Generalized Two-Dimensional Shuffle-Exchange Problem

Anonymous

Anonymous

Abstract

In this paper, we consider the following combinatorial puzzle: given two $n \times m$ matrices M_1, M_2 with elements from $\{1, 2, ..., nm\}$ (not necessarily distinct) and two groups $G \leq S_{[m]}, H \leq S_{[n]}$ where S_T is the symmetric group on set T, the task is to transform M_1 to M_2 with minimum steps. In each step, there are two kinds of allowable operations: choose $g_1, ..., g_n \in G$ and perform each g_i on the i-th row in parallel; or choose $h_1, ..., h_m \in H$ and perform each h_i on the i-th column in parallel. This problem is a special case of the Cayley graph diameter problem on matrices. It can also be considered as a generalized version of the two-dimensional *Shuffle-Exchange problem*.

Here are our main results. First, we give a reduction from Graph Isomorphism to the Reachability problem, i.e., determining whether M_1 can be transformed to M_2, but we also show that it is unlikely to be **NP**-hard. Second, we prove that determining whether M_1 can be transformed to M_2 within k steps is **NP**-complete, where k is part of the input. In addition, we prove that the minimum number of steps for the transformation, if reachable, is upper bounded by $\text{poly}(n, m)$. This implies the underlying Cayley graph has exponential vertices but a polynomial diameter.

We also focus on a special case of the puzzle where M_1, M_2 contain distinct elements and G, H are cyclic groups. For this task, we present two algorithms, whose combination gives asymptotically optimal answer when $\min\{n, m\} = O(1)$ or $n = \Theta(m)$. The main idea is to simulate the algorithm of the two-dimensional Shuffle-Exchange problem with cyclic operation and further accelerate the process with *Periodic Switching Network*.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases Cayley graph, Permutation rearrangement, Computational group theory, Switching network

Digital Object Identifier 10.4230/LIPIcs...

© Anonymous;
licensed under Creative Commons License CC-BY

Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

1.1 Overview

Cayley graph is a widely studied object. Given a group G with its generator set S, we can construct its Cayley graph $\Gamma(G, S)$, a directed graph whose vertices are all elements in G, and there is an arc from x to y if and only if there exists an $s \in S$ such that $y = sx$ [8]. One of the key problems is that given G and S, how long the diameter of $\Gamma(G, S)$ will be [2, 15].

An application of the Cayley graph diameter problem in switching network theory is Shuffle-Exchange Conjecture [5, 19], which also plays a important role in parallel processing, sorting networks, etc. To begin with, its two-dimensional version can be stated as follows:

▶ **Problem** (Two-dimensional Shuffle-Exchange problem (2dSE)). *Given permutation $\sigma \in S_{[n]\times[m]}$. Decompose σ into $\sigma_1 \circ \cdots \circ \sigma_t$, where σ_k is constructed by one of the following ways:*

- *Choose $g_1, g_2, \ldots, g_n \in S_{[m]}$ and $\sigma_k := \prod_{i=1}^{n} g_i^{\text{Row-}i}$;*
- *Choose $h_1, h_2, \ldots, h_m \in S_{[n]}$ and $\sigma_k := \prod_{j=1}^{m} h_j^{\text{Col-}j}$.*

where $g_i^{\text{Row-}i}$ and $h_j^{\text{Col-}j}$ mean performing g_i and h_j in row-i and column-j respectively.

According to [19], 2dSE can be solved with $t \leq 3$. This is also discussed in Section 5.2. The general case of Shuffle-Exchange Conjecture is rather involved; and since it is not typically needed here, we refer interested readers to [5, 19] for detail.

Inspired by the parallel (different) group action described in 2dSE, we raise the following problem, which can be seen as a generalization of the two-dimensional Shuffle-Exchange problem:

▶ **Problem** (Generalized two-dimensional Shuffle-Exchange problem (G2dSE)). *Given matrices $M_1, M_2 \in [nm]^{[n]\times[m]}$ and two groups $G \leq S_{[m]}, H \leq S_{[n]}$, transform M_1 to M_2 by the following operations:*

- *Choose $g_1, g_2, \ldots, g_n \in G$ and perform $\prod_{i=1}^{n} g_i^{\text{Row-}i}$;*
- *Choose $h_1, h_2, \ldots, h_m \in H$ and perform $\prod_{j=1}^{m} h_j^{\text{Col-}j}$.*

The aim is spontaneously to determine whether we can do this transformation; and if the answer is Yes, determine at least how many steps are required.

▶ Remark. Note that for any group $G \leq S_{[n]}$, although the size of G may be large, [11] shows that it always has a small generator set which has size up to $O(\log_2 |G|) = O(\log_2(n!)) = O(\text{poly}(n))$. Therefore, the cost to input G, H is still polynomial in n, m.

Since the group G and H are arbitrary, G2dSE may give a framework to many problems related to Cayley graph with specific groups. Under this general framework, we provide several results on both computational complexity and algorithm. In particular, we show that the decision version of this problem is harder than Graph Isomorphism but unlikely to to be **NP**-hard unless $\mathbf{PH} = \mathbf{\Pi}_2^p$. As a contrast, determining whether M_1 can be transformed to M_2 within k steps is **NP**-complete. Second, we give an algorithm to show that the Cayley graph of G2dSE has diameter of polynomial length. It is worth mentioning that this result is rather non-trivial, since [17] gives an example of a Cayley graph with diameter of exponential length. Finally, in a special case where G and H are cyclic groups and all elements in the matrix are distinct, we also give an efficient algorithm whose answer is nearly optimal. The main idea is to simulate the algorithm for 2dSE, and use periodic switching network to accelerate the process.

The organization of this paper is as follows. In Section 2, some essential notations are given. In Section 3, we discuss the difficulty in determining the reachability and distance of

G2dSE. In Section 4, we study the length of the diameter of the Cayley graph in G2dSE. In Section 5, we focus on a special case where two groups are cyclic groups and all elements in the matrix are distinct. In Section 6, we summarize this paper and point out further direction.

1.2 Main Results

First, we show the main results on computational complexity:

▶ **Theorem 1.** *In G2dSE, given integer $k \geq 2$, it is* **NP***-complete to determine if M_1 can be transformed to M_2 within k steps.*

The membership in **NP** of Theorem 1 is naturally implied by the following theorem.

▶ **Theorem 2.** *In G2dSE, if M_2 is reachable from M_1, then M_1 can be transformed to M_2 within $O(\mathrm{poly}(n, m))$ steps.*

In addition, we also consider the complexity of examining the reachability. As a corollary of [11], when M_1 (or M_2) contains exactly nm distinct elements, this task is in **P**. We, however, investigate more carefully into the structure of this problem and obtain the following result. (We will define *fix-point free* formally later.)

▶ **Theorem 3.** *In G2dSE, for general groups G, H, it is* **GI***-hard, but unlikely to be* **NP***-hard as it belongs to* **coAM***, to determine if M_1 can be transformed to M_2, where* **GI** *means* Graph Isomorphism*. However, it is polynomial-time solvable when G and H are fix-point free.*

Last but not least, for a specific case where rows and columns can only be shifted, we provide two efficient algorithms for it.

▶ **Theorem 4.** *In G2dSE, when both M_1 and M_2 contain distinct elements, nm is even and $G = C_m, H = C_n$, then (without loss of generality) assuming $n \leq m$, M_1 can be transformed to M_2 within $O(\min\{m, n \log m\})$ steps, and there is a polynomial-time algorithm to output a transformation sequence. On the other hand, there exist cases which require $\Omega(n \log_n m)$ steps.*

Note that it is reasonable to restrict nm to be even, since if otherwise, we will show in Lemma 24 that only even permutations can be achieved. Meanwhile, with algorithm in Theorem 4, we can also provide valid transformation sequence when M_1, M_2 are arbitrary in $[nm]^{[n]\times[m]}$ by assigning different labels to identical elements.

1.3 Related Works

Determining whether a permutation can be generated by a given set is widely studied. Furst, Hopcroft and Luks [11] gave an algorithm to show that this problem is in **P**. However, finding a minimum generating sequence is **PSPACE**-complete, which was proven by Jerrum [17]. More achievements on this topic can be seen in [16].

For general version of Cayley graph problem, there is a long history. Cayley graph was first introduced by [8], which represents a group by a directed graph. The motivation is to study the relationship between a group and its generator sets. For detailed discussion, [20] is an excellent survey. A natural problem on this topic is that given G and one of its generator sets S, how long the diameter of the Cayley graph is. This is also an important problem in computational group theory [2, 15].

For special cases of the Cayley graph diameter problem, many intriguing results are obtained on the aspect of both algorithm and complexity, especially in puzzles and games. For example, for the (n^2-1)-PUZZLE, Ratner and Warmuth [22] showed that this problem is **NP**-complete and offered an $O(n^3)$ algorithm. Recently, Demaine, Eisenstat and Rudoy [9] gave a proof on the **NP**-completeness of RUBIK'S CUBE PUZZLE and an $O\left(\frac{n^2}{\log n}\right)$ algorithm for any $n \times n \times n$ Rubik's cube. For more examples in puzzles and games, see [14].

Another special case of Cayley graph diameter problem is Shuffle Exchange Conjecture [5]. Given a permutation, the aim is to rearrange it by a switching network with specific ordered layers. People have made a lot of efforts on this topic. [7] gave a proof of this conjecture, but later [3] showed the proof is incomplete.

For permutation rearrangement by switching networks, recently [21] gave an algorithm to a problem on drawing permutation networks. Later on, [10] showed that this problem is **NP**-hard.

2 Preliminaries

Specific Notations

In this paper, all log are base 2. For simplicity, we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$ and define $\oplus_t$ to represent modulo t, where $a \oplus_t b = (a + b - 1 \mod t) + 1$. Then we say $T \subseteq [n]$ is *non-adjacent* if and only if for any $i \in T$, $i \oplus_n 1 \notin T$.

For a Boolean string $v \in \{0,1\}^s$, its complement $\overline{v}$ is a Boolean string with same length satisfying $v_i + \overline{v}_i = 1$ for all $i \in [s]$.

FLATTEN is an operator which transforms an $n \times m$ matrix into a vector of length nm in row-major order. With a little abuse of notation, the result vector can also be seen as a 1 by nm matrix.

