Smoothed Complexity of SWAP in Local Graph Partitioning*

Xi Chen†
Columbia University
xichen@cs.columbia.edu

Chenghao Guo‡
MIT
chenghao@mit.edu

Emmanouil V. Vlatakis-Gkaragkounis§
University California, Berkeley
emvlatakis@berkeley.edu

Mihalis Yannakakis¶
Columbia University
mihalis@cs.columbia.edu

Abstract

We give the first quasipolynomial upper bound $\phi n^{\text{polylog}(n)}$ for the smoothed complexity of the SWAP algorithm for local Graph Partitioning (also known as Bisection Width) under the full perturbation model, where n is the number of nodes in the graph and ϕ is a parameter that measures the magnitude of perturbations applied on its edge weights. More generally, we show that the same quasipolynomial upper bound holds for the smoothed complexity of the 2-FLIP algorithm for any binary Maximum Constraint Satisfaction Problem, including local Max-Cut, for which similar bounds were only known for 1-FLIP. Our results are based on an analysis of a new notion of useful cycles in the multigraph formed by long sequences of double flips, showing that it is unlikely for every double flip in a long sequence to incur a positive but small improvement in the cut weight.

1 Introduction

Local search has been a powerful machinery for a plethora of problems in combinatorial optimization, from the classical Simplex algorithm for linear programming to the gradient descent method for modern machine learning problems, to effective heuristics (e.g. Kernighan-Lin) for basic combinatorial problems such as the Traveling Salesman Problem and Graph Partitioning. A local search algorithm begins with an initial candidate solution and then follows a path by iteratively moving to a better neighboring solution until a local optimum in its neighborhood is reached. The quality of the obtained solutions depends of course on how rich is the neighborhood structure that is explored by the algorithm. Local search is a popular approach to optimization because of the general applicability of the method and the fact that the algorithms typically run fast in practice. In contrast to their empirical fast convergence, however, many local search algorithms have exponential running time in the worst case due to delicate pathological instances that one may never expect to encounter in practice. To bridge this striking discrepancy, Spielman and Teng [26] proposed the framework of *smoothed analysis*, a hybrid of the classical worst-case and average-case analyses. They used it to provide rigorous justifications for the empirical performance of the Simplex algorithm by showing its smoothed complexity to be polynomial. Since then, the smoothed analysis of algorithms and problems from combinatorial optimization [5, 14, 23], among many other research areas such as numerical methods [12, 6, 23, 16], machine learning [8, 2, 3, 25] and algorithmic game theory [9, 10, 4, 27], has been studied extensively .

In this paper we study the smoothed complexity of local search algorithms for the classical problem of *Graph Partitioning* (also known as *Bisection Width* in the literature). In the problem we are given edge weights

*The full version of the paper can be accessed at https://arxiv.org/abs/2305.15804

†Supported by NSF IIS-1838154, CCF-2106429 and CCF-2107187.
‡Supported by NSF TRIPODS program award DMS-2022448 and by NSF Career Award CCF-1940205, CCF- 2131115.
§Supported by Postdoctoral FODSI Simons-Fellowship.
¶Supported by NSF CCF-2107187 and CCF-2212233, and by Amazon.

$X = (X_e : e \in E_{2n})$ of a complete graph $K_{2n} = (V_{2n}, E_{2n})$ with $X_e \in [-1, 1]$, and the goal is to find a *balanced* partition (U, V) of V_{2n} into two equal-size subsets U and V to minimize the weight of the corresponding cut (i.e., the sum of weights of edges with one node in U and the other node in V). Graph Partitioning has been studied extensively, especially in practice. It forms the basis of divide and conquer algorithms and is used in various application domains, for example in laying out circuits in VLSI. It has also served as a test bed for algorithmic ideas [19].

Given its NP-completeness [18], heuristics have been developed to solve Graph Partitioning in practice. A commonly used approach is based on local search: starting with an initial balanced partition, local improvements on the cut are made iteratively until a balanced partition that mimimizes the cut within its *neighborhood* is reached. The simplest neighborhood is the SWAP neighborhood, where two balanced partitions are neighbors if one can be obtained from the other by swapping two nodes, one from each part. A locally optimal solution under the SWAP neighborhood can be found naturally by the SWAP algorithm, which keeps swapping two nodes as long as the swap improves the cut. A more sophisticated neighborhood structure, which yields much better locally optimal solutions in practice, is that of the Kernighan-Lin (KL) algorithm which performs in each move a sequence of swaps [21].

These local search algorithms for Graph Partitioning typically converge fast in practice. (For a thorough experimental analysis of their performance, and comparison with simulated annealing, regarding both the quality of solutions and running time, see [19].) In contrast, it is known that their worst-case complexity is exponential. (Finding a locally optimal solution under the sophisticated Kernighan-Lin neighborhood, and even under the simpler SWAP neighborhood is complete in PLS [20, 24]. The hardness reductions give instances on which these algorithms take exponential time to converge.) *This significant gap in our understanding motivates us to work on the smoothed complexity of the SWAP algorithm for Graph Partitioning in this paper.*

We work on the full perturbation model, under which edge weights are drawn independently from a collection of distributions $\mathcal{X} = (\mathcal{X}_e : e \in E_{2n})$, where each $\mathcal{X}_e$ is supported on $[-1, 1]$ and has its density function bounded from above by a parameter $\phi > 0$. Our goal is to understand the expected number of steps the SWAP algorithm takes, as a function of n and ϕ, against any edge weight distributions $\mathcal{X}$. [1] Note that the SWAP algorithm, similar to the Simplex algorithm, is a family of algorithms since one can implement it using different pivoting rules, deterministic or randomized, to pick the next pair of nodes to swap when more than one pairs can improve the cut. We would like to establish upper bounds that hold for every implementation of the SWAP algorithm.

1.1 Related work: Smoothed analysis of 1-FLIP for Max-Cut There has not been any previous analysis on SWAP under the smoothed setting, as far as we are aware. In contrast, much progress has been made on the smoothed analysis of the *1-FLIP algorithm for Max-Cut* [13, 15, 1, 7, 11]. The major challenge for the analysis of SWAP, as we discuss in more details in Section 1.3, is to overcome substantial new obstacles posed by the richer neighborhood structure of SWAP, which are not present in the simpler 1-*change neighborhood* behind 1-FLIP.

Recall in Max-Cut, we are given edge weights $X = (X_e : e \in E_n)$ of a complete graph $K_n = (V_n, E_n)$ with $X_e \in [-1, 1]$, and the goal is to find a (*not necessarily balanced*) partition of V_n to maximize the cut.[2] The simplest neighborhood structure for local search on Max-Cut is the so-called 1-*change neighborhood*, where two partitions are neighbors if one can be obtained from the other by moving a single node to the other side. The 1-FLIP algorithm finds such a locally optimal solution by repeatedly moving nodes, one in each round, as long as each move strictly improves the cut. Similar to SWAP, the worst-case complexity of 1-FLIP is exponential, and the problem of finding a locally optimal solution of Max-Cut under the 1-change neighborhood is PLS-complete [24]. For the structured perturbation model, where a graph G (not necessarily a complete graph) is given and only weights of edges in G are perturbed, [15] showed that the expected number of steps 1-FLIP

[1] Note that any upper bound under the full perturbation model applies to the alternative, simpler model where an adversary commits to a weight w_e for each edge and then all edge weights are perturbed independently by a random noise $\mathcal{Z}_e$ (for example, drawn uniformly from a small interval), i.e. the weights are $\mathcal{X}_e = w_e + \mathcal{Z}_e$. The parameter ϕ in the full perturbation model is a bound on the pdf of the perturbations $\mathcal{Z}_e$.

[2] Since we allow weights in $[-1, 1]$, maximizing the cut is the same as minimizing the cut after negating all edge weights. Hence the only difference of Max-Cut, from Graph Partitioning, is that the partition does not have to be balanced.

takes to terminate is at most $\phi n^{\log n}$. Subsequently, the bound was improved by [1] to $\phi \cdot \text{poly}(n)$ for the full perturbation model, with further improvements in [7] on the polynomial function of n. The upper bound of [15] for the structured model was recently improved to $\phi n^{\sqrt{\log n}}$ in [11].

1.2 Our Contributions We present the first smoothed analysis of the SWAP algorithm for Graph Partitioning. Our main result for SWAP is a quasipolynomial upper bound on its running time:

THEOREM 1.1. *Let $\mathcal{X} = (\mathcal{X}_e : e \in E_{2n})$ be distributions of edge weights such that each $\mathcal{X}_e$ is supported on $[-1, 1]$ and has its density function bounded from above by a parameter $\phi > 0$. Then with probability at least $1 - o_n(1)$ over the draw of edge weights $X \sim \mathcal{X}$, every implementation of SWAP terminates within $\phi n^{O(\log^{10} n)}$ steps.*

The proof of Theorem 1.1 uses techniques we develop for a more challenging problem: the smoothed analysis of the 2-*FLIP* algorithm for Max-Cut. Starting with any initial partition, 2-FLIP in each round can move either one node (like 1-FLIP) or two nodes (not necessarily in different parts) as long as the cut is improved. Given that 2-FLIP generalizes 1-FLIP, its worst-case complexity is also exponential (every improving sequence of moves for 1-FLIP is also an improving sequence for 2-FLIP), and the problem of finding a locally optimal solution of Max-Cut under the 2-change neighborhood is also PLS-complete (every locally optimal partition for 2-FLIP is also locally optimal for 1-FLIP). If we restrict the algorithm to only use double flips (i.e., move exactly two nodes) in every move, then we call this variant *pure* 2-FLIP. Feasible moves in SWAP are clearly feasible in pure 2-FLIP as well but not vice versa. Thus, an improving sequence of SWAP in the Graph Partitioning problem is also an improving sequence of pure 2-FLIP in the Max-Cut problem on the same instance but not vice versa, which makes the smoothed analysis of 2-FLIP even more challenging.

Similar to our result for SWAP, we do not make any assumption on the pivoting rule used by 2-FLIP, except that the algorithm never moves a pair of nodes u and v when moving just one of them alone (i.e. either just moving u or just moving v) would yield an even better cut. Our main result on 2-FLIP is a similar quasipolynomial upper bound. The same result holds also for any implementation of the pure 2-FLIP algorithm that performs only 2-flips. This is the first smoothed analysis of 2-FLIP:

THEOREM 1.2. *Let $\mathcal{X} = (\mathcal{X}_e : e \in E_n)$ be distributions of edge weights such that each $\mathcal{X}_e$ is supported on $[-1, 1]$ and has its density function bounded from above by a parameter $\phi > 0$. Then with probability at least $1 - o_n(1)$ over the draw of $X \sim \mathcal{X}$, every implementation of the 2-FLIP algorithm terminates within $\phi n^{O(\log^{10} n)}$ steps.*

A more general class of problems that is related to Max-Cut is the class of *Maximum Binary Constraint Satisfaction Problems* (MAX 2-CSP). In a general Max-2CSP, the input consists of a set of Boolean variables and a set of weighted binary constraints over the variables; the problem is to find an assignment to the variables that maximizes the total weight of satisfied constraints. Max-Cut is the special case when every constraint is a $\neq$ (XOR) constraint. Other examples are Max 2SAT and Max Directed Cut (i.e., the Max Cut problem on weighted directed graphs). More generally, in a *Binary Function Optimization Problem* (BFOP), instead of binary constraints the input has a set of weighted binary functions on the variables, and the objective is to find an assignment that maximizes the sum of the weights of the functions (see Section 7 for the formal definitions). It was shown in [11] that the results for 1-FLIP for Max-Cut generalize to all Max 2-CSP and BFOP problems. We prove that this is the case also with 2-FLIP.

We say an instance of Max 2-CSP or BFOP is *complete* if it includes a constraint or function for every pair of variables.

