Green Open Access added to TU Delft Institutional Repository

'You share, we take care!' - Taverne project

https://www.openaccess.nl/en/you-share-we-take-care

Otherwise as indicated in the copyright section: the publisher is the copyright holder of this work and the author uses the Dutch legislation to make this work public.
Birkhoff’s Decomposition Revisited: Sparse Scheduling for High-Speed Circuit Switches

Víctor Valls®, George Iosifidis®, and Leandros Tassiulas, Fellow, IEEE

Abstract—Data centers are increasingly using high-speed circuit switches to cope with the growing demand and reduce operational costs. One of the fundamental tasks of circuit switches is to compute a sparse collection of switching configurations to support a traffic demand matrix. Such a problem has been addressed in the literature with variations of the approach proposed by Birkhoff in 1946 to decompose a doubly stochastic matrix exactly. However, the existing methods are heuristic and do not have theoretical guarantees on how well a collection of switching configurations (i.e., permutations) can approximate a traffic matrix (i.e., a scaled doubly stochastic matrix). In this paper, we revisit Birkhoff’s approach and make three contributions. First, we establish the first theoretical bound on the sparsity of Birkhoff’s algorithm (i.e., the number of switching configurations necessary to approximate a traffic matrix). In particular, we show that by using a subset of the admissible permutation matrices, Birkhoff’s algorithm obtains an $\epsilon$-approximate decomposition with at most $O(\log(1/\epsilon))$ permutations. Second, we propose a new algorithm, Birkhoff+, which combines the wealth of Frank-Wolfe with Birkhoff’s approach to obtain sparse decompositions in a fast manner. And third, we evaluate the performance of the proposed algorithm numerically and study how this affects the performance of a circuit switch. Our results show that Birkhoff+ is superior to previous algorithms in terms of throughput, running time, and number of switching configurations.

Index Terms—Scheduling algorithms, machine learning algorithms, switches, Birkhoff decomposition.

I. INTRODUCTION

Data centers are increasingly adopting hybrid switching designs that combine traditional electronic packet switches with high-speed circuit switches [1]–[3]. In short, packet switches are flexible at making forwarding decisions at a packet level, but have limited capacity and are becoming increasingly expensive in terms of cost, heat, and power. In contrast, circuit switches provide significantly higher data rates at a lower cost but are less flexible at making forwarding decisions. The main drawback of circuit switches is that they have high reconfiguration times, which limit the amount of traffic they can carry [1], [4], [5]. For instance, circuit switches have reconfiguration times in the order of milliseconds (e.g., 25 ms for off-the-self circuit switches [6], [7]), whereas the reconfiguration times in electronic switches are in the scale of microseconds. As a result, hybrid switching architectures load balance and use circuit switches for high-intensity/bursty flows [8], [9] and electronic switches for traffic that needs of a more fine-grained scheduling (e.g., delay-sensitive applications).

The problem of computing switching configurations for circuit switches is central to networking and has a direct impact on the performance of nowadays data centers. Mathematically, we can model a circuit switch as a crossbar, and cast the problem of finding a small collection of switching configurations as decomposing a doubly stochastic matrix as a sparse convex combination of permutations matrices. In brief, for a given $n \times n$ doubly stochastic matrix $X^*$ (i.e., a scaled traffic matrix) and an $\epsilon \geq 0$, the goal is to find a small collection of permutation matrices $P_1, P_2, \ldots, P_k$ (i.e., switching configurations) and weights $\theta_1, \theta_2, \ldots, \theta_k > 0$ (i.e., the fraction of time the switching configurations will be used) with $\sum_{i=1}^{k} \theta_i \leq 1$ such that

$$\left\| X^* - \sum_{i=1}^{k} \theta_i P_i \right\|_F \leq \epsilon,$$

where $\| \cdot \|_F$ is the Frobenius norm (see definition in Section III-A). The smaller $\epsilon$ is, the higher the throughput. However, practical systems cannot use as many switching configurations as desired as each incurs a reconfiguration time $\delta$ that affects the fraction of time the switch can carry traffic. Or put differently, there is a constraint on the number of configurations a switch can use to approximate a traffic matrix.

Previous work has addressed the problem above with variations (e.g., [8], [14]–[16]) of the approach proposed by Birkhoff in 1946 [17] to decompose a doubly stochastic matrix:

1. Traffic in data centers is often bursty [10], [11] and uses few input/output ports [12].
2. See, for example, [13, Section 4.1].
3. A matrix is doubly stochastic if its entries are non-negative and the sum of every row and column is equal to one. A permutation matrix is a binary doubly stochastic matrix.

Technically, a traffic matrix $X^*$ is valid for a time window period $W$, and the decomposition must satisfy $\sum_{i=1}^{k} (\theta_i + \delta) \leq W$. That is, the time spent transmitting ($\sum_{i=1}^{k} \theta_i$) and reconfiguring ($\delta k$) cannot exceed the time window duration ($W$).
exactly (i.e., $\epsilon = 0$). However, little is known about the behavior or convergence properties of Birkhoff’s algorithm, and so fundamental questions remain still unanswered. In particular, how does the decomposition approximation $\epsilon$ in Eq. (1) depend on the number of switching configurations? How much does an additional switching configuration contribute to increasing a circuit switch throughput? How is Birkhoff’s algorithm related to other numerical methods in other fields, such as optimization and machine learning? Answering these questions is crucial to better understand the structure of the problem and to design new algorithms that improve the performance of circuit switches. To this end, the main contributions of the paper are the following:

(i) Revisiting Birkhoff’s Approach: We revisit Birkhoff’s algorithm and establish the first theoretical bound on its sparsity (i.e., the number of permutations necessary to approximate a doubly stochastic matrix). In particular, we show that by selecting permutations from a subset of admissible permutations, Birkhoff’s algorithm has sparsity $O(\log(1/\epsilon))$ (Theorem 1). That is, the number of permutations required to obtain an $\epsilon$-approximate decomposition increases logarithmically with the decomposition error. Our results also show that previous Birkhoff-based algorithms that select permutations using a Max-Min criterion (e.g., [8]) have logarithmic sparsity (Corollary 1), and that Birkhoff’s algorithm is strongly connected to block-coordinate descent and Frank-Wolfe methods in convex optimization (Section IV-D.2 and Section V).

(ii) New Algorithm (Birkhoff+): We propose a new algorithm that combines Birkhoff’s approach and Frank-Wolfe. Specifically, permutation matrices are selected using a Frank-Wolfe-type update with a barrier function, while the weights as in Birkhoff’s approach. The proposed algorithm has theoretical guarantees (Corollary 2) and is non-trivial as a direct combination of Birkhoff’s approach with Frank-Wolfe may not converge (Theorem 4). Furthermore, Birkhoff+ is faster than previous algorithms as it computes a new permutation/configuration by solving a single linear program (LP). Table I contains a summary of the main differences between Birkhoff+ and the state-of-the-art algorithms discussed in Section II.

(iii) Numerical Evaluation: We evaluate Birkhoff+’s performance in a circuit switch application and compare it against existing algorithms for a range of matrices (dense, sparse, skewed) that capture the characteristics of traffic in data centers. Our results show that Birkhoff+ is superior to previous algorithms in terms of throughput, running time, and number of switching configurations. For instance, when $\delta/W = 10^{-2}$ (the reconfiguration time over the time available for transmission), Birkhoff+ has 7% more throughput than the best state-of-the-art algorithm. If we consider, in addition, the time to compute the switching configurations as an overhead, the throughput gain increases to 34% (switch with $n = 100$ ports).

The outline of the paper is as follows. Section II presents related work and Section III the preliminaries, which include the notation and how to find a permutation matrix by solving a linear program. In Section IV, we revisit Birkhoff’s approach in a general form, establish its sparsity rate, and show how this is connected to block-coordinate descent methods in convex optimization. The latter also clarifies that selecting a permutation matrix can be seen as choosing a (gradient) descent direction. Section V shows how to use Frank-Wolfe algorithms to decompose a doubly stochastic matrix, and how Frank-Wolfe chooses a permutation matrix that provides “steepest descent.” In Section VI, we present the new algorithm (Birkhoff+) and in Section VII evaluate its performance against the state-of-the-art algorithms. Finally, Section VIII concludes. All the proofs are in the Appendix.