Matrix Representation

For convenience, describe some notations related with matrix $M \in [nm]^{[n]\times[m]}$. Define $M[\![i,j]\!]$ as the element in position-(i,j) in M. Define $M[\![S]\!]$ as the partial matrix ranged by S where $S \subseteq [n] \times [m]$. Formally, $M[\![S]\!] : S \to [nm]$ is define as $(M[\![S]\!])[\![i,j]\!] := M[\![i,j]\!]$ for all $(i,j) \in S$. Define $M[\![i,*]\!] := M[\![\{i\} \times [m]]\!]$ and $M[\![*,j]\!] := M[\![[n] \times \{j\}]\!]$.

Permutations and Group Theory

The permutation group is a crucial term in our work. We list some definitions used in this paper about permutations. In the following context, we may use abbreviation $G \leq H$ to represent G is a subgroup of H. For a permutation π, we say it is even if it can be decomposed into even number of swaps, and odd otherwise. We also define an indicator function $\mathrm{sgn}(\pi)$, $\mathrm{sgn}(\pi) = 1$ if π is odd permutation; otherwise $\mathrm{sgn}(\pi) = 0$.

Define *permutation group* S_T as the set of all permutations over finite set T, i.e.,

$$S_T := \left\{\sigma \in T^T \mid \sigma \text{ is a bijection}\right\}.$$

Define *alternating group* A_T of S_T as the subgroup containing all even permutation in S_T. *Cyclic group* C_n is a subgroup of $S_{[n]}$, which is generated by a permutation σ satisfying $\sigma(i) := i \oplus_n 1$. Furthermore, given $s, t, n \in \mathbb{N}$, define $(s \rightsquigarrow t)_n$ as the unique permutation $\sigma \in C_n$ satisfying $\sigma(s) = t$. When n is specified, it may also be denoted as $(s \rightsquigarrow t)$ or $s \rightsquigarrow t$. Elements in C_n can be visualized as "shift" as well. For detail, see Section 5.

When T is a matrix, define natural group action $\varphi : S_{[n]\times[m]} \times [nm]^{[n]\times[m]} \to [nm]^{[n]\times[m]}$ as

$$\varphi(\sigma, M)[\![i,j]\!] = M[\![\sigma(i,j)]\!]$$

for all $i \in [n]$ and $j \in [m]$. For simplicity, $\varphi(\sigma, M)$ is abbreviated as $\sigma(M)$.

Given permutation $\sigma \in S_{[m]}$ and $i \in [n]$, define $\sigma^{\text{Row-}i} \in S_{[n]\times[m]}$ as a permutation rearranging the position in row-i with σ, i.e.,

$$\sigma^{\text{Row-}i}(j,k) := \begin{cases} (j,\sigma(k)) & \text{if } j = i, \\ (j,k) & \text{otherwise.} \end{cases}$$

In the same way, for $\sigma \in S_{[n]}$ define $\sigma^{\text{Col-}i} \in S_{[n]\times[m]}$. For $p_1, p_2 \in [n] \times [m], p_1 \neq p_2$, define $\textsc{Swap}(p_1, p_2)$ as the swap $(p_1\ p_2) \in S_{[n]\times[m]}$. Given group $G \subseteq S_{[n]}$, we say $i \in [n]$ is a *fix-point* if $g(i) = i, \forall g \in G$. If there is no fix-point in G, then the group G is *fix-point free*.

Computational Complexity Classes

In computational complexity theory, there are several well-known complexity classes appearing in this paper: $\mathbf{P}$, $\mathbf{NP}$, $\mathbf{AM}$, $\mathbf{PH}$, $\mathbf{\Pi}_2^p$ and $\mathbf{GI}$-hard (the set of problems at least as hard as Graph Isomorphism). For more detail, see [1] and [18].

3 Computational Complexity Perspective

In this section, we investigate the reachability from M_1 to M_2 in G2dSE and the minimum steps required for such transformation from the perspective of computational complexity. We show that it may be hard to determine the reachability in general. However, it is somewhat "not so hard", as it is unlikely to be $\mathbf{NP}$-hard. Due to the limit of pages, all proofs of lemmas and theorems in this section are in Appendix A.

First, the following two lemmas induces the first part of Theorem 3:

▶ **Lemma 5.** *In G2dSE, it is* $\mathbf{GI}$*-hard to determine whether* M_1 *can be transformed to* M_2*.*

▶ **Lemma 6.** *In G2dSE, it is not* $\mathbf{NP}$*-hard to determine whether* M_1 *can be transformed to* M_2 *unless* $\mathbf{PH} = \mathbf{\Pi}_2^p$*.*

When we turn to the steps required for such transformation, the task becomes much more unwieldy. Consider a special case where both M_1, M_2 are $\{0,1\}$-value matrices, G, H are cyclic groups, and the task is to check whether M_1 can be transformed to M_2 within 2 steps. Under such restriction, this problem, however, is still as hard as the 3-SAT problem. Note that the size of the matrices is nm and C_n, C_m can be represented within $O(n \log n + m \log m)$ bits, thus the input size is $\Theta(\text{poly}(n,m))$.

▶ **Theorem 7.** *Given two matrices* $M_1, M_2 \in \{0,1\}^{[n]\times[m]}$*, it is* $\mathbf{NP}$*-hard to determine whether* M_1 *can be transformed to* M_2 *within 2 steps by following operations:*

- *Choose* $g_1, g_2, \ldots, g_n \in C_m$ *and perform* $\prod_{i=1}^n g_i^{\text{Row-}i}$*;*
- *Choose* $h_1, h_2, \ldots, h_m \in C_n$ *and perform* $\prod_{j=1}^m h_j^{\text{Col-}j}$*.*

▶ **Corollary** (Theorem 1 $\mathbf{NP}$-hard part)**.** *In G2dSE, given integer* $k \geq 2$*, it is* $\mathbf{NP}$*-hard to determine if* M_1 *can be transformed to* M_2 *within* k *steps.*

4 Algorithmic Perspective

In this section, we study the reachability and minimum required steps of G2dSE from algorithmic perspective. The hardness result for reachability in the previous section strongly relies on the fix-points in G or H (see the proof in Appendix A), actually it is polynomial-time solvable when G, H are fix-point free. Here we present an algorithm for this case, which surprisingly incorporates the idea of linear representation from linear algebra.

Before introducing the main algorithm, several auxiliary functions is needed. First, TransPart divides a row (or column) into several parts, on each of which the given permutation group is transitive. Second, RowBasis and ColBasis compute the linear basis in $\mathbb{F}_2^{[c]\times[r]}$ from generators of the permutation group, which also takes the partition, i.e., $\{T_j\}$ in Algorithm 2, of row (or column) from Algorithm 1 as input. Third, LinearDependent checks the linear dependency between a vector (u) with wildcards $*$ and a set of ordinary basis (V). Note that a standard Gaussian elimination process is used when all the wildcards are removed.

Algorithm 1 Group Dividing and Linear Dependency Checking

function TransPart$(s, \{\sigma_i\}_{i\in[t]})$ $\qquad\triangleright\ \sigma_i \in S_{[s]}$
    $S \leftarrow \{\{1\},\{2\},\ldots,\{s\}\}$
    repeat
        for all $i \in [s], j \in [t]$ **do**
            let $T, T' \in S$ with $i \in T, \sigma_j(i) \in T'$
            $S \leftarrow \{T \cup T'\} \cup (S \backslash \{T, T'\})$
    until S gets no update
    $k \leftarrow |S|$ and list elements in S as $S_1,\ldots,S_k$
    return $k, S_1, \ldots, S_k$

function LinearDependent(c, r, u, V) $\qquad\triangleright\ u \in \{0,1,*\}^{cr}, V \subseteq \{0,1\}^{cr}$
    $S \leftarrow \{i \in [cr] | u_i \neq *\}, s \leftarrow |S|$
    $\tilde{u} \leftarrow u|_S, \tilde{V} \leftarrow \{v|_S | v \in V\}$ $\qquad\triangleright\ \tilde{u} \in \{0,1\}^s, \tilde{V} \subseteq \{0,1\}^s$
    if $\tilde{u}$ lies in the linear subspace in $\mathbb{F}_2^s$ spanned by vectors in $\tilde{V}$ **then**
$\qquad\triangleright$ standard Gaussian elimination algorithm
        return Reachable
    else
        return Not Reachable

Algorithm 2 Row and Column Linear Basis

function RowBasis$(c, \{\sigma_i\}_{i\in[t]}, \{T_j\}_{j\in[r]})$
    $S \leftarrow \emptyset$
    for all $i \in [t], j \in [c]$ **do**
        $\hat{v} \leftarrow 0^{[c]\times[r]}$
        for all $k \in [r]$ **do**
            $\hat{v}_{j,k} \leftarrow \mathrm{sgn}(\sigma_i|_{T_k})$
        $S \leftarrow S \cup \{\mathrm{FLATTEN}(\hat{v})\}$
    return S

function ColBasis$(r, \{\sigma_i\}_{i\in[t]}, \{T_j\}_{j\in[c]})$
    $S \leftarrow \emptyset$
    for all $i \in [t], j \in [r]$ **do**
        $\hat{v} \leftarrow 0^{[c]\times[r]}$
        for all $k \in [c]$ **do**
            $\hat{v}_{k,j} \leftarrow \mathrm{sgn}(\sigma_i|_{T_k})$
        $S \leftarrow S \cup \{\mathrm{FLATTEN}(\hat{v})\}$
    return S

Finally, we are able to present the main algorithm in Algorithm 3. The proof of its correctness is deferred to Appendix B.

As a corollary, last piece of Theorem 3 is completed.

▶ **Corollary** (Theorem 3 polynomial-time part)**.** *In G2dSE, it is polynomial-time solvable when G and H are fix-point free.*

Actually, from Algorithm 3 we could say something more than reachability. It also gives a way to instantiate a valid transformation of $\mathrm{poly}(n, m)$ steps when G, H are fix-point free.