THEOREM 1.3. *Let I be an arbitrary complete instance of MAX 2-CSP (or BFOP) with n variables and m constraints (functions) with independent random weights in $[-1, 1]$ with density at most $\phi > 0$. With probability at least $1 - o_n(1)$ over the draw of weights, every implementation of 2-FLIP terminates within $m\phi n^{O(\log^{10} n)}$ steps.*

For all the aforementioned problems, by controlling the tail-bound of the failure probability, we can strengthen our analysis to derive the same bound for the expected number of steps needed to terminate as in the standard smoothed analysis prototype (See Corollary 3.2).

1.3 Our Approach We give an overview of our approach, focusing on 2-FLIP for Max-Cut (Theorem 1.2). Many details are omitted to help the reader get an overview of some of the key ideas and the structure of the proof.

1.3.1 Analysis of 1-FLIP: Rank of Arcs First let's briefly review the approach of [15] on the simpler 1-FLIP. As the weight of any cut lies in $[-n^2, n^2]$, for an execution of 1-FLIP to be long, most of its moves must induce small improvements in the cut weight, say in $(0, \epsilon]$ for some small $\epsilon > 0$. Now the improvement in cut weight of any single move is a linear combination of edge weights X_e with coefficients in $\{-1, 0, 1\}$, which indicate whether each edge is removed, unchanged or added in the cut as a result of the move. As a result, the probability of a single move having its improvement lie in $(0, \epsilon]$ is small ($\leq \phi\epsilon$) over $X_e \sim \mathcal{X}_e$. If improvements of different moves were uncorrelated, the probability that a sequence improves the cut by no more than ϵ would go down exponentially with the length of the sequence ($\leq (\phi\epsilon)^\ell$ for any sequence of length ℓ).

Of course, improvements of different moves are correlated. But the same effect holds if they are *linearly independent* in the following sense. Given a sequence S of moves, the *improvement vector* of each move is defined as the $\{-1, 0, 1\}$-vector indexed by edges of the graph as coefficients of each X_e in the improvement. Most work along this line of research is based on the following fact (Corollary 2.1): Let $\mathsf{rank}(S)$ be the rank of improvement vectors of moves in S. Then the sequence has overall improvement at most ϵ with probability at most $(\phi\epsilon)^{\mathsf{rank}(S)}$. If $\mathsf{rank}(S) = \Omega(n)$ for all sequences S of length $\Theta(n)$, then by a union bound, with probability $1 - 2^n n^{\Theta(n)} (\phi\epsilon)^{\Omega(n)}$, every sequence of length $\Theta(n)$ improves the cut by at least ϵ. (We pay 2^n in the union bound to fix the initial configuration of nodes at the beginning of S.) On the one hand, when $\epsilon := (\phi \cdot \mathrm{poly}(n))^{-1}$, the probability above is $1 - o_n(1)$; on the other hand, when the event happens, 1-FLIP must terminate within $\Theta(n) \cdot (2n^2/\epsilon) = \phi \cdot \mathrm{poly}(n)$ steps as desired.

However, $\mathsf{rank}(S)$ of an $\Theta(n)$-move sequence can be much smaller than n.[3] A key idea of [15] is to work with *arcs* instead of individual moves. An arc consists of two adjacent moves of the same node.[4] The improvement vector of an arc is defined to be the sum of improvement vectors of its two moves. A simple but crucial observation is that edges that involve *inactive* nodes (i.e. nodes that never move in S) are cancelled in the improvement vector of an arc. [15] proved a combinatorial lemma to show that every sequence S of length $\Theta(n)$ must have a substring T such that the rank of improvement vectors of arcs in T, denoted by $\mathsf{rank}_{\mathsf{arcs}}(T)$, is at least $\Omega(\mathsf{len}(T)/\log n)$, where $\mathsf{len}(T)$ denotes the length of T (i.e., the number of moves in it). By a union bound, with probability at least

$$1 - \sum_{\ell \leq \Theta(n)} 2^\ell n^\ell (\phi\epsilon)^{\Omega(\ell/\log n)},$$

all sequences of length $\Theta(n)$ improve the cut weight by at least ϵ. (Note that the ℓ above is the length of T, and we only pay 2^ℓ in the union bound because we do not need to know the initial configuration of inactive nodes since we work with arcs, and the number of active nodes is trivially at most the length of T.) Setting $\epsilon := (\phi n^{\log n})^{-1}$ finishes the proof of the $\phi \cdot n^{\log n}$ upper bound of [15] for 1-FLIP.

1.3.2 Inadequacy of Arcs in 2-FLIP Sequences Now let's return to our analysis of 2-FLIP and for simplicity, we focus on the simpler case of *pure* 2-FLIP, where every round the algorithm makes a so-called 2-*move* that flips two nodes $\{u, v\}$. To avoid the union bound over inactive nodes, a natural generalization of the notion of arcs is to work with two adjacent 2-moves where exactly the same pair of nodes $\{u, v\}$ are flipped. This, however, fails for the following two reasons: (1) Such arcs may occur very sparsely and may not even exist unless the sequence has length $\Omega(n^2)$; (2) Inactive nodes do not necessarily cancel out in the sum of improvement vectors of the two moves; indeed this happens only when both u and v get flipped an even number of times between the two moves of $\{u, v\}$, which is in general not the case. For example, consider the sequence $\{u, v\}, \{w, x\}, \{v, y\}, \{s, t\}, \{u, v\}$. Edges that involve inactive nodes do not get cancelled out if we sum up improvement vectors of the first and the last moves, since v gets flipped once between them.

[3] For example, any sequence in which only $n^{0.1}$ distinct nodes moved has $\mathsf{rank}(S) \leq n^{0.2}$.

[4] For example, in $v_1, v_2, v_3, v_4, v_1, v_5, v_2$, the first and fifth moves form an arc, and the second and last moves form an arc.

1.3.3 Cancellation of Inactive Nodes with Useful Cycles In general (as the simple example above suggests), cancelling out edges that involve inactive nodes under the 2-FLIP setting is much more challenging compared to that of 1-FLIP. To overcome this new obstacle, we associate an *auxiliary graph* to a sequence of 2-moves, and introduce a notion of *useful cycles*. Given a sequence S of 2-moves, we define a (multi)graph H which has the same set V_n of nodes as the original graph and whose edges correspond to the 2-moves of the sequence: an undirected edge between u and v is added to H for each 2-move $\{u, v\}$ in S. A *cycle* of S is a set of 2-moves in S such that their corresponding edges form a cycle in H. We identify a subset of cycles as *useful*: these are the cycles that have the property that there is a $\{\pm 1\}$-linear combination of the improvement vectors of the moves of the cycle that cancels out all inactive nodes. We associate this linear combination to each useful cycle as its improvement vector. Given a sequence S of 2-moves we write $\mathsf{rank}_{\mathsf{cycles}}(S)$ to denote the rank of improvement vectors of useful cycles in S.

With the notion of useful cycles in place, the goal now is to show that every sequence of length $\tilde{\Theta}(n)$ must have a substring T that satisfies $\mathsf{rank}_{\mathsf{cycles}}(T) \geq \mathsf{len}(T)/\mathrm{polylog}(n)$. This is the goal of our main technical lemma, Lemma 3.2. Assuming Lemma 3.2, the rest of proof applies a similar union bound to show that with probability at least

$$1 - \sum_{\ell \leq \tilde{\Theta}(n)} 2^{\ell} n^{\ell} \left(\phi\epsilon\right)^{\Omega(\ell/\mathrm{polylog}(n))},$$

every sequence of length $\tilde{\Theta}(n)$ must improve the cut weight by at least ϵ. Setting $\epsilon := (\phi \cdot n^{\mathrm{polylog}(n)})^{-1}$ finishes the proof of our quasipolynomial upper bound.

The challenge behind the proof of Lemma 3.2 comes in two aspects:

(1) What is a sufficient condition for T that allows us to argue $\mathsf{rank}_{\mathsf{cycles}}(T) \geq \mathsf{len}(T)/\mathrm{polylog}(n)$?

(2) Can we show that every S of length $\tilde{\Theta}(n)$ contains a substring T that satisfies the condition in (1)?

1.3.4 Algorithmically Finding Many Linearly Independent Useful Cycles A first attempt for the sufficient condition on T would be the following: The auxiliary graph H satisfies that every node has degree between $\Omega(\mathrm{polylog}(n))$ and $O(\mathrm{polylog}(n))$ (i.e., at least $\Omega(\log^c n)$ for some c and at most $O(\log^d n)$ for some d). Assuming this is case, one can use the following process to find enough useful cycles that are linearly independent:

1. Find a useful cycle C in the auxiliary graph H and add it to the collection;
2. Pick a nonzero entry of the improvement vector of C and let e be its index (recall the vector is indexed by edges in the original complete graph); we call e the *witness* of the cycle C.
3. Delete both nodes of e from the auxiliary graph H, and repeat.

On the one hand, we only delete two nodes from H in each round. Given that every node has degree between $\Omega(\mathrm{polylog}(n))$ and $O(\mathrm{polylog}(n))$, one can argue that the process above can find a collection of at least $\mathsf{len}(T)/\mathrm{polylog}(n)$ many cycles. On the other hand, since the witness of the current cycle can never be a witness of a future cycle (as both nodes of the witness are deleted), the improvement vectors of cycles in the collection would form a triangular matrix and thus, have full rank. Together we have that $\mathsf{rank}_{\mathsf{cycles}}(T) \geq \mathsf{len}(T)/\mathrm{polylog}(n)$.

However, it is not hard to construct sequences of length $\tilde{\Theta}(n)$ (or any polynomial length), such that the auxiliary graph of any substring T has nodes with very unbalanced degrees: Most edges in the auxiliary graph are adjacent to a small number of nodes with very high degrees. As a result, deleting a high-degree node would have a significant impact on the auxiliary graph and the cycle-finding process described above may have to terminate within a few rounds (e.g., after all the high-degree nodes are deleted). So the challenge is to run a similar process, but reuse high-degree nodes carefully.

Skipping many details, we prove Lemma 3.2 in the following two steps. First, we prove in Section 4 that every sequence S of 2-FLIP of length $\tilde{\Theta}(n)$ must have a substring T such that its auxiliary graph H contains a bipartite graph with the following properties: (1) The number of edges in this bipartite graph is $\tilde{\Omega}(\mathsf{len}(T))$; (2) Every node on the LHS of the bipartite graph, as desired, has degree bounded between $\Omega(\mathrm{polylog}(n))$ and

$O(\text{polylog}(n))$; and (3) Every node on the RHS has degree at least $\Omega(\text{polylog}(n))$ but there is no upper bound on their degrees (so these are the high-degree nodes that can break the cycle-finding process described above).

Next for any sequence T with an auxiliary graph H that satisfies the condition above, we show that $\mathsf{rank}_{\mathsf{cycles}}(T) \geq \mathsf{len}(T)/\text{polylog}(n)$ by introducing a so-called *splitting* operation on the auxiliary graph. After a useful cycle is found in the current auxiliary graph H, we do not just delete the two nodes in its witness from H since one of them could lie on the RHS and has a high degree. Instead, we apply the splitting operation on the high-degree node in the witness: we "split" the node into several nodes and partition the edges of the node among the new nodes. On the one hand, the splitting operation has a much smaller impact on the auxiliary graph H; the impact is similar to the deletion of a node with $O(\text{polylog}(n))$ degree so the cycle-finding process can repeat at least $\mathsf{len}(T)/\text{polylog}(n)$ many rounds as before. On the other hand, the splitting operation and partition of the edges is done in a way that makes sure that the witness never appears again in future cycles so the cycles found are still linearly independent. The design of the splitting operation and the analysis of the cycle-finding process (recall that in the real proof we need to find not just any cycles but useful ones) are technically the most challenging parts of the paper.