### II. HISTORY AND RELATED WORK

#### A. Birkhoff’s Approach

This is the method employed by Birkhoff in 1946 to decompose a doubly stochastic matrix exactly [17, first theorem],\(^5\)

In brief, the method consists of finding permutations matrices sequentially (e.g., with the Hungarian algorithm) and terminates when it obtains an exact decomposition, which happens with at most $k = (n - 1)^2 + 1$ iterations/permute [21], [22] by Carathéodory’s theorem. The method, however, does not guarantee that the decomposition (i.e., $\sum_{i=1}^{k} \theta_i P_i$) is close to the doubly stochastic matrix it aims to approximate (i.e., $X^*$). In fact, the approximation is typically very poor until the algorithm converges exactly in the last iteration (see Figure 3a in Section VII for an example).

#### B. Related Mathematical Problems

The problem of finding the Birkhoff decomposition with the minimum number of permutation matrices ($\min k \text{ s.t. } X^* = \sum_{i=1}^{k} \theta_i P_i$) was addressed in [15] and shown to be NP-hard. In [23], the authors also show that the problem is not tractable when the minimal decomposition can be expressed with $k \geq 4$ permutations. The work in [24] formulates a similar problem. For a demand matrix $D = X^* - S$ with $S \in [0,1]^{n \times n}$, the goal is to find a collection of weights $\{\theta_i\}_{i=1}^{k}$ and permutation matrices $\{P_i\}_{i=1}^{k}$ that minimizes $\sum_{i=1}^{k} \theta_i$ subject to $\sum_{i=1}^{k} \theta_i P_i \geq D$ entry-wise. The problem is shown to be NP-complete. The problem addressed in this paper is different.

\(^5\)The result is also known as Birkhoff-von Neumann (BvN) as it was discovered independently by von Neumann [20]. We use Birkhoff instead of BvN as the algorithm used in the literature is based on the method of proof used by Birkhoff in [17].

\(^6\)The entries of the demand matrix $D$ are non-negative. Matrix $S$ adds a non-negative virtual load to demand matrix so that $D + S$ is doubly stochastic.
in spirit from the mathematical problems in [15], [24] because we do not aim to find a (small) collection of objects (i.e., \( k \)) subject to decomposition constraints (i.e., \( X^* = \sum_{i=1}^{k} \theta_i P_i \) or \( \sum_{i=1}^{k} \theta_i P_i \geq D \)). Instead, our goal is to design an algorithm that minimizes \( \|X^* - X_k\|_F \) where \( X_k = \sum_{i=1}^{k} \theta_i P_i \). The convergence rate of the numerical method corresponds to the number of permutations required to obtain an \( \epsilon \)-approximate decomposition.

C. Algorithms

The paper in [8] proposes Solstice, a Birkhoff-based heuristic for finding a Birkhoff decomposition with few permutations/configurations. Solstice picks permutation matrices using a Max-Min type criterion, and the weights or configurations durations are selected as large as possible provided \( X^* - \sum_{i=1}^{k} \theta_i P_i \) is non-negative entry-wise. The work in [9] proposes Eclipse, a sub-modular-type algorithm for solving the problem of the type introduced in [24]. Permutation matrices and weights are selected jointly to maximize an effective utilization criterion, which takes into account the reconfiguration penalty \( \delta \). Also, [9] shows that the final decomposition satisfies the optimal approximation ratio in sub-modular optimization with cover constraints. Both algorithms [8], [9] select permutation matrices by solving multiple linear programs (LPs) with a simplex type method [25]. Finally, we note the recent works in [16] and [26]. The first extends Eclipse to use a special type of weights/time coefficients that do not constraint the decomposition to be a scaled doubly stochastic matrix. The second addresses the online version of the problem in [9]—in the machine learning sense [27]—where the traffic matrix is learned a posteriori.

To conclude, we note the Frank-Wolfe algorithms [18], [28] used extensively in machine learning. The Frank-Wolfe setup is the following. Given a collection of discrete objects \( D \) and a convex set \( X \subseteq \text{conv}(D) \), the goal is to minimize a convex function by making convex combinations of the discrete objects. The problem addressed in this paper can be seen as a special case for Frank-Wolfe. The permutation matrices correspond to the discrete objects, the Birkhoff polytope is the convex set, and the objective function a metric that captures the distance between the approximate decomposition and \( X^* \) (e.g., Frobenius norm or Euclidean distance). Also, unlike Birkhoff-based approaches, Frank-Wolfe algorithms provide sparsity guarantees and ensure that the approximate decomposition is always a doubly stochastic matrix.

III. PRELIMINARIES

A. Notation

We use \( \mathbb{R}_+ \) and \( \mathbb{R}^d \) to denote the set of nonnegative real numbers and \( d \)-dimensional real vectors. Vectors and matrices are written in lower and upper case respectively, and all vectors are in column form. The transpose of a vector \( x \in \mathbb{R}^d \) is indicated with \( x^T \), and we use \( 1 \) to indicate the all ones vector—the dimension of the vector will be clear from the context. We use parenthesis to indicate an element in a vector, i.e., \( x(j) \) is the \( j \)’th element of vector \( x \). Similarly, the element in the \( i \)’th row and \( j \)’th column of a matrix \( X \) is indicated with \( X(i,j) \).

Algorithm 1  General Birkhoff

<table>
<thead>
<tr>
<th>Input:</th>
<th>Doubly stochastic matrix ( X^* ), ( \epsilon \geq 0 ), and ( k_{\text{max}} \geq 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set:</td>
<td>( k = 1 ) and ( X_0 = {0}^{n \times n} )</td>
</tr>
<tr>
<td>while</td>
<td>( |X_{k-1} - X^*|<em>F &gt; \epsilon ) and ( k \leq k</em>{\text{max}} )</td>
</tr>
<tr>
<td>do</td>
<td>Compute ( P_k, \theta_k ) that satisfy Eqs. (3)-(5)</td>
</tr>
<tr>
<td>( X_k \leftarrow X_{k-1} + \theta_k P_k )</td>
<td></td>
</tr>
<tr>
<td>( k \leftarrow k + 1 )</td>
<td></td>
</tr>
<tr>
<td>end while</td>
<td></td>
</tr>
<tr>
<td>return</td>
<td>((P_1, \ldots, P_{k-1}), (\theta_1, \ldots, \theta_{k-1}))</td>
</tr>
</tbody>
</table>

B. Finding Extreme Points by Solving Linear Programs

We will present algorithms that find extreme points (i.e., permutation matrices) by solving linear programs (LPs) over a convex set (i.e., the Birkhoff polytope or set of doubly stochastic matrices). We recall the following result from linear programming.

**Lemma 1:** Let \( X \) be a bounded polytope from \( \mathbb{R}^d \), and \( \mathcal{E} \) denote its extreme points. For any vector \( c \in \mathbb{R}^d \), we have that \( \{\arg \min_{x \in X} c^T x\} \cap \mathcal{E} \neq \emptyset \).

That is, an extreme point in \( \mathcal{E} \) is always a solution to \( \min_{x \in X} c^T x \). In our case, \( X \) is the Birkhoff polytope and \( \mathcal{P} \) the set of permutation matrices. Throughout the paper, we will cast linear programs as

\[
LP(c, X) : \text{ minimize } c^T x \text{ subject to } x \in X,
\]

and we will assume that the solution returned is always an extreme point—which is the case if we solve the LP with a simplex-type method [25].

IV. REVISITING BIRKHOFF’S ALGORITHM

This section revisits Birkhoff’s algorithm. The main technical contribution is Theorem 1, which establishes that the number of permutation matrices in Birkhoff’s approach increases logarithmically with the decomposition error.

A. Approximate Birkhoff Decomposition Problem

The mathematical problem we want to solve is the following. For a given \( n \times n \) doubly stochastic matrix \( X^* \) and an \( \epsilon \geq 0 \), our goal is to find a small collection of permutation matrices \( P_1, P_2, \ldots, P_k \) and weights \( \theta_1, \theta_2, \ldots, \theta_k > 0 \) with \( \sum_{i=1}^{k} \theta_i = 1 \) such that \( \|X^* - \sum_{i=1}^{k} \theta_i P_i\|_F \leq \epsilon \). Recall a matrix \( X \in [0, 1]^{n \times n} \) is doubly stochastic if every row and column sums to one. That is, \( X1 = 1 \) and \( 1^T X = 1^T \). Also, a doubly stochastic matrix is a permutation if its entries are binary.
B. Algorithm Description

The original Birkhoff algorithm is described in Algorithm 1, and consists of two steps. First, the algorithm computes a permutation \( P_k \) and a weight \( \theta_k > 0 \) that satisfy the following three conditions:

\[
X_{k-1}(a, b) + \theta_k P_k(a, b) \leq X^*(a, b) \quad \forall a, b \in [n] \tag{3}
\]

\[
\theta_k > 0 \quad \forall k \geq 1 \tag{4}
\]

\[
\sum_{i=1}^{k} \theta_i \leq 1 \quad \forall k \geq 1 \tag{5}
\]

The second step is to add \( \theta_k P_k \) to the previous approximate decomposition, i.e., \( X_k = X_{k-1} + \theta_k P_k \). Hence, \( X_k(a, b) \leq X^*(a, b) \) for all \( a, b \in \{1, \ldots, n\} \). The algorithm terminates when the approximate decomposition \( X_k \) is \( \epsilon \) close to \( X^* \), or when the maximum number of admissible permutations \( (k_{\text{max}}) \) is reached. See [29, Fig. 2] for a decomposition example.