Algorithm 3 Check reachability (fix-point free)

Input: $G = \langle \hat{g}_i \rangle_{i\in[\ell_1]}, H = \langle \hat{h}_j \rangle_{j\in[\ell_2]}$ and $M_1, M_2 \in [nm]^{[n]\times[m]}$
Assert: G, H are fix-point free
$(r, T_1^R, \dots, T_r^R) \leftarrow \textsc{TransPart}(m, \{\hat{g}_i\}_{i\in[\ell_1]})$
$(c, T_1^C, \dots, T_c^C) \leftarrow \textsc{TransPart}(n, \{\hat{h}_i\}_{i\in[\ell_2]})$
$V \leftarrow \textsc{RowBasis}(c, \{\hat{g}_i\}_{i\in[\ell_1]}, \{T_j^R\}_{j\in[r]}) \cup \textsc{ColBasis}(r, \{\hat{h}_i\}_{i\in[\ell_2]}, \{T_j^C\}_{j\in[c]})$
$u \leftarrow *^{[c]\times[r]}$
for all $i \in [c], j \in [r]$ **do**
  if $M_1[\![T_i^C \times T_j^R]\!], M_2[\![T_i^C \times T_j^R]\!]$ has different elements **then**
    return Not Reachable
  if $M_1[\![T_i^C \times T_j^R]\!]$ contains distinct elements **then**
    $\pi \leftarrow$ the unique permutation that $\pi\left(M_1[\![T_i^C \times T_j^R]\!]\right) = M_2[\![T_i^C \times T_j^R]\!]$
    $u_{i,j} \leftarrow \mathrm{sgn}(\pi)$
return $\textsc{LinearDependent}(c, r, \mathrm{FLATTEN}(u), V)$

The rigorous proof is in Lemma 24 in Appendix B. With this, we can also finish the proof of Theorem 2.

When M_2 is reachable from M_1, for any fix-point t under G, there must exist $\pi \in H$ satisfying $\pi^{\text{Col-}t}(M_1)[\![*, t]\!] = M_2[\![*, t]\!]$. In other words, if $\sigma(M_1) = M_2$ and σ is achievable, $\sigma|_{[n]\times\{t\}}$ contributes at most one step to the overall step cost. Similar analysis fits for fix-points in H as well. Thus Theorem 2 holds immediately.

▶ **Corollary** (Theorem 2 restated). *In G2dSE, if M_2 is reachable from M_1, then M_1 can be transformed to M_2 within $O(\mathrm{poly}(n, m))$ steps.*

5 Algorithms for CnxCm

Here, we investigate CnxCm, a special case of G2dSE where $G = C_m, H = C_n$ and both M_1, M_2 contain nm distinct elements. We focus on the minimum required steps for the transformation.

For convenience, we say an algorithm for CnxCm is $T(n, m)$-operation if for any valid M_1, M_2, the output sequence (to achieve the transformation) is of length at most $T(n, m)$. Restate our problem in permutation version.

▶ **Problem** ($C_n \times C_m$ problem (CnxCm)). *Assume nm is even. Given permutation $\sigma \in S_{[n]\times[m]}$, achieve σ with as few steps as possible by the following operations:*

- *Choose $g_1, g_2, \dots, g_m \in C_n$ and perform $\prod_{i=1}^m g_i^{\mathit{Row}\text{-}i}$;*
- *Choose $h_1, h_2, \dots, h_n \in C_m$ and perform $\prod_{j=1}^n h_j^{\mathit{Col}\text{-}j}$.*

Note that C_n contains only even permutations for odd n. Using Lemma 24, it is easy to find that when nm is odd, σ can be achieved if and only if σ is even. Whereas when nm is even, any permutation can be achieved. Thus we only design algorithms for the latter one.

This section is structured as follows. First, we give this problem a lower bound of $\Omega(n \log_n m)$, assuming $n \le m$, as part of Theorem 4. The proof is deferred to Appendix C.1.

▶ **Lemma 8** (Theorem 4 lower bound part). *In CnxCm, assuming $n \le m$, there exists a permutation σ requiring $\Omega(n \log_n m)$ steps to achieve.*

Second, as a warm-up, we show how to swap elements in different rows (or columns) parallel under some restrictions. However, if implemented naively, the algorithm may perform much worse than the lower bound. To improve this trivial algorithm, we come to the case where $G = S_{[m]}, H = S_{[n]}$, i.e., 2dSE. The main idea to efficiently solve CnxCm is to simulate the algorithm of 2dSE with shifts. In addition, we accelerate the process with periodic switching network. Finally, a polynomial-time algorithm giving $O(\min\{m, n\log m\})$ operations (assuming $n \le m$), which is asymptotically optimal when $n = \Theta(1)$ or $n = \Theta(m)$, is obtained.

5.1 Warm-up I: How to Swap Elements in Parallel

Parallel shifts in rows (or columns) can be represented as a vector. For $w \in \mathbb{Z}_m^n$ and $v \in \mathbb{Z}_n^m$, their corresponding shifts $\textsc{CycRows} : \mathbb{Z}_m^n \to S_{[n]\times[m]}$ and $\textsc{CycColumns} : \mathbb{Z}_n^m \to S_{[n]\times[m]}$ are defined as:

$$\textsc{CycRows}(w) := \prod_{i=1}^{n} \left(1 \rightsquigarrow (w_i \oplus_m 1)\right)^{\text{Row-}i} \text{ and } \textsc{CycColumns}(v) := \prod_{i=1}^{m} \left(1 \rightsquigarrow (v_i \oplus_n 1)\right)^{\text{Col-}i}$$

The following algorithm swaps pairs of elements in parallel, with some by-product (see the example below).

Algorithm 4 parallel swapping pairs in different rows

procedure $\textsc{SwapRows}(v,\, w)$ ▷ $v \in \{0, \pm 1\}^m$, $w \in \mathbb{N}^n$
  $\textsc{CycColumns}(v)$
  $\textsc{CycRows}(-w)$
  $\textsc{CycColumns}(-v)$
  $\textsc{CycRows}(w)$
  $\textsc{CycColumns}(v)$

The input of Algorithm 4 should satisfy the following restrictions:

- For any distinct $i, j \in [m]$, and $k \in [n]$, if $v_i \neq 0, v_j \neq 0,\ w_k \neq 0$, then $i \oplus_m w_k \neq j$;
- For any distinct $i, j \in [n]$, and $k \in [m]$, if $w_i \neq 0, w_j \neq 0,\ v_k \neq 0$, then $i \oplus_n v_k \neq j$.

$$\begin{array}{|c|c|c|c|c|}\hline 1&2&3&4&5\\\hline a&&&d&\\\hline b&&&e&\\\hline 6&7&8&9&0\\\hline c&&&f&\\\hline\end{array} \Rightarrow \begin{array}{|c|c|c|c|c|}\hline c&2&3&f&5\\\hline 1&&&4&\\\hline a&&&d&\\\hline b&7&8&e&0\\\hline 6&&&9&\\\hline\end{array} \Rightarrow \begin{array}{|c|c|c|c|c|}\hline 2&3&f&5&c\\\hline 1&&&4&\\\hline a&&&d&\\\hline 7&8&e&0&b\\\hline 6&&&9&\\\hline\end{array} \Rightarrow \begin{array}{|c|c|c|c|c|}\hline 1&3&f&4&c\\\hline a&&&d&\\\hline 7&&&0&\\\hline 6&8&e&9&b\\\hline 2&&&5&\\\hline\end{array} \Rightarrow \begin{array}{|c|c|c|c|c|}\hline c&1&3&f&4\\\hline a&&&d&\\\hline 7&&&0&\\\hline b&6&8&e&9\\\hline 2&&&5&\\\hline\end{array} \Rightarrow \begin{array}{|c|c|c|c|c|}\hline 2&1&3&5&4\\\hline c&&&f&\\\hline a&&&d&\\\hline 7&6&8&0&9\\\hline b&&&e&\\\hline\end{array}$$

Figure 1 Calling Algorithm 4 with $v = (1, 0, 0, 1, 0), w = (1, 0, 0, 1, 0)$. The 4 pairs $(1, 2)$, $(4, 5)$, $(6, 7)$ and $(9, 0)$ are swapped parallel, while some elements represented by letters are moved .

See Figure 1 as an example. It swaps 4 pairs in row-1 and row-4. As a consequence, column-1 and column-4 (excluding the first and forth elements) are shifted. Formally, Algorithm 4 performs the following permutation:

$$\textsc{SwapRows}(v, w) = \pi \cdot \prod_{i,j} \textsc{Swap}\left(j, j \oplus_m (w_i \cdot v_j^2)\right)^{\text{Row-}i},$$

where the restriction of inputs ensures it is well-defined and π will not permute elements on row-i where $w_i \neq 0$. It is easy to see in the example, calling Algorithm 4 for another two times helps to reverse the unwanted column effect.

5.2 Warm up II: Algorithm for 2dSE

Before introducing our main algorithms, we revisit the case where $G = S_{[m]}$, $H = S_{[n]}$, and both M_1, M_2 contain nm distinct elements. In fact, this is exactly the 2dSE.

▶ **Theorem 9.** *Given a permutation $\sigma \in S_{[n]\times[m]}$, σ can be achieved within 3 steps by the following operations:*

- *Choose $g_1, g_2, \ldots, g_m \in S_{[n]}$ and perform $\prod_{i=1}^{m} g_i^{Row\text{-}i}$;*
- *Choose $h_1, h_2, \ldots, h_n \in S_{[m]}$ and perform $\prod_{j=1}^{n} h_j^{Col\text{-}j}$.*

Algorithm 5 solves 2dSE and verifies Theorem 9. Its correctness comes from Lemma 10, which is a direct application of Hall's marriage theorem [13].

▶ **Lemma 10.** *k-regular bipartite graph can be decomposed into k perfect matchings.*

Algorithm 5 Decomposition for 2dSE

Input: $M_1, M_2 \in [nm]^{[n]\times[m]}$
Assert: Elements in both M_1 and M_2 are a permutation of $[nm]$
$E \leftarrow \emptyset$
for all $i \in [n], j \in [m]$ **do**
  $(k, \ell) \leftarrow \sigma(i, j)$
  add (i, k) to E and label it with (i, j)
$G \leftarrow ([n], [n], E)$ ▷ G is a bipartite graph
let $M_1, \ldots, M_m$ be m disjoint perfect matchings of G ▷ Followed from Lemma 10
for all $t \in [m]$ and label $(i, j) \in M_t$ **do**
  $(k, \ell) \leftarrow \sigma(i, j)$
  $\sigma_1(i, j) \leftarrow (i, t)$ ▷ Row operation
  $\sigma_2(i, t) \leftarrow (k, t)$ ▷ Column operation
  $\sigma_3(k, t) \leftarrow (k, \ell)$ ▷ Row operation
return $\sigma_1, \sigma_2, \sigma_3$

It can be verified $\sigma_3(\sigma_2(\sigma_1(i, j))) = \sigma(i, j)$ holds for any $i \in [n], j \in [m]$ as the desired transformation from M_1 to M_2.

5.3 $O(n + m)$-operation Algorithm

In this subsection, we design an $O(n+m)$-operation algorithm for CnxCm, which decomposes permutations in Algorithm 5 into swaps in each row and performs them parallel. In this subsection, we assume m is even. All proofs of lemmas are in Appendix C.2.

