PLS-complete problems with lexicographic cost functions: Max-k-SAT and Abelian Permutation Orbit Minimization

Dominik Scheder
TU Chemnitz, Germany
dominik.scheder@informatik.tu-chemnitz.de

Johannes Tantow
TU Chemnitz, Germany
johannes.tantow@informatik.tu-chemnitz.de

December 5, 2025

Abstract

How hard is it to find a local optimum? If we are given a graph and want to find a locally maximal cut–meaning that the number of edges in the cut cannot be improved by moving a single vertex from one side to the other–then just iterating improving steps finds a local maximum since the size of the cut can increase at most $|E|$ times. If, on the other hand, the edges are weighted, this problem becomes hard for the class PLS (Polynomial Local Search) [20].

We are interested in optimization problems with *lexicographic costs*. For Max-Cut this would mean that the edges $e_1, \dots, e_m$ have costs $c(e_i) = 2^{m-i}$. For such a cost function, it is easy to see that finding a *global* Max-Cut is easy. In contrast, we show that it is PLS-complete to find an assignment for a 4-CNF formula that is locally maximal (when the clauses have lexicographic weights); and also for a 3-CNF when we relax the notion of "local" by allowing to switch two variables at a time.

We use these results to answer a question in Scheder and Tantow [21], who showed that finding a lexicographic local minimum of a string $s \in \{0,1\}^n$ under the action of a list of given permutations $\pi_1, \dots, \pi_k \in S_n$ is PLS-complete. They ask whether the problem stays PLS-complete when the $\pi_1, \dots, \pi_k$ commute, i.e., generate an Abelian subgroup G of S_n. In this work, we show that it does, and in fact stays PLS-complete even (1) when every element in G has order two and also (2) when G is cyclic, i.e., all $\pi_1, \dots, \pi_k$ are powers of a single permutation π.

Additionally, we use it to reprove the hardness of computing an α approximate Nash equilibria in congestion games by Skopalik and Vöcking[22] and extend the result from step functions to exponential functions.

1 Introduction

Given a set $V = \{1, \ldots, n\}$ of positions, a list $\pi_1, \ldots, \pi_m \in S_V$ of permutations thereon, and a string $s : V \to \{0, 1\}$, what is the lexicographically smallest string we can get by starting with s and repeatedly applying permutations π_i on it? Formally, which string in the orbit

$$\{s \circ \sigma \mid \sigma \in \langle \pi_1, \ldots, \pi_m \rangle\}$$

is the lexicographically smallest? This problem is known to be NP-hard even for $m = 1$ [21]. A more modest goal is to find a *local minimum*, i.e., a $v = s \circ \sigma$ for some $\sigma \in \langle \pi_1, \ldots, \pi_m \rangle$ such that none of $v \circ \pi_1, v \circ \pi_2, \ldots, v \circ \pi_m$ is lexicographically smaller than v. Such a local minimum obviously exists and can be verified efficiently (this is less obvious and we discuss it later). This means finding a local minimum is in the complexity class PLS (*polynomial local search*), more formally defined in the next sections. We call this problem Permutation Orbit Local Minimization, short Polm. Scheder and Tantow [21] showed that Polm is PLS-complete and thus unlikely to admit an efficient algorithm. They left it as an open question whether Polm is still PLS-complete if all given permutations commute, i.e., if $\langle \pi_1, \ldots, \pi_m \rangle$ is Abelian. We answer this question: Yes, Abelian Polm is still PLS-complete.

1.1 Motivation

The motivation behind Polm comes from SAT solving. Kołodziejczyk and Thapen [13] investigate whether proof systems including *symmetry breaking rules* can be simulated by proof systems without them. For a concrete example, suppose we want to prove that the Ramsey number $R(5, 5)$ is at least 43. The statement "G is a graph on 43 vertices and has no clique nor independent set of size 5" can be written as a CNF formula (here: a 10-CNF formula with $\binom{43}{2}$ variables) and fed to a SAT solver. If there is a solution, then every graph isomorphic to it is also a solution. Thus, we can try to speed things up by adding certain *symmetry breaking constraints*. For example, we might want to add constraints formalizing the statement "and G should be lexicographically minimal". If by "minimal" we mean "a global minimum" then this statement cannot be written efficiently as a CNF formula; if we mean "the graph cannot be made smaller by exchanging the names of two vertices" then we can encode this as a CNF formula; we are now in fact looking for a solution that is a lexicographic local minimum with respect to the permutations on the set of edges defined by swapping two vertices.

1.2 Total search problems

A function problem or search problem asks us to compute for a given input x a value y, called a *solution*, such that $(x, y) \in R$ for some problem-specific relation R. In the case of SAT, R would be the relation of formulas with their models.

A search problem is in FNP if y is polynomially bounded, i.e. $|y| = poly(|x|)$, and $(x, y) \in R$ can be checked in polynomial time.

Another example of an FNP-problem would be FACTORING, where x is a number and y is a prime factor of x, which can be checked in polynomial time. In contrast to SAT, every instance x has at least one solution y and hence it is a *total search problem*. These class of total search problems is called TFNP. If a TFNP-problem were to be NP-complete, then NP = coNP would follow[17]. Hence NP-hardness is not the right tool to study the hardness of total search problems. As TFNP is a semantic class, no complete problems are known. Instead, it is usually studied via its subclasses based on the combinatorial principle used to prove the totality of the problem. Here are some examples of subclasses, together with their defining combinatorial principle.

- PLS: Every DAG has a sink [12].
- PPAD: If a directed graph has a vertex u with more edges going out than coming in, then there must be some other vertex v with more edges coming in than going out [18]
- PPA: If an undirected graph has a vertex u of odd degree, it must have another such vertex v [18].
- PPP: The pigeonhole principle [18].

All these classes are conjectured to be different from FP (the class of efficiently solvable search problems). This list is not exhaustive: FACTORING for example is not known to be in any of them.

1.3 PLS

Local search is a powerful technique to solve optimization problems in practice. Nonetheless, there are instances where local search takes a long time to converge. In order to study local search from a theoretical perspective, the class PLS was introduced in 1988 [12]. Following [9], a combinatorial optimization problem can be stated as follows:

1. The set of feasible solutions $S \subseteq \{0, 1, \ldots, M-1\}^v \times \{0, 1\}^{\bar{v}}$ for $M, v, \bar{v} \in \mathbb{N}$. Each feasible solution can be expressed as a tuple $s = (p, x)$ with the cost part $p \in \{0, 1, \ldots, M-1\}^v$ and the non-cost part[1] $x \in \{0, 1\}^{\bar{v}}$.
2. A vector of costs (or payoffs) $c \in \mathbb{R}^v$ which leads to the cost function for a feasible solution $\text{cost}(s) := \sum_{i=1}^{v} c_i p_i$
3. A neighborhood function $N : S \to 2^S$.

[1] This was introduced for technical reasons, as multiple solutions can have the same costs, but a different neighborhood.

For example, Weighted Local Max Cut can be formalized by using the non-cost part to encode the 2-coloring of the vertices (setting $\bar{v} := |V|$) and having the cost part encode which edge crosses the cut and which doesn't (setting $v := |E|$). The cost vector $c \in \mathbb{R}^{|E|}$ lists the cost of each edge. A solution (p, x) is feasible if the cost part p truthfully represents which edges are two-colored under the coloring x. The neighborhood function computes all 2-colorings that can be achieved by changing the color of a single vertex, along with the new vector indicating which edges are two-colored.

PLS is now defined as the class of optimization problems where a start solution and the neighborhood function are computable in polynomial time, and the set S of feasible solutions can be recognized efficiently. A local optimum is a feasible solution s such that $\text{cost}(s) \geq \text{cost}(s')$ for all $s' \in N(s)$.[2]

A local optimum can be found with the *standard algorithm*: We compute a starting solution in polynomial time and then always select an improving neighbor until no such neighbor exists. There are instances where this can take exponentially many steps[12]. In fact, the standard algorithm can take exponentially long even when the optimization problem itself is in P: think about linear programming and the simplex algorithm.

PLS contains problems that are complete under search problem reductions, for example finding a locally optimal max-cut in a weighted graph[20], finding locally optimal assignments for weighted CNF-formulas[14] or finding pure Nash equilibria in congestion games[7].

1.4 Lexicographic costs

If the costs are polynomially bounded in the input size, then the PLS-problem becomes easy to solve: The cost function is then also bounded by a polynomial, and each step of the standard algorithm improves the cost by at least 1. Hence, the standard algorithm finds a local optimum after at most polynomially many steps.

Therefore, the costs in PLS-reductions must be large, usually of the form 2^i. However, often multiple entries of the cost vector are the same. In order to compare two solutions, one would have to compute the costs of both and then compare them, which requires multiple additions and multiplications. If the cost function is like $c_i = 2^{-i}$ and $M = 2$, then we get a *lexicographic* cost function: $\text{cost}(p_1, x_2) < \text{cost}(p_2, x_2)$ if and only if $p_1 <_{\text{lex}} p_2$. Thus, the statement that $s = (p, x)$ is lexicographically minimal can be encoded by a simple CNF formula, which is important in contexts such as [13].[3] Thus, we call a cost function *lexicographic* if the coefficients of the cost vector fulfill $c_i = C \cdot M^{-i}$, where M is usually 2, and the common factor C is just for convenience, to allow for a vector $(16, 8, 4, 2, 1)$, for example. If a cost function is lexicographic, then to compare (p, x) and (p', x') it suffices to look for the first position i with $p_i \neq p'_i$;

[2] Alternatively, we can also view it as a minimization problem, where the cost of s is smaller or equal compared to all of its neighbors.