C. Convergence

We proceed to establish the convergence of Birkhoff’s algorithm. We start by presenting the following lemma, which lower and upper bounds \( \|X_k - X^*\|_F \).

**Lemma 2:** Consider the setup in Algorithm 1. The following two bounds hold:

\[
\|X_k - X^*\|_F \geq \left( 1 - \sum_{i=1}^{k} \theta_i \right) \tag{6}
\]

\[
\|X_k - X^*\|_F \leq \sqrt{n} \prod_{i=1}^{k} \left( 1 - \frac{n \theta_i^2}{\|X_{i-1} - X^*\|_F^2} \right) \tag{7}
\]

where \( \theta_i \leq \frac{1}{\sqrt{n}} \|X_{i-1} - X^*\|_F \).

**Proof:** See the Appendix.

The bounds in Lemma 2 are very general as they hold for any collection of permutation matrices and weights that satisfy the conditions in Eqs. (3)-(5). The lower bound in Eq. (6) tells us that the approximate decomposition error is at least \( 1 - \sum_{i=1}^{k} \theta_i \), and so we will have an exact decomposition (i.e., serve 100% of the traffic demand) only if \( \sum_{i=1}^{k} \theta_i = 1 \). The upper bound in Eq. (7) shows how the decomposition error depends on the weights \( \theta_i \) and the previous approximations \( \|X_{i-1} - X^*\|_F^2 \), \( i = 1, \ldots, k \). In particular, on the ratio \( n \theta_i^2/\|X_{i-1} - X^*\|_F^2 \), which captures how large \( \theta_i \) is with respect to the previous approximation. Note the values that \( \theta_i \) can take depend on \( \|X_{i-1} - X^*\|_F^2 \) as we must always satisfy the conditions in Eqs. (3)-(5). Finally, we note that finding a joint collection of weights and permutation matrices that minimize the RHS of Eq. (7) for a fixed \( k \) is as difficult as minimizing \( \|X_k - X^*\|_F^2 \) directly, since the RHS of Eq. (7) depends on \( \|X_{i-1} - X^*\|_F \), \( i = 1, \ldots, k \). Because of the latter, we study how to minimize the RHS of Eq. (7) in an iterative manner: for a given collection of permutation matrices \( P_i \) and weights \( \theta_i \) with \( i = 1, \ldots, k-1 \), our goal is to find a permutation matrix \( P_k \) and weight \( \theta_k \) that decrease the RHS of \( \epsilon \)-approximate decomposition and to know how to select “good” permutation matrices. To start, let \( \mu_i := n \theta_i^2/\|X_{i-1} - X^*\|_F^2 \) and rewrite Eq. (7) as

\[
\|X_k - X^*\|_F \leq \sqrt{n} \prod_{i=1}^{k} (1 - \mu_i) \tag{8}
\]

Note that \( (1 - \mu_i) \in [0,1) \) for all \( i = 1, 2, \ldots \) since \( \theta_i \geq 0 \) and \( \theta_i \leq \frac{1}{\sqrt{n}} \|X_{i-1} - X^*\|_F \) by Lemma 2. Hence, we have that \( X_k \rightarrow X^* \) as \( k \rightarrow \infty \) and so the algorithm converges. Now, suppose there exists a constant \( \mu_{\text{min}} > 0 \) such that \( \mu_{\text{min}} \leq \mu_i \) for all \( i \geq 1 \). Then, the bound in Eq. (8) simplifies to

\[
\|X_k - X^*\|_F \leq \sqrt{n} (1 - \mu_{\text{min}})^k/2 \tag{9}
\]

The last equation tells us that the approximation error decreases exponentially with the number of permutations. For example, if \( \mu_{\text{min}} = 1/2 \), we have that \( \|X_k - X^*\|_F \leq \sqrt{n}(1/2)^k/2 \), which means that every additional permutation in the decomposition decreases the approximation error by at least half. The ratio \( \kappa := 1/\mu_{\text{min}} \geq 1 \) can be regarded as the condition number in convex optimization with a smooth and strongly convex objective function [30][Section 9.1.2 and 9.3.1].

The following lemma establishes an upper bound on the number of permutations required to obtain an \( \epsilon \)-approximate decomposition provided that a constant \( \mu_{\text{min}} > 0 \) exists.

**Lemma 3:** Suppose \( \mu_{\text{min}} > \mu_{\text{min}} \) for all \( i = 1, \ldots, k \) for some constant \( \mu_{\text{min}} > 0 \). Then, Algorithm 1 obtains an \( \epsilon \)-approximate decomposition with at most

\[
k \leq 2 \log^{-1} \left( \frac{1}{1 - \mu_{\text{min}}} \right) \log \left( \frac{\sqrt{n}}{\epsilon} \right)
\]

permutation matrices.

Lemma 3 says that if a constant \( \mu_{\text{min}} \) exists, then the number of permutations required to obtain an \( \epsilon \)-approximate decomposition has a logarithmic dependence with \( \epsilon \). Hence, it remains to show whether such constant exists. Or equivalently, that we can select a \( \theta_i \) such that \( n \theta_i^2/\|X_{i-1} - X^*\|_F^2 \) is uniformly lower bounded by a strictly positive constant. We show that in the following theorem, which is one of the main contributions of the paper.

---

3The procedure proposed by Birkhoff in 1946 [17] does not specify how to compute such permutation and weight. The subroutine `PERM` described in Algorithm 2—which we will present in Theorem 1—returns a permutation \( P_k \) and a weight \( \theta_k \) that satisfy Eqs. (3)-(5).
Algorithm 3 BIRKHOFF_STEP

1: Input: $X^*$, $X_{k-1}$, and $P_k$
2: return $\min_{a,b} \{(X^*(a,b) - X_{k-1}(a,b) - 1)P_k(a,b) + 1\}$

Theorem 1: Consider Algorithm 1 and replace step $\bullet$ with the subroutine PERM (Algorithm 2) where

$$\mathcal{I}_k(\alpha) = \{P \in \mathcal{P} \mid X_{k-1}(a,b) + \alpha P(a,b) \leq X^*(a,b)\}$$

with $\alpha = \sqrt{\frac{\mu_{\min}}{n}}(1 - \sum_{i=1}^{k} \theta_i)$ and $\mu_{\min} = 1/n^3$. Algorithm 1 obtains an $\epsilon$-approximate decomposition with at most

$$k \leq 2 \log^{-1}\left(1 - \min_{i \in \{1,\ldots,k\}} \frac{n\theta^2_{\epsilon}}{\|X_{i-1} - X^*\|_F^2}\right) \log \left(\frac{\sqrt{n}}{\epsilon}\right)$$

permutation matrices.

Proof: See the Appendix.

Theorem 1 establishes that by selecting permutation matrices from set $\mathcal{I}_k(\alpha) \subseteq \mathcal{P}$, and weights as indicated in Algorithm 2, then the number of permutation matrices required to obtain an $\epsilon$-decomposition increases logarithmically with $\epsilon$. Set $\mathcal{I}_k(\alpha)$ is necessary to enforce that the conditions in Eqs. (3)–(5) are satisfied, but also to push Birkhoff’s algorithm to make sufficient progress in every iteration. Observe that the threshold $\alpha$ is bounded away from zero and that this depends on the constant $\mu_{\min} = 1/n^3$. Finally, we have written $\min_{i \in \{1,\ldots,k\}} n\theta^2_{\epsilon}/\|X_{i-1} - X^*\|_F^2$ instead of $\mu_{\min}$ in Eq. (11) (c.f. Lemma 3) to emphasize two points. The first one is that $\mu_{\min}$ is over-conservatively small, and that we can in general obtain a much sharper upper bound. In the numerical evaluation (Section VII-B.1), we show the condition numbers ($\kappa = 1/\mu_{\min}$) of different algorithms. The second point is that $n\theta^2_{\epsilon}/\|X_{i-1} - X^*\|_F^2$ is a quantity that we can measure and so use as a criterion for selecting a “good enough” permutation matrix. Importantly, note that the PERM subroutine does not specify which specific permutation to select from set $\mathcal{I}_k(\alpha)$, which is in marked contrast to previous approaches (e.g., [8], [9], [16]), which use a predefined criterion for selecting permutation matrices and weights.