First, we show the following lemma, as an essential ingredient in this algorithm.

▶ **Lemma 11.** *In CnxCm, given a non-adjacent set $T \subseteq [n]$, there exists a permutation $\pi \in S_{[n]\times[m]}$ keeping the positions out of $([n]\backslash T) \times \{1\}$ invariant, such that for any $i \in T$, $s_i, t_i \in [m], s_i \neq t_i$,*

$$\tau := \pi \cdot \prod_{i\in T} \textsc{Swap}(s_i, t_i)^{Row\text{-}i} \text{ and } \tau^{-1} := \pi^{-1} \cdot \prod_{i\in T} \textsc{Swap}(s_i, t_i)^{Row\text{-}i}$$

can be achieved within $O(1)$ steps.

Next, we define a routine DECOMPOSE which takes as input an integer m and $\sigma \in A_{[m]}$. DECOMPOSE(m, σ) outputs $2m$ pairs $((s^{(1)}, t^{(1)}), \ldots, (s^{(2m)}, t^{(2m)}))$ such that

$$\sigma = \text{SWAP}(s^{(1)}, t^{(1)}) \circ \cdots \circ \text{SWAP}(s^{(2m)}, t^{(2m)}),$$

and $s^{(i)} < t^{(i)}$ for all $i \in [2m]$. Its correctness is guaranteed by the following fact.

▶ **Fact 12.** *Any $\sigma \in A_{[m]}$ can be decomposed into $2m$ swaps.*

Note that DECOMPOSE can be implemented in polynomial-time. In Lemma 13, we design a routine EPERMROWS, which takes a non-adjacent set T and $\sigma \in A_{[m]}$, then performs even permutations parallel in non-adjacent rows.

▶ **Lemma 13.** *In CnxCm, given a non-adjacent set $T \subseteq [n]$, and $\sigma_i \in A_{[m]}, i \in T$, then $\prod_{i \in T} \sigma_i^{Row\text{-}i}$ can be achieved within $O(m)$ steps by Algorithm 6.*

Algorithm 6 Perform even permutations in a non-adjacent row set

procedure EPERMROWS(T, σ) ▷ $T \subseteq [n], \sigma : T \to A_{[m]}$
  for all $i \in [n]$ **do**
    if $i \in T$ **then**
      $\left((s_i^{(1)}, t_i^{(1)}), \cdots, (s_i^{(2m)}, t_i^{(2m)})\right) \leftarrow$ DECOMPOSE$(\sigma(i))$
    else
      $\left((s_i^{(1)}, t_i^{(1)}), \ldots, (s_i^{(2m)}, t_i^{(2m)})\right) \leftarrow ((1,1), \ldots, (1,1))$
  for all $j \in [2m]$ **do**
    CYCROWS$(1^n - s^{(j)})$
    SHIFTROW$((-1)^j \cdot e_1, t^{(j)} - s^{(j)})$
    CYCROWS$(s^{(j)} - 1^n)$

Furthermore, in Lemma 14 we can improve EPERMROWS to PERMROWS and change the restriction $\sigma \in A_{[m]}$ to m is even.

▶ **Lemma 14.** *In CnxCm when m is even, given a non-adjacent set $T \subseteq [n]$ and permutations $\sigma_i \in S_{[m]}, i \in T$, then $\prod_{i \in T} \sigma_i^{Row\text{-}i}$ can be achieved within $O(m)$ steps by Algorithm 7.*

Algorithm 7 Perform permutations in a non-adjacent row set

procedure PERMROWS(T, σ) ▷ $T \subseteq [n]$, $\sigma = \prod_{i \in T} \sigma_i^{\text{Row-}i}$
  $v \leftarrow 0^n, \sigma' \leftarrow \sigma$
  for all $i \in T$ such that σ_i' is odd **do**
    $\sigma_i' \leftarrow \sigma_i' \circ (1 \rightsquigarrow 2)^{\text{Row-}i}$
    $v_i \leftarrow 1$
  EPERMROWS(T, σ')
  CYCROWS$(-v)$

Note that doing the same thing for columns seems to be more difficult since n may be odd. Lemma 15 provides PERMCOLS to overcome this problem with a little extra cost.

▶ **Lemma 15.** *In CnxCm when m is even, given a non-adjacent set $T \subseteq [n]$ and permutations $\sigma_i \in S_{[n]}, i \in T$, then $\prod_{i \in T} \sigma_i^{Col\text{-}i}$ can be achieved within $O(t_1 + t_2)$ steps by Algorithm 8, if*

- *for all $g_1, g_2, \ldots, g_m \in A_{[n]}$, $\prod_{i=1}^{m} g_i^{Col\text{-}i}$ can be achieved within t_1 steps;*

- *for all $h \in S_{[m]}$ and $r \in [n]$, $h^{Row\text{-}r}$ can be achieved within t_2 steps.*

Algorithm 8 Perform permutations in a non-adjacent column set

procedure PermCols(T, σ) ▷ $T \subseteq [m], \sigma = \prod_{i \in T} \sigma_i^{\text{Col-}i}$
  $v \leftarrow 0^n, \sigma' \leftarrow \sigma$
  $T' \leftarrow \{i \in T \mid \sigma_i \text{ is odd}\}$
  for all $i \in T'$ **do**
    $\sigma'_i \leftarrow \sigma_i \circ \text{Swap}(1,2)^{\text{Col-}i}$
    $v_i \leftarrow 1$
  EPermCols(T, σ')
  SwapCols(e_1, v) ▷ Let $\pi \in S_{[n]\times[m]}$ be the permutation performed
  $\pi' \leftarrow \mathsf{id}$
  for all $i \in [m] \backslash T'$ **do**
    $(1, \pi'(i)) \leftarrow \pi^{-1}(1, i)$ ▷ $(1, i)$ is mapped to the first row by π.
  PermRow$(1, \pi')$

In Algorithm 8, EPermCols and SwapCols are column versions of EPermRows and SwapRows. PermRow takes $r \in [n], \pi' \in S_{[m]}$ and achieves $\pi'^{\text{Row-}r}$. Lemma 13 shows EPermCols requires $O(n)$ steps and Lemma 14 shows PermRow requires $O(m)$ steps. Thus, Algorithm 8 offers a solution within $O(n+m)$ steps.

In general, $[m]$ (or $[n]$) can be divided into at most 3 non-adjacent sets. By Lemma 14 and Lemma 15, the row (or column) permutations in 2dSE can be simulated within $O(n+m)$ steps. Combining Algorithm 5, an $O(n+m)$ algorithm is obtained.

5.4 $O(\min\{n \log m, m \log n\})$-operation Algorithm

Without loss of generality, we assume $n \le m$. In this subsection, we present a new algorithm, whose operation cost is $O(n \log m)$.

The core of the algorithm is a decomposition for even permutations. Given $\sigma \in A_{[m]}$, we claim σ can be decomposed as $\sigma_D \circ \cdots \circ \sigma_1$, where $D = O(\log m)$ and for all $i \in [D]$, σ_i can be further decomposed as

$$\text{Swap}(s_{i,k_i}, s_{i,k_i} + d_i) \circ \cdots \circ \text{Swap}(s_{i,2}, s_{i,2} + d_i) \circ \text{Swap}(s_{i,1}, s_{i,1} + d_i)$$

for some integer d_i, such that $s_{i,j}, s_{i,k}, s_{i,j} + d_i$ and $s_{i,k} + d_i$ are distinct for all $j, k \in [k_i]$. The innovative idea here is to use *periodic switching network*, which is introduced in Appendix C.3.

Based on Lemma 25, a decomposing routine PeriodicSwitchingNetwork can be constructed, which takes an integer m and a permutation $\sigma \in S_{[m]}$ as inputs. Then it outputs $(s_1, \ldots, s_D), (t_1, \ldots, t_D)$, depth D, and the period in each level indicated by a vector d, which can be implemented in polynomial-time.

Recover is an auxiliary procedure involved in Algorithm 9 which is described in Appendix C.3, as well as the whole proof of Lemma 16.

▶ **Lemma 16.** *In CnxCm, given $r \in [n]$ and $\sigma \in S_{[m]}$, $\sigma^{Row\text{-}r}$ can be achieved within $O(\log m)$ steps by Algorithm 9.*

By Lemma 16 and the proof of Lemma 14, we have $t_2 = O(\log m)$ for Lemma 15. On the other hand, an obvious $O(n \log m)$ algorithm is obtained by achieving row permutations one by one. Therefore, we have the following result.

Algorithm 9 Perform an even permutation in a row

```
procedure PermRow(r, σ)                                  ▷ r ∈ [n], σ ∈ S_[m]
    (s, t, D, d) ← PeriodicSwitchingNetwork(m, σ)
    w ← 0^m
    for all i ∈ [D] do
        v ← 0^n
        for all j ∈ [k_i] do
            v_{s_{i,j}} ← (−1)^{w_{s_{i,j}}}
            w_{s_{i,j}} ← w_{s_{i,j}} ⊕ 1
        SwapRows(d_i · e_r, v)
    Recover(r, w)
```

▶ **Corollary 17.** *In CnxCm when m is even, then*

- *given $\sigma_i \in S_{[m]}$ and $i \in [n]$, $\prod_{i=1}^{n} \sigma_i^{\text{Row-}i}$ can be achieved within $O(n \log m)$ steps;*
- *given $\sigma_j \in S_{[n]}$ and $j \in [m]$, $\prod_{j=1}^{m} \sigma_j^{\text{Col-}j}$ can be achieved within $O(n + \log m)$ steps.*

When m is odd and n is even, column permutations become simple by Lemma 14. And the algorithm for row permutations follows from Lemma 15, where $t_2 = O(\log n)$ now and by repeating Lemma 16 we have $t_1 = O(n \log m)$.

▶ **Corollary 18.** *In CnxCm when m is odd and n is even, then*

- *given $\sigma_i \in S_{[m]}$ and $i \in [n]$, $\prod_{i=1}^{n} \sigma_i^{\text{Row-}i}$ can be achieved within $O(n \log m)$ steps;*
- *given $\sigma_j \in S_{[n]}$ and $j \in [m]$, $\prod_{j=1}^{m} \sigma_j^{\text{Col-}j}$ can be achieved within $O(n)$ steps.*

Combining the algorithm for 2dSE, Corollary 17 and Corollary 18, we prove any permutation in $S_{[n]\times[m]}$ can be achieved within $O(n \log m)$ steps assuming $n \le m$, which finishes the proof of Theorem 4.