[3] For larger values of M we would define lexicographic costs as $c_i = M^{-i}$, but we do not encounter this case in our reductions.

everything to the right pales in comparison. Arguably, this is one of the most simple cost function available. The central question is: Which problems are PLS-complete problems when we insist on lexicographic costs?

The first known PLS-complete problem FLIP[12] fulfills this property. We are given a circuit C with n inputs and m outputs and want to compute a locally minimal output. Formally, this optimization problem is defined by the relation $S = \{(C(x), x) \mid x \in \{0,1\}^n\}$, with the output $C(x)$ being the cost part and input x being the non-cost part. The neighborhood of $(C(x), x)$ is the set of all $(C(x \oplus \mathbf{e}_i), x \oplus \mathbf{e}_i)$, i.e., we are allowed to flip one bit of the input. The cost vector is $c_i = 2^{-i}$, i.e., we want to lexicographically maximize the output. To the best of our knowledge, this has for the longest time been the only known PLS-complete problem with lexicographic costs. Recently, Polm joined this club [21].

Lexicographic costs are often easy. Multiple PLS-complete problems are easy with a lexicographic cost function. Consider the weighted independent set problem[20]. Our greedy algorithm sorts the vertices by weight from high to low and starts with $I = \emptyset$. For $i = 1, \ldots, n$ it adds v_i to I if $I \cup \{v_i\}$ is still independent. Similarly, we can find a global optimum of a max-cut instance with lexicographic weights. Here we have to check whether a given set of edges can be realized in a cut. This is easy because we just have to check whether they form a bipartite graph. More generally, Max-2-SAT with lexicographic costs can be solved efficiently because a given subset of clauses can be efficiently tested for satisfiability. It is not obvious how this works for larger clauses, as there is no known way to check satisfiability of a subset of clauses. In fact, we show that finding a local maximum under lexicographic clause weights is PLS-hard for 4-CNF formulas. We should note that even though a global lex-maximum for 2-SAT can be found efficiently, we do not know whether the standard algorithm terminates after a polynomial number of steps.

Lexicographic cost problems as starting point of PLS reductions. Since lexicographic cost function are intuitively easier than general ones, it is not surprising that reducing *from* a lexicographic PLS-complete problem can be particularly easy. This is helpful if we are also interested in the hardness of approximation. As each cost entry is unique and we are only interested in the lexicographic order between two solutions, we can increase the base of the cost from 2 to be larger than the desired approximation ratio. In fact, we give a simple PLS-reduction from local max 4-SAT with lexicographic costs to computing an α-Nash equilibrium for congestion games, thus greatly simplifying a proof by Skopalik and Vöcking [22].

1.5 Reductions in PLS

In order to talk about PLS-completeness, we need a notion of reductions. A PLS reduction[12] from P to Q is a pair of functions f and g. f maps an instance

I of P to an instance $f(I)$ of Q. g maps a solution s of $f(I)$ to a solution $g(I,s)$ of I, so that if s is a local optimum for $f(I)$ then $g(I,s)$ is a local optimum for I. Additionally, both f and g are computable in polynomial time. One checks that this fulfills the purpose of a reduction: if we can efficiently solve Q (i.e., find local minimum) then we can solve P. The transition graph $TG(I)$ of an instance I of a PLS-problem P is the directed graph where the vertices are the feasible solutions of I and an edge (u,v) is present if v is a neighbor of u according to the neighborhood function.

Given an optimization problem P, the following problem is more difficult than finding a local optimum: given a start solution s, find the solution on which the standard algorithm terminates. We call this the standard algorithm end point problem (SAEPP). It is known that SAEPP for FLIP is PSPACE-complete.

A PLS-reduction (f,g) from P to Q is called *tight*[20] if for every instance I of P there exists a set $\mathcal{R}$ of feasible solutions of the instance $f(I)$ of Q such that

1. $\mathcal{R}$ contains all local optima of $f(I)$.
2. For every solution s of I we can construct in polynomial time a solution t of $f(I)$ such that $g(I,t) = s$
3. If the transition graph $TG(f(I))$ has a path from q to q' such that q and q' are the only vertices of that path in $\mathcal{R}$, then either $g(I,q) = g(I,q')$ or there is an edge from $g(I,q)$ to $g(I,q')$ in $TG(I)$.

Tight reductions express that the graph structures of the two problems is essentially the same (with possibly more intermediate steps in Q) and therefore preserve the PSPACE-completeness of the associated SAEPP problem.

1.6 Previous results

The Polm problem arose from proof-complexity based investigations into the dominance rule by Kołodziejczyk and Thapen [13]. Its PLS-completeness was shown by Scheder and Tantow in [21]. If the output of Polm is the locally minimized string $x \in \{0,1\}^n$ itself, then the problem of verifying that x is in the orbit, i.e., that there is a permutation $\pi \in \langle \pi_1, \ldots, \pi_k \rangle$ with $x = s \circ \pi$ is known as the *string isomorphism problem*, which is at least as hard as graph isomorphism. Thus, to make sure Polm is in PLS, we want the output to not be x but the permutation π itself; now whether $\pi \in \langle \pi_1, \ldots, \pi_k \rangle$ can be verified efficiently using an algorithm of Furst, Hopcroft, and Luks [8]. For the hardness proof, they used a direct reduction from FLIP using gate gadgets and the test circuit technique by Krentel[14]. While the gate gadgets could be built with commuting permutations, the test circuit technique introduced permutations that do not commute with the permutations of the gate gadgets. Hence, the question of the complexity of Polm for Abelian permutation groups arose.

Given a CNF formula with clause weights and an assignment α, the *payoff* of α is the total weight of satisfied clauses. We denote by k-SAT/d-FLIP the

optimization problem, where we are given a k-CNF formula with clause weights and want to find a truth assignment α whose payoff is locally optimal: switching α at at most d variables does not increase the payoff. For arbitrary costs (i.e., not necessarily lexicographic) it has been known that k-SAT/1-FLIP is PLS-complete [15].[4] Later, [20] showed that Max-Cut is PLS-compete, which immediately implies that 2-SAT/1-FLIP is PLS-complete.

Further general PLS reductions can be found in [6, 12, 14, 20]. In game theory the PLS-completeness of finding a pure Nash equilibria was shown via different routes in[7] and [1]. Buchheim and Jünger proved NP-completeness for multiple problems regarding permutations in [2].

1.7 Our results

Theorem 1. *The 4-SAT/2-FLIP problem with lexicographic payoffs is* PLS*-complete.*

The proof, given in Section 3, follows the approach of Krentel [14, 15], who reduces from FLIP, but deviates from it in one crucial detail. With some additional work, we can show that 3-SAT/2-FLIP with lexicographic payoffs is PLS-complete.

Theorem 2. *The 3-SAT/2-FLIP problem with lexicographic payoffs is* PLS*-complete.*

We cannot further reduce the clause width k, since Max-2-SAT with lexicographic clause payoffs can be solved in polynomial time, as argued above, so even a global maximum can be found efficiently. Using additional helper variables, we manage to reduce from the 2-Flip neighborhood to a more natural 1-Flip neighborhood.

Theorem 3. *The 4-SAT/1-FLIP problem with lexicographical costs is* PLS*-complete.*

We apply the previous results to answer an open question from Scheder and Tantow [21]

Theorem 4. Polm *is* PLS*-complete even when restricted to commuting permutations.*

We prove the theorem in Section 6 The proof follows from a reduction very similar to Buchheim and Jünger [2]. The permutations constructed in the reduction are commuting involutions (permutations of order 2) and thus the generated subgroup of the symmetric group is isomorphic to the additive group of $\mathbb{F}_2^n$, the vector space over $\mathbb{F}_2$.

We will give a second reduction, following the proof of Theorem 2 in [21], where all permutations are powers of one single permutation π. In other words, we prove the following theorem:

[4] In fact, Krentel gives a reduction to $(3, 4)$-CSP and then claims that it is easy to see that it also works for SAT.

Theorem 5. POLM *is* PLS*-complete, even when restricted to permutations* $\pi^{i_1}, \pi^{i_2}, \ldots, \pi^{i_n}$ *that are powers of one single permutation* π*.*

The proof is in Section 7.

Finally, we extend our approach to game theory and the complexity of finding approximate Nash equilibria. This was proven to be PLS-complete in [22]. The hardness of lexicographic 4-SAT/1-Flip allows an easy proof and generalizes the original statement to exponential functions.

Theorem 6. *Finding* α*-Nash equilibria in congestion games is* PLS*-complete for step functions and exponential functions as delay functions.*

The proof is in Section 8

2 Krentel's approach and why it does not directly work for lexicographic Max-k-SAT

The weighted CNF-SAT/1-Flip problem consists of a CNF-formula and weights for each clause. The payoff of an assignment is the sum of all weights of the clauses that are satisfied by the assignment. The neighborhood of an assignment are all assignments that are achieved by changing the value of a single variable.

Krentel proved this to be PLS-complete[14] with the test circuit technique by reducing from FLIP. We sketch this proof again and show why it does not work for lexicographic weights. We are given a circuit C with n inputs and m outputs. For the reduction, we imagine $n+1$ copies of C, called $C^{(0)}, C^{(1)}, \ldots, C^{(n)}$. We want $C^{(0)}$ to compute $C(x)$ for the current input x, and each other $C^{(i)}$ $(1 \leq i \leq n)$ to compute $C(x \oplus \mathbf{e}_i)$. For each circuit $C^{(i)}$ $(0 \leq i \leq n)$ we create a vector of n input variables $\mathbf{x}^{(i)} = (x_1^{(i)}, \ldots, x_n^{(i)})$. We wish for $\mathbf{x}^{(i)} = \mathbf{x}^{(0)} \oplus \mathbf{e}_i$. For each circuit i and gate j we create a gate variable $g_j^{(i)}$. The variables involved in a gate can be consistent or inconsistent with the gate, and we call the gate *correct* or *incorrect* accordingly. We wish for all gates to be correct, but have to tolerate them being temporarily incorrect.