D. Discussion

1) Max-Min Birkhoff Algorithms: The most popular variant of Birkhoff’s algorithm (e.g., [8], [15]) aims to find a permutation matrix with the largest associated weight. Such approach corresponds to solving the following optimization problem:

$$\begin{align*}
\text{maximize} & \quad \theta \\
\text{subject to} & \quad X_{k-1}(a,b) + \theta P(a,b) \leq X^*(a,b) \\
& \quad \forall a, b \in [n] \\
\end{align*}$$

(12)

where $\theta > 0, P \in \mathcal{P}$ and $\theta_k > 0$ is a step size and $M_k \in \{-1,0,1\}^{n \times n}$ a matrix that indicates the direction in which to update each of the coordinates. Birkhoff’s approach can be regarded as a special case where the $M_k$ matrices are permutations, and so have constraints on the group of coordinates can be jointly updated. Also, there are no negative coordinates since by selecting $X_0 = \{0\}^{n \times n}$ as starting point the algorithm only needs to “move forward.” To conclude, we note that our sparsity result is connected to the linear convergence rate obtained by convex optimization algorithms that exploit the smoothness and strong convexity of the objective function [30, Ch. 4].

V. FRANK-WOLFE FOR THE APPROXIMATE BIRKHOFF DECOMPOSITION

In this section, we show how the Frank-Wolfe (FW) algorithm and its fully corrective variant (FCFW) can be used to decompose a doubly stochastic matrix. The main contributions are to give explicit sparsity bounds for the FW and FCFW algorithms (Theorem 2 and 3) and to discuss their drawbacks (low sparsity and high complexity, respectively). We also compare how Frank-Wolfe and Birkhoff select permutation

The method is usually expressed in vector form. In our case, we can create a vector by stacking the matrix columns.
Algorithm 4 Birkhoff (Vector Form)

1. **Input:** Birkhoff polytope \( \mathcal{B} \), \( x^* \in \mathcal{B} \), \( \epsilon \geq 0 \), \( k_{\text{max}} \geq 1 \)
2. **Set:** \( k = 1 \) and \( x_0 = 0 \)
3. **while** \( \|x^* - x_{k-1}\|_2 > \epsilon \) and \( k \leq k_{\text{max}} \) **do**
   4. \( p_k \leftarrow \text{LP} \left( -[x^* - x_{k-1}], \mathcal{B} \right) \)
   5. \( \star: \theta_k \leftarrow \text{BIRKHOFF\_STEP}(x^*, x_{k-1}, p_k) \)
   6. \( x_k \leftarrow x_{k-1} + \theta_k p_k \)
   7. \( k \leftarrow k + 1 \)
8. **end while**
9. **return** \((p_1, \ldots, p_{k-1}), (\theta_1, \ldots, \theta_{k-1})\)

matrices (Observation 1) and show that Frank-Wolfe provides the “steepest descent” permutation (Observation 2). The latter will be key for selecting permutation matrices in the Birkhoff+ algorithm we will present in Section VI.

A. Birkhoff Polytope and Algorithm in Vector Form

In the rest of the paper, it will be more convenient to write \( n \times n \) doubly stochastic matrices as \( n^2 \)-dimensional vectors\(^9\) in the set

\[
\mathcal{B} := \{ x \in \mathbb{R}^d \mid x \succeq 0, \ Ax = b \},
\]

where \( d = n^2 \), \( A \in \{0, 1\}^{2n \times d} \), and \( b := \{1\}^{2n} \). Matrix \( A \) contains the \( 2n \) equality constraints that characterize the Birkhoff polytope (i.e., the sum of the columns and rows of a doubly stochastic matrix must be equal to 1). The specific structure of \( A \) can be derived easily and is given in the Appendix. As before, we use set \( \mathcal{P} \subset \{0, 1\}^{2n} \) to denote the set of permutation matrices or extreme points, but now these are in column form. The terms extreme point and permutation matrix will be used interchangeably in the rest of the paper. Finally, Algorithm 4 contains the procedure of the classic Birkhoff algorithm [17] in vector form,\(^10\) which is a special case of the more general Algorithm 1. Permutation matrices are selected by solving the linear program \( \text{LP} \left( -[x^* - x_{k-1}], \mathcal{B} \right) \) and the step sizes as large as possible provided \( x_k \succeq x^* \) for all \( k \geq 1 \) (see Section III-B).\(^11\) The \( \text{LP} \left( -[x^* - x_{k-1}], \mathcal{B} \right) \) returns any admissible permutation matrix (see Section III) and \([\cdot]\) denotes the entry-wise ceiling of a vector.

B. Frank-Wolfe Overview

In short, the Frank-Wolfe algorithm is a numerical method for minimizing a convex function \( f \) over a convex set contained in the convex hull of a set of discrete points or atoms [28]. In our case, the convex set is the Birkhoff polytope (\( \mathcal{B} \)) and the atoms the set of permutation matrices (\( \mathcal{P} \)). In each iteration, the algorithm selects an extreme point with the update

\[
p_k \in \arg\min_{u \in \mathcal{P}} \nabla f(x_{k-1})^T u
\]

\(^9\)Instead of having a matrix \( Z \in \mathbb{R}^{n \times n} \) such that \( Z1 = Z^T 1 = 1 \), we work with a vector \( x := (z_1, \ldots, z_n) \) where \( z_i \) is the \( i \)th column of \( Z \).

\(^10\)The algorithm corresponds to the method of proof employed by Birkhoff to show that a doubly stochastic matrix is an arithmetic mean of permutation matrices. See [17], theorem on page 1.

\(^11\)This permutation choice satisfies Eqs. (3)-(5).

and chooses a step size \( \theta > 0 \) such that \( f(x_{k-1} + \theta p_k - x_{k-1}) \) is convex and \( f(x_{k-1}) \) is the extreme point that is a direction in which it is possible to improve the objective function. The step size can be selected in a variety of ways (e.g., constant, line search, etc.) and differently from the previous section, Frank-Wolfe does not require that \( x_{k-1} + \theta p_k \succeq x^* \). Frank-Wolfe adds a constant \( \theta \) to streamline exposition but also because it allows us to make the following observations:

**Observation 1 (Weighted Search Direction):** For this particular choice of objective function, we have that \( \nabla f(x_{k-1}) = -(x^* - x_{k-1}) \). Hence, the update in Eq. (14) becomes

\[
p_k \in \arg\min_{u \in \mathcal{P}} -(x^* - x_{k-1})^T u,
\]

which is equivalent to solving the linear program \( \text{LP} \left(-[x^* - x_{k-1}], \mathcal{B}\right) \). That is, computing an extreme point with Frank-Wolfe and Birkhoff \( \mathcal{B} \times \mathbb{R}^n \) is the same except for the ceiling.\(^12\) Recall also that with FW there is not requirement that \( x^* \succeq x^\ast \).

**Observation 2 (Steepest Descent Permutation):** The extreme points selected by Frank-Wolfe corresponds to the target value \( x^* \). Note that \( x^* - x_{k-1} = -\nabla f(x_{k-1}) \) is the direction that goes straight to the target value \( x^* \), and that

\[
\arg\min_{u \in \mathcal{P}} \nabla f(x_{k-1})^T u
\]

\[= \arg\min_{u \in \mathcal{P}} \| \nabla f(x_{k-1}) \|_2 \| u \|_2 \cos \phi(\nabla f, u)
\]

\[= \arg\min_{u \in \mathcal{P}} \cos \phi(\nabla f, u)
\]

\[^{12}\]There exists a constant \( L \) such that \( f(y) \leq f(x) + \nabla f(x)^T (y - x) + \frac{L}{2} \| y - x \|_2^2 \) for all \( x, y \in \mathcal{B} \).

Authorized licensed use limited to: TU Delft Library. Downloaded on January 04,2022 at 07:10:09 UTC from IEEE Xplore. Restrictions apply.
where (a) follows from the dot product and (b) since \( \| p \|_2 = \sqrt{n} \) for all \( p \in P \), and \( \| \nabla f(x) \|_2 \) does not depend on \( p \). The RHS of the last equation corresponds to maximizing \( \cos(\rho_{-f,p}) \), which is equivalent to finding the \( L \) in \( P \) that minimizes the angle between \(-\nabla f(x) = (x^* - x)\) and \((p-x)\). Furthermore, since the Birkhoff polytope is regular and the number of extreme points increases factorially with \( n \), we can expect \( \phi_{-f,p} \) to be small. Figure 1 shows, schematically, how Frank-Wolfe selects the extreme points that has the smallest angle with respect to \((x^* - x)\). The black dots with a red cross are “non-descent” permutations that will not improve the decomposition approximation.