6 Conclusion and Open Problems

Through out this paper, we discuss the problem of transforming a matrix to another by parallel group actions on rows/columns. We prove that determining whether this transformation can be done within k steps is **NP**-complete. When we focus on the cyclic group action, we also provide efficient algorithms. The work all above may give a general framework on how to solve some combinatorial problems related to group action and Cayley graph.

Some problems remain unsolved.

- On the aspect of computational complexity, we conjecture that the reachablity version of G2dSE is actually **GI**-complete. That is, we believe that there exists a polynomial-time reduction from determining whether M_1 can be transformed into M_2 to Graph Isomorphism.
- On the aspect of algorithm, when G, H are both cyclic groups, we provide two efficient algorithms much better than brute force and direct simulation, but there is still a logarithmic gap between the lower and upper bound when $\omega(1) \le n \le o(m)$. We conjecture our algorithm is optimal and the lower bound can be improved by a more careful analysis.

References

1. Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.
2. László Babai and Ákos Seress. On the diameter of permutation groups. *European journal of combinatorics*, 13(4):231–243, 1992.
3. Xuewen Bao, Frank K Hwang, and Qiao Li. Rearrangeability of bit permutation networks. *Theoretical computer science*, 352(1-3):197–214, 2006.
4. Václad E Beneš. Optimal rearrangeable multistage connecting networks. *Bell system technical journal*, 43(4):1641–1656, 1964.
5. VE Beneš. Proving the rearrangeability of connecting networks by group calculations. *Bell System Technical Journal*, 54(2):421–434, 1975.
6. Ravi B Boppana, Johan Hastad, and Stathis Zachos. Does co-np have short interactive proofs? *Information Processing Letters*, 25(2):127–132, 1987.
7. Hasan Cam. Rearrangeability of (2n-1)-stage shuffle-exchange networks. *SIAM Journal on Computing*, 32(3):557–585, 2003.
8. Arthur Cayley. Desiderata and suggestions: No. 2. the theory of groups: graphical representation. *American Journal of Mathematics*, 1(2):174–176, 1878.
9. Erik D Demaine, Sarah Eisenstat, and Mikhail Rudoy. Solving the rubik's cube optimally is np-complete. In *35th Symposium on Theoretical Aspects of Computer Science*, 2018.
10. Oksana Firman, Philipp Kindermann, Alexander Ravsky, Alexander Wolff, and Johannes Zink. Computing optimal tangles faster. *arXiv preprint arXiv:1901.06548*, 2019.
11. Merrick Furst, John Hopcroft, and Eugene Luks. Polynomial-time algorithms for permutation groups. In *21st Annual Symposium on Foundations of Computer Science*, pages 36–41. IEEE, 1980.
12. Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 59–68. ACM, 1986.
13. Philip Hall. On representatives of subsets. In *Classic Papers in Combinatorics*, pages 58–62. Springer, 2009.
14. Robert A Hearn and Erik D Demaine. *Games, puzzles, and computation*. AK Peters/CRC Press, 2009.
15. Harald A Helfgott and Ákos Seress. On the diameter of permutation groups. *Annals of mathematics*, pages 611–658, 2014.
16. Derek F Holt, Bettina Eick, and Eamonn A O'Brien. *Handbook of computational group theory*. Chapman and Hall/CRC, 2005.
17. Mark Jerrum. The complexity of finding minimum-length generator sequences. In *International Colloquium on Automata, Languages, and Programming*, pages 270–280. Springer, 1984.
18. Johannes Kobler, Uwe Schöning, and Jacobo Torán. *The graph isomorphism problem: its structural complexity*. Springer Science & Business Media, 2012.
19. Vadim Lioubimov. Shuffle-exchange conjecture. URL: `http://www.openproblemgarden.org/op/shuffle_exchange_conjecture`.
20. Wilhelm Magnus, Abraham Karrass, and Donald Solitar. *Combinatorial group theory: Presentations of groups in terms of generators and relations*. Courier Corporation, 2004.
21. Maya Olszewski, Jeff Meder, Emmanuel Kieffer, Raphaël Bleuse, Martin Rosalie, Grégoire Danoy, and Pascal Bouvry. Visualizing the template of a chaotic attractor. In *International Symposium on Graph Drawing and Network Visualization*, pages 106–119. Springer, 2018.
22. Daniel Ratner and Manfred Warmuth. The (n^2-1)-puzzle and related relocation problems. *Journal of Symbolic Computation*, 10(2):111–137, 1990.

23 Charles C Sims. Computational methods in the study of permutation groups. In *Computational problems in abstract algebra*, pages 169–183. Elsevier, 1970.

Appendix

A Proofs of Lemmas and Theorems in Section 3

Proof of Lemma 5. In Graph Isomorphism, given two graphs W_1 and W_2 with same number of vertices and no multi-edge, the task is to determine if they are same in isomorphic sense.

Suppose the number of vertices in W_1 (and W_2) is k, then label all vertices in each graph from 1 to k in any order. Now in G2dSE, set $n = 1$, $m = k^2$, and $M_1, M_2 \in \{0,1\}^{[1]\times[m]}$ be the flattened adjacency matrix of W_1 and W_2. Then define the two groups $H = \mathsf{id}$ and $G = \langle \hat{g}_{u,v} \rangle_{\{u,v\}\in\binom{[k]}{2}}$, where $\hat{g}_{u,v}^{\text{Row-1}}(M_i)$ is the flattened adjacency matrix of W_i after swapping vertex u, v. Therefore M_1 can be transformed to M_2 under G and H if and only if W_1 and W_2 are isomorphic. ◀

Proof of Lemma 6. By the Theorem 19, it suffices to show the language

$$\{\langle M_1, M_2\rangle \mid M_1 \text{ cannot be transformed to } M_2 \text{ in } \mathsf{G2dSE}\}$$

is in **AM**.

▶ **Theorem 19** ([6]). *If any* **coNP**-*hard problem* $L \in \mathbf{AM}$, *then* $\mathbf{PH} = \mathbf{\Pi}_2^p$.

Define $I \le S_{[n]\times[m]}$ as $I = \left\langle G^{\text{Row-1}}, \dots, G^{\text{Row-}n}, H^{\text{Col-1}}, \dots, H^{\text{Col-}m} \right\rangle$ and

$$T = \{(M, \pi) \mid (M \cong_I M_1 \vee M \cong_I M_2) \wedge \pi \in aut_I(M)\},$$

where $a \cong_I b$ means there exists $g \in I$ such that $g(a) = b$. If $M_1 \cong_I M_2$, then $|T| = |I|$ holds. Otherwise, $|T| = 2|I|$ holds. Note that $|I|$ can be computed by Schreier-Sims algorithm [23]. Due to Goldwasser-Sipser set lower bound protocol [12], the language is in **AM**. ◀

Proof of Theorem 7. It suffices to show a polynomial-time reduction from 3-SAT to this problem. The main idea is that, given a 3-SAT instance ϕ with n clauses $\mathcal{C}_1, \dots, \mathcal{C}_n$ and m variables $x_1, \dots, x_m$, we will build a Boolean matrix $X_\phi = \{a_{i,j}\}$ so that ϕ is satisfiable if and only if X_ϕ can move all its 1's to the first row within 2 steps.

To describe the construction of X_ϕ, define the binary fingerprint v_i, $i \in [n+m]$ as

$$v_i = 01\,bin_{\lceil \log(n+m)\rceil}(i)\overline{bin_{\lceil \log(n+m)\rceil}(i)}10,$$

where $bin_t(i)$ is the binary representation of i with length $t \ge \log i$ (if the length of $(i)_2$ is smaller than $\lceil \log(n+m)\rceil$, we add some 0s in front of $(i)_2$). Also define the co-fingerprint of i as $\overline{v_i}$. Let ℓ be the length of the binary fingerprint, i.e., $\ell = 4 + 2\lceil \log(n+m)\rceil$. Note that in each fingerprint, there are exact $\ell/2$ ones, which indicates two fingerprints v, w satisfy $v \oplus w = 1^\ell$ if and only if v, w are complementary. Assign a fingerprint for each clause and each variable, i.e., let $v_{\mathcal{C}_i} := v_i$ and $v_{x_j} := v_{j+n}$ for all $i \in [n]$ and $j \in [m]$.

X_ϕ contains $1 + 2m + 10n$ rows. The first row is treated as a basic structure which is denoted by a_{bas}. Row-2 to row-$(2m+1)$ correspond to m variables in ϕ, i.e., rows a_{i+1} and a_{i+m+1} correspond to x_i, which are denoted by $a_{x_i}^{(1)}$ and $a_{x_i}^{(2)}$ respectively. Row-$(2m+2)$ to row-$(1+2m+10n)$ correspond to n clauses in ϕ. For $i \in [n]$,

- $a_{\mathcal{C}_i}^{(1,1)}, \dots, a_{\mathcal{C}_i}^{(1,4)}$ denote $a_{1+2m+10(i-1)+1}, \dots, a_{1+2m+10(i-1)+4}$;
- $a_{\mathcal{C}_i}^{(2,1)}, \dots, a_{\mathcal{C}_i}^{(2,4)}$ denote $a_{1+2m+10(i-1)+5}, \dots, a_{1+2m+10(i-1)+8}$;
- $a_{\mathcal{C}_i}^{(3)}$ denotes $a_{1+2m+10(i-1)+9}$;
- $a_{\mathcal{C}_i}^{(4)}$ denotes $a_{1+2m+10(i-1)+10}$.

Define $6m$ intervals on row-1 to row-$(2m+1)$ respectively in matrix X_ϕ,

$$r_{i,j,k} := \big[(36i + 12j + 6k - 53)\ell + 1, (36i + 12j + 6k - 48)\ell\big],$$

which corresponds to the j-th literal in $\mathcal{C}_i$, for all $i \in [m], j \in [3]$ and $k \in [2]$. See Figure 2 as an example. For convenience, define $r_{i,j,k}[pos]$ as the position of the pos-th element in the interval $r_{i,j,k}$ with respect to the whole row.

For any string s, let $s[r]$ be the sub-string $s[i,j]$ where r is the interval $[i,j]$; and $s]r[$ be the part of s outside interval r.

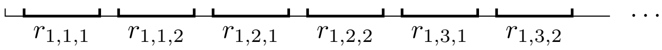

Figure 2 Intervals on row-1 to row-$(2m+1)$.