We create n *flip variables* $f^{(1)}, \ldots, f^{(n)}$ that indicate which circuit is currently *active*. If all $f^{(n)} = \cdots = f^{(n)} = 0$ we say that the main circuit is active. A circuit being active means that we "harvest" its output, i.e., its output should contribute to the payoff function. The idea is now that when $C^{(i)}$ is correct and its output is better than that of $C^{(0)}$, we can mark $C^{(i)}$ as active and thus harvest its output instead. We should then replace $\mathbf{x}^{(0)}$ by the "better" $\mathbf{x}^{(i)}$, change gate variables $g_j^{(0)}$ to make all of $C^{(0)}$ correct; make $C^{(0)}$ active; adapt the inputs $\mathbf{x}^{(i)}$ for $1 \leq i \leq n$ to face the new reality of $\mathbf{x}^{(0)}$ having changed; then make all circuits correct again, rinse and repeat. We design clauses that describe these wishes. Their weights descend along the list. The clauses describe:

1. At most one flip variable must be true.
2. All gates in the active circuit must be correct (and if $i \neq 0$ is active then its input must be correct, i.e., $\mathbf{x}^{(i)} = \mathbf{x}^{(0)} \oplus \mathbf{e}_i$.

3. The value of the output of the active circuit should be high.

4. If $i \geq 1$ is active then $\mathbf{x}^{(0)}$ should be identical to $\mathbf{x}^{(i)}$.

5. All flip variables should be 0 (i.e., the active circuit should preferably be $C^{(0)}$).

6. All circuits should be correct.

If one chooses suitable clause weights, then those six groups guarantee that there is always a variable flip that improves the current assignment, unless the input to the main circuit is a local optimum. The problematic part is the third group; here the weight of the satisfied clauses should correspond to the computed output of the circuit. If $o_j^{(i)}$ denotes the j-th output of $C^{(i)}$ then those clauses are

$$\begin{aligned} O_j^{(0)} &:= (\neg f_1 \wedge \cdots \wedge \neg f_n) \rightarrow o_j^{(0)} \\ O_j^{(1)} &:= f_1 \rightarrow o_j^{(1)} \\ &\vdots \\ O_j^{(n)} &:= f_n \rightarrow o_j^{(n)} \end{aligned} \tag{2.1}$$

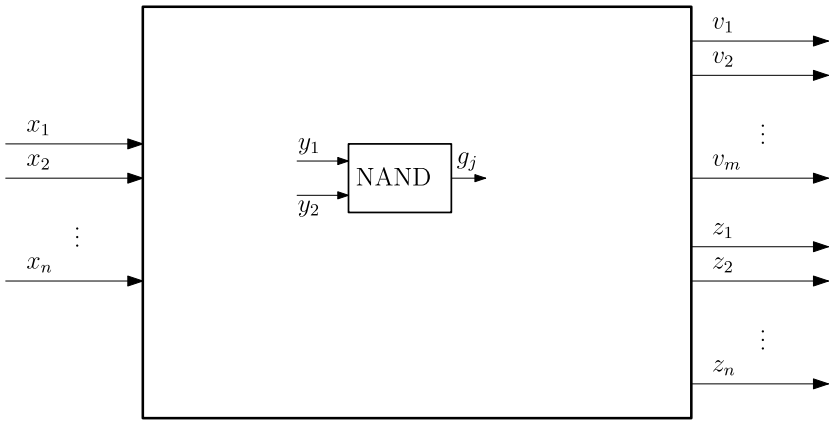
(2.2)

The only violated clauses in this group are those belonging to 0-outputs of the active input. The weight of $O_j^{(i)}$ is proportional to 2^{-j} and equal for all i, thus the total payoff of group 3 is a constant plus something proportional to the output of the active circuit (read as a binary number). Thus, the total payoff of group 3 increases when we activate a circuit with a better output. With lexicographic costs, this is no longer possible. Suppose $C^{(0)}$ and $C^{(1)}$ currently both have a 0 in the first output position: $o_1^{(0)} = o_1^{(1)} = 0$. If $C^{(0)}$ is active then $O_1^{(0)}$ is violated and $O_1^{(1)}$ is satisfied; if $O_1^{(1)}$ has higher weight than $O_1^{(0)}$, then making $C^{(1)}$ active violates a more valuable clause, which cannot be compensated by gaining any number of less valuable clauses. It does not pay to make $C^{(1)}$ active. If, on the other hand, $O_1^{(0)}$ is more valuable than $O_1^{(1)}$, then we have a very strong incentive to make $C^{(1)}$ active (setting $f^{(1)}$ to 1); but then we will never switch $f^{(1)}$ back to 0 to active $C^{(0)}$, lest we lose $O_1^{(0)}$. Krentel's construction works only when we are allowed to make all of $O_j^{(0)}, O_j^{(1)}, \ldots, O_j^{(n)}$ equally valuable.

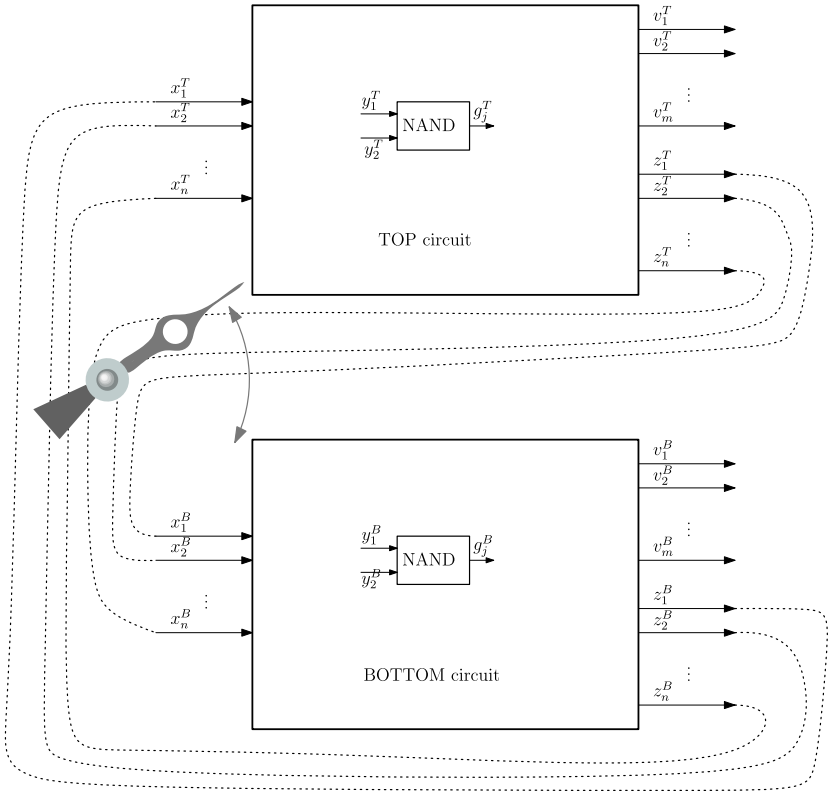
Thus, to make the cost function truly lexicographic, we need a new idea.

3 Lexicographic 4-SAT/2-Flip: Proof of Theorem 1

We reduce from FLIP. Let C' be the input circuit to FLIP. It has n input variables and m output variables. The output $C'(\vec{x})$, when read as a binary number, describes the payoff of the solution $\vec{x}$. We build a new circuit C with a *payoff output* $\vec{v} = (v_1, \ldots, v_m)$ and a *best-neighbor output* $\vec{z} = (z_1, \ldots, z_n)$ with the following properties: the first m output bits $\vec{v}$ are simply $C'(\vec{x})$, describing the payoff of $\vec{x}$. The n output bits $\vec{z}$ describe a best neighbor of $\vec{x}$ among its n neighbors of Hamming distance one. We assume without loss of generality that all gates are NAND gates and that every path from input to output passes through at least one gate.



In a second step, we replace this circuit by two copies, called TOP and BOTTOM. The idea is that the best-neighbor output of one is lazily fed back as the input to the other:



With this arrangement in mind, we will now construct a 4-CNF formula $F = C_1 \wedge C_2 \wedge \cdots \wedge C_t$ with the property that every locally optimal assignment α (meaning that the t-bit vector of satisfied clauses) cannot be improved by flipping one or two variables) corresponds to a local optimum of FLIP.

The variables. For every input variable x_i of C we construct two copies x_i^T and x_i^B. For every gate g_j of C we create g_j^T and g_j^B. The output variables $v_j^T, z_j^T, v_j^B, z_j^B$ are not separate variables, but simply synonyms for the gate variable of the gate that outputs them. Consider a gate g_j^T (or similarly g_j^B) and call its two input variables y_1, y_2 for the moment. Let α be a truth assignment to the variables. We call the gate *correct under* α if $\alpha(g_j^T) = \text{NAND}(\alpha(y_1), \alpha(y_2))$. We call the circuit TOP correct if all its gates are correct (and similarly for BOTTOM).

We have a switch variable f (the needle in the above figure) that indicates whether TOP is considered the *active circuit* (if $f = 1$) or BOTTOM is (if $f = 0$). Finally, and this differs from [14], we have a *global output register* $\vec{o} = (o_1, o_2, \ldots, o_m)$.