Both observations rely on the objective function being quadratic; however, we can expect similar properties for other smooth convex objectives. For example, we could use \( f(x) = (x^* - x)^T Q (x^* - x) \) where \( Q \) is a positive definite matrix that emphasizes which of the components in vector \( x^* - x \) to minimize. In Section VI, we will include a log-barrier function to the objective. In the rest of the section, we will use a quadratic objective function to streamline exposition.

C. Frank-Wolfe With Line Search

The procedure of the Frank-Wolfe algorithm is given in Algorithm 5. Differently from Birkhoff’s approach, \( F^W \) uses an extreme point as a starting point instead of the origin. Note that \( 0 \not\in \mathcal{B} \). The choice of step size is indicated in step 3, and corresponds to carrying out line search. This can be easily verified. Let \( x_k := x_{k-1} + \theta_k (p_k - x_{k-1}) \) be the \( k \)'th iterate, and observe that we can write

\[
\frac{1}{2} \| x_k - x^* \|^2_2 - \frac{1}{2} \| x_{k-1} - x^* \|^2_2 = \frac{1}{2} \| x_{k-1} + \theta_k (p_k - x_{k-1}) - x^* \|^2_2 - \frac{1}{2} \| x_{k-1} - x^* \|^2_2 = \theta_k (x_{k-1} - x^*)^T (p_k - x_{k-1}) + \frac{\theta_k^2}{2} \| p_k - x_{k-1} \|^2_2.
\]

The RHS of the last equation is a quadratic function in \( \theta_k \), and its minimizer can be obtained in closed form. And since equality holds in the last equation, minimizing the quadratic function on the RHS is equivalent to minimizing the LHS with line search. Hence, Algorithm 5 corresponds to Frank-Wolfe with line search, and so from [28, Theorem 1] we have the bound

\[
\| x_k - x^* \|^2_2 \leq \frac{4n}{k+2}.
\]  

Algorithm 5 Frank-Wolfe (F\textsuperscript{W}) With Quadratic Objective and Line Search

1: As Algorithm 4, but set \( x_0 \in P \) and replace lines \( \circ, \circ, \circ \) with
2: \( p_k \leftarrow \text{LP} \left(- (x^* - x_{k-1}), B \right) \)
3: \( \theta_k \leftarrow \frac{\| x_{k-1} + \theta_k (p_k - x_{k-1}) - x^* \|^2_2 - \| x_{k-1} - x^* \|^2_2}{\| p_k - x_{k-1} \|^2_2} \)
4: \( x_k \leftarrow x_{k-1} + \theta_k (p_k - x_{k-1}) \)

D. Fully Corrective Frank-Wolfe (FC\textsuperscript{FW})

This variant of Frank-Wolfe algorithm recomputes the weights assigned to vectors \( p_1, \ldots, p_k \) in every iteration \( k \). The FC\textsuperscript{FW} procedure is described in Algorithm 6. Like the F\textsuperscript{W} algorithm, FC\textsuperscript{FW} selects a \( x_0 \in P \) and computes a new permutation/extreme point by solving a linear program \( \text{LP} \left( \nabla f(x_{k-1}), B \right) \). The permutations are collected in matrix \( V_k \), and the weights \( (\theta_1, \ldots, \theta_k) \) are selected to minimize \( \| V_k (\theta_1, \ldots, \theta_k) - x^* \|^2_2 \) subject to \( \sum_{i=1}^k \theta_i = 1 \) and \( \theta_i \geq 0 \) for all \( i = 1, \ldots, k \). Importantly, the (re)computation of weights in Algorithm 6 involves solving a quadratic program (QP) of dimension \( i = 1, \ldots, k \). By Theorem 1 in [19], we have the bound

\[
\| x_k - x^* \|^2_2 \leq \| x_0 - x^* \|^2_2 \exp \left( -\frac{\mu}{4L} \left( \frac{\lambda}{M} \right)^2 k \right),
\]

where \( \mu/L \) is the condition number and \( (\lambda/M)^2 \) the eccentricity of the Birkhoff polytope. These two parameters are usually not known, however, not in our problem since the Birkhoff polytope and the objective function \( f(x) = (1/2) \| x - x^* \|^2_2 \) have remarkable structure. We establish the eccentricity of the Birkhoff polytope in the next lemma.

Lemma 4: The eccentricity \( (\lambda/M)^2 \) of the Birkhoff polytope is lower bounded by \( 1/(2n^3) \).

Proof: See the Appendix.

Using the last lemma and the fact that the condition number of the objective function \( (\mu/L) \) is equal to 1, we can obtain the FC\textsuperscript{FW}’s sparsity.

Algorithm 6 Fully Corrective Frank-Wolfe (FC\textsuperscript{FW})

1: As Algorithm 4, but set \( x_0 \in P \) and define \( V_0 = \emptyset \). Let \( \Delta_k \) be the \( k \)-simplex. Replace lines \( \circ, \circ, \circ \) with
2: \( p_k \leftarrow \text{LP} \left( \nabla f(x_{k-1}), B \right) \)
3: \( V_k \leftarrow [V_{k-1}, p_k] \)
4: \( (\theta_1, \ldots, \theta_k) \leftarrow \arg \min_{\theta \in \Delta_k} \| V_k \theta - x^* \|^2_2 \)
5: \( x_k \leftarrow V_k (\theta_1, \ldots, \theta_k) \)

Rearranging terms in Eq. (15), we can obtain an upper bound on the sparsity of F\textsuperscript{W}.

Theorem 2 (F\textsuperscript{W} Sparsity): Algorithm 5 obtains an \( \epsilon \)-approximate decomposition with at most \( k \leq 4n/\epsilon^2 \) permutation matrices, where \( \epsilon = \| x_k - x^* \|_2 \).

The bound in Theorem 2 says that the sparsity of F\textsuperscript{W} is of the order of \( O(1/\epsilon^2) \), so we may not be able to obtain sparse decompositions if \( \epsilon \) is small. One of the issues with first-order-methods such as F\textsuperscript{W} is the zig-zagging phenomenon when the approximate decomposition is close to \( x^* \). Hence, even though F\textsuperscript{W} selects the steepest descent direction, the step size choice \( (\theta_k) \) is not enough. One way to avoid zig-zagging is to recompute the weights of all the atoms or extreme points discovered so far, which is in essence what the fully corrective variant of the algorithm does.
decompositions with accuracies below $10^{-2}$ but finding a good permutation matrix is slow as it requires to represent the convex hull of $x_{k-1}$ and the permutation matrices in $\mathcal{I}_k(\alpha)$.

**Theorem 3 (FCFW Sparsity):** Algorithm 6 obtains an $\epsilon$-approximate decomposition with at most $k \leq 8n^3 \log(2n/\epsilon^2)$ permutation matrices, where $\epsilon = \|x_k - x^*\|_2$.

**Proof:** See the Appendix.

VI. NEW ALGORITHM

Birkhoff and FCFW algorithms have both logarithmic sparsity, but they are very different algorithmically. On the one hand, weights are easy to compute in Birkhoff’s approach, but finding a good permutation matrix is slow as it requires to solve multiple linear programs (e.g., [8]). In contrast, FCFW can obtain a good permutation matrix by solving a single linear program (see Observation 2), but it requires to solve a quadratic program to (re)calculate the weights.

In this section, we present Birkhoff+ (Algorithm 7), a variation of the original Birkhoff’s algorithm that uses the intuition behind Frank-Wolfe to obtain sparse decompositions in a fast manner. The performance of Birkhoff+ is evaluated in Section VII.

A. Approach

The intuition behind our approach is shown schematically in Figure 2. In brief, Birkhoff’s algorithm (Figure 2a) can be seen as constructing a path from the origin ($x_0 = 0$) to the target value ($x^*$) while always remaining in the dotted box (i.e., $x_k \leq x^*$ for all $k = 0, 1, 2, \ldots$). Frank-Wolfe (Figure 2b), on the other hand, constructs a path from a permutation matrix $x_0 \in \mathcal{P}$ to the target value $x^*$ within the polytope of doubly stochastic matrices (blue surface). Our approach (Figure 2c) can be regarded as using Frank-Wolfe within the polytope $\text{conv}(\mathcal{I}_k(\alpha) \cup x_{k-1})$ (yellow polygon in Figure 2c) with the additional constraint that the approximate decomposition must be within the dotted box. That is, we want to use the path or permutations that Frank-Wolfe would select while remaining in the box that characterizes the Birkhoff’s approach. It is important to use $\text{conv}(\mathcal{I}_k(\alpha) \cup x_{k-1})$ instead of $\text{conv}(\mathcal{P} \cup x_{k-1})$ (i.e., all permutations) as the algorithm may otherwise not converge. The latter is shown formally in the following theorem.