In a_{bas}, the bits outside intervals $\{r_{i,j,k}\}$ are set to 1 and the bits in the intervals are determined by

$$a_{\text{bas}}[r_{i,j,1}] := 0^{2\ell}\overline{v_{vbs(i,j)}}0^{\ell}v_{\mathcal{C}_i}$$
$$a_{\text{bas}}[r_{i,j,2}] := v_{\mathcal{C}_i}0^{\ell}\overline{v_{vbs(i,j)}}0^{2\ell}$$

for each $i \in [n], j \in [3]$ where $vbs : [n] \times [3] \to \{x_1, \ldots, x_m\}$ means the j-th literal in i-th clause is $vbs(i,j)$ (or $\neg vbs(i,j)$). Then, construct two rows for each variable x_i. Set

$$a^{(1)}_{x_i}]r_{j,k,1}[:= 0^*, \quad a^{(1)}_{x_i}[r_{j,k,1}] := \begin{cases} \overline{v_{\mathcal{C}_j}}1^{\ell}v_{x_i}v_{\mathcal{C}_j}\overline{v_{\mathcal{C}_j}}, & x_i \in \mathcal{C}_j \\ \overline{v_{\mathcal{C}_j}}v_j v_{x_i}1^{\ell}\overline{v_{\mathcal{C}_j}}, & \neg x_i \in \mathcal{C}_j \end{cases}$$
$$a^{(2)}_{x_i}]r_{j,k,2}[:= 0^*, \quad a^{(2)}_{x_i}[r_{j,k,2}] := \begin{cases} \overline{v_{\mathcal{C}_j}}v_j v_{x_i}1^{\ell}\overline{v_{\mathcal{C}_j}}, & x_i \in \mathcal{C}_j \\ \overline{v_{\mathcal{C}_j}}1^{\ell}v_{x_i}v_{\mathcal{C}_j}\overline{v_{\mathcal{C}_j}}, & \neg x_i \in \mathcal{C}_j \end{cases}$$

for all $j \in [m]$ and $k \in [3]$ such that $vbs(j,k) = x_i$. Furthermore, construct $10n$ "filling" rows. For each clause $\mathcal{C}_i$ in ϕ, set

$$a^{(1,j)}_{\mathcal{C}_i} := v_{\mathcal{C}_i}0^*, \quad a^{(2,j)}_{\mathcal{C}_i} := \overline{v_{\mathcal{C}_i}}0^*$$

for all $j \in [4]$ and

$$a^{(3)}_{\mathcal{C}_i} := \overline{v_{\mathcal{C}_i}}v_{\mathcal{C}_i}0^*, \quad a^{(4)}_{\mathcal{C}_i} := v_{\mathcal{C}_i}\overline{v_{\mathcal{C}_i}}0^*.$$

For example, given a 3-CNF $\phi = (x_1 \vee \cdots) \wedge (\neg x_1 \vee \cdots)$, for the variable x_1 we construct an instance in Figure 3, where a_{bas} is the first row, $a^{(1)}_{x_1}$ is the second row, and $a^{(2)}_{x_1}$ is the third row.

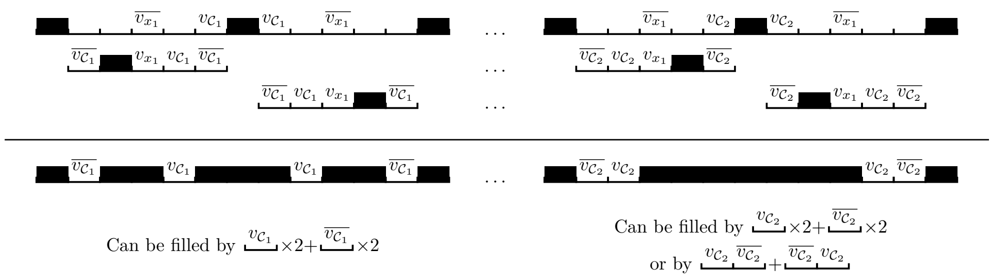

Figure 3 Construction for $\phi = (x_1 \vee \cdots) \wedge (\neg x_1 \vee \cdots)$

If $a_{\text{bas}}[r_{i,j,1}]$ and $a_{\text{bas}}[r_{i,j,2}]$ can be filled by $\underbrace{v_{\mathcal{C}_i}\,\overline{v_{\mathcal{C}_i}}}$ and $\underbrace{\overline{v_{\mathcal{C}_i}}\,v_{\mathcal{C}_i}}$, the literal of $x_{vbs_{i,j}}$ in $\mathcal{C}_i$ is satisfied. In the case of Figure 3, x_1 is set to FALSE and $\neg x_1$ in $\mathcal{C}_2$ is satisfied initially. Note that the second row can be shifted by 6ℓ while the third row is shifted by -6ℓ. If so, the left part in turn can be filled by $\underbrace{\overline{v_{\mathcal{C}_1}}\,v_{\mathcal{C}_1}}$ and $\underbrace{v_{\mathcal{C}_1}\,\overline{v_{\mathcal{C}_1}}}$, which means after setting x_1 to TRUE, the literal of x_1 in $\mathcal{C}_1$, i.e., x_1 itself, is satisfied. Thus, the value of variables can be switched by shifting the corresponding rows.

With the above preparation, it is enough to show that by our construction at least one literal can be satisfied in every clause if and only if the constructed instance can be recovered within 2 steps. Formally, we prove the equivalence between the satisfiability of a given 3-CNF ϕ and 2-step solvability of X_ϕ by Lemma 20 and Lemma 21.

▶ **Lemma 20.** *Given a 3-CNF Boolean formula ϕ and the corresponding matrix X_ϕ, if there exists a feasible assignment y for ϕ, X_ϕ can be recovered within 2 steps. Here the word 'recovered' means X_ϕ can be transformed to a matrix with same size in which all elements in the first row are 1 while other elements are 0.*

Proof. Based on an feasible assignment $x = y$, the first operation is a row permutation on X_ϕ as follows:

1. Keep the first row not shifted.
2. For any $y_i =$ FALSE, keep row $a^{(1)}_{x_i}$ and $a^{(2)}_{x_i}$ not shifted.
3. For any $y_i =$ TRUE, perform

$$r_{1,1,1}[1] \rightsquigarrow r_{1,1,2}[1] \text{ on } a^{(1)}_{x_i}, \quad r_{1,1,2}[1] \rightsquigarrow r_{1,1,1}[1] \text{ on } a^{(2)}_{x_i}.$$

4. For all clause $\mathcal{C}_i$, there exists a literal, assuming it is the c_i-th literal, satisfied under the assignment y. Perform

$$1 \rightsquigarrow r_{i,c_i,1}[1] \text{ on } a^{(4)}_{\mathcal{C}_i}, \quad 1 \rightsquigarrow r_{i,c_i,2}[4\ell+1] \text{ on } a^{(3)}_{\mathcal{C}_i}.$$

For other literal of $vbs(i,j)$ where $j \in [3]\backslash\{c_i\}$, let $t = j-(3-j)(j-1)(2-c_i) \in [4]$, then

- if the literal of $vbs(i,j)$ is satisfied, then perform

$$\begin{aligned} 1 \rightsquigarrow r_{i,j,1}[1] \text{ on } a^{(1,t)}_{\mathcal{C}_i}, &\quad 1 \rightsquigarrow r_{i,j,2}[4\ell+1] \text{ on } a^{(1,t+1)}_{\mathcal{C}_i}, \\ 1 \rightsquigarrow r_{i,j,1}[\ell+1] \text{ on } a^{(2,t)}_{\mathcal{C}_i}, &\quad 1 \rightsquigarrow r_{i,j,2}[3\ell+1] \text{ on } a^{(2,t+1)}_{\mathcal{C}_i}. \end{aligned}$$

- if the literal of $vbs(i,j)$ is not satisfied, then perform

$$\begin{aligned} 1 \rightsquigarrow r_{i,j,1}[1] \text{ on } a^{(1,t)}_{\mathcal{C}_i}, &\quad 1 \rightsquigarrow r_{i,j,2}[4\ell+1] \text{ on } a^{(1,t+1)}_{\mathcal{C}_i}, \\ 1 \rightsquigarrow r_{i,j,1}[3\ell+1] \text{ on } a^{(2,t)}_{\mathcal{C}_i}, &\quad 1 \rightsquigarrow r_{i,j,2}[\ell+1] \text{ on } a^{(2,t+1)}_{\mathcal{C}_i}. \end{aligned}$$

It can be verified there is a single 1 in each column of the shifted matrix, which means it can be recovered by another column permutation. ◀

▶ **Lemma 21.** *Given a 3-CNF Boolean formula ϕ and the corresponding matrix X_ϕ, if X_ϕ can be recovered within 2 steps, ϕ is satisfiable.*

Proof. Note that the target state is a fix point under row permutations. Thus, if a matrix can be recovered within 2 operations, the first operation must be a row permutation. Without loss of generality, assume the first row is not shifted in this row permutation. Then the feasible shift of the other rows will give rise to a valid assignment for ϕ.

For some variable x_k, suppose $a^{(1)}_{x_k}[r_{i,j,1}] \neq 0^{6\ell}$ and $a^{(2)}_{x_k}[r_{i,j,2}] \neq 0^{6\ell}$, which means x_k or $\neg x_k$ is the j-th literal in $\mathcal{C}_i$. Since the first and the last bit in the interval are 1 and there does not exist continuous ℓ zeros, $a^{(1)}_{x_k}$ and $a^{(1)}_{x_k}$ can only be shifted by distance of multiple of 6ℓ. Furthermore, to avoid the conflict of fingerprints, there are only two feasible shifts for rows corresponding to each variable x_k:

- Both $a^{(1)}_{x_k}$ and $a^{(2)}_{x_k}$ are kept invariant.
- Perform $0 \rightsquigarrow 6\ell$ shift on $a^{(1)}_{x_k}$, and $6\ell \rightsquigarrow 0$ shift on $a^{(2)}_{x_k}$.