The idea now is that (1) $\vec{o}$ should be identical to the value output of the active circuit; (2) the neighbor output of the active circuit should be the input

of the inactive circuit; (3) the value output of the inactive circuit should not be better than $\vec{o}$. If all three conditions are met (and both circuits correctly compute the output), then $\vec{x}^T$ is a local optimum for FLIP.

A difficulty. Simulating a correct evaluation of all gates is relatively easy to do with k-SAT/1-FLIP. The problem arises when all circuits are correct, but the payoff output of the inactive circuit is better than that of the active output. We need a mechanism that allows us to make the inactive circuit active (and vice versa) while improving the payoff (the total value of satisfied clauses). In [14] this is achieved by introducing clauses $(f \to v_i^T)$ and $(\neg f \to v_i^B)$, both of weight 2^{c-j} for some constant c. Making the BOTTOM active and TOP inactive (for example) will increase the total satisfied weight if and only if $\vec{v}^B >_{\text{lex}} \vec{v}^T$. However, this only works if $(f \to v_i^T)$ and $(\neg f \to v_i^B)$ really have the same weight. If we insist on lexicographic weights and the first bit of both outputs is 0, say, and BOTTOM is active, then among $(f \to v_1^T)$ and $(\neg f \to v_1^B)$ only the first clause is satisfied, and it would never pay to make TOP active. This is where we need to introduce the global output and allow double flips.

The clauses. We will now describe in detail the clauses of our 4-CNF formula F. They come in groups, which we list from most-important to least-important.

1. **Every gate in the active circuit should be correct.** When a gate g_j has inputs y_1 and y_2, we write

$$f \to (g_j^T \leftrightarrow \text{NAND}(y_1^T, y_2^T)) \tag{3.1}$$

   and express this as a conjunction of four 4-clauses. Similarly, for the case that BOTTOM is active:

$$\neg f \to (g_j^B \leftrightarrow \text{NAND}(y_1^B, y_2^B)) \tag{3.2}$$

   Within this group, the ordering of the clauses follow the topological ordering of the gates (when g feeds into h then g has comes before h and the clauses thusly produced come first, too. Within the eight clauses of (3.1-3.2) for a single gate, the order is arbitrary.

2. **Each payoff output bit of the active circuit should be large, but not larger than the global output.** For each payoff output position $1 \le i \le m$ we create three clauses:

$$C_i^T := f \to (o_i \le v_i^T) \tag{3.3}$$

$$C_i^B := \neg f \to (o_i \le v_i^B) \tag{3.4}$$

$$o_i \tag{3.5}$$

   Clauses C_i^T and C_i^B are called the i^{th} *control clauses* and (o_i) the *payoff clause*. Point 2 creates a total of $3m$ clauses; they are ordered from $i = 1$

to $i = m$, and for each i they are ordered as written, meaning the control clauses are more important than the payoff clause (o_i), which in turn is more important than C^T_{i+1}, C^B_{i+1} and so on.

3. **Best-neighbor output of active circuit should be input of inactive circuit.** We write this as

$$f \to (z^T_j = x^B_j) \tag{3.6}$$
$$\neg f \to (z^B_j = x^T_j) \tag{3.7}$$
$$\tag{3.8}$$

   The ordering within this group is arbitrary.

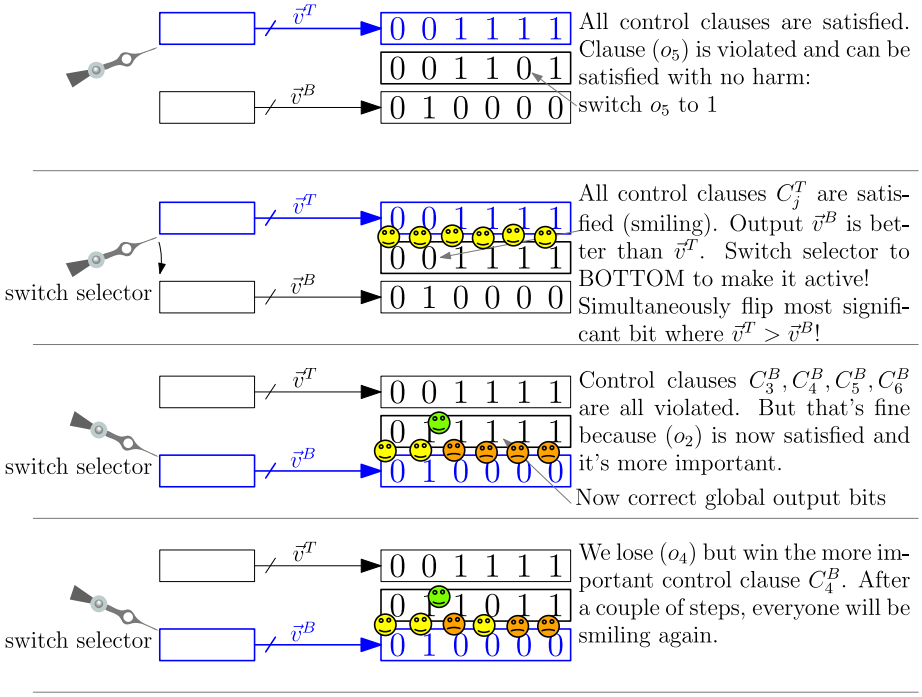
4. **All gates should be correctly evaluated,** whether in the active or inactive circuit.

$$g^T_j \leftrightarrow \text{NAND}(y^T_1, y^T_2) \tag{3.9}$$
$$g^B_j \leftrightarrow \text{NAND}(y^B_1, y^B_2) \tag{3.10}$$

Lemma 1. *Let α be an assignment to the variables of F. If α cannot be improved by a 1-Flip then all clauses of (1), (3), and (4) are satisfied.*

Lemma 2. *Suppose all clauses in (1), (3), and (4) are satisfied by α but the inactive circuit has a lexicographically larger output than the active circuit. Then there is a 1-variable or 2-variable flip that improves α.*

Proof. The following figure shows how we imagine typical flips involving the output register $\vec{o}$:



Without loss of generality, let us assume that TOP is active and BOTTOM is inactive, thus $\alpha(f) = 1$. Now, if $\vec{v}_T \neq \vec{o}$ then a single flip can improve α: Indeed, suppose they differ in position i. If $\alpha(v_i^T) = 1$ and $\alpha(o_i) = 0$ then control clause C_i^T is satisfied, but payoff clause (o_i) isn't, thus setting $\alpha(o_i) := 1$ satisfies it and violates no additional clause; the payoff increases. If, on the other hand $\alpha(v_i^T) = 0$ and $\alpha(o_i) = 1$ then the control clause C_i^T is violated; setting $\alpha(o_i) = 0$ satisfies it; it only additionally violates (o_i), which has lower weight, and thus the total payoff increases.

Thus, we can assume that $\vec{v}_T = \vec{o}$. Since $\vec{v}^B >_{\text{lex}} \vec{v}^T$ the two output vectors agree up to coordinate $i-1$ and differ at i with $v_i^B = 1$ and $v_i^T = 0$. We now switch two variables, setting $\alpha(f) = 0$ and setting $\alpha(o_i) = 1$. The control clauses for positions $1, \ldots, i$ are still satisfied; output clause (o_i) becomes satisfied; we might lose output clauses and control clauses among positions $i+1, \ldots, m$ but their total weight is less than that of (o_i).

Clauses of group (1) stay satisfied because BOTTOM is correctly evaluated by assumption; clauses of group (3) might become violated because the input of formerly active TOP is usually not the best-neighbor output of formerly inactive BOTTOM; still, the weight of the newly satisfied clause (o_i) is greater than the total weight of newly violated clauses, so the payoff increases. □

Now suppose that α cannot be improved by a 1-flip or 2-flip. Furthermore, suppose without loss of generality that $f = 1$ (i.e., TOP is active). Let $\vec{a} := \alpha(\vec{x}^T)$ and $\vec{b} := \alpha(\vec{z}^T)$. Since both circuits are correct (by group (1) and (4))

and the input of BOTTOM is $\vec{b}$ (by group (3)) the value output of TOP is $C'(\vec{a})$ and that of BOTTOM is $C'(\vec{b})$. Since no 2-flip can increase α, we conclude with Lemma 2 that $C'(\vec{b}) \leq C'(\vec{a})$. But $\vec{b}$ is the best neighbor of $\vec{a}$ (because TOP is correct), and thus $\vec{a}$ is indeed a local optimum of FLIP for circuit C'. This concludes the proof of Theorem 1.

We note that the above reduction is not *tight*. Since we are allowed to flip up to two variables, we can flip two inputs positions of the circuit, which is not an edge in the transition graph of the flip instance. Hence, we can potentially find a local optimum in fewer steps. We can however modify the problem by forbidding these type of flips. In the proof, we never have to flip two input variables at the same time, so the problem stays PLS-complete.

Theorem 7. *The 4-SAT/2-FLIP problem with forbidden flips, in which we are given an explicit set $L \subseteq \binom{V}{2}$ of pairs of variables which must not be flipped simultaneously, is* PLS*-complete with a tight* PLS*-reduction.*

Proof. We use the previous proof for the completeness and forbid flipping two input variables at the same time. These are only polynomially many pairs we forbid, so that the reduction is possible in polynomial time.