1.4 Organization of the paper. The rest of the paper is organized as follows. Section 2 gives basic definitions of the problems and the smoothed model, defines the central concepts of arcs, and useful cycles, their improvement vectors, and proves a set of basic lemmas about them that are used throughout in the subsequent analysis. Section 3 states the main lemma on the existence of a nice window in the move sequence such that the arcs and cycles in the window have high rank, and shows how to derive the main theorem from this lemma. Sections 4 and 5 prove the main lemma in the case that all the moves are 2-moves (this is the more challenging case). First we show in Section 4 the existence of a nice window (in fact a large number of nice windows, since this is needed in the general case that includes both 1- and 2-moves) such that many moves in the window have the property that both nodes of the move appear a substantial number of times in the window (at least $\text{polylog}(n)$ times), and one of them does not appear too many times (at most a higher $\text{polylog}(n)$). In Section 5 we show how to find in such a nice window a large number of useful cycles whose improvement vectors are linearly independent. Section 6 extends the proof of the main lemma to the general case where the sequence of moves generated by 2-FLIP contains both 1- and 2-moves. Finally, in Section 7 we extend the results to the class of Maximum Binary Constraint Satisfaction and Function Optimization problems. For space reasons, several proofs are deferred to the full version of the paper.

2 Preliminaries

We write $[n]$ to denote $\{1,\dots,n\}$. Given two integers $a \leq b$, we write $[a:b]$ to denote $\{a,\dots,b\}$. Given $\gamma,\gamma' \in \{\pm 1\}^n$ we use $d(\gamma,\gamma')$ to denote the Hamming distance between γ and γ', i.e., the number of entries $i \in [n]$ such that $\gamma_i \neq \gamma'_i$.

2.1 Local Max-Cut and the FLIP Algorithm Let $K_n = (V_n, E_n)$ with $V_n = [n]$ be the complete undirected graph over n nodes. Given edge weights $X = (X_e : e \in E_n)$ with $X_e \in [-1,1]$, the *k-local Max-Cut problem* is to find a partition of V_n into two sets V_1 and V_2 such that the weight of the corresponding cut (the sum of weights of edges with one node in V_1 and the other in V_2) cannot be improved by moving no more than k nodes to the other set. Formally, the objective function of our interest is defined as follows: Given any *configuration* $\gamma \in \{\pm 1\}^n$ (which corresponds to a partition V_1, V_2 with $V_1 = \{u \in V_n : \gamma(u) = -1\}$ and $V_2 = \{u \in V_n : \gamma(u) = 1\}$), the objective function is

$$\mathsf{obj}_X(\gamma) := \sum_{(u,v)\in E_n} X_{(u,v)} \cdot \mathbf{1}\{\gamma(u) \neq \gamma(v)\} = \frac{1}{2} \sum_{(u,v)\in E_n} X_{(u,v)} \cdot \big(1 - \gamma(u)\gamma(v)\big). \tag{2.1}$$

Our goal is to find a configuration $\gamma \in \{\pm 1\}^n$ that is a k-local optimum, i.e., $\mathsf{obj}_X(\gamma) \geq \mathsf{obj}_X(\gamma')$ for every configuration $\gamma' \in \{\pm 1\}^n$ with Hamming distance no more than k from γ.

A simple local search algorithm for k-Local Max-Cut is the following k-FLIP algorithm:

Start with some initial configuration $\gamma = \gamma_0 \in \{\pm 1\}^n$. While there exists a configuration γ' with $d(\gamma', \gamma) \leq k$ such that $\mathsf{obj}_X(\gamma') > \mathsf{obj}_X(\gamma)$, select one such configuration γ' (according to some pivoting criterion), set $\gamma = \gamma'$ and repeat, until no such configuration γ' exists.

The execution of k-FLIP on K_n with edge weights X depends on both the initial configuration γ_0 and the pivoting criterion used to select the next configuration in each iteration. The larger the value of k, the larger the neighborhood structure that is being explored, hence the better the quality of solutions that is expected to be generated. However, the time complexity of each iteration grows rapidly with k: there are $\Theta(n^k)$ candidate moves, and with suitable data structures we can determine in $O(n^k)$ if there is an improving move and select one. Thus, the algorithm is feasible only for small values of k. For $k = 1$, it is the standard FLIP algorithm. Here we are interested in the case $k = 2$. We will not make any assumption on the pivoting criterion in our results, except that we assume that the algorithm does not choose to flip in any step two nodes when flipping only one of them would produce a strictly better cut. This is a natural property satisfied by any reasonable implementation of 2-FLIP. For example, one approach (to optimize the time of each iteration) is to first check if there is an improving 1-flip (n possibilities), and only if there is none (i.e. the partition is locally optimal with respect to 1-FLIP), proceed to search for an improving 2-flip ($O(n^2)$ possibilities). Clearly any implementation that follows this approach satisfies the above property. Also, the greedy approach, that examines all $O(n^2)$ possible 1-flips and 2-flips in each step and chooses one that yields the maximum improvement, obviously satisfies the above property.

Our results hold also for the variant of 2-FLIP that uses only 2-flips (no 1-flips), under any pivoting rule. We refer to this variant as *Pure 2-FLIP*.

2.2 Graph Partitioning and the SWAP Algorithm In the Graph Partitioning (or Bisection Width) problem, we are given a graph G on $2n$ nodes with weighted edges; the problem is to find a partition of the set V of nodes into two equal-sized subsets V_1, V_2 to minimize the weight of the cut.[5] As in the Max Cut problem, in this paper we will assume the graph is complete and the edge weights are in $[-1, 1]$. A simple local search algorithm is the SWAP algorithm: Starting from some initial partition (V_1, V_2) with n nodes in each part, while there is a pair of nodes $u \in V_1, v \in V_2$ whose swap (moving to the other part) decreases the weight of the cut, swap u and v. We do not make any assumption on the pivoting rule, i.e. which pair is selected to swap in each iteration if there are multiple pairs whose swap improves the cut. At the end, when the algorithm terminates it produces a locally optimal balanced partition, i.e. one that cannot be improved by swapping any pair of nodes. The SWAP algorithm is clearly a restricted version of Pure 2-FLIP (restricted because the initial partition is balanced, and in each step the 2-flip must involve two nodes from different parts of the partition).

The SWAP algorithm is the simplest local search algorithm for the Graph Partitioning problem, but it is a rather weak one, in the sense that the quality of the locally optimal solutions produced may not be very good. For this reason, more sophisticated local search algorithms have been proposed and are typically used, most notably the Kernighan-Lin algorithm [21], in which a move from a partition to a neighboring partition involves a sequence of swaps. If a partition has a profitable swap, then Kernighan-Lin (KL) will perform the best swap; however, if there is no profitable swap then KL explores a sequence of n greedy steps, selecting greedily in each step the best pair of nodes to swap that have not changed sides before in the current sequence, and if this sequence of swaps produces eventually a better partition, then KL moves to the best such partition generated during this sequence. A related variant, to reduce the time cost of each iteration, was proposed by Fiduccia and Matheyses [17]. This idea of guided deep neighborhood search is a powerful method in local search that was introduced first in the [21] paper of Kernighan and Lin on Graph Partitioning, and was applied subsequently successfully to the Traveling Salesman Problem and other problems.

2.3 Smoothed Analysis We focus on the 2-FLIP algorithm from now on. Under the smoothed complexity model, there is a family $\mathcal{X} = (\mathcal{X}_e : e \in E_n)$ of probability distributions, one for each edge in $K_n = (V_n, E_n)$. The edge weights $X = (X_e : e \in E_n)$ are drawn independently with $X_e \sim \mathcal{X}_e$. We assume that each $\mathcal{X}_e$ is a

[5] Since the weights can be positive or negative, there is no difference between maximization and minimization. The Graph Partitioning problem is usually stated as a minimization problem.

distribution supported on $[-1,1]$ and its density function is bounded from above by a parameter $\phi > 0$. (The assumption that the edge weights are in $[-1,1]$ is no loss of generality, since they can be always scaled to lie in that range.) Our goal is to bound the number of steps the 2-FLIP algorithm takes to terminate when running on K_n with edge weights $X \sim \mathcal{X}$, in terms of n and the parameter ϕ.

2.4 Move Sequences We introduce some of the key definitions that will be used in the smoothed analysis of 2-FLIP.

A *move sequence* $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_\ell)$ is an ℓ-tuple for some $\ell \geq 1$ such that $\mathcal{S}_i$ is a subset of V_n of size either one or two. We will refer to the i-th move in $\mathcal{S}$ as a 1-*move* if $|\mathcal{S}_i| = 1$ and a 2-*move* if $|\mathcal{S}_i| = 2$, and write $\mathsf{len}(\mathcal{S}) := \ell$ to denote its length. Additionally, let 1-*move*$(\mathcal{S})$ and 2-*move*$(\mathcal{S})$ denote the corresponding subsequence of single flip or double flips correspondingly. We say a node $u \in V_n$ is *active* in $\mathcal{S}$ if u appears in $\mathcal{S}_i$ for some i, and is *inactive* otherwise. We write $V(\mathcal{S}) \subseteq V_n$ to denote the set of active nodes in $\mathcal{S}$.

Given $\gamma_0 \in \{\pm 1\}^n$ as the initial configuration, a move sequence $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_\ell)$ naturally induces a sequence of configurations $\gamma_0, \gamma_1, \ldots, \gamma_\ell \in \{\pm 1\}^n$, where γ_{i+1} is obtained from γ_i by flipping the nodes in $\mathcal{S}_{i+1}$. We say $(\gamma_0, \mathcal{S})$ is *improving* with respect to edge weights X if

$$\mathsf{obj}_X(\gamma_i) > \mathsf{obj}_X(\gamma_{i-1}), \quad \text{for all } i \in [\ell]$$

and is ϵ-*improving* with respect to edge weights X, for some $\epsilon > 0$, if

$$\mathsf{obj}_X(\gamma_i) - \mathsf{obj}_X(\gamma_{i-1}) \in (0, \epsilon], \quad \text{for all } i \in [\ell].$$

For each $i \in [\ell]$, the change $\mathsf{obj}_X(\gamma_i) - \mathsf{obj}_X(\gamma_{i-1})$ from the i-th move $\mathcal{S}_i$ can be written as follows:

1. When $\mathcal{S}_i = \{u\}$,

$$\mathsf{obj}_X(\gamma_i) - \mathsf{obj}_X(\gamma_{i-1}) = \sum_{w \in V_n : w \neq u} \gamma_{i-1}(w)\gamma_{i-1}(u)X_{(u,w)}. \tag{2.2}$$

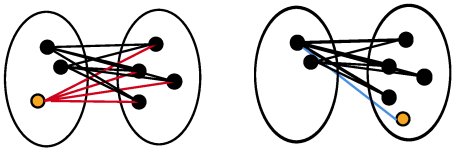

Figure 1: Example of a *1-move*, showing edges in the cut only.

2. When $\mathcal{S}_i = \{u, v\}$,

$$\mathsf{obj}_X(\gamma_i) - \mathsf{obj}_X(\gamma_{i-1}) = \sum_{w \in V_n : w \notin \{u,v\}} (\gamma_{i-1}(w)\gamma_{i-1}(u)X_{(w,u)} + \gamma_{i-1}(w)\gamma_{i-1}(v)X_{(w,v)}). \tag{2.3}$$

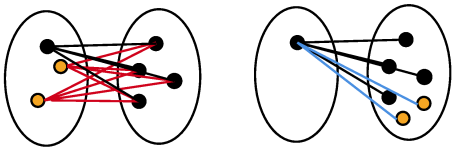

Figure 2: Example of a *2-move* , showing edges in the cut only.