▶ **Fact 22.** *For each clause $\mathcal{C}_i$, $a^{(3)}_{\mathcal{C}_i}$ and $a^{(4)}_{\mathcal{C}_i}$ must be shifted to be embedded into a_{bas}, so that there exists $j \in [3]$ satisfying*

$$a^{(1)}_{vbs(i,j)} \text{ is } \begin{cases} \text{shifted}, & \text{if } a_{vbs(i,j)}[r_{i,j,1}] = \overline{v_{\mathcal{C}_j}} 1^{\ell} v_{x_i} v_{\mathcal{C}_j} \overline{v_{\mathcal{C}_j}} \\ \text{kept invariant}, & \text{otherwise.} \end{cases}$$

Then, construct an assignment for ϕ based on the first operations when recovering X_ϕ. If $a^{(1)}_{x_i}$ and $a^{(2)}_{x_i}$ are shifted, let $x_i = \text{TRUE}$; otherwise, let $x_i = \text{FALSE}$. Note that $a_{vbs(i,j)}[r_{i,j,1}] = \overline{v_{\mathcal{C}_j}} 1^{\ell} v_{x_i} v_{\mathcal{C}_j} \overline{v_{\mathcal{C}_j}}$ if and only if $vbs(i,j) \in \mathcal{C}_i$. Thus, Fact 22 implies there exists at least one satisfied literal in each clause under the assignment and ϕ is satisfiable. ◀

Combining all these results, Theorem 7 holds. ◀

B Correctness of Algorithm 3

For convenience, we use some extra notations here.

Define the notation $\sqcup\mathcal{B} := \bigcup_{B\in\mathcal{B}} B$ if $\mathcal{B}$ is a set containing sets. Then we say permutation $\sigma \in S_{\sqcup\mathcal{B}}$ is a *$\mathcal{B}$-local permutation*, if $\sigma(i) \in B$ holds for any $B \in \mathcal{B}$ and any $i \in B$. Moreover, we say a permutation group $G \leq S_{\sqcup\mathcal{B}}$ is a *$\mathcal{B}$-local group*, if any element in G is $\mathcal{B}$-local permutation and for any $B \in \mathcal{B}$, any $i, j \in B$, there exists $\sigma \in G$ such that $\sigma(i) = j$.

For a $\mathcal{B}$-local permutation σ and a set $B \in \mathcal{B}$, define a mapping $\sigma|_B \in S_B$ that $\sigma|_B(i) := \sigma(i)$ for all $i \in B$, representing that σ is restricted on B. Then for a $\mathcal{B}$-local group $G = \langle \hat{g}_1, \hat{g}_2, \ldots, \hat{g}_\ell \rangle$ and a set $B \in \mathcal{B}$ satisfying that $\forall j \in [l]$ $\hat{g}_j|_B$ is a permutation over B, define $G|_B := \langle \hat{g}_1|_B, \hat{g}_2|_B, \ldots, \hat{g}_\ell|_B \rangle, \hat{g}_i \in S_{\sqcup\mathcal{B}}$. Note that if $G|_B$ is well-defined, then for all $g \in G$, $g|_B \in G|_B$ holds.

Now back to our problem. Given $G = \langle \hat{g}_1, \ldots, \hat{g}_{\ell_1} \rangle$ in G2dSE, we divide $[m]$ into several disjoint subsets $T^R_1, \ldots, T^R_r$ gathered by $\mathcal{B}^R$ satisfying that G is $\mathcal{B}^R$-local, which means $G|_{T^R_i}$ is transitive for any $i \in [r]$. Note that two elements $i, j \in [m]$ are contained in the same block if and only if there exists a sequence of permutations in the generator set $g_1, g_2, \ldots, g_k$ such that $g_k \circ \cdots \circ g_2 \circ g_1(i) = j$. Checking the latter condition can be reduced to a problem in **P** class that determines the reachability between two vertices in a graph. We design TransPart to compute the local structure of a specific group given with a set containing its generators. Furthermore, $[m]$ is divided by such structure.

Similarly, given the group $H = \langle \hat{h}_1, \hat{h}_2, \ldots, \hat{h}_{\ell_2} \rangle$ in G2dSE, we divide $[n]$ into disjoint subsets $T^C_1, \ldots, T^C_c$. Therefore, $[n] \times [m]$ is broken down into blocks $B_{i,j} := T^C_i \times T^R_j, i \in [c], j \in [r]$, which are gathered by $\mathcal{B}$. Note that $\mathcal{B}$ is polynomial-time computable. Also, given M_1 and M_2, it is easy to check whether $M_1[\![B_{i,j}]\!]$ and $M_2[\![B_{i,j}]\!]$ contain the same elements for all $i \in [c]$ and $j \in [r]$. Thus, we assume this necessary condition is satisfied.

In essence, the transformation is achieved by performing a $\mathcal{B}$-local permutation σ, the existence of which is guaranteed when promised each block in M_1 and M_2 contains the same elements, though σ may not be legal. First, we will show that σ can be partly achieved in $B_{i,j}$ if $|T^C_i|, |T^R_j| > 1$ and $\sigma|_{B_{i,j}}$ is even, within $O(\text{poly}(n, m))$ steps.

▶ **Lemma 23.** *In G2dSE, given $i,j \in [c]\times[r]$ where $|T_i^C|>1, |T_j^R|>1$, any $\sigma \in S_{B_{i,j}}$ over $B_{i,j}$ can be achieved within $O(\mathrm{poly}(n,m))$ steps.*

Proof. Consider arbitrary 3 distinct positions in $p_1,p_2,p_3 \in B_{i,j}$ to be shifted. Note that $H|_{T_i^C}$ and $G|_{T_j^R}$ are transitive, which implies there exists $\pi \in S_{[n]\times[m]}$ achieved by $O(1)$ steps, such that $\pi(p_1),\pi(p_2)$ are in the same row v, and $\pi(p_1),\pi(p_3)$ are in the same column u. Furthermore, there also exists $\sigma_1 \in G, \sigma_2 \in H$ such that

$$\sigma_1^{\text{Row-}v}\circ\pi(p_2) = \sigma_2^{\text{Col-}u}\circ\pi(p_3) = \pi(p_1).$$

Thus, a permutation can be achieved as:

$$\pi^{-1}\circ(\sigma_2^{\text{Col-}u})^{-1}\circ(\sigma_1^{\text{Row-}v})^{-1}\circ\sigma_2^{\text{Col-}u}\circ\sigma_1^{\text{Row-}v}\cdot\pi$$

which is equal to $(p_1\ p_2\ p_3)$. Since that σ can be decomposed into $O(\mathrm{poly}(n,m))$ 3-cycle, it can be achieved within $O(\mathrm{poly}(n,m))$ steps. ◀

For convenience, for any $\mathcal{B}$-local permutation π, define *the parity matrix* $\mathrm{PAR}(\pi) \in \mathbb{F}_2^{[c]\times[r]}$ where $\mathrm{PAR}(\pi)[\![i,j]\!] = \mathrm{sgn}(\pi|_{B_{i,j}})$. Furthermore, given two groups G and H in G2dSE and notations above, for any M_1 and M_2 which are equivalent under a $\mathcal{B}$-local permutation σ, define *the extended parity matrix class* $\mathrm{EPAR}(M_1,M_2) \subseteq \mathbb{F}_2^{[c]\times[r]}$, where any matrix $M \in \mathrm{EPAR}(M_1,M_2)$ satisfies

$$M[\![i,j]\!] = \begin{cases}\mathrm{PAR}(\sigma)[\![i,j]\!], & \text{if } M_1[\![B_{i,j}]\!] \text{ contains distinct elements;}\\ 0 \text{ or } 1, & \text{otherwise.}\end{cases}$$

Note that although σ may not be unique, when $M_1[\![B_{i,j}]\!]$ contains distinct elements, $\sigma|_{B_{i,j}}$ is unique. Thus the extended parity matrix class is well defined.

Given two matrices M_1, M_2 and a $\mathcal{B}$-local permutation σ satisfying $\sigma(M_1) = M_2$, suppose there are $p_1,p_2 \in B_{i,j}, (i,j)\in[c]\times[r]$ such that $p_1 \neq p_2$ and $M_1[\![p_1]\!] = M_1[\![p_2]\!]$. Thus $\sigma(M_1) = (\sigma\circ\textsc{Swap}(p_1,p_2))(M_1) = M_2$ but $\mathrm{PAR}(\sigma)[\![i,j]\!] \neq \mathrm{PAR}(\sigma\circ\textsc{Swap})[\![i,j]\!]$, which indicates that for any $A \in \mathrm{EPAR}(M_1,M_2)$, there exists a $\mathcal{B}$-local permutation σ such that $\sigma(M_1) = M_2$ and $\mathrm{PAR}(\sigma) = A$.

The next lemma finishes the proof of Theorem 3:

▶ **Lemma 24.** *In G2dSE where $G = \langle\hat{g}_1,\hat{g}_2,\ldots,\hat{g}_{\ell_1}\rangle$ and $H = \langle\hat{h}_1,\hat{h}_2,\ldots,\hat{h}_{\ell_2}\rangle$ are fix-point free, define $\mathcal{S} = \mathrm{span}\,(S_{base})$ where*

$$S_{base} := \left\{\mathrm{PAR}(g_i^{\textit{Row-}j})\middle| i\in[\ell_1], j\in[n]\right\}\cup\left\{\mathrm{PAR}(h_i^{\textit{Col-}j})\middle| i\in[\ell_2], j\in[m]\right\},$$

then M_1 can be transformed to M_2 if and only if $\mathrm{EPAR}(M_1,M_2)\cap\mathcal{S}\neq\varnothing$. Moreover, if achievable, the transformation requires $O(\mathrm{poly}(n,m))$ steps.

Proof. Consider two $\mathcal{B}$-local permutations π_1,π_2. Note that, for any $(i,j)\in[c]\times[r]$, $(\pi_1\circ\pi_2)|_{B_{i,j}}$ is odd if and only if the parity of $\pi_1|_{B_{i,j}}$ and $\pi_2|_{B_{i,j}}$ is different, which implies

$$\mathrm{PAR}(\pi_1\circ\pi_2)[\![i,j]\!] = \mathrm{PAR}(\pi_1)[\![i,j]\!] + \mathrm{PAR}(\pi_1)[\![i,j]\!],$$

thus $\mathrm{PAR}(\pi_1\circ\pi_2) = \mathrm{PAR}(\pi_1)+\mathrm{PAR}(\pi_2)$.

If M_2 is reachable from M_1, there exists $\mathcal{B}$-local permutation $\sigma\in\mathrm{EPAR}(M_1,M_2)$ and σ can be decomposed into a sequence of permutations $\sigma = \sigma_k\circ\cdots\circ\sigma_2\circ\sigma_1, \sigma_i\in S_{\text{base}}$. Thus, $\mathrm{PAR}(\sigma) = \sum_{i=1}^k \mathrm{PAR}(\sigma_i)\in\mathcal{S}$.