For the tightness proof, we use the set of all assignments as $\mathcal{R}$ which trivially contains all local optima. We can create an assignment that is mapped to the desired input of the flip instance in polynomial time by flipping the input variables of the original circuit one by one to the desired values. Finally, any path that only contains the endpoints of the path in $\mathcal{R}$ must be an edge. Mapping the endpoints of an edge to inputs for flip, we either gain the same input as we flip only axillary variables, or they differ in a single input position as we forbid flipping two input variables in the same step. In the latter case, the resulting solutions for the FLIP instance are neighbors. □

4 3-SAT/2-FLIP, Proof of Theorem 2

The reduction is almost identical to the one in the previous section, with one crucial modification: for each gate g_j we introduce not one but *three* variables $g_j, g_{j,1}, g_{j,2}$, and then two copies of each corresponding to TOP and BOTTOM: $g_j^T, g_{j,1}^T, g_{j,2}^T, g_j^B, g_{j,1}^B, g_{j,2}^B$. The idea is that $g_{j,1}$ and $g_{j,2}$ represent the two inputs of the gate and g_j the output. We introduce a "Group 0" of 3-clauses with the highest priority that state gates should be correct at all times:

$$g_j^T \leftrightarrow \text{NAND}(g_{j,1}, g_{j,2}) \ . \qquad \text{(Group 0)}$$

We modify the clauses in Group 1; we replace the 4-clauses constructed in (3.1) and (3.2), namely

$$\begin{aligned} f &\rightarrow (g_j^T \leftrightarrow \text{NAND}(y_1^T, y_2^T)) \\ \neg f &\rightarrow (g_j^B \leftrightarrow \text{NAND}(y_1^B, y_2^B)) \end{aligned} \qquad \text{(The Group 1 in the previous reduction)}$$

by clauses requiring that the input of gate g_j must agree with the output of the gate feeding into it; in other words, if the output of gate g_i feeds into the first output of gate g_j, we add the following 3-clauses to Group 1:

$$\begin{aligned} f &\to (g_i^T \leftrightarrow g_{j,1}^T) \\ \neg f &\to (g_i^B \leftrightarrow g_{j,1}^B) \end{aligned} \qquad \text{(The new Group 1)}$$

and similarly for the second input $g_{j,2}$; we call those clauses *wire clauses* because they check whether the value in the circuit are consistent along the wires. Also, if the first input of the gate g_j is an input x_i to the circuit, we produce the clause

$$\begin{aligned} f &\to (x_i^T \leftrightarrow g_{j,1}^T) \\ \neg f &\to (x_i^B \leftrightarrow g_{j,1}^B) \end{aligned} \qquad \text{(The new Group 1)}$$

These are also called wire clauses.

Finally, we replace the lowest-priority group, Group 4, which previously stated that all gates should be correct, including those in the inactive circuit, by wire clauses:

$$\begin{aligned} g_i^T &\leftrightarrow g_{j,1}^T \\ g_i^B &\leftrightarrow g_{j,1}^B \end{aligned} \qquad \text{(The new Group 4)}$$

Lemma 3. *Suppose α is a local maximum, i.e., cannot be improved by a 1-flip or 2-flip. Then all clauses in groups 0, 1, 3, and 4 are satisfied.*

Proof. A clause in Group 0 can always be satisfied by changing α in one position, namely the output of the incorrect gate. Now suppose a clause in the new group 1 is violated, for concreteness $(f \to (g_i^T \leftrightarrow g_{j,1}^T))$, so $f = 1$ (TOP is active) and $\alpha(g_i^T) \neq \alpha(g_{j,1}^T)$ (the wire from gate g_i^T to gate g_j^T is inconsistent); we can satisfy this by flipping $\alpha(g_{j,1}^T)$; this might violate gate g_j^T and thus a clause in Group 0, which has higher priority. However, we are allowed to flip up to *two* variables: we flip $g_{j,1}^T$ and, if need be, also g_j^T. This keeps Group 0 satisfied; it might violate a wire clause for a wire leaving gate j; however, this is in Group 1 and has a lower priority due to the topological ordering of the circuit.

A clause in Group 3 requires that the output of the active circuit agree with the input of the inactive one; suppose one is violated, for concreteness $(f \to (z_j^T \leftrightarrow x_j^B))$, so $f = 1$ (TOP is active). We will simply flip x_j^B. The wires going from x_j to the gates into which it feeds will now be inconsistent; however, the corresponding wire clauses in Group 1 will be satisfied because $f = 1$ (BOTTOM is inactive); some wire clauses of Group 4 will become violated, but that's fine because they have lower priority than the clause in Group 3.

Finally, if a clause in Group 4 is violated, it must be in the inactive circuit (all wire clauses for the active circuit are satisfied due to Group 1); for concreteness, BOTTOM is inactive and $(g_i^B \leftrightarrow g_{j,1}^B)$ is violated. We flip $g_{j,1}^B$ and, if need be,

the output variable g_j^B. Clauses in Group 0 are still satisfied because all gates are still correct; clauses in Group 1 are satisfied because we only fiddled in BOTTOM, which is inactive, so all its Group-1-clauses are satisfied anyway; Group 3 stays satisfied because outputs of the inactive circuit BOTTOM are not required to agree with the inputs of the active circuit TOP; in Group 4 further wire clauses involving g_{j^B} might become violated, but they have lower priority than the one we fixed. □

Lemma 2 still holds in the setting; if Group 0, 1, 3, and 4 are satisfied, and the output of the inactive circuit is lexicographically larger than the output of the active circuit, we can still flip f and one bit of $\vec{o}$ to improve α, simply because the clauses of Group 2 are identical to the ones in the reduction to 4-SAT/2-FLIP. This concludes the proof of Theorem 2.

Note that our reduction to 4-SAT/2-FLIP uses 2-flips very sparingly, only when we switch f (switch which circuit is active). In the reduction to 3-SAT/2-FLIP we use a 2-flip potentially every time we propagate a value through the circuit.

5 Reducing 2-flips to 1-flips - Proof of theorem 3

In section 3, we showed our general construction for lexicographical sat problems. So far, we have always used 2-flips as we needed to both flip f and the specific output. Flipping only the output variable violates the output constraints in some positions and there is no incentive to flip f as there is no improvement.

In order to break this barrier, we introduce new variables, the so-called shadow variables $s_{i,j}$. These can temporarily disable an output constraint for the variable o_j with the circuit i (they hide the constraint under a shadow) and allow afterward to flip the global output variable.

In order to not run into unwelcome local minima, we have to introduce at first constraints that prevent the shadow variables from being turned on prematurely. We do this by adding further constraints to the first group. This group contains again that all gates in the active circuit are correct. Additionally, we require that if a shadow variable is active, then the circuit must also be correct. Both are again arranged in a topological order of the circuit.

Active circuit must be correct. We have already seen those clauses above. They are the most important clauses and among them their importance follows the topological order of the circuit.

$$f \to (g_j^T \leftrightarrow \text{NAND}(y_1^T, y_2^T)) \tag{5.1}$$

$$\neg f \to (g_j^B \leftrightarrow \text{NAND}(y_1^B, y_2^B)) \tag{5.2}$$

Proper use of shadow variables. Shadow variables should be turned on only when the corresponding circuit is correctly evaluated and, in case it is the inactive circuit, it is reading its correct input, namely the best-neighbor output of the active circuit. Thus, for every output position $1 \leq i \leq m$ and every input position $1 \leq j \leq n$ we produce

$$(s_{B,i} \wedge f) \rightarrow (z_j^T = x_j^B) \tag{5.3}$$

$$(s_{T,i} \wedge \neg f) \rightarrow (z_j^B = x_j^T) \tag{5.4}$$

$$\tag{5.5}$$

saying that a "shadowed inactive" circuit must read the correct input. Second, we require a shadowed circuit to be correct in all gates. Thus, if a gate g_j has inputs y_1 and y_2, we produce

$$s_{T,i} \rightarrow (g_j^T \leftrightarrow \text{NAND}(y_1^T, y_2^T)) \tag{5.6}$$

$$s_{B,i} \rightarrow (g_j^B \leftrightarrow \text{NAND}(y_1^B, y_2^B)) \tag{5.7}$$

Never should shadow variables from both circuits be turned on. Our intention is that shadow variables are turned on for only the circuit that wants to transition from inactive to active, and are turned off shortly after this transition (although we do not encode this intention in clauses).

$$\forall i, j : \neg s_{i,B} \vee \neg s_{j,T} \tag{5.8}$$

Lastly, shadow variables must not be used in vain—meaning they should be used only when they indicate a position where the inactive output is better than the active one. Why is this necessary? When i is the most significant bit in which the active output is 1 but the inactive is 0 (so i is where we can improve things), we can be sure that all shadow variables of the positions $1, ..., i-1$ are off, which will be important later.

$$s_{i,B} \rightarrow v_i^B \tag{5.9}$$

$$s_{i,B} \rightarrow \neg v_i^T \tag{5.10}$$

$$s_{i,T} \rightarrow v_i^T \tag{5.11}$$

$$s_{i,T} \rightarrow \neg v_i^B \tag{5.12}$$

The order among the clauses in this group is arbitrary. All clauses so far (correctness clauses and proper use clauses) stay fulfilled along the canonical path and are never violated when only doing improving steps. On the other hand, we can always fulfill these by correcting the output of gates (first group) or by deactivating shadow variables (second group).

