harder than single-pair max-flow: Gomory-hu trees in almost-linear time*, in 2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS), 2023, pp. 2204–2212, https://doi.org/10.1109/FOCS57990.2023.00137.

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

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

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

[5] B. Dezsö, A. Jüttner, and P. Kovács, *LEMON – an open source C++ graph template library*, Electronic Notes in Theoretical Computer Science, 264 (2011), pp. 23–45, https://doi.org/10.1016/j.entcs.2011.06.003. Proceedings of the Second Workshop on Generative Technologies (WGT) 2010.

[6] R. Duan, S. Pettie, and H.-H. Su, *Scaling algorithms for weighted matching in general graphs*, ACM Transactions on Algorithms (TALG), 14 (2018), pp. 1–35.

[7] 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 (2018), pp. 68–78, https://doi.org/10.1016/j.laa.2018.05.017.

[8] F. Dufossé and B. Uçar, *Notes on Birkhoff–von Neumann decomposition of doubly stochastic matrices*, Linear Algebra and its Applications, 497 (2016), pp. 108–115, https://doi.org/10.1016/j.laa.2016.02.023.

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

[10] M. X. Goemans, *Uncrossing*. Available at http://www.science.unitn.it/cirm/Goemans.pdf.

[11] M. X. Goemans, *Minimum bounded degree spanning trees*, in 2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06), IEEE, 2006, pp. 273–282.

[12] R. E. Gomory and T. C. Hu, *Multi-terminal network flows*, Journal of the Society for Industrial and Applied Mathematics, 9 (1961), pp. 551–570, https://doi.org/10.1137/0109047.

[13] T. Hartmann and D. Wagner, *Dynamic Gomory-Hu tree construction – fast and simple*, 2013, https://arxiv.org/abs/1310.0178.

[14] P. A. Knight and D. Ruiz, *A fast algorithm for matrix balancing*, IMA Journal of Numerical Analysis, 33 (2013), pp. 1029–1047, https://doi.org/10.1093/imanum/drs019.

[15] P. A. Knight, D. Ruiz, and B. Uçar, *A symmetry preserving algorithm for matrix scaling*, SIAM Journal on Matrix Analysis and Applications, 35 (2014), pp. 931–955, https://doi.org/10.1137/110825753.

[16] V. Kolmogorov, *A computational study of Gomory-Hu construction tree algorithms*, 2022, https://arxiv.org/abs/2204.10169.

[17] V. Kolmogorov, *OrderedCuts: A new approach for computing Gomory-Hu tree*, 2023, https://arxiv.org/abs/2208.02000.

[18] D. Lesens, J. E. Cohen, and B. Uçar, *Orthogonal matching pursuit-based algorithms for the Birkhoff-von Neumann decomposition*, in EUSIPCO 2024 - 24th European Signal Processing Conference, Lyon, France, Aug. 2024, p. 12, https://hal.science/hal-04500014.

[19] M. W. Padberg and M. R. Rao, *Odd minimum cut-sets and b-matchings*, Mathematics of Operations Research, 7 (1982), pp. 67–80, http://www.jstor.org/stable/3689360.

[20] M. W. Padberg and L. A. Wolsey, *Fractional covers for forests and matchings*, Mathematical Programming, 29 (1984), pp. 1–14, https://doi.org/10.1007/BF02591725.

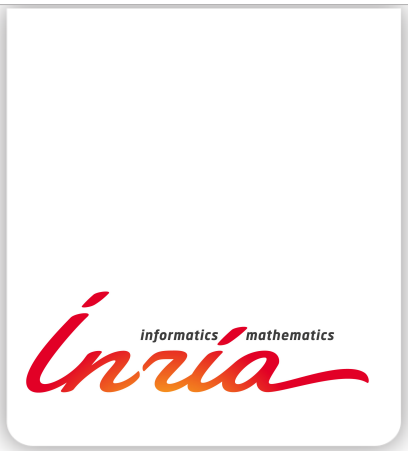
[21] I. Panageas, T. Tröbst, and V. V. Vazirani, *Time-efficient algorithms for nash-bargaining-based matching market models*, 2024, https://arxiv.org/abs/2106.02024.

[22] 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.

[23] A. E. Roth, T. Sönmez, and M. U. Ünver, *Pairwise kidney exchange*, Journal of Economic Theory, 125 (2005), pp. 151–188.

[24] R. Sinkhorn and P. Knopp, *Concerning nonnegative matrices and doubly stochastic matrices*, Pacific J. Math., 21 (1967), pp. 343–348.

[25] V. V. Vazirani, *An extension of the Birkhoff–von Neumann theorem to non-bipartite graphs*, 2020, https://arxiv.org/abs/2010.05984.



RESEARCH CENTRE
Centre Inria de Lyon

Bâtiment CEI-2, Campus La Doua
56, Boulevard Niels Bohr - CS 52132
69603 Villeurbanne

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399