By RowBasis and ColBasis, Algorithm 3 gets $\mathcal{S}_{\text{base}}$. If $\text{EPAR}(M_1, M_2) \cap \mathcal{S} \neq \varnothing$, there exists permutation $\sigma \in \text{EPAR}(M_1, M_2) \cap \mathcal{S}$. Since $\sigma \in \mathcal{S}$, $\text{PAR}(\sigma) = \sum_{i=1}^{k} \text{PAR}(\sigma_i), \sigma_i \in \mathcal{S}_{\text{base}}$ and $k \leq \operatorname{rank} \mathcal{S} \leq r + c$. Thus $\text{PAR}(\sigma') = 0^{[c]\times[r]}$ where $\sigma' := \sigma \circ \sigma_1^{-1} \circ \cdots \circ \sigma_k^{-1}$. By Lemma 23, σ' can be achieved with $O(\text{poly}(n, m))$ steps, which means M_1 can be transformed to M_2 within $O(\text{poly}(n, m) + k) = O(\text{poly}(n, m))$ steps. ◀

Finally, Algorithm 3 calls LinearDependent to judge whether the intersection of $\text{span}(\mathcal{S}_{\text{base}})$ and $\text{EPAR}(M_1, M_2)$ is empty. From the perspective of computational geometry, $\text{EPAR}(M_1, M_2)$ is an affine subspace and $\mathcal{S}$ is a linear subspace, thus computing the intersection between $\text{EPAR}(M_1, M_2)$ and $\mathcal{S}$ is obviously in $\mathbf{P}$ due to Gaussian elimination algorithm over $\mathbb{F}_2^{[c]\times[r]}$, which completes the proof of Theorem 3.

C Proofs of Lemmas and Theorems in Section 5

C.1 Proof of Lemma 8

Proof. Suppose all permutations can be achieved within k steps. Note that in each step, at most $n^m + m^n$ different permutations can be performed. Thus, to generate $(nm)!$ different permutations, $(n^m + m^n)^k \geq (nm)!$ should be satisfied, which implies $k = \Omega(n \log_n m)$. ◀

C.2 Proofs of Lemmas in Section 5.3

Proof of Lemma 11. Since $(s_i \rightsquigarrow 1)^{\text{Row-}i}$ costs only 1 step, we may assume $s_i = 1$ for all $i \in T$. Then after calling $\text{SwapRows}(e_1, t - 1^n)$.

- For any $i \in T$, $(i, 1)$ and (i, t_i) are swapped and $(i, j), j \in [m] \backslash \{1, t_i\}$ remains invariant;
- For any $i \in [n] \backslash T$ and any $j \in [m] \backslash \{1\}$, (i, j) remains invariant;
- For any $i \in [n] \backslash T$, $(i, 1)$ is shifted to $(i', 1)$, where i' is the "next" element in $[n] \backslash T$.

Thus after the procedure, elements in $([n] \backslash T) \times \{1\}$ are shifted by one. Denote π as such shift. Therefore, $\text{SwapRows}(e_1, t - 1^n)$ is exactly τ. Similar analysis fits for τ^{-1} by calling $\text{SwapRows}(-e_1, t - 1^n)$. ◀

Proof of Lemma 13. Since σ_i is even, it can be decomposed into $2m$ swaps $\pi_{i,1}, \ldots, \pi_{i,2m}$ by Fact 12. For $j \in [m]$, perform τ in Lemma 11 to achieve $\pi_{i,2j-1}, \forall i \in T$. After then, perform τ^{-1} in Lemma 11 to achieve $\pi_{i,2j}, \forall i \in T$, which erases the changes in the first column caused by $\text{ShiftRow}(\cdot)$. Thus $\prod_{i \in T} \sigma_i^{\text{Row-}i}$ can be achieved in $O(m)$ steps. ◀

Proof of Lemma 14. Note that $\pi := 1 \rightsquigarrow 2$ is odd since m is even. Then let $\sigma_i' = \sigma_i, \tilde{\sigma}_i = \mathsf{id}$ if σ_i is even and $\sigma_i' = \sigma_i \pi, \tilde{\sigma}_i = \pi^{-1}$ otherwise. Thus, σ_i' is even, which can be achieved in $O(m)$ steps by Lemma 13. Then performing Algorithm 6 on $\prod_{i \in T} \tilde{\sigma}_i^{\text{Row-}i}$ will make sense. ◀

Proof of Lemma 15. Denote $S \subseteq T$ as the set of all the column index i such that σ_i is odd. Decompose σ_i for each $i \in S$ as an even permutation σ_i' and an extra swap π_i. By assumption, $\sigma_i', i \in S$ and $\sigma_i, i \in T \backslash S$ can be achieved in t_1 steps. Then, in $O(1)$ steps, Lemma 11 helps to achieve $\pi_i, i \in S$ while changes the first row. Therefore, it is necessary to call a t_2-step routine to perform a permutation on the first row to counteract the side effect. ◀

C.3 Proofs of Lemmas in Section 5.4

Periodic switching network is the key point to speed up our algorithm in Section 5. Define the *switching network* as a permuting procedure. A d-depth switching network for n items contains n lines and each of them is divided into d levels. In each level, we can connect pairs of different lines and each line can be connected at most once. For example, Figure 4 is a 5-depth network with 8 lines.

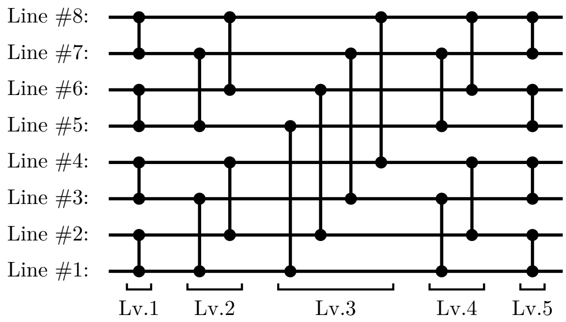

Figure 4 Switching network

Given a switching network with n lines, we can achieve permutations over n items as follows. Going through all the d levels in turn, in each level we choose whether to swap item-s and item-t if s-th and t-th line are connected in this level. We say the network can achieve a permutation σ if there exists a switching procedure along the network to achieve σ. Furthermore, we say a switching network is *periodic* if and only if in each level the distance between any connected pair is the same, i.e., if $(s_1, t_1), \ldots, (s_k, t_k)$ are connected in some level in a periodic switching network, then $|s_i - t_i|, i \in [k]$ are the same.

Note that there exists a well-constructed class of periodic switching network called *Beneš network*, which realizes arbitrary permutation over $[2^n]$ with $(2n-1)$-depth [4]. Also, we define the composition of two switching network N_1 and N_2 as $N_2 \circ N_1$, where N_1 is a d_1-depth switching network and N_2 is d_2-depth. Then $N_2 \circ N_1$ is a $(d_1 + d_2)$-depth switching network; the first d_1 levels are constructed as N_1 and the last d_2 levels as N_2. By composing several *Beneš networks*, we can construct an $O(\log m)$-depth periodic switching network to achieve $S_{[m]}$ for arbitrary m.

▶ **Lemma 25.** *There exists an $O(\log m)$-depth periodic switching network to perform arbitrary permutation $\sigma \in S_{[m]}$ on $[m]$.*

Proof. Let $N = 2^{\lfloor \log m \rfloor}$. Pick arbitrary 3 intervals S_1, S_2, S_3 over $[m]$, such that S_1, S_2 and S_2, S_3 share at least $N/2$ positions respectively and $S_1 \cup S_3 = [m]$.

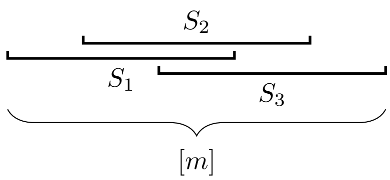

Figure 5 Intervals cover $[m]$

Then construct 3 Beneš networks N_1, N_2 and N_3 to achieve permutations on S_1, S_2 and S_3 respectively. We claim the composed network $N_4 = N_1 \circ N_2 \circ N_1$ can achieve any permutations on $S_1 \cup S_2$. In N_1, switch all the elements in $S_1 \backslash S_2$, whose target positions are in $S_2 \backslash S_1$, into $S_1 \cap S_2$. Note that it is feasible since $|S_1 \cap S_2| \geq N/2$. Then in N_2, achieve the permutation on $S_2 \backslash S_1$, and in second N_1, achieve the permutation on S_1. Similarly, construct $N_5 = N_2 \circ N_3 \circ N_2$ which achieves all permutations on $S_2 \cup S_3$. Furthermore,

$|(S_1 \cup S_2) \cap (S_2 \cup S_3)| = |S_2| \geq \max\{|S_1 \cup S_2|, |S_2 \cup S_3|\}/2$ holds, which means $N_6 = N_4 \circ N_5 \circ N_4$ achieves all permutation on $[m]$. The final network N_6 is $18\lfloor \log n \rfloor - 9 = O(\log n)$. ◀

Proof of Lemma 16. SwapRows can perform all swaps in only one level of a periodic switching network, with some elements out of row-r shifted. Thus, calling SwapRows for $O(\log m)$ times is sufficient to achieve σ in row-r.

Before Recover is called, some columns are partly shifted by 1 and the others are invariant, which is recorded by a vector w. It is easy to see that $|w|$ shares the same parity with σ. Thus, the number of partly shifted columns is even. Now we propose an $O(\log m)$-operation implement of Recover.

Algorithm 10 Column recovery

```
procedure RECOVER(r, v)
    ℓ ← m
    repeat
        ℓ' = ⌊ℓ/2⌋, s ← ℓ − ℓ', v̂ ← 0^{ℓ'} and ṽ ← 0^{ℓ'}
        for all i ∈ [ℓ'] such that v_{s+i} = 1 do
            v̂_{s+i} ← −1, ṽ_i ← 1 − 2v_i
        SWAPROWS(v̂, −s · e_r)
        SWAPROWS(ṽ, s · e_r)
        ℓ ← s, v ← v + v̂ + ṽ
    until ℓ < 2
```

Since ℓ is halved by at most $O(\log m)$ times, this algorithm performs $O(\log m)$ calls. Also, it can be easily verified that after SwapRows$(\hat{v}, -s \cdot e_1)$ and SwapRows$(\tilde{v}, s \cdot e_1)$, row-1 remains unchanged and column-i for any $i \in [s+1, s+\ell']$ is recovered. In addition, v_i represents exactly if columns are partly shifted by 1 after every update. In a word, this algorithm restores at least $\sum_{i=1}^{t} 1/2^t$ of the columns in t rounds without interfering other positions.

Therefore, after the two stages, $\sigma^{\text{Row-1}}$ is performed and the other rows are invariant and it costs totally $O(\log m)$ steps. ◀