For each $i \in [\ell]$, we write $\mathsf{imprv}_{\gamma_0,\mathcal{S}}(i)$ to denote the improvement vector in $\{0, \pm 1\}^{E_n}$ such that

$$\mathsf{obj}_X(\gamma_i) - \mathsf{obj}_X(\gamma_{i-1}) = \mathsf{imprv}_{\gamma_0,\mathcal{S}}(i) \cdot X. \tag{2.4}$$

Next, let $E(\mathcal{S})$ denote the set of edges $(u,v) \in E_n$ such that both u and v are active in $\mathcal{S}$. We write $\mathsf{imprv}^*_{\gamma_0,\mathcal{S}}(i) \in \{0,\pm 1\}^{E(\mathcal{S})}$ to denote the projection of $\mathsf{imprv}_{\gamma_0,\mathcal{S}}(i)$ on entries that correspond to edges in $E(\mathcal{S})$. We note that $\mathsf{imprv}^*_{\gamma_0,\mathcal{S}}(i)$ only depends on the initial configuration of active nodes $V(S)$ in γ_0. Given a (partial) configuration $\tau_0 \in \{\pm 1\}^{V(S)}$ of $V(S)$, we let

$$\mathsf{imprv}_{\tau_0,\mathcal{S}}(i) := \mathsf{imprv}^*_{\gamma_0,\mathcal{S}}(i) \in \{0,\pm 1\}^{E(\mathcal{S})},$$

where $\gamma_0 \in \{\pm 1\}^n$ is an arbitrary (full) configuration that is an extension of τ_0. (To aid the reader we will always use γ to denote a full configuration and τ to denote a partial configuration in the paper.)

Note that if $\mathcal{S}$ is a sequence of moves generated by an execution of the 2-FLIP algorithm then $\mathcal{S}$ must be improving, because every move must increase the weight of the cut and therefore every $1-$ or $2-$ move is improving. On the other hand, if every move in $\mathcal{S}$ increases the cut weight by no more than ϵ then we can not directly guarantee that after $poly(|\mathcal{S}|, n, 1/\epsilon)$ steps the algorithm would certainly terminate. From probabilistic perspective, in order to provide a smoothed upper bound on the running time of 2-FLIP method, it suffices to show that it is exponentially small probability for every move in a long enough sequence to incur only a $o(1/poly(n))$ improvement in our objective.

Indeed, in an idealized scenario where the improvements of different moves of a sequence were disentangled, the event for a linear-length sequence to be at most $\epsilon-$improving would have exponentially small probability. Unfortunately, going back to the 2-FLIP algorithm, there could be improving steps that are strongly correlated (as an extreme situation there could be two flips with almost the same improvement vector). Thus, as one may expect the probability exponential decay holds only for linearly independent $\mathsf{imprv}_{\tau_0,\mathcal{S}}(\cdot)$, introducing the necessity of analysis of the $\mathsf{rank}\left(\{\mathsf{imprv}_{\tau_0,\mathcal{S}}(i) | i \in \mathcal{S}'\}\right)$, for some neatly chosen subset $\mathcal{S}'$ of moves from the sequence $\mathcal{S}$.

Corollary 2.1. ([15]) *Let $X_1, ..., X_m$ be independent real random variables and let $f_i : \mathbb{R} \to [0, \phi]$ for some $\phi > 0$ denote the density of X_i for each $i \in [m]$. Additionally, let $\mathcal{C}$ be a collection of k not necessarily linearly independent integer row vectors, namely $\mathcal{C} = \{V_1, \cdots, V_k\}$. Then it holds that for any interval $I \subset \mathbb{R}$*

$$\Pr[F_\epsilon] = \Pr\left[\bigcap_{i\in[k]} \{V_i \cdot X \in I\}\right] \leq (\phi \mathit{len}(I))^{\mathit{rank}(\mathcal{C})}$$

However, one standard issue, which typically occurs with the direct usage of improvement vectors of sequence's moves, is their dependence also on the initial configuration γ of inactive nodes that do not appear in the sequence $\mathcal{S}$. Their number may be much larger than the rank of the active nodes, and thus considering all their possible initial values in a union-bound will overwhelm the probability $(\phi\epsilon)^r$. For these reasons, in the literature [15, 7, 11] more complex combinatorial structures have been proposed, like pairs of (consecutive) moves of the same node.

2.5 Arcs

Definition 2.1. *An* arc *α in a move sequence $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_\ell)$ is an ordered pair (i,j) with $i < j \in [\ell]$ such that $\mathcal{S}_i = \mathcal{S}_j = \{u\}$ for some node $u \in V_n$ and for any $i < k < j$, $\mathcal{S}_k \neq \{u\}$.*

Let $\tau_0 \in \{\pm 1\}^{V(\mathcal{S})}$ be a configuration of active nodes in $\mathcal{S}$, and let $\tau_0, \tau_1, \ldots, \tau_\ell \in \{\pm 1\}^{V(\mathcal{S})}$ be the sequence of configurations induced by $\mathcal{S}$, i.e., τ_i is obtained from τ_{i-1} by flipping nodes in $\mathcal{S}_i$. We make the following observation:

Lemma 2.1. *For any configuration $\gamma_0 \in \{\pm 1\}^n$ that is an extension of τ_0, letting $\gamma_0, \gamma_1, \ldots, \gamma_\ell \in \{\pm 1\}^n$ be the sequence of configurations induced by $\mathcal{S}$ and letting $w[u,i,j] := \gamma_i(u) \cdot \mathsf{imprv}_{\gamma_0,\mathcal{S}}(i) - \gamma_j(u) \cdot \mathsf{imprv}_{\gamma_0,\mathcal{S}}(j)$, we have that*

$$(w[u,i,j]_e)_{e\in E_n} = \begin{cases} (\tau_i(u) \cdot \mathsf{imprv}_{\tau_0,\mathcal{S}}(i) - \tau_j(u) \cdot \mathsf{imprv}_{\tau_0,\mathcal{S}}(j))_e & \text{for every entry } e \in E(\mathcal{S}), \\ 0 & \text{otherwise.} \end{cases}$$

for any arbitrary choice of $u \in V(\mathcal{S})$.

Motivated by Lemma 2.1, we define for an arc $\alpha = (i, j)$ of a node u,

$$\mathsf{imprv}_{\tau_0,\mathcal{S}}(\alpha) := \tau_i(u) \cdot \mathsf{imprv}_{\tau_0,\mathcal{S}}(i) - \tau_j(u) \cdot \mathsf{imprv}_{\tau_0,\mathcal{S}}(j) \in \mathbb{Z}^{E(\mathcal{S})}. \tag{2.5}$$

Let $\mathsf{arcs}(\mathcal{S})$ denote the set of all arcs in $\mathcal{S}$. We will be interested in the rank of

$$Q_{\mathsf{arcs}} := \left\{\mathsf{imprv}_{\tau_0,\mathcal{S}}(\alpha) : \alpha \in \mathsf{arcs}(\mathcal{S})\right\} \tag{2.6}$$

It is easy to show that the rank does not depend on the choice of τ_0 so we will denote it by $\mathsf{rank}_{\mathsf{arcs}}(\mathcal{S})$.

LEMMA 2.2. *The rank of the set of vectors in (2.6) does not depend on the choice of τ_0.*

The notion of arcs concerns only 1-moves. The problem becomes much more involves in the presence of 2-moves. To address this, we associate an auxiliary (multi)graph to the 2-moves, use the cycles of this graph, and introduce the notion of useful cycles.

2.6 Graph of 2-Moves and Useful Cycles Given a move sequence $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_\ell)$, we associate with it an auxiliary (multi)graph H over the set of nodes V_n that has one edge (u, v) for every 2-move $\{u, v\}$ of $\mathcal{S}$. Of particular interest are the cycles of H, and the corresponding subsets of moves of $\mathcal{S}$.

DEFINITION 2.2. *A* cycle *C in a move sequence $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_\ell)$ is an ordered tuple $C = (c_1, \ldots, c_t)$ for some $t \geq 2$ such that $c_1, \cdots c_t$ are distinct, and $\mathcal{S}_{c_j} = \{u_j, u_{j+1}\}$ for all $j \in [t-1]$ and $\mathcal{S}_{c_t} = \{u_t, u_1\}$ for some nodes $u_1, \ldots, u_t \in V_n$. (Every $\mathcal{S}_{c_j}$ is a 2-move. The same vertex may appear in multiple $\mathcal{S}_{c_j}$'s).*

DEFINITION 2.3. *Given a configuration $\tau_0 \in \{\pm 1\}^{V(\mathcal{S})}$, we say a cycle $C = (c_1, \ldots, c_t)$ in $\mathcal{S}$ is* useful *with respect to τ_0 if there exists $b \in \{\pm 1\}^t$ such that*

$$\textit{For all } j \in [t-1] \textit{ we have that } b_j \cdot \tau_{c_j}(u_{j+1}) + b_{j+1} \cdot \tau_{c_{j+1}}(u_{j+1}) = 0 \textit{ and } b_t \cdot \tau_{c_t}(u_1) + b_1 \cdot \tau_{c_1}(u_1) = 0,$$

where $\tau_0, \tau_1, \ldots, \tau_\ell$ are configurations induced by $\mathcal{S}$ starting from τ_0.

We note that such a vector b, if it exists, it has the form $b = b_1 \cdot \left(1, \cdots, (-1)^{k-1} \prod_{i \in [2:k]} \tau_{c_{i-1}}(u_i)\tau_{c_i}(u_i), \cdots\right)^\top$ and hence it is unique if we further require $b_1 = 1$. After elimination of the above equations, we see the following equivalent criterion:

REMARK 2.1. (USEFULNESS CRITERION) *A cycle C is* useful $\Leftrightarrow (-1)^t = \tau_{c_t}(u_1)\tau_{c_1}(u_1) \cdot \prod_{i=2}^{t} \tau_{c_{i-1}}(u_i)\tau_{c_i}(u_i)$.

We will refer to the unique vector $b \in \{\pm 1\}^t$ as the *cancellation vector* of C. Notice that whether a cycle C in $\mathcal{S}$ is useful or not actually does not depend on the choice of τ_0.

LEMMA 2.3. *If a cycle C of $\mathcal{S}$ is useful with respect to some $\tau_0 \in \{\pm 1\}^{V(\mathcal{S})}$ using b as its cancellation vector, then it is useful with respect to every configuration $\tau_0' \in \{\pm 1\}^{V(\mathcal{S})}$ using the same b as its cancellation vector.*

As a result, we can refer to cycles of $\mathcal{S}$ as useful cycles without specifying a configuration τ_0; the same holds for cancellation vectors. Next we prove a lemma that is similar to Lemma 2.1 for arcs:

LEMMA 2.4. *Let $C = (c_1, \ldots, c_t)$ be a useful cycle of $\mathcal{S}$ and let b be its cancellation vector. Then for any configurations $\tau_0 \in \{\pm 1\}^{V(\mathcal{S})}$ and $\gamma_0 \in \{\pm 1\}^n$ such that γ_0 is an extension of τ_0, letting $w[C] := \sum_{j \in [t]} b_j \cdot \mathsf{imprv}_{\gamma_0,\mathcal{S}}(c_j)$, we have that*

$$(w[C]_e)_{e \in E_n} = \begin{cases} (\sum_{j \in [t]} b_j \cdot \mathsf{imprv}_{\tau_0,\mathcal{S}}(c_j))_e & \textit{for every entry } e \in E(\mathcal{S}), \\ 0 & \textit{otherwise.} \end{cases}$$

Given $\tau_0 \in \{\pm 1\}^{V(\mathcal{S})}$ and a useful cycle C of $\mathcal{S}$ with b as its cancellation vector, we define

$$\mathsf{imprv}_{\tau_0,\mathcal{S}}(C) := \sum_{j\in[t]} b_j \cdot \mathsf{imprv}_{\tau_0,\mathcal{S}}(c_j). \tag{2.7}$$

Let $\mathsf{cycles}(\mathcal{S})$ denote the set of all useful cycles in $\mathcal{S}$. We will be interested in the rank of

$$Q_{\mathsf{cycles}} := \left\{\mathsf{imprv}_{\tau_0,\mathcal{S}}(C) : C \in \mathsf{cycles}(\mathcal{S})\right\} \tag{2.8}$$

Similarly we note that the rank does not depend on the choice of τ_0 so we denote it by $\mathsf{rank}_{\mathsf{cycles}}(\mathcal{S})$.