Circuit outputs, global output, and shadow variables are consistent. For each bit $1 \leq i \leq m$ of the output, we construct a block of clauses, with clauses in block i being more important than those in $i+1$ and so on. Within each block i, the order of importance is as listed now:

$$C_i^T := f \to (o_i \leq v_i^T) \vee s_{B,i} \tag{5.13}$$
$$C_i^B := \neg f \to (o_i \leq v_i^B) \vee s_{T,i} \tag{5.14}$$
$$o_i \tag{5.15}$$
$$(o_i \wedge s_{B,i}) \to \neg f \tag{5.16}$$
$$(o_i \wedge s_{T,i}) \to f \tag{5.17}$$

The first two clauses are again the control clauses, requiring in words, that the global output must be bit-wise at most the active output (as in the previous sections), but now we allow this to be violated if the corresponding shadow variable is turned on. The third clause simply states that we want a lexicographically large output (which is what this is all about, after all). Clauses (5.16) and (5.17) give us the incentive to switch active and inactive circuits after the shadow variable has been used to increase the output.

Best-neighbor output of active circuit should be input of inactive circuit. This group is identical to the one in Point 3.

$$f \to (z_j^T = x_j^B)$$
$$\neg f \to (z_j^B = x_j^T)$$

All gates should be correctly evaluated, also in the inactive circuit. This group is identical to the one in Point 4.

$$g_j^T \leftrightarrow \mathrm{NAND}(y_1^T, y_2^T)$$
$$g_j^B \leftrightarrow \mathrm{NAND}(y_1^B, y_2^B)$$

Use shadow variables at improvable positions. If the inactive output has a 1 at position i but the active has a 0 (and so the inactive as better, at least locally here), then we should tentatively put a shadow variable on position i to prepare a switch. The following clause gives us the (mild) incentive to turn the shadow variable on:

$$(f \wedge v_i^B \wedge \neg v_i^T) \to s_{B,i} \tag{5.18}$$
$$(\neg f \wedge v_i^T \wedge \neg v_i^B) \to s_{T,i} \tag{5.19}$$

The order among these clauses is arbitrary.

Turn off shadow variables.

$$\neg s_{B,i} \tag{5.20}$$

$$\neg s_{T,i} \tag{5.21}$$

The order among these clauses is arbitrary.

Lemma 4. *Any local optimum α of this instance is a local optimum for the flip instance.*

Proof. We assume without loss of generality that in α we have $f = 1$. In α all gates must be correct and the bottom circuit receives as an input the best neighbor of the input of the top circuit or else we could correct the gates as previously.

In α, all shadow variables of the type $s_{T,i}$ for all i must be false. If not, we can gain additional payoff from flipping $s_{T,i}$ from eq. (5.21) as the control clauses where the shadow variable is used are not active and eq. (5.19) is similarly not active due to the value of f.

If both circuits have the same output in α or the output of top is better in every position, then the input of the top circuit is a local optima for flip and the proof is complete as long as the global output is corrected. Else there must be a smallest index j such that $v_j^B = 1$ and $v_j^T = 0$ in α. We know that all shadow variables $s_{B,k}$ for $k < j$ must be false in α since j is the smallest index where this happens, so that $s_{B,k}$ being true in α would violate a hard constraint.

If $s_{B,j}$ and o_j are false in α, then we would gain additional payoff from activating it from eq. (5.19), since eq. (5.16) is not fulfilled. If o_j were true and $s_{B,j}$ is false, then flipping $s_{B,j}$ would reap additional payoff, since the control clause is then satisfied. If $s_{B,j}$ is true and o_j is false in α, then α is not a local optima as we can flip o_j, since the control clauses are active now.

Finally, if both $s_{B,j}$ and o_j are true in α, we again have two cases. Flipping f may bring additional payoff from eq. (5.16). If the bottom circuit has a better output than the top circuit, then this is possible and α is not a local optimum. This violates later control clauses, which are however less valuable with lexicographical weights. This corresponds to an improving step in the flip instance. Otherwise, there is an earlier position $k < j$, where the top circuit is better. Then flipping f would violate a higher control clause and hence we are in a local optima, which is also a local optima for the flip instance. □

Using similar arguments as before, this reduction is also tight.

6 Reduction to Abelian Polm—Proof of Theorem 4

We adapt a proof technique from [2] to reduce 4-SAT/1-FLIP to Abelian Polm. Let F be our 4-CNF formula, let $x_1, \ldots, x_n$ be its variables, and let $C_1, \ldots, C_m$ be its clauses.

We assume for simplicity that each clause in F has exactly four literals; the reduction for the general case is almost identical, but notation would become less readable.

The positions. For each 4-clause C and each $b \in \{0,1\}^4$ we construct a position C_b. Thus, we will have $16m$ positions.

The initial bit string. Our initial bit string $\vec{s}$ puts a 1 on $C_{\vec{0}}$ for each clause and a 0 on all other C_b. This corresponds to the assignment that sets all variables to 0.

The permutations. For a variable x and a clause C, let $\pi_{x,C}$ be the permutation defined as follows:

1. If x does not show up in C, then $\pi_{x,C}$ is the identity.
2. If x is the i-th variable of C (with $1 \leq i \leq 4$), then $\pi_{x,C}$ is the involution that swaps each position C_b with position $C_{b \oplus e_i}$.

So $\pi_{x,C}$ basically switches the value of x *in clause* C. We define π_x to be

$$\pi_x := \prod_{C \in F} \pi_{x,C}$$

so π_x switches the value of x in every clause. Our final set of generators is

$$G := \{\pi_x \mid x \in V\}$$

1. Let $\vec{v}$ be a string in the orbit, meaning $\vec{v} = \vec{s} \circ \sigma$ for some $\sigma \in \langle G \rangle$. Then for each clause C, exactly one of the sixteen positions C_b has a 1 in $\vec{v}$; this corresponds to some assignment of the variables in C.
2. Those assignments agree globally and define an assignment $\alpha_v : V \to \{0,1\}$.

For each clause C, there is one assignment $b \in \{0,1\}^4$ violating it. We denote the corresponding position C_b by C^*. We now define the ordering on the positions: first come all the "violating positions" $C_1^*, \ldots, C_m^*$, using the same priority order as the clauses (with C_1 having the highest priority). Then come the $15m$ non-violating positions, in some arbitrary order. We observe:

Proposition 1. *Let $1 \leq j \leq m$ and let $\vec{v}$ be in the orbit. Then v_j is 1 if and only if the corresponding assignment α_v violates C_j.*

Thus, if α_v can be improved by flipping some variable x, then $\vec{v} \circ \pi_x$ decreases the string lexicographically; Thus, a locally minimal string in the orbit corresponds to an assignment that cannot be improved by 1-flips i.e., a local optimum for 4-SAT/1-FLIP. This concludes the proof of Theorem 4.

The group generated by the π_x is actually isomorphic to $\big(\{0,1\}^V, \oplus\big)$. Additionally, this reduction is tight, as both transition graphs are isomorphic, since every application of a permutation is directly a flip of a variable. We can also reduce from 3-SAT/2-FLIP with forbidden flips by including $\pi_x \circ \pi_y$ in our list of generators only when the pair $\{x, y\}$ is not forbidden.

7 Reduction to Cyclic Group Polm—Proof of Theorem 5

We reduce from 4-SAT/1-FLIP with lexicographic payoffs, which is PLS-complete according to Theorem 2. Let F be a 3-CNF formula $x_1, \ldots, x_n$ its variables, and $C_1, \ldots, C_m$ its clauses from highest-priority to lowest-priority.

Consider the first n prime numbers $p_1 = 2, 3, 5, \ldots, p_n$ and set $N := p_1 \cdot p_2 \cdots p_n$. We let $\mathbb{Z}_k$ denote the group $(\mathbb{Z}/k\mathbb{Z}, +)$ and also, by a slight abuse of notation, the set $\{0, 1, \ldots, k-1\}$. By the Chinese remainder theorem, the function

$$\begin{aligned} f : \mathbb{Z}_N &\to \mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_n} \\ l &\mapsto (i \bmod p_1, \ldots, l \bmod p_n) \end{aligned}$$

is bijective, and both the function and its inverse can be computed in polynomial time.

The positions. Our positions will be arranged in *cycles*, one for each variable and one for each clause. In each cycle, the positions are numbered as $0, \ldots, l-1$, where l is the length of the cycle. The initial bit string s will place a 1 on each 0-position in each cycle and 0s elsewhere. Each permutation will rotate each cycle by a given amount; thus, each element in the orbit of s will have exactly one 1 on each cycle.

Variable positions. For every variable x_i we create a cycle of length p_i with positions $0, 1, \ldots, p_i - 1$. An element in the orbit has one 1 per cycle and thus assigns each variable x_i a value in $\{0, 1, \ldots, p_i - 1\}$. We call the positions $2, \ldots, p_i - 1$ bad-color-positions and make them very expensive (giving them the highest priority) to make sure that a local minimum only assigns values 0 and 1 to the variables. Among bad-color-positions, the order is arbitrary.

Clause positions. Let C be a clause of F and suppose it contains the variables x_i, x_j, x_k. We create a cycle of length $p_i p_j p_k$. Each $l \in \{0, \ldots, p_i p_j p_k - 1\}$ becomes, when taking remainders modulo p_i, p_j, p_k, a triple $(r_i, r_j, r_k) \in \mathbb{Z}_{p_i} \times \mathbb{Z}_{p_j} \times \mathbb{Z}_{p_k}$. Should this triple be in $\{0,1\}^3$ and should it be the unique truth assignment violating clause C, we call l the *clause-violating position* on that cycle. In our lexicographic ordering, we place the clause-violating positions directly after the bad-color-positions; among themselves, we order them as the

ordering of the clauses of F dictates: the clause-violating position of clause C_i has higher priority than that of C_{i+1}.