**Theorem 4:** Consider Algorithm 4 and replace line o with $\text{LP}(-x^* - x_{k-1})$. Then, there may not exist a $k$ for which $\|x_k - x^*\|_2^2 \leq \epsilon$ for any $\epsilon > 0$.

We can prove the theorem by example. Suppose we want to decompose the following $n \times n$ doubly stochastic matrix

$$
\begin{bmatrix}
\frac{1}{n-1} & 0 & \cdots & 0 & 1 \\
0 & n-2 & 1 \\
\vdots & \ddots & \ddots & \vdots \\
0 & \cdots & n-2 & 1 \\
1 & 1 & \cdots & 1 & 0
\end{bmatrix}
$$

(17)

That is, (i) the first $n-1$ entries in the diagonal are equal to $(n-2)/(n-1)$, (ii) the first $n-1$ entries of the last row are equal to $1/(n-1)$, and (iii) the first $n-1$ entries of the last column are equal to $1/(n-1)$. Note the sum of each row and column is equal to one. Next, suppose that $f(x) = (1/2)\|x^* - x\|_2^2$ where $x^*$ is the matrix in Eq. (17) in column form. In the first iteration $(x_0 = 0)$, Frank-Wolfe selects a permutation by solving the linear program $\text{LP}(-x^*, B)$, the solution of which is the identity matrix since the doubly stochastic matrix in Eq. (17) is diagonally dominant. And because the last entry of the matrix in Eq. (17) is equal to zero, we have that $\text{BIRKHOFF\_STEP}(x^*, x_{k-1}, p_k) = 0$ and therefore $x_k = x_{k-1}$. That is, the algorithm will be “stuck.”

In sum, a Birkhoff-type algorithm that selects permutation matrices with Frank-Wolfe using all the permutation matrices $\mathcal{P}$ may not converge. However, we can use Frank-Wolfe with the permutations in the set $\mathcal{I}_k(\alpha)$, which ensures not only that the algorithm converges but that this has logarithmic sparsity (Theorem 1).

1) Objective Function With Barrier: Since Birkhoff’s approach restricts $x_k$ to remain in the Birkhoff’s dotted box (see Figure 2), it is reasonable to use an objective function that aims to construct a path to $x^*$ from within the box. For that, we define

$$
f_\beta(x) = f(x) - \beta \sum_{j=1}^d \log(x^*(j) - x(j)) + \epsilon/d, \quad (18)
$$

where $\beta \geq 0$ and $x(j)$ is the $j$'th component of vector $x$. Note that $f_\beta$ is convex as this is the composition of $f$ plus a convex penalty/barrier function $-\beta \sum_{j=1}^d \log(x^*(j) - x_{k-1}(j)) + \epsilon/d$. The term $\epsilon/d$ in the barrier is used for numerical stability as otherwise the barrier goes to $+\infty$ when $x^*(j) = x(j)$. The motivation for using a barrier function comes from interior point methods in optimization, where parameter $\beta$ is typically tuned throughout the algorithm to allow $x_k \to x^*$. Note that $f_\beta \to f$ as $\beta \to 0$. 
Algorithm 7 Birkhoff+
1: As Algorithm 4, but take $\beta \geq 0$ also as input. Replace line 2 with
2: $\alpha \leftarrow (1 - \sum_{i=1}^{k} \theta_i)/n^2$
3: $p_k \leftarrow \text{LP}(\nabla f_{\beta}(x_{k-1}), \text{conv}(I_k(\alpha)))$

Algorithm 8 Birkhoff+(max_rep) — With Permutation Selection Refinement
1: As Algorithm 7, but replace line 2 with
2: for $i = 1, \ldots, \text{max_rep}$ do
3: $p_i \leftarrow \text{LP}(\nabla f_{\beta}(x_{k-1}), \text{conv}(I_k(\alpha)))$
4: $\theta_i \leftarrow \text{BIRKHOFF_STEP}(x^*, x_{k-1}, p_k)$
5: if $(\theta_i > \alpha) \alpha \leftarrow \text{BIRKHOFF_STEP}(x^*, x_{k-1}, p_k)$
6: else exit while loop
7: $p_k \leftarrow p_i$
8: end for

B. Birkhoff+ Algorithm Description and Complexity

The procedure of Birkhoff+ is described in Algorithm 7, and consists of replacing how permutation matrices are selected in Algorithm 4 with $\text{LP}(\nabla f_{\beta}(x_{k-1}), \text{conv}(I_k(\alpha)))$, where $f_{\beta}$ is as defined in Eq. (18). Parameter $\beta$ can be selected to emphasize the barrier over the objective function $f$. In our case, we do not need $\beta \rightarrow 0$ as by selecting permutations from $I_k(\alpha)$ is enough to allow the algorithm to make progress. The convergence of the algorithm is stated formally in the following corollary.

Corollary 2: Algorithm 7 obtains an $\epsilon$-approximate decomposition with at most $k \leq O(\log(1/\epsilon))$ permutation matrices.

The complexity of Birkhoff+ per iteration is equal to solving a linear program with a simplex type method. The linear program $\text{LP}(\nabla f_{\beta}(x_{k-1}), \text{conv}(I_k(\alpha)))$ can be carried out with $\text{LP}(\nabla f_{\beta}(x_{k-1}) + b_k, B)$ where $b_k = d/\epsilon \cdot I_{[0,1]}(x^* - x_{k-1} \geq \alpha)$ is a penalty vector to force the solver to do not select the components of vector $x^* - x_k$ smaller than $\alpha$.

Finally, we note that Birkhoff+ depends on how we define set $I_k(\alpha)$. Algorithm 8 is a meta-heuristic for selecting $\alpha$ based on Birkhoff’s step size. In particular, $\alpha$ is set to $(1 - \sum_{i=1}^{k} \theta_i)/n^2$ in the first iteration and then equal to the largest step size for the permutation selected using the Frank-Wolfe-type update. The search for a large $\alpha$ terminates when the maximum number of repetitions (max_rep) is reached or the value of $\alpha$ does not increase. We call Algorithm 8 Birkhoff+(#), where # indicates the maximum number of permutation refinements. Birkhoff+(1) is equivalent to Birkhoff+ as it computes only one permutation matrix.

VII. NUMERICAL EVALUATION

In this section, we evaluate performance of Birkhoff+ and compare it to existing algorithms. Our goal is to illustrate the algorithms’ characteristics and how different traffic matrices affect the performance of a circuit switch in terms of throughput, configurations computation time, and number of configurations. The code of Birkhoff+ is available as a Julia package in [37].

A. Setup

The Birkhoff, FW, FCFW, Birkhoff+ and Birkhoff+(#) algorithms are implemented in Julia [38] and as indicated in Algorithms 4, 5, 6, 7 and 8 respectively. Parameter $\beta$ is fixed to 1 and the maximum number of permutation refinements in Birkhoff+(#) to 10 — however, we observe in the experiments that the actual number of permutation refinements is usually less than 3. Solstice corresponds to Algorithm 2 in [8], and Eclipse to Algorithm 2 in [9]. The linear programs $\text{LP}(\cdot, \cdot)$ are carried out with Clp [39] in all algorithms and return an extreme point/permutation matrix. The quadratic programs in the FCFW algorithm are carried out with Ipopt [35]. Both solvers are open-source.

Traffic demand matrices are generated by sampling permutations uniformly at random, and weights are selected to model the type of load in data centers. In particular, we follow the evaluation scenario in [9], where traffic matrices are sparse and consist of 12 flows. Three of the flows are large and carry the 70% of the load, while the rest are small flows and carry the remaining 30% of the traffic. We note that the traffic matrix in practical scenarios may be below the switch’s capacity (i.e., the sum of each row or column may be smaller than one), and so we first need to add a virtual load to the traffic matrix to make it doubly stochastic.18 For simplicity, we assume in the evaluation that the demand matrices are doubly stochastic.

Finally, the numerical evaluation is carried out on a computer equipped with an Intel i7 8700B (3.2 GHz) CPU and 32 GB of memory. The version of Julia is 1.3.1.

B. Experiments

We first study the algorithms’ characteristics, and then show how those affect the performance of a circuit switch.