Lemma 2.5. *The rank of the set of vectors in (2.8) does not depend on the choice of τ_0.*

For the sake of readability we defer the proofs of initial configuration invariance for the rank of improvement vectors of arcs and cycles to Appendix B in the full version of the paper. Having defined the sets of $\mathsf{arcs}(\mathcal{S})$ and $\mathsf{cycles}(\mathcal{S})$, we conclude this section by showing that for a fixed parameter $\epsilon > 0$, a move sequence $\mathcal{S}$ and an initial configuration $\tau_0 \in \{\pm 1\}^{V(\mathcal{S})}$, if either $\mathsf{rank}_{\mathsf{arcs}}(\mathcal{S})$ or $\mathsf{rank}_{\mathsf{cycles}}(\mathcal{S})$ is high, then most likely (over $X \sim \mathcal{X}$) $(\gamma_0, \mathcal{S})$ is not ϵ-improving for every $\gamma_0 \in \{\pm 1\}^n$ that is an extension of τ_0.

Lemma 2.6. *Let $\epsilon > 0$. With probability at least*

$$1 - \left(2\mathsf{len}(\mathcal{S}) \cdot \phi\epsilon\right)^{\max\left(\mathsf{rank}_{\mathsf{arcs}}(\mathcal{S}),\, \mathsf{rank}_{\mathsf{cycles}}(\mathcal{S})\right)}$$

over $X \sim \mathcal{X}$, we have that $(\gamma_0, \mathcal{S})$ is not ϵ-improving for every $\gamma_0 \in \{\pm 1\}^n$ that is an extension of τ_0.

Proof. Let $\mathcal{E}_{moves}$ be the event of a given $(\gamma_0, \mathcal{S})$ being ϵ-improving with respect to edge weights X, for some fixed $\epsilon > 0$:

$$\mathcal{E}_{moves} : \left\{\mathsf{imprv}_{\gamma_0,\mathcal{S}}(i) \cdot X \in (0, \epsilon], \quad \text{for all } i \in [\ell]\right\}$$

where $\mathsf{imprv}_{\gamma_0,\mathcal{S}}(i)$ correspond to the improvement vector of $\mathcal{S}_i$ move(See (2.2),(2.3)). Now, notice that the improvement vector of an arc (See (2.5)) or of a useful cycle (See (2.7)) can be written as the $\{-1, 0, 1\}$ sum of all the improvement vectors of either 1 or 2-moves in $\mathcal{S}$. Thus, we define the corresponding event for cycles and arcs for a given sequence $(\gamma_0, \mathcal{S})$ with respect to edge weights X:

$$\mathcal{E}_{arcs/cycles} : \left\{\mathsf{imprv}_{\gamma_0,\mathcal{S}}(\beta) \cdot X \in [-\mathsf{len}(\mathcal{S})\epsilon, \mathsf{len}(\mathcal{S})\epsilon], \quad \text{for any } \beta \in \mathsf{arcs}(\mathcal{S})/\mathsf{cycles}(\mathcal{S})\right\}$$

So it is easy to see that $\mathcal{E}_{moves}$ implies $\mathcal{E}_{arcs/cycles}$, or equivalently $\Pr[\mathcal{E}_{moves}] \leq \min\{\Pr[\mathcal{E}_{arcs}], \Pr\left[\mathcal{E}_{cycles}\right]\}$. Thus, by leveraging Corollary 2.1 for vectors in Q_{arcs} and Q_{cycles}), we get that:

$$\Pr\left[(\gamma_0, \mathcal{S}) \text{ being an } \epsilon\text{-improving sequence}\right] \leq \left(2\mathsf{len}(\mathcal{S}) \cdot \phi\epsilon\right)^{\max\left(rank_{arcs}(\mathcal{S}),\, rank_{cycles}(\mathcal{S})\right)}$$

This finishes the proof of the lemma. ∎

3 Main Lemma and the Proof of Theorem 1.2 and Theorem 1.1

We start with the definition of *valid* move sequences:

Definition 3.1. *We say a move sequence $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_\ell)$ is* valid *if it satisfies the following property: For every $i < j \in [\ell]$, at least one node $w \notin \mathcal{S}_i$ appears an odd number of times in $\mathcal{S}_i, \ldots, \mathcal{S}_j$.*

Lemma 3.1. *The move sequence generated by 2-FLIP (or by pure 2-FLIP), for any pivoting rule and any instance, is valid .*

Proof. Let $\mathcal{S}$ be a move sequence generated by 2-FLIP (or pure 2-FLIP). If there are two moves $\mathcal{S}_i, \mathcal{S}_j$, $i \leq j$, such that no node appears an odd number of times in $\mathcal{S}_i, \ldots, \mathcal{S}_j$, then the configurations before $\mathcal{S}_i$ and after $\mathcal{S}_j$ are

the same, contradicting the fact that all the moves increase the weight of the cut. Therefore, the set O of nodes that appear an odd number of times in $\mathcal{S}_i, \ldots, \mathcal{S}_j$ is nonempty. Suppose that $i < j$ and $O \subseteq \mathcal{S}_i$. If $O = \mathcal{S}_i$, then the set of nodes that appear an odd number of times in $\mathcal{S}_{i+1}, \ldots, \mathcal{S}_j$ would be empty, a contradiction to the above property. Therefore, $O \neq \mathcal{S}_i$.

In the case of pure 2-FLIP, since all moves are 2-flips, O has even size, and hence $O \neq \emptyset$ and $O \neq \mathcal{S}_i$ imply the claim.

In the case of 2-FLIP, $O \neq \emptyset$, $O \neq \mathcal{S}_i$ and $O \subseteq \mathcal{S}_i$ imply that $\mathcal{S}_i$ has size 2, say $\mathcal{S}_i = \{u, v\}$ and $O = \{u\}$ or $O = \{v\}$. If $O = \{u\}$ then the configuration γ_j differs from γ_{i-1} only in that node u is flipped. Thus, at configuration γ_{i-1}, flipping node u results in configuration γ_j which has strictly greater cut than the configuration γ_i that results by flipping the pair $\{u, v\}$, contradicting our assumption about 2-FLIP. A similar argument holds if $O = \{v\}$. In either case we have a contradiction to $O \subseteq \mathcal{S}_i$. The claim follows. ∎

Given a move sequence $\mathcal{S}$, a *window* W of $\mathcal{S}$ is a substring of $\mathcal{S}$, i.e., $W = (\mathcal{S}_i, \ldots, \mathcal{S}_j)$ for some $i < j \in [\ell]$ (so W itself is also a move sequence). Our main technical lemma below shows that every long enough valid move sequence has a window W such that either $\mathsf{rank}_{\mathsf{arcs}}(W)$ or $\mathsf{rank}_{\mathsf{cycles}}(W)$ is large relative to $\mathsf{len}(W)$.

LEMMA 3.2. *Let $\mathcal{S}$ be a valid move sequence with $\mathsf{len}(\mathcal{S}) \geq n \log^{10} n$. Then $\mathcal{S}$ has a window W such that*

$$\max\left(\mathsf{rank}_{\mathsf{arcs}}(W), \mathsf{rank}_{\mathsf{cycles}}(W)\right) \geq \Omega\left(\frac{\mathsf{len}(W)}{\log^{10} n}\right). \tag{3.9}$$

We prove Lemma 3.2 when $\mathcal{S}$ consists of 2-moves only in Section 4 and 5, and then generalize the proof to work with general move sequences in Section 6. Assuming Lemma 3.2, we use it to establish our main theorem, restated below:

THEOREM 3.1. *Let $\mathcal{X} = (\mathcal{X}_e : e \in E_n)$ be distributions of edge weights such that each $\mathcal{X}_e$ is supported on $[-1, 1]$ and has its density function bounded from above by a parameter $\phi > 0$. Then with probability at least $1 - o_n(1)$ over the draw of $X \sim \mathcal{X}$, every implementation of the 2-FLIP algorithm terminates within $\phi n^{O(\log^{10} n)}$ steps.*

Proof. Let $\epsilon > 0$ be specified as follows:

$$\epsilon := \frac{1}{\phi n^{c_1 \log^{10} n}}$$

for some large enough constant $c_1 > 0$ to be specified later. We write $\mathcal{F}$ to denote the following event on the draw of edge weights $X \sim \mathcal{X}$:

> Event $\mathcal{F}$: For every move sequence W of length at most $n \log^{10} n$ such that (letting $a > 0$ be the constant hidden in (3.9))
>
> $$\max\left(\mathsf{rank}_{\mathsf{arcs}}(W), \mathsf{rank}_{\mathsf{cycles}}(W)\right) \geq \frac{a}{\log^{10} n} \cdot \mathsf{len}(W). \tag{3.10}$$
>
> and every configuration $\gamma_0 \in \{\pm 1\}^n$, (γ_0, W) is not ϵ-improving with respect to X.

We break the proof of the theorem into two steps. First we show that $\mathcal{F}$ occurs with probability at least $1 - o_n(1)$. Next we show that when $\mathcal{F}$ occurs, any implementation of 2-FLIP must terminate in at most $\phi n^{O(\log^{10} n)}$ many steps.

For the first step, we apply Lemma 2.6 on every move sequence W of length at most $n \log^{10} n$ that satisfies (3.10) and every configuration $\tau_0 \in \{\pm 1\}^{V(W)}$. It then follows from a union bound that $\mathcal{F}$ occurs with probability at least

$$1 - \sum_{\ell \in [n \log^{10} n]} n^{2\ell} \cdot 2^{2\ell} \cdot (\ell \phi \epsilon)^{\frac{a\ell}{\log^{10} n}} = 1 - \sum_{\ell \in [n \log^{10} n]} \left((2n)^{\frac{2 \log^{10} n}{a}} \cdot \ell \phi \epsilon\right)^{\frac{a\ell}{\log^{10} n}} = 1 - o_n(1),$$

where the factor $n^{2\ell}$ is an upper bound for the number of W of length ℓ and $2^{2\ell}$ is an upper bound for the number of configurations τ_0 since $|V(W)| \leq 2\ell$. The last equation follows by setting the constant c_1 sufficiently large.

For the second step, we assume that the event $\mathcal{F}$ occurs, and let $\gamma_0, \ldots, \gamma_N \in \{\pm 1\}^n$ be a sequence of N configurations that is the result of the execution of some implementation of 2-FLIP under X. Let $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_N)$ denote the move sequence induced by $\gamma_0, \ldots, \gamma_N$. So $(\gamma_0, \mathcal{S})$ is improving with respect to edge weights X. By Lemma 3.1, $\mathcal{S}$ is a valid sequence.