The permutations. Let π be the permutation that rotates each cycle by one (maps position t on the cycle to position $t+1$, and so on). For $l \in \mathbb{N}$ the permutation π^l rotates the variable cycle of x_i by l mod p_i and the clause cycle of a clause with variables x_i, x_j, x_k by l mod $p_i p_j p_k$. Thus, the bit string $v = s \circ \pi^l$ has a 1 on a bad-color position if v does not represent a Boolean assignment to the variables; if it does represent a Boolean assignment α, then v has a 1 on the clause-violating position of clause C if and only if α violates C.

For each variable x_i and each $r \in \mathbb{Z}_{p_i}$ let l be such that

$$\begin{aligned} l &\equiv r \mod p_i \\ l &\equiv 0 \mod p_j \qquad \text{(for all } j \neq i\text{)} \end{aligned}$$

and set $\pi_{x_i,r} := \pi^l$. The idea is that the permutation $\pi_{x_i,r}$ changes the value of x_i but not of any other variable. The generators for our Polm instances are now all $\pi_{x_i,r}$. This gives a polynomially long list of permutations $\pi^{e_1}, \pi^{e_2}, \ldots, \pi^{e_M}$, all powers of a single permutation π.

Lemma 5. *Let l be such that $s \circ \pi^l$ is a local minimum, i.e., $s \circ \pi^l \circ \pi^{e_i} \geq_{\text{lex}} s \circ \pi^l$ for all $1 \leq i \leq M$. Then $s \circ \pi^l$ encodes an assignment $\{x_1, \ldots, x_n\} \to \{0, 1\}$ that is locally optimal for F under 2-flips.*

Proof. Let $v := s \circ \pi^l$. Each variable cycle contains exactly one position where v has a 1. This assigns each x_i a value $b \in \{0, 1, \ldots, p_i - 1\}$. Now if $b \notin \{0, 1\}$ then v has a 1 at position b of this cycle—a bad-color-position. Applying π_{x_i,p_i-b} moves the 1 on this cycle to the 0-position but leaves all other variable cycles as they are. We have moved a 1 away from a bad-color-position and thus decreased the cost of v; hence v is not a local minimum.

Conversely, if v is a local minimum, then v has no 1 at any bad-color position and thus v encodes, as described above, a truth assignment α to the variables $x_1, \ldots, x_n$. Suppose α is not locally optimal for F but can be improved by flipping a variable x_i. This flip can be realized by the permutation $\pi_{x_i,r}$ where r is 1 if we want to flip x_i from 0 to 1 and $p_i - 1$ if we want to flip it from 1 to 0. Since flipping x_i is an improving step, it additionally satisfies some clause C and does not additionally violate any higher priority clause C'; thus, applying $\pi_{x_i,r}$ moves the 1 of v away from the clause-violating position of C but places no new 1 on any clause-violating position of any higher-priority clause C'. Also, it does not move a 1 onto any bad-color-position. Thus, v is not a local minimum, either.

We infer that every local minimum $v = s \circ \pi^l$ under the permutations $\pi^{e_1}, \ldots, \pi^{e_M}$ corresponds to a local maximum of the 4-SAT/1-FLIP instance F. This concludes the proof of Theorem 5. □

8 An application to game theory

Lexicographic k-SAT can also help to show hardness for results, whose cost functions are in general not lexicographic. A congestion game can be represented as a tuple $(N, E, (d_e)_{e\in E}, (S_i)_{i\in N})$, where N is a set of players and E is a set of resources. Each resource $e \in E$ has a delay $d_e : \mathbb{N} \to \mathbb{R}$ and each player i has a set of strategies $S_i \subseteq 2^E$.

Each player i plays a strategy s_i that are combined in a strategy profile $s = (s_1, s_2, \ldots, s_n)$. The cost of such a strategy profile s for a player i is the sum of the delays of each resource in s_i, i.e. $C_i(s) = \sum_{e\in s_i} d_e(|\{j \in N \mid e \in s_j\}|)$. The players act selfishly and switch their strategy, if they can reduce their cost. A pure Nash-equilibrium is a strategy profile s such that for any strategy profile s' that differs only in the strategy of player i, we have that $C_i(s) \leq C_i(s')$, i.e. there is no reason to deviate alone. If a strategy is not a Nash equilibrium, then at least one player has an improving move that decreases his costs.

A simple reduction from local NAE-3-SAT to finding pure Nash equilibria is shown in [7].

If the players are lazy and only change their strategy if their cost decreases a significant amount, then we ask for approximate Nash-equilibria. More formally, for a real $\alpha > 1$, an α-equilibrium is a strategy profile s such that for any strategy profile s' that differs only in the strategy of player i, we have that $C_i(s) \leq \alpha\, C_i(s')$. In [22], this was show to be PLS-complete for any computable $\alpha > 1$ using an involved reduction from FLIP.

It is not possible to reuse the general reduction from 3-SAT as there could be two clauses C and C' with the same high weight w and a variable x, such that $x \in C$ and $\neg x \in C'$. Suppose now that in the 3-SAT instance, we flip the value of x to improve a small weight $w' \ll w$. Then this might not be an α-improving move, as the change is only very little in comparison to the large weight w. These problems don't occur with lexicographic weights.

Proposition 2 ([22])**.** *For any computable $\alpha > 1$ and any congestion game G, finding an α-equilibrium is* PLS*-complete.*

Proof. We reduce from lexicographic 4-SAT/1-FLIP. Let

$$\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$$

be a weighted 4-SAT formula with the clauses ordered from low-weight to high-weight (for example, clause C_j having weight 2^j would do; as would any c^j for $c \geq 2$); The set of resources are the clauses of ϕ. We have for each variable x in ϕ a player x with the strategies $true = \{C \in \phi \mid \neg x \in C\}$ and $false = \{C \in \phi \mid x \in C\}$. Note that strategy profiles in the game are in a 1-to-1 correspondence with the Boolean assignments to the variables of ϕ. The delay d_C of clause/resource C_j is defined to be zero if fewer than four players use it, and $(1+\alpha)^j$ if four players do (note that a resource can never be used by more than four players). Apart from the value $1+\alpha$, this is roughly as in [7].

We must now show that a pure α-Nash equilibrium corresponds to a local minimum of the 4-SAT/1-FLIP problem. Contrapositively, we show that a

variable flip that improves the weight of satisfied clauses corresponds to a player that will change her strategy. Consider an improving flip of variable x (from False to True, without loss of generality). Let j be the highest position such that C_j used to be unsatisfied before the flip and is now satisfied. The delay for player x before the flip is therefore at least

$$(1+\alpha)^j \ . \tag{8.1}$$

Because our costs are lexicographic, the status of all higher clauses $C_{j+1}, \ldots, C_m$ does not change; in particular, all those that contain $\bar{x}$ are satisfied—before and after the flip. Thus, the delay for player x after the flip is at most

$$\sum_{i \le j-1} (1+\alpha)^i = \frac{(1+\alpha)^j}{\alpha} \ . \tag{8.2}$$

Thus, changing the strategy reduces the delay of player x at least by a factor of α and player x would change her strategy. □

In fact, we can even replace the step functions in the reduction by exponential functions: if k players choose resource C_j, we define the delay to be

$$(1+\alpha)^{km+j} \ . \tag{8.3}$$

The term in (8.1) would now be $(1+\alpha)^{4m+j}$ and the one in (8.1) would be $\sum_{i \le 4m+j-1}(1+\alpha)^i$ and thus the two terms would still be a factor of α apart. A technical issue arises when some of our clauses have fewer than four literals; in this case, we either have to multiply the cost function in (8.3) by $(1+\alpha)^{m(4-|C|)}$; alternatively, we could create up to three dummy players whose only strategy is to sit in short clauses and occupy those resources.

9 Conclusion and Open Problems

We have shown that commutativity does not help when solving the orbit minimization problem, strengthening the result from [21].

Open problem 1. *What is the complexity of* Polm *when the number of permutations is a constant k?*

If the reader is pessimistic and suspects hardness, the following problem might be easier to solve (in the negative):

Open problem 2. *When the number of permutations is constant, does the standard algorithm terminate after a polynomial number of steps? Maybe if the permutations commute? Maybe if $k = 2$ and they commute?*

We currently do not know this, even when $k = 2$ and the two permutations commute.

With the lexicographic 3-SAT/2-FLIP and 4-SAT/1-FLIP problems, we gave additional PLS-complete problems that use lexicographic weights, which should allow easier reductions than reducing from FLIP directly.

Open problem 3. *What is the complexity of* 3*-SAT/1-FLIP?*

Our current reduction does not work for 3-SAT/1-FLIP because we need 3 variables per gate and a switching variable for the highest clauses. Hence, a new approach is needed here. As computing a global optimum is also NP-hard for 3-SAT with lexicographic weights (just reduce from the plain old decision problem of 3-SAT), one might again suspect hardness. On the other hand, finding a globally optimal max cut on cubic graphs is NP-hard, while the local optima can be found in polynomial time[16, 19]. While 2-SAT is tractable for lexicographic weights, it is harder than the unweighted case which is in NL, since our algorithm needs to remember a set of clauses that can be fulfilled so far. This is unlikely to improve:

Proposition 3. 2*-SAT/1-FLIP is* FP*-complete.*

Proof. We reduce from the Circuit-Evaluation problem, where every gate is a NAND-gate, and introduce a variable for each gate, input and output. The clauses with the highest weight are 1-clauses that fix the input. For each gate g with inputs y_1 and y_2, we add the following 3 clauses:

1. $\neg y_1 \to g$
2. $\neg y_2 \to g$
3. $\neg g$

The last clause has the lowest weight in that group, and the clause groups are ordered in the topological order of the circuit. The inner order of the gate clauses lead to the simulation of a NAND-gate. □

Although Max-2-SAT is easy under lexicographic costs, our greedy algorithm differs greatly from the standard algorithm:

Open problem 4. *Does the standard algorithm on 2-SAT/1-FLIP with lexicographic costs terminate after a polynomial number of steps?*

Another interesting PLS-problem with lexicographical weights is the travelling salesman problem. The usual approach does not work similarly as to 4-SAT, since we would have to check whether a Hamiltonian cycle still exists without these edges. Recently, Heimann, Hoang and Hougardy investigated the complexity of this problem for general costs[11]. Their reduction however starts from Max-Cut, which is easy in the lexicographical setting. We believe that a reduction similar to the NP-completeness proof for Hamiltonian cycle would work, if the number of occurrences of each variable in the k-SAT instance is bounded by some constant.