1) Decomposition Approximation vs. Number of Permutations and Time: We set $n = 32$ and sample traffic matrices as indicated in Section VII-A. Also, we fix $\epsilon = 10^{-4}$, $k_{\max} = 300$ and $\delta = 10^{-2}$ (just for Eclipse).19 Figure 3 shows the algorithms decomposition error in terms of permutations and time. The results are the average of 50 realizations.

Observe from Figure 3a that the decomposition error of Birkhoff is large until it converges exactly in the last iteration ($k \approx 250$). On the other hand, FW progresses quickly, but it slows down drastically around $\epsilon = 0.9$. The latter is due to the $O(1/\epsilon^2)$ sparsity rate and the zig-zagging phenomenon typical in first-order-methods (see Section V-C). The FCFW has a better sparsity performance than FW, but it cannot obtain decomposition with an $\epsilon$ below $0.5 \cdot 10^{-3}$ due to the numerical accuracy of the quadratic solvers (see Section V-D). Eclipse has a better performance than previous algorithms until it gets stuck between $\epsilon \in [10^{-2}, 10^{-1}]$.

18The work in [8] (see Section 4.2.1) uses the term “stuffing” for adding virtual load to the traffic matrix. Stuffing can be seen as a special type of projection of the demand matrix onto the Birkhoff polytope. Technically, for a demand matrix $D$, we need to find a matrix $S \in B - D$. Matrix $S$ may not be unique and finding the best virtual load matrix for our algorithm is an interesting problem but out of the scope of the paper.

19The value corresponds to having a switching cost of 10 ms.
weights (i.e., solve a quadratic program) every time it adds a new permutation to the decomposition.

2) Circuit Switch Performance: We now evaluate the algorithm’s performance when used to compute the switching configurations for a circuit switch with $n = 100$ ports. The performance metrics we evaluate are the throughput, the configurations computation time, and the number of switching configurations. We carry out three experiments where we vary the reconfiguration cost, the skewness and sparsity of the traffic matrix, and the configurations computation overhead. Importantly, now the traffic matrix $X_t$ is associated with a time window $W$ that enforces the decomposition to satisfy $\sum_{i=1}^{k} (\theta_i + \delta) \leq W$, i.e., the time spent transmitting ($\sum_{i=1}^{k} \theta_i$) and reconfiguring ($\delta k$) cannot exceed the time window duration ($W$). Finally, we only evaluate Solstice, Eclipse, Birkhoff+, and Birkhoff+(10) as (i) Birkhoff and FW have a poor performance, and (ii) FCFW is very slow when $n \geq 32$ (see times in Figure 3b).

Experiment 1 (Impact of Reconfiguration Time): Figure 4 shows the algorithms’ performance in terms of throughput, running time, and the number of configurations depending on the ratio $\delta/W$ (the impact of the reconfiguration delay proportionally to the time window duration). Observe from the figure that Birkhoff+(10) outperforms the other algorithms in terms of throughput. For instance, for $\delta/W = 10^{-2}$, Birkhoff+(10) achieves a 7% more throughput than Eclipse and Solstice. Birkhoff+ has almost the same throughput than Eclipse and Solstice. Regarding the time required to compute the switching configurations, Solstice and Eclipse are slower than Birkhoff+ and Birkhoff+(10) by an order of magnitude; however, the difference decreases as $\delta/W$ increases because we have fewer switching configurations as a result of larger reconfiguration penalties (c.f. Figure 4b and Figure 4c). Finally, observe from Figure 4c that Birkhoff+(10) can obtain decompositions with half of the configurations compared to other algorithms when $\epsilon$ is small (i.e., $10^{-4}$). Conclusions: Birkhoff+ has the same performance in terms of throughput and number of switching configurations than Solstice and Eclipse, but it is 10 times faster. Birkhoff+(10) obtains higher throughput than all algorithms and it is only slightly slower than Birkhoff+.

Experiment 2 (Sparsity and Skewness): Now we set $\delta/W = 10^{-2}$ and evaluate the algorithms’ performance depending on the skewness and the sparsity of the demand matrix. In Figure 5, we vary the fraction of the load carried by the small flows. Observe that as before, Birkhoff+(10) outperforms the other algorithms, and that Birkhoff+, Solstice, and Eclipse are almost the same in terms of throughput for different demand matrices. Furthermore, there is little variation on the running time and the number of switching configurations—despite a slight bend in the curves when the traffic matrix contains the same fraction of large and small flows.

In Figure 6, we show the results when we vary the number of permutations used to generate the demand matrix. Each permutation matrix is sampled as explained in Section VII-A. Observe from the figure that the sparsity of the traffic matrix
Fig. 4. Circuit switch performance (throughput, running time, and number of configurations) depending on $\delta/W$, where $\delta$ is the switching time and $W$ the time window duration. The figures show the average of 50 realizations.

Fig. 5. Circuit switch performance (throughput, running time, and number of configurations) depending on the load carried by the small flows. The figures show the average of 50 realizations.

Fig. 6. Circuit switch performance (throughput, running time, and number of configurations) depending on the number of permutations matrices used to generate the traffic matrix. The figures show the average of 50 realizations.

has a significant impact on the throughput, running time, and the number of the switching configurations. In particular, observe from Figure 6a that the throughput of all algorithms decreases and that Eclipse is comparable to Birkhoff+(10) as the traffic matrix becomes denser. However, the running time of Birkhoff+(10) does not explode (see Figure 6b) and Birkhoff+(10) does not get stuck when the traffic demand matrix is very sparse.\(^{22}\) Regarding Birkhoff+, observe that now the running times difference with Birkhoff+(10) becomes more noticeable as the demand matrix becomes denser. Finally, observe from Figure 6c that the number of switching configurations increases with the density of the traffic matrix for all algorithms.

Conclusions: The skewness of the demand matrix has little impact on to the performance of all algorithms. The sparsity, on the other hand, plays an important role. Eclipse has a similar performance than Birkhoff+(10), but it is notably slower.

Experiment 3 (Configurations Computation Overhead): This experiment shows how time to compute the switching configurations affects the circuit switch’s throughput. In particular, we set $\delta/W = 10^{-2}$ and truncate the decomposition to satisfy $\sum_{i=1}^{k} (\theta_i + \delta) \leq W - T$, where $T$ is the time to compute the switching configurations. The values of $T$ for this particular setting are given in Figure 4b. Figure 7 shows the throughput for different values of $W$ in seconds. Observe from the figure that the throughput increases with $W$ for all algorithms. When $W$ is small (i.e., the decomposition computation overhead is large), Birkhoff+ has a higher throughput than Birkhoff+(10) because it is faster—recall Birkhoff+...
when an upper bound on the Frobenius norm of a doubly stochastic program. However, Birkhoff+ (10)’s throughput is higher when \( W > 5 \) since the reconfiguration time is larger than the time to compute the switching configurations. Regarding Eclipse and Solstice, observe that both are affected heavily by the decomposition overhead. For instance, when \( W = 5 \), Birkhoff+ has 67% and 34% more throughput than Solstice and Eclipse respectively. Also, note that when \( W = 1 \), Birkhoff+ can serve 80% of the traffic whereas Solstice and Eclipse almost nothing. **Conclusions:** Birkhoff+ outperforms Solstice and Eclipse and it is slightly better than Birkhoff+ (10) when the time windows are short. As with the reconfiguration costs, the benefit of computing switching configurations fast diminishes as the time window duration increases.

### VIII. Conclusion

This paper studies how to compute switching configurations for circuit switches. We have revisited Birkhoff’s approach and established its properties in terms of the number of switching configurations required to obtain an approximate representation of a traffic matrix. A new algorithm (Birkhoff+) is proposed, which obtains representations with fewer switching configurations than previous work (Solstice, Eclipse) and is 10-100 times faster depending on the setting. The latter is important in terms of throughput when traffic bursts are short-lived, and so the time required to compute the switching configurations is a non-negligible overhead. We also propose a variant of Birkhoff+ that is slightly slower but obtains representations with even fewer switching configurations. The performance of the proposed algorithms is evaluated through exhaustive numerical experiments for traffic demand matrices that capture traffic characteristics in data centers.

### APPENDIX

#### A. Proofs of Section IV

We start by presenting two lemmas. The first lemma gives an upper bound on the Frobenius norm of a doubly stochastic matrix.

**Lemma 5:** \( \|X\|_F \leq \sqrt{n} \) for any doubly stochastic matrix \( X \).

**Proof:** Let \( r_i \) be the \( i \)’th row of \( X \) and note \( \|r_i\|_2 = 1 \) for all \( i \in \{1, \ldots, n\} \), i.e., the sum of a row is equal to 1. Observe

\[
\|X\|_F = \sqrt{\text{Tr}(XX^*)} = \sqrt{\sum_{i=1}^{n} \|r_i\|^2_2} \leq \sqrt{n},
\]

where the first inequality follows because \( \| \cdot \|_2 \leq \| \cdot \|_1 \).