We use the event $\mathcal{F}$ to bound the length of $\mathcal{S}$. Because of $\mathcal{F}$ and that $\mathcal{S}$ is a valid move sequence, it follows from Lemma 3.2 that the objective function gets improved by at least ϵ for every $n \log^{10} n$ consecutive moves in $\mathcal{S}$. Given that the objective function lies between $[-n^2, n^2]$, we have

$$\mathsf{len}(\mathcal{S}) \leq n \log^{10} n \cdot \frac{2n^2}{\epsilon} \leq \phi n^{O(\log^{10} n)}. \tag{3.11}$$

This finishes the proof of the theorem. ∎

Corollary 3.1. *Under the same setting of Theorem 1.2, the same result holds for Pure 2-FLIP.*

Proof. The only property of the sequence of moves used in the proof of Theorem 1.2 is that it is a valid sequence, and this property holds for Pure 2-FLIP as well. ∎

Notice that by twining the constant c_1 in the exponent, we can control the tail-bound of the failure probability. Thus, we can strengthen our proof to get the same bound for the expected number of steps needed to terminate as in the standard smoothed analysis prototype :

Corollary 3.2. *Under the same setting of Theorem 1.2, any implementation of the 2-FLIP algorithm (or Pure 2-FLIP) takes at most $\phi n^{O\left(\log^{10} n\right)}$ many steps to terminate on expectation.*

Proof. We let $\mathcal{F}_\epsilon$ denote the event $\mathcal{F}$ in the proof of Theorem 1.2 with a specified $\epsilon > 0$. Let $\epsilon_0 = 1/\left(\phi \cdot n^{c_1 \log^{10} n}\right)$, where $c_1 > 0$ is a constant to be fixed shortly. For any $\epsilon < \epsilon_0$, we have that

$$\begin{aligned}
\Pr[\neg \mathcal{F}_\epsilon] &\leq \sum_{\ell \in [n \log^{10} n]} n^{2\ell} \cdot 2^{2\ell} \cdot (\ell \phi \epsilon)^{\max\left(\mathsf{rank}_{\mathsf{arcs}}(W), \mathsf{rank}_{\mathsf{cycles}}(W)\right)} \leq \sum_{\ell \in [n \log^{10} n]} n^{2\ell} \cdot 2^{2\ell} \cdot (\ell \phi \epsilon \cdot \frac{\epsilon_0}{\epsilon_0})^{\lceil \frac{a\ell}{\log^{10} n} \rceil} \\
&\leq \sum_{\ell \in [n \log^{10} n]} n^{2\ell} \cdot 2^{2\ell} \cdot \left(\ell \phi \epsilon_0\right)^{\lceil \frac{a\ell}{\log^{10} n} \rceil} \cdot (\frac{\epsilon}{\epsilon_0})^{\lceil \frac{a\ell}{\log^{10} n} \rceil} \leq \sum_{\ell \in [n \log^{10} n]} \frac{1}{n^3} \left(\frac{\epsilon}{\epsilon_0}\right)^{\lceil \frac{a\ell}{\log^{10} n} \rceil} \leq \frac{\epsilon}{c_2 n \epsilon_0}
\end{aligned}$$

where $c_1 = 2 + 6/a$ (letting $a > 0$ be the constant hidden in (3.9)) and $c_2 = 10^8$. From the proof of Theorem 1.2, conditionally to the event $\mathcal{F}_\epsilon$ for any $\epsilon \leq \epsilon_0$, $\mathsf{len}(\mathcal{S}) \leq L(\epsilon) := \frac{2n^3 \log^{10} n}{\epsilon}$. Notice that for any $\epsilon(\rho) := \epsilon_0/\rho$, $L(\epsilon(\rho)) = \rho L(\epsilon_0)$. Thus, the probability that $\mathsf{len}(\mathcal{S})$ is larger than $cL(\epsilon)$ for any $\rho \geq 1$ is

$$\Pr\left[\mathsf{len}(\mathcal{S}) > \rho L(\epsilon)\right] \leq \Pr\left[\neg F_{\epsilon_0/\rho}\right] \leq \frac{1/\rho}{c_2 \cdot n}.$$

Note that L is always trivially bounded by the total number of configurations, 2^n. Therefore, we have

$$\begin{aligned}
\mathbb{E}[\mathsf{len}(\mathcal{S})] &= \sum_{s=1}^{2^n} \Pr[\mathsf{len}(\mathcal{S}) \geq s] \leq \sum_{s=1}^{\lceil L(\epsilon_0) \rceil} \Pr[L \geq s] + \sum_{s=\lceil L(\epsilon_0) \rceil}^{2^n} \Pr[L \geq s] \leq L(\epsilon_0) + \sum_{s=\lceil L(\epsilon_0) \rceil}^{2^n} \Pr[L \geq s] \\
&\leq L(\epsilon_0) + \sum_{s=\lceil L(\epsilon_0) \rceil}^{2^n} \Pr\left[L \geq \tfrac{s}{L(\epsilon_0)} \cdot L(\epsilon_0)\right] \leq L(\epsilon_0) + \sum_{s=\lceil L(\epsilon_0) \rceil}^{2^n} \frac{L(\epsilon_0)/s}{c_2 n} = O(n) \cdot L(\epsilon_0) = \phi n^{O(\log^{10} n)}.
\end{aligned}$$

This finishes the proof of Corollary 3.2. ∎

The same results hold for the Graph Partitioning problem and the SWAP neighborhood.

THEOREM 3.2. *Let $\mathcal{X} = (\mathcal{X}_e : e \in E_{2n})$ be distributions of edge weights such that each $\mathcal{X}_e$ is supported on $[-1, 1]$ and has its density function bounded from above by a parameter $\phi > 0$. Then with probability at least $1 - o_n(1)$ over the draw of edge weights $X \sim \mathcal{X}$, every implementation of SWAP terminates within $\phi n^{O(\log^{10} n)}$ steps.*

Proof. Every move sequence $\mathcal{S}$ generated by SWAP (for any pivoting rule, any weights, and any initial balanced partition) is also a legal move sequence for Pure 2-FLIP on the same instance, except that the sequence may be incomplete for Pure 2-FLIP, that is, the final partition may not be locally optimal for Pure 2-FLIP, since there may be a 2-move (but not a swap) that improves the weight of the cut (the resulting partition would not be balanced), and Pure 2-FLIP would continue and produce a longer sequence. Hence, the number of steps of SWAP is upper bounded by the number of steps of Pure 2-FLIP, and thus it is at most $\phi n^{O\left(\log^{10} n\right)}$ with probability $1 - o_n(1)$, as well as in expectation. ∎

4 Windows in a Valid Sequence of 2-Moves

We will start with the proof of Lemma 3.2 for the case when $\mathcal{S}$ consists of 2-moves only in Section 4 and 5, and generalize it to deal with general move sequences in Section 6.

We start with a combinatorial argument about sets and subsequences of $[N]$, where $N = poly(n)$ for any polynomial at n. Let I be a subset of $[N]$ with $|I| \geq \log^{10} n$. Intuitively, later in this section I will be chosen to be the set I_u representing the appearances of some frequently appeared active node $u \in V(\mathcal{S})$ in a move sequence $\mathcal{S}$. We will write $\mathsf{order}(i)$ to denote the order of $i \in I$. In other words, the smallest index in I has order 1 and the largest index in I has order $|I|$. To give an example if $I = \{2, 5, 9, 11\}$ then $\mathsf{order}(2) = 1, \mathsf{order}(5) = 2$ and so on. Let $\delta = 0.01$. We start by quantifying how much large windows centered around an index $i \in I$ should be to cover the majority of a set I. Afterwards, we present the combinatorial lemmas about subset I.

DEFINITION 4.1. *Let $I \subseteq [N]$. We say an index $i \in I$ is* ℓ-good *for some positive integer ℓ if*

$$\left[i - \lceil (1+2\delta)L' \rceil : i + \lceil (1+2\delta)L' \rceil\right] \subseteq [N], \quad \text{where } L' = \lceil (1+\delta)^{\ell-1} \rceil$$

and I satisfies

$$\left|I \cap [i - L' : i + L']\right| \geq \log^3 n \quad \text{and} \quad \left|I \cap \left[i - \lceil (1+2\delta)L' \rceil : i + \lceil (1+2\delta)L' \rceil\right]\right| \leq \log^7 n. \tag{4.12}$$

If there exists no such constant ℓ, we call the corresponding index bad.

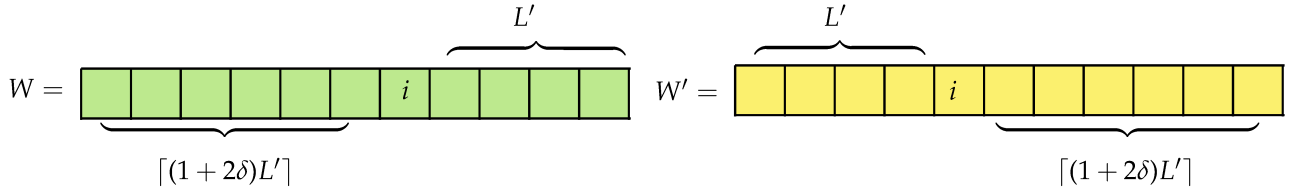

Figure 3: Two examples of windows whose intersection in I is between $[\log^3 n, \log^7 n]$.

REMARK 4.1. *Some motivation behind the definition 4.1: When $i \in I$ is ℓ-good (letting $L = L' + \lceil (1+2\delta)L' \rceil + 1$ and $L' = \lceil (1+\delta)^{\ell-1} \rceil$), it implies that all the $\lceil (1+2\delta)L' \rceil - L' \geq 2\delta L' = \Omega(L)$[6] windows W of length L, i.e., those start at*

[6] Indeed, by definition $L = L' + 1 + \lceil (1+2\delta)L' \rceil \leq 2((1+\delta)L' + 1) \leq 2((1+\delta)L' + 1) \leq 2(2+\delta)L'$, since $L' \geq 1$

$i - \lceil (1+2\delta)L' \rceil, \ldots, i - L'$, *satisfy*

$$i \in W, \quad |I \cap W| \geq \log^3 n \quad \textit{and} \quad |I \cap W| \leq \log^7 n.$$

The first and last windows that satisfy the above property are illustrated in Figure 3.

REMARK 4.2. *By definition 4.1, ℓ can get at most $\log_{1+\delta} N = \Theta(\log n)$, for any $N = poly(n)$.*

LEMMA 4.1. *Suppose I is a subset of $[N]$ with $|I| \geq \log^8 n$. Then at least a $(1 - O(1/\log n))$-fraction of $i \in I$ is ℓ_i-good for some nonnegative integer ℓ_i.*

Proof. See proof in the full version. ∎

Now we return to work on our problem and an arbitrary move sequence $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_N)$. Let W be a window (move sequence) of $\mathcal{S}$. For each active node $u \in V(W)$, we write $\#_W(u)$ to denote the number of occurrences of u in W. The main result in this section is the following lemma:

LEMMA 4.2. *Let $\mathcal{S}$ be a move sequence of length $N = n \log^{10} n$ that consists of 2-moves only. There exists a positive integer L such that $\mathcal{S}$ has at least $\Omega((N - L + 1)/\log n)$ many windows $W = (W_1, \ldots, W_L)$ of length L such that at least $\Omega(L/\log n)$ moves $W_i = \{u, v\}$ of W satisfy*

$$\log^3 n \leq \#_W(u) \leq \log^7 n \quad \textit{and} \quad \#_W(v) \geq \log^3 n \tag{4.13}$$

Proof. See proof in the full version. ∎

5 Finding Cycles

Let $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_N)$ be a valid move sequence of length $N = n \log^{10} n$ that consists of 2-moves only. By Lemma 4.2, $\mathcal{S}$ has a window $W = (W_1, \ldots, W_L)$ of length L such that the number of moves in W that satisfy (4.13) is at least $\Omega(L/\log n)$. We show in this section that such a W satisfies

$$\mathsf{rank}_{\mathsf{cycles}}(W) = \Omega\left(\frac{L}{\log^{10} n}\right). \tag{5.14}$$

This will finish the proof of Lemma 3.2 when $\mathcal{S}$ consists of 2-moves only.

To this end, let $\tau_0 \in \{\pm 1\}^{V(W)}$ be the configuration with $\tau_0(u) = -1$ for all $u \in V(W)$ so that we can work on vectors $\mathsf{imprv}_{\tau_0,W}(i)$ and $\mathsf{imprv}_{\tau_0,W}(C)$ for useful cycles of W (at the same time, recall from Lemma 2.5 that $\mathsf{rank}_{\mathsf{cycles}}(W)$ does not depend on the choice of τ_0). Let $\tau_0, \ldots, \tau_L$ denote the sequence of configurations induced by W.