We showed a way to solve lexicographic problems like 2-SAT in polynomial time. Is there an alternative algorithm that solves lexicographic problems? We also don't know if there still exist exponentially long improving sequences, even if the optimum can be found in polynomial time.

Lexicographic k-SAT was useful to show hardness of finding approximate Nash equilibria for congestion games. An important subclass are (weighted or unweighted) congestion games where the delay at a resource is given by a polynomial in the number of players using this resource (resp. their total weight). We were not able to extend our results to this case. Note that in polynomial weighted congestion games finding $d^{\mathcal{O}(d)}$-equilibria is in P[4, 10], while equilibria are known to exists only for $\alpha \geq d$ where d is the maximum degree of the polynomial functions[3]. For $\alpha \in \mathcal{O}(\frac{d}{\log d})$ deciding if an equilibria exists is NP-hard[5].

References

[1] Heiner Ackermann, Heiko Röglin, and Berthold Vöcking. "On the Impact of Combinatorial Structure on Congestion Games". In: *47th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2006, Berkeley, California, USA, October 21-24, 2006, Proceedings*. IEEE Computer Society, 2006, pp. 613–622. DOI: 10.1109/FOCS.2006.55. URL: https://doi.org/10.1109/FOCS.2006.55.

[2] Christoph Buchheim and Michael Jünger. "Linear optimization over permutation groups". In: *Discret. Optim.* 2.4 (2005), pp. 308–319. DOI: 10.1016/J.DISOPT.2005.08.005. URL: https://doi.org/10.1016/j.disopt.2005.08.005.

[3] Ioannis Caragiannis and Angelo Fanelli. "On approximate pure Nash equilibria in weighted congestion games with polynomial latencies". In: *J. Comput. Syst. Sci.* 117 (2021), pp. 40–48. DOI: 10.1016/J.JCSS.2020.10.007. URL: https://doi.org/10.1016/j.jcss.2020.10.007.

[4] Ioannis Caragiannis, Angelo Fanelli, Nick Gravin, and Alexander Skopalik. "Approximate Pure Nash Equilibria in Weighted Congestion Games: Existence, Efficient Computation, and Structure". In: *ACM Trans. Economics and Comput.* 3.1 (2015), 2:1–2:32. DOI: 10.1145/2614687. URL: https://doi.org/10.1145/2614687.

[5] George Christodoulou, Martin Gairing, Yiannis Giannakopoulos, Diogo Poças, and Clara Waldmann. "Existence and Complexity of Approximate Equilibria in Weighted Congestion Games". In: *Math. Oper. Res.* 48.1 (2023), pp. 583–602. DOI: 10.1287/MOOR.2022.1272. URL: https://doi.org/10.1287/moor.2022.1272.

[6] Dominic Dumrauf and Burkhard Monien. "On the *PLS*-complexity of maximum constraint assignment". In: *Theor. Comput. Sci.* 469 (2013), pp. 24–52. DOI: 10.1016/J.TCS.2012.10.044. URL: https://doi.org/10.1016/j.tcs.2012.10.044.

[7] Alex Fabrikant, Christos H. Papadimitriou, and Kunal Talwar. "The complexity of pure Nash equilibria". In: *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*. Ed. by László Babai. ACM, 2004, pp. 604–612. DOI: 10.1145/1007352.1007445. URL: https://doi.org/10.1145/1007352.1007445.

[8] Merrick L. Furst, John E. Hopcroft, and Eugene M. Luks. "Polynomial-Time Algorithms for Permutation Groups". In: *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980*. IEEE Computer Society, 1980, pp. 36–41. DOI: 10.1109/SFCS.1980.34. URL: https://doi.org/10.1109/SFCS.1980.34.

[9] Yiannis Giannakopoulos, Alexander Grosz, and Themistoklis Melissourgos. "On the Smoothed Complexity of Combinatorial Local Search". In: *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*. Ed. by Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson. Vol. 297. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, 72:1–72:19. DOI: 10.4230/LIPICS.ICALP.2024.72. URL: https://doi.org/10.4230/LIPIcs.ICALP.2024.72.

[10] Yiannis Giannakopoulos, Georgy Noarov, and Andreas S. Schulz. "Computing Approximate Equilibria in Weighted Congestion Games via Best-Responses". In: *Math. Oper. Res.* 47.1 (2022), pp. 643–664. DOI: 10.1287/MOOR.2021.1144. URL: https://doi.org/10.1287/moor.2021.1144.

[11] Sophia Heimann, Hung P. Hoang, and Stefan Hougardy. "The k-Opt Algorithm for the Traveling Salesman Problem Has Exponential Running Time for k $\geq$ 5". In: *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*. Ed. by Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson. Vol. 297. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, 84:1–84:18. DOI: 10.4230/LIPICS.ICALP.2024.84. URL: https://doi.org/10.4230/LIPIcs.ICALP.2024.84.

[12] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. "How Easy is Local Search?" In: *J. Comput. Syst. Sci.* 37.1 (1988), pp. 79–100. DOI: 10.1016/0022-0000(88)90046-3. URL: https://doi.org/10.1016/0022-0000(88)90046-3.

[13] Leszek Aleksander Kolodziejczyk and Neil Thapen. "The Strength of the Dominance Rule". In: *27th International Conference on Theory and Applications of Satisfiability Testing, SAT 2024, August 21-24, 2024, Pune, India*. Ed. by Supratik Chakraborty and Jie-Hong Roland Jiang. Vol. 305. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, 20:1–20:22. DOI: 10.4230/LIPICS.SAT.2024.20. URL: https://doi.org/10.4230/LIPIcs.SAT.2024.20.

[14] Mark W. Krentel. "On Finding Locally Optimal Solutions". In: *Proceedings: Fourth Annual Structure in Complexity Theory Conference, University of Oregon, Eugene, Oregon, USA, June 19-22, 1989*. IEEE Computer Society, 1989, pp. 132–137. DOI: 10.1109/SCT.1989.41819. URL: https://doi.org/10.1109/SCT.1989.41819.

[15] Mark W. Krentel. "Structure in Locally Optimal Solutions (Extended Abstract)". In: *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*. IEEE Computer Society, 1989, pp. 216–221. DOI: 10.1109/SFCS.1989.63481. URL: https://doi.org/10.1109/SFCS.1989.63481.

[16] Martin Loebl. "Efficient Maximal Cubic Graph Cuts (Extended Abstract)". In: *Automata, Languages and Programming, 18th International Colloquium, ICALP91, Madrid, Spain, July 8-12, 1991, Proceedings*. Ed. by Javier Leach Albert, Burkhard Monien, and Mario Rodríguez-Artalejo. Vol. 510. Lecture Notes in Computer Science. Springer, 1991, pp. 351–362. DOI: 10.1007/3-540-54233-7_147. URL: https://doi.org/10.1007/3-540-54233-7%5C_147.

[17] Nimrod Megiddo and Christos H. Papadimitriou. "On Total Functions, Existence Theorems and Computational Complexity". In: *Theor. Comput. Sci.* 81.2 (1991), pp. 317–324. DOI: 10.1016/0304-3975(91)90200-L. URL: https://doi.org/10.1016/0304-3975(91)90200-L.

[18] Christos H. Papadimitriou. "On the Complexity of the Parity Argument and Other Inefficient Proofs of Existence". In: *J. Comput. Syst. Sci.* 48.3 (1994), pp. 498–532. DOI: 10.1016/S0022-0000(05)80063-7. URL: https://doi.org/10.1016/S0022-0000(05)80063-7.

[19] Svatopluk Poljak. "Integer Linear Programs and Local Search for Max-Cut". In: *SIAM J. Comput.* 24.4 (1995), pp. 822–839. DOI: 10.1137/S0097539793245350. URL: https://doi.org/10.1137/S0097539793245350.

[20] Alejandro A. Schäffer and Mihalis Yannakakis. "Simple Local Search Problems That are Hard to Solve". In: *SIAM J. Comput.* 20.1 (1991), pp. 56–87. DOI: 10.1137/0220004. URL: https://doi.org/10.1137/0220004.

[21] Dominik Scheder and Johannes Tantow. "PLS-Completeness of String Permutations". In: *33rd Annual European Symposium on Algorithms, ESA 2025, September 15-17, 2025, Warsaw, Poland*. Ed. by Anne Benoit, Haim Kaplan, Sebastian Wild, and Grzegorz Herman. Vol. 351. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025, 56:1–56:14. DOI: 10.4230/LIPICS.ESA.2025.56. URL: https://doi.org/10.4230/LIPIcs.ESA.2025.56.

[22] Alexander Skopalik and Berthold Vöcking. "Inapproximability of pure nash equilibria". In: *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*. Ed. by Cynthia Dwork. ACM, 2008, pp. 355–364. DOI: 10.1145/1374376.1374428. URL: https://doi.org/10.1145/1374376.1374428.