The second lemma establishes that \( X^* - X_k \) is a scaled doubly stochastic matrix.

**Lemma 6:** Let \( X_k = \sum_{i=1}^{k} \theta_i P_i \) and suppose \( X_k(a, b) \leq X^*(a, b) \) for all \( a, b \in \{1, \ldots, n\} \) and \( k \geq 1 \). Then,

\[
\text{(a)} \quad \frac{X^* - X_k}{1 - \sum_{i=1}^{k} \theta_i} \quad \text{is doubly stochastic}
\]

\[
\text{(b)} \quad \|X^* - X_k\|_F \leq \sqrt{n} \left( 1 - \sum_{i=1}^{k} \theta_i \right)
\]

**Proof:** We start with (a). By assumption, \( 0 \leq X_k(a, b) \leq X^*(a, b) \leq 1 \) for all \( a, b \in \{1, \ldots, n\} \). Hence, we only need to show that the sum of each row and column is equal to one. Observe

\[
(1 - \sum_{i=1}^{k} \theta_i)^{-1}(X^* - X_k)P_i = (1 - \sum_{i=1}^{k} \theta_i)^{-1}P_i \quad \text{for all } i \in \{1, \ldots, n\}
\]

The same argument above can be used to show that \( (1 - \sum_{i=1}^{k} \theta_i)^{-1}P_i = 1 \), i.e., the sum of each column is equal to one.

For (b), observe \( \|X^* - X_k\|_F = \|1 - \sum_{i=1}^{k} \theta_i\|^{-1}\|X^* - X_k\|_F \leq \sqrt{n} \) by Lemma 5. Rearranging terms yields the result.

**Proof of Lemma 2**

We start by proving the lower bound. We first note \( \|X\|_F \geq 1 \) for any doubly stochastic matrix \( X \). Recall

\[
\|X\|_F \geq \frac{\|Xu\|_2}{\|u\|_2} \quad \text{for } u \in \mathbb{R}^n \quad \text{s.t. } u \neq 0
\]

Let \( u = 1 \) in the equation above to obtain

\[
\|X\|_F \geq \frac{\|X1\|_2}{\|1\|_2} = \frac{\|1\|_2}{\|1\|_2} = 1,
\]

where \( X1 = 1 \) follows since \( X \) is doubly stochastic. Next, since \( \frac{X^* - X_k}{1 - \sum_{i=1}^{k} \theta_i} \) is doubly stochastic by Lemma 6, we have

\[
1 \leq \frac{\|X^* - X_k\|_F}{\|1 - \sum_{i=1}^{k} \theta_i\|_F} = (1 - \sum_{i=1}^{k} \theta_i)^{-1}\|X^* - X_k\|_F
\]

Rearranging terms yields the lower bound.

For the upper bound, observe

\[
\|X_k - X^*\|_F^2 = \left\|X_{k-1} + \theta_k P_k - X^*\right\|_F^2 = \left\|X_{k-1} - X^*\right\|_F^2 + 2\theta_k \sum_{a,b} P_k(a,b)(X_{k-1}(a,b) - X^*(a,b))
\]

Authorized licensed use limited to: TU Delft Library. Downloaded on January 04,2022 at 07:10:09 UTC from IEEE Xplore. Restrictions apply.
holds for all \( a, b \in \{1, \ldots, n\} \) and \( j \in \{1, \ldots, (n-1)^2 + 1\} \). Furthermore, since \( \sum_{j=1}^{(n-1)^2+1} \beta_j = 1 \), we have that

\[
\beta_j \geq \frac{1}{(n-1)^2 + 1} \tag{21}
\]

for at least one \( j \in \{1, \ldots, (n-1)^2 + 1\} \). Let \( \alpha = \beta_j \) such that the last equation holds. Combining Eq. (20) and Eq. (21), we obtain that

\[
1 - \sum_{i=1}^{k-1} \theta_i P(a, b) = \alpha P(a, b) \leq X^*(a, b) - X_{k-1}(a, b)
\]

That is, there exists at least a permutation \( P \) such that \( X_{k-1} + \alpha P(a, b) \leq X^*(a, b) \), and so \( I_k(\alpha) \) is non-empty.

We are now in position to present the proof of Theorem 1. By Lemma 7, set \( I_k(\alpha') \) with \( \alpha' = \frac{1 - \sum_{i=1}^{k-1} \theta_i}{(n-1)^2 + 1} \) is non-empty. Now, observe that since \( \alpha = \frac{\sqrt{\sum_{i=1}^k \theta_i}}{\sqrt{1 - \mu_{\text{min}}} (1 - \mu_{\text{min}})^{k/2}} \) because \( \mu_{\text{min}} = 1/n^2 \), we have that \( I_k(\alpha') \subseteq I_k(\alpha) \) and so \( I_k(\alpha) \) is non-empty. The rest of the proof follows as in Lemma 3 with \( \mu_{\text{min}} = -\log_{n}(1 - \mu_{\text{min}}) / n \).

B. Proofs of Section V

**Birkhoff Polytope Representation in Vector Form:** The Birkhoff polytope is the set that contains all doubly stochastic matrices. Recall we say that a nonnegative matrix is doubly stochastic if the sum of its rows and columns is equal to one. This corresponds to having \( 2n \) equality constraints. We can express these in vector form by defining matrices

\[
A'(in + 1, in + j) = \begin{cases} 1, & i = 0, \ldots, n-1, \ j = 1, \ldots, n \\ 0, & \text{otherwise} \end{cases}
\]

\[
A''(j, in + j) = \begin{cases} 1, & i = 0, \ldots, n-1, \ j = 1, \ldots, n \\ 0, & \text{otherwise} \end{cases}
\]

and then collecting them in \( A = [A'; A''] \). Next, define \( b \in \{1\}^{2n} \). Any vector from \( \mathbb{R}_+^d \) such that \( Ax = b \) correspond to having doubly stochastic matrix in vector form.

For example, with \( n = 3 \) we have

\[
A = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}, \ b = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}
\]

**Proof of Lemma 4**

The eccentricity consists of two parameters. The diameter of the polytope \( (M) \) and its pyramidal width \( (\lambda) \). The diameter of the Birkhoff polytope is the maximum distance between two points in \( B \), which is the maximum distance between two vertices. Specifically, this is equal to \( \|p - p'\| = \sqrt{2n} \) where \( p, p' \in P \) are two vertices such that \( p^T p' = 0 \), i.e., have ones in different components.

It is possible to obtain a lower bound on the pyramidal width of the Birkhoff polytope by using the fact that its extreme
points are a subset of the extreme points of the unit cube in $d$ dimensions. Formally, $\mathcal{P} \subset \{0, 1\}^d$ and so $\text{conv}(\mathcal{P}) := B \subset C := \text{conv}(\{0, 1\}^d)$. The latter means that the unit cube is “extreme-point-wise denser” than the Birkhoff polytope and so it has smaller pyramidal width. From Lemma 4 in [19] we can obtain that the pyramidal width of the Birkhoff polytope is lower bounded by $1/\sqrt{d} = 1/n$.²³ Hence, $(\lambda/M)^2 \geq 1/(2n^3)$ as claimed.

**Proof of Theorem 3**

This theorem is an application of Theorem 1 in [19] with the quadratic objective function $f(x) = (1/2)||x - x^*||_2^2$ and set $B$. This theorem says that

$$||x_k - x^*||_2^2 \leq ||x_0 - x^*||_2^2 \exp \left( -\frac{\mu}{4L} \frac{(\lambda/M)^2}{2} \right).$$

The term $||x_0 - x^*||_2$ can be upper bounded by $\sqrt{n}$, which is the maximum Euclidean distance between two points in $B$ (see the proof of Lemma 4). The condition number $\mu/L$ is equal to 1 because the objective function is quadratic and $(\lambda/M)^2 \geq 1/(2n^3)$ by Lemma 4. Hence,

$$||x_k - x^*||_2 \leq 2n \exp \left( -\frac{k}{8n^3} \right).$$

To conclude, let $\epsilon^2 = ||x_k - x^*||_2^2$ and write $\epsilon^2 \leq 2n \exp \left( -\frac{k}{8n^3} \right)$. By expressing $k$ as a function of $\epsilon$ in the last equation, we obtain the stated result.

**ACKNOWLEDGMENT**

Víctor Valls would like to thank Ehsan Kazemi (Yale Institute for Network Science) for many helpful conversations and the counter-example in the proof of Theorem 4.

**REFERENCES**


²³Recall that $d = n^2$.