Next, let us construct an auxiliary graph $H = (V(W), E)$, where every move $W_i = \{u, v\}$ adds an edge between u and v in E. Note that we allow parallel edges in H so $|E| = L$ and $\#_W(u)$ is exactly the degree of u in H. There is also a natural one-to-one correspondence between cycles of W and cycles of H. The following lemma shows the existence of a nice looking bipartite graph in H:

LEMMA 5.1. *There are two disjoint sets of nodes $V_1, V_2 \subset V(W)$ and a subset of edges $E' \subseteq E$ such that*

1. *Every edge in E' has one node in V_1 and the other node in V_2;*
2. *$|V_1 \cup V_2| = O(L/\log^3 n)$ and $|E'| = \Omega(L/\log n)$;*
3. *$\#_W(u) \leq \log^7 n$ for every node $u \in V_1$.*

Proof. See proof in the full version. ∎

Recall the definition of useful cycles of W (and their cancellation vectors) from Section 2. Since we only care about the rank of vectors induced by useful cycles, we give the following definition which classify each edge of H into two types and then use it to give a sufficient condition for a cycle of W to be useful:

Definition 5.1. *We say the i-th move $W_i = \{u, v\}$ of W is of the same sign if $\tau_i(u) = \tau_i(v)$, and is of different signs if $\tau_i(u) \neq \tau_i(v)$.*

Lemma 5.2. *Let $C = (c_1, \ldots, c_t)$ be a cycle of W and assume that t is even. If all of $W_{c_1}, \ldots, W_{c_t}$ are of different signs, then C is a useful cycle; If all of $W_{c_1}, \ldots, W_{c_t}$ are of the same sign, then C is a useful cycle of W.*

Proof. Recall the Usefulness Criterion (Remark 2.1)

$$C \text{ is a useful cycle of } W \Leftrightarrow (-1)^t = \tau_{c_t}(u_1)\tau_{c_1}(u_1) \cdot \prod_{i=2}^{t} \tau_{c_{i-1}}(u_i)\tau_{c_i}(u_i)$$

If all of $W_{c_1}, \ldots, W_{c_t}$ are of different signs, then $\tau_{c_j}(u_j)\tau_{c_j}(u_{j+1}) = \tau_{c_t}(u_1)\tau_{c_t}(u_t) = -1$, and the above expression equals to $(-1)^t = (-1)^t$. If all of $W_{c_1}, \ldots, W_{c_t}$ are of the same sign, then $\tau_{c_j}(u_j)\tau_{c_j}(u_{j+1}) = \tau_{c_t}(u_1)\tau_{c_t}(u_t) = 1$, the above expression is also $1 = (-1)^t$, which holds since t is even. ∎

We assume in the rest of the proof that at least half of edges in E' are of the same sign; the case when at least half of E' are of different signs can be handled similarly. Let E'' be the subset of E' that consists of edges of the same sign, with $|E''| \geq |E'|/2$. In the following discussion, cycles in E'' always refer to cycles that do not use the same edge twice (parallel edges are counted as different edges, since they correspond to different moves in the window W).

The aforementioned discussion leads to the following corollary which reduces the existence of useful cycle of W to a simple cycle in auxiliary graph H:

Corollary 5.1. *Since every cycle in a bipartite graph has even length, every cycle in E'' corresponds to a useful cycle of W. For convenience, given any cycle C of E'' we will write $\mathsf{imprv}_{\tau_0,W}(C)$ to denote the vector of its corresponding useful cycle of W.*

We first deal with the case when E'' contains many parallel edges:

Lemma 5.3. *Let D be the subset of nodes in V_1 that have parallel edges in E''. Then $\mathsf{rank}_{\mathsf{cycles}}(W) \geq |D|/2$.*

Proof. We prove the lemma even if the sequence contains both 1-moves and 2-moves so that we can use it also in the general case in the next section. We note first that if $S_i = \{u, v\}$, $S_j = \{u, v\}, i < j$ are two moves that involve the same two nodes, then there is at least one node $z \neq u, v$ that appears an odd number of times between the two moves. This follows from the definition of a valid move sequence.

We will construct a set Q of at least $|D|/2$ 2-cycles, where each 2-cycle consists of two parallel edges in E''. We use the following procedure.

1. While there is a 2-cycle (u, v) with $u \in D$, $v \in V_2$, such that some node $z \neq u$ of V_1 moves an odd number of times between the two $\{u, v\}$ moves of the 2-cycle, pick any such 2-cycle (u, v) and add it to our set Q, pick any such node $z \neq u$ that moves an odd number of times between the two $\{u, v\}$ moves, and delete u and z from D (if z is in D).
2. Suppose now that there are no more 2-cycles as in step 1. While D is not empty, let u be any remaining node in D, take any two incident parallel edges $\{u, v\}$ in E'', add the corresponding 2-cycle to Q, and delete u from D.

Firstly, notice that for every new entry at Q in the procedure, we delete at most 2 nodes from D. Hence, this procedure will generate clearly a set Q of at least $|D|/2$ 2-cycles. Let $(u_1, v_1), (u_2, v_2), \ldots, (u_k, v_k)$ be the sequence of 2-cycles selected, where the first d were selected in step 1, and the rest in step 2. The nodes u_i are distinct, while the nodes v_i may not be distinct. For each $i = 1, \ldots, d$, let z_i be the node in V_1 that appears an odd number of times between the two $\{u_i, v_i\}$ moves that was selected by the algorithm. Note that node $z_i \neq u_j$ for all $j \geq i$, since z_i was deleted from D when u_j was selected. For each $i = d + 1, \ldots, k$, let z_i be any node, other than u_i, v_i, that appears an odd number of times between the two $\{u_i, v_i\}$ moves. Then z_i is not in V_1 because in step 2 there are no odd nodes in V_1. For each $i = 1, \ldots, k$, we view the edge $\{u_i, z_i\}$ of the complete graph as a witness for the 2-cycle (u_i, v_i).

Consider the matrix with columns corresponding to the selected 2-cycles (u_i, v_i), $i = 1, \ldots, k$, and rows corresponding to the witness edges $\{u_i, z_i\}$. The entry for the corresponding witness edge $\{u_i, z_i\}$ is nonzero. Indeed, by definition 2.3, $\mathsf{imprv}_{\tau_0,W}(C = (u_i, v_i))_{\{u_i,z_i\}} = -b_1(\tau_{c_1}(u_i)\tau_{c_1}(z_i)) - b_2(\tau_{c_2}(u_i)\tau_{c_2}(z_i))$

$$\text{and } \begin{cases} \tau_{c_1}(z_i) = -\tau_{c_2}(z_i) \\ b_1 = 1 \ \& \ b_2 = -\tau_{c_1}(v_i)\tau_{c_2}(v_i) \\ \tau_{c_m}(v_i) = \tau_{c_m}(u_i), \text{ for } m \in \{1,2\} \end{cases} \quad \text{which yields } \mathsf{imprv}_{\tau_0,W}(C = (u_i, v_i))_{\{u_i,z_i\}} = 2 \cdot \tau_{c_1}(z) \neq 0.$$

Consider the column for a 2-cycle (u_i, v_i) selected in step 1. The entry for any other witness edge $\{u_j, z_j\}$ with $j < i$ is 0 because $u_j, z_j \neq u_i, v_i$. (The entries for witness edges $\{u_j, z_j\}$ with $j > i$ could be nonzero.)

Consider the column for a 2-cycle (u_i, v_i) selected in step 2. The entry for any witness edge $\{u_j, z_j\}$ from step 1 (i.e. with $j \leq d$) is 0 because $u_j, z_j \neq u_i, v_i$. The entry for any witness edge $\{u_j, z_j\}$ from step 2 (i.e. with $j > d$) is also 0 because (1) $u_j \neq u_i, v_i$, (2) $z_j \notin V_1$ hence $z_j \neq u_i$, and (3), even if $z_j = v_i$, all nodes of V_1 –hence also u_j –occur an even number of times between the two $\{u_i, v_i\}$ moves, therefore the entry for $\{u_j, v_i\}$ is 0.

$$\begin{cases} \mathcal{M}_{1\to d}^{[\text{step 1}]} = \begin{bmatrix} \mathsf{imprv}_{\tau_0,W}(C_1)_{\{x_1,y_1\}} \neq 0 & 0 & 0 \\ * & \ddots & \vdots \\ * & * & \mathsf{imprv}_{\tau_0,W}(C_d)_{\{x_d,y_d\}} \neq 0 \end{bmatrix} \\ \mathcal{M}_{d+1\to k}^{[\text{step 2}]} = \mathrm{diag}_{t\in(d+1)\to k}(\mathsf{imprv}_{\tau_0,W}(C_t)_{\{x_t,y_t\}} \neq 0) \end{cases} \Rightarrow \mathcal{M} = \begin{bmatrix} \mathcal{M}_{1\to d}^{[\text{step 1}]} & \mathbf{0} \\ * & \mathcal{M}_{d+1\to k}^{[\text{step 2}]} \end{bmatrix}$$

Thus, the matrix with columns corresponding to the selected 2-cycles (u_i, v_i) and rows corresponding to their witness edges $\{u_i, z_i\}$ is a lower triangular matrix with non-zero diagonal entries. It follows that the columns are linearly independent. ∎

As a result, it suffices to deal with the case when $|D|$ is $o(L/\log^8 n)$. Let E^* denote the subset of edges obtained from E'' after deleting all nodes of D and their incident edges. The remaining bipartite graph has no parallel edges. Then we have

$$|E^*| \geq |E''| - |D| \cdot \log^7 n = \Omega(L/\log n).$$

We list all properties of the bipartite graph $H^* = (V_1 \cup V_2, E^*)$ we need as follows:

1. H^* is a bipartite graph with no parallel edges;
2. $|V_1 \cup V_2| \leq O(L/\log^3 n)$ and $|E^*| \geq \Omega(L/\log n)$; and
3. $\#_W(u) \leq \log^7 n$ for every node $u \in V_1$.
4. Every edge $e = \{u, v\} \in E^*$ corresponds to a move $W_i = \{u, v\}$ which is of the same sign.

Recall that $E(W)$ denotes the set of edges in K_n which have both nodes in $V(W)$. These edges are indices of $\mathsf{imprv}_{\tau_0,\mathcal{S}}(i)$ and $\mathsf{imprv}_{\tau_0,\mathcal{S}}(C)$ for a given useful cycle C of W. Our main lemma is the following:

LEMMA 5.4. *Fix an arbitrary $s \in [0 : L/\log^{10}(n)]$. Assume additionally that there exists a set of edges $\mathcal{E}_s = \{\{x_1, y_1\}, \cdots \{x_s, y_s\}\} \subseteq E(W)$ such that $x_i \in V_1$ for all $i \in [s]$. Then there exists a cycle C in H^* and an edge $\{u, v\} \in E(W)$ with $u \in V_1 \setminus \{x_1, y_1, \ldots, x_s, y_s\}$ such that*

$$\left(\mathsf{imprv}_{\tau_0,W}(C)\right)_{\{u,v\}} \neq 0 \quad \textit{and} \quad \left(\mathsf{imprv}_{\tau_0,W}(C)\right)_{\{x_i,y_i\}} = 0, \quad \textit{for all } i \in [s]. \tag{5.15}$$

We can use Lemma 5.4 to prove (5.14):

Proof. [Proof of (5.14) Assuming Lemma 5.4]

Start with $\mathcal{E}_{s=0} = \emptyset$, For integer s going from 0 to $\lfloor L/\log^{10} n \rfloor$, using Lemma 5.4, find cycle C_{s+1} and an edge $\{u, v\}$ satisfying (5.15), let $\mathcal{E}_{s+1} = \mathcal{E}_s \cup \{x_{s+1}, y_{s+1}\} = \{u, v\}$ and repeat the above process.

In the end, we get a set of cycles $C_1, \cdots, C_k$ where $k = \lfloor L/\log^{10} n \rfloor$. And for any $j \in [k]$, we have

$$\left(\mathit{imprv}_{\tau_0,W}(C_j)\right)_{\{x_j,y_j\}} \neq 0 \quad \text{and} \quad \left(\mathit{imprv}_{\tau_0,W}(C_j)\right)_{\{x_i,y_i\}} = 0, \quad \text{for all } i \in [j-1].$$

Let $\mathcal{M}$ be the $k \times k$ square matrix where $\mathcal{M}_{ij} = \left(\mathit{imprv}_{\tau_0,W}(C_j)\right)_{\{x_i,y_i\}}$.

$$\mathcal{M} = \begin{bmatrix} \mathsf{imprv}_{\tau_0,W}(C_1)_{\{x_1,y_1\}} \neq 0 & 0 & 0 & \cdots & 0 \\ * & \mathsf{imprv}_{\tau_0,W}(C_2)_{\{x_2,y_2\}} \neq 0 & 0 & \cdots & 0 \\ \vdots & * & \mathsf{imprv}_{\tau_0,W}(C_3)_{\{x_3,y_3\}} \neq 0 & \cdots & 0 \\ \vdots & \vdots & * & \ddots & 0 \\ * & * & * & * & \mathsf{imprv}_{\tau_0,W}(C_k)_{\{x_k,y_k\}} \neq 0 \end{bmatrix}$$

As we can see, the matrix is lower triangular with non-zero diagonal entries, so it has full rank k. Note that $\mathcal{M}$ is a submatrix of the matrix formed by taking $\mathit{imprv}_{\tau_0,W}(C_j)$ as column vectors, therefore we have $\mathsf{rank}_{\mathsf{cycles}}(W) \geq k \geq L/\log^{10} n$. ∎

5.1 Proof of Lemma 5.4 Given a cycle C in H^*, we say $\{u, v\} \in E(W)$ is a *witness* of C if

$$\left(\mathsf{imprv}_{\tau_0,W}(C)\right)_{\{u,v\}} \neq 0.$$

So the goal of Lemma 5.4 is to find a cycle C of H^* such that none of $(u_i, v_i) \in \mathcal{E}_s$ are witnesses of C and at the same time, C has a witness edge $\{u, v\}$ with u being a new node in V_1 not seen in $\mathcal{E}_s$ before. The proof consists of two steps. First we introduce a so-called *split auxiliary graph* G using H^* and $\mathcal{E}_s$, by deleting certain nodes and creating extra copies of certain nodes in H^*. We show in Lemma 5.5 that certain simple cycles in G correspond to cycles in H^* that don't have any edge in $\mathcal{E}_s$ as witnesses. Next we show in Lemma 5.7 how to find such a simple cycle in G that has a new witness (u, v) such that $u \in V_1$ and does not appear in $\mathcal{E}_s$.

Let $\mathsf{wit}_1(\mathcal{E}_s)$ be the set of $u \in V_1$ that appear in $\mathcal{E}_s$ and let $\mathsf{wit}_2(\mathcal{E}_s)$ be the set of $v \in V_2$ that appear in $\mathcal{E}_s$. For each $v \in \mathsf{wit}_2(\mathcal{E}_s)$, we write $\mathsf{wit}_1(v) \neq \emptyset$ to denote the set of nodes $u \in \mathsf{wit}_1(\mathcal{E}_s)$ such that $(u, v) \in \mathcal{E}_s$, and let k_v denote the number of moves in W that involve at least one node in $\mathsf{wit}_1(v)$. We have $k_v \leq |\mathsf{wit}_1(v)| \cdot \log^7 n$ since $\#_W(u) \leq \log^7 n$ for all $u \in V_1$. Below, we give in Fig. 4 an example of such an auxiliary graph:

We now define our split auxiliary (bipartite) graph G. We start with its set of nodes $V_1' \cup V_2'$:

1. $V_1' = V_1 \setminus \mathsf{wit}_1(\mathcal{E}_s)$; and
2. $V_2' = \cup_{v \in V_2} C(v)$, where $C(v) = \{v^{(0)}\}$ if $v \notin \mathsf{wit}_2(\mathcal{E}_s)$ and $C(v) = \{v^{(0)}, v^{(1)}, \ldots, v^{(k_v)}\}$ if $v \in \mathsf{wit}_2(\mathcal{E}_s)$.

So we deleted nodes $\mathsf{wit}_1(\mathcal{E}_s)$ from V_1 and replaced each node $v \in \mathsf{wit}_2(\mathcal{E}_s)$ by $k_v + 1$ new nodes. Next we define the edge set $E(G)$ of G. Every move $W_i = \{u, v\}$ in W that corresponds to an edge (u, v) in H^* with $u \in V_1 \setminus \mathsf{wit}_1(\mathcal{E}_s)$ and $v \in V_2$ will add an edge in G as follows:

1. If $v \notin \mathsf{wit}_2(\mathcal{E}_s)$, then we add $(u, v^{(0)})$ to G; and
2. Otherwise ($v \in \mathsf{wit}_2(\mathcal{E}_s)$), letting $\mu_i \in [0 : k_v]$ be the number of moves before W_i that contain at least one node in $\mathsf{wit}_1(v)$ (note that W_i does not contain $\mathsf{wit}_1(v)$; actually W_i cannot contain $\mathsf{wit}_1(\mathcal{E}_s)$), we add $(u, v^{(\mu_i)})$ to G.

Therefore, every edge in G corresponds to a move in W which corresponds to an edge in H^* that does not contain a node in $\mathsf{wit}_1(\mathcal{E}_s)$. It is clear that each simple cycle of G corresponds to a cycle of H, which in turn corresponds to a useful cycle of W (Since we assume w.l.o.g that all edges of auxiliary graph, and its split one, correspond to moves of the same sign (See Corollary 5.1)). So $\mathsf{imprv}_{\tau_0,W}(C)$ is well defined for simple cycles C of G. Our motivation for constructing and working on G is because of the following lemma:

LEMMA 5.5. *Let C be a simple cycle of G. Then none of the edges in $\mathcal{E}_s$ is a witness of C.*

Proof. See proof in the full version. ∎

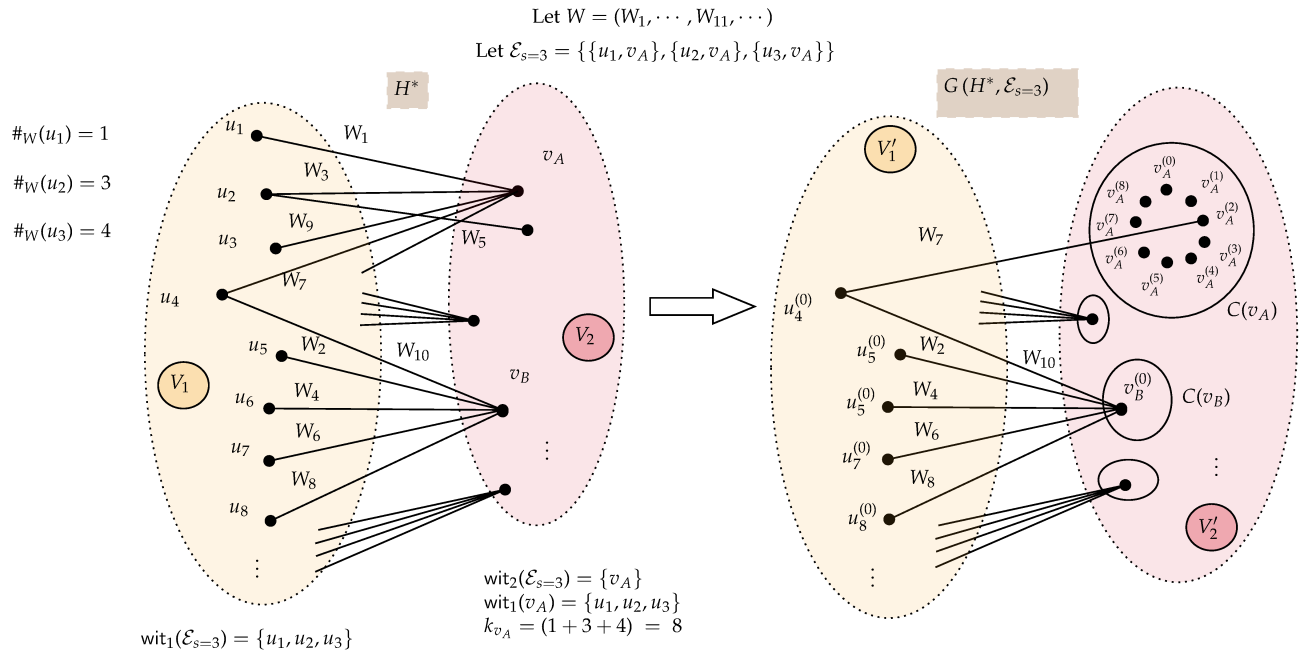

Figure 4: An exemplifying case of an auxiliary graph H^* and splitting graph $G(H^*, \mathcal{E}_s)$

To finish the proof, it suffices now to find a simple cycle C of G that has a witness $(u, v) \in E(W)$ with one of its vertices $u \in V_1'$. We start by checking that all conditions for H^* still hold for G. It is clear that G is a bipartite graph with no parallel edges. By the definition of $\mathsf{wit}_1(v)$ for each $v \in \mathsf{wit}_2(\mathcal{E}_s)$, we have $\sum_{v \in \mathsf{wit}_2(\mathcal{E}_s)} |\mathsf{wit}_1(v)| \le 2s$, also $|\mathsf{wit}_1(\mathcal{E}_s)| \le 2s$. The number of nodes $|V_1' \cup V_2'|$ in G is at most

$$O(L/\log^3 n) + \sum_{v \in \mathsf{wit}_2(\mathcal{E}_s)} k_v \le O(L/\log^3 n) + \sum_{v \in \mathsf{wit}_2(\mathcal{E}_s)} |\mathsf{wit}_1(v)| \cdot \log^7 n = O(L/\log^3 n).$$

where the last equality used that $s \le L/\log^{10} n$. The number of edges in G is at least

$$\Omega(L/\log n) - |\mathsf{wit}_1(\mathcal{E}_s)| \cdot \log^7 n = \Omega(L/\log n).$$

Let's work on another preprocessing of G to simplify the proof. Note that the average degree of nodes in G is at least $\Omega\left((L/\log n)/(L/\log^3 n)\right) = \Omega(\log^2 n)$. The following simple lemma shows that one can clean up G to get a bipartite graph G^* such that every node has degree at least $100 \log n$ and the number of edges in G^* remains to be $\Omega(L/\log n)$:

LEMMA 5.6. *There is a bipartite graph $G^* = (V_1^* \cup V_2^*, E(G^*))$ with $V_1^* \subseteq V_1'$, $V_2^* \subseteq V_2'$ and $E(G^*) \subseteq E(G)$ such that every node in G^* has degree at least $100 \log n$ and $|E(G^*)| = \Omega(L/\log n)$.*

Proof. See proof in the full version. □

Let us list the properties of $G^* = (V_1^* \cup V_2^*, E(G^*))$ we will use in the rest of the proof:

1. $V_1^* \subseteq V_1' = V_1 \setminus \mathsf{wit}_1(\mathcal{E}_s)$ and $V_2^* \subseteq V_2'$ so each node in V_2^* is in $C(v)$ for some $v \in V_2$.
2. The degree of any node is at least $100 \log n$; and
3. For any $u \in V_1^*$ and $v \in V_2$, the number of neighbors of u in $V_2^* \cap C(v)$ is at most one.
4. $E(G^*)$ has no parallel edges, $|E(G^*)| \ge \Omega(L/\log n)$ and w.l.o.g. each edge in $E(G^*)$ correspond to a move of same sign.

We prove the following lemma to finish the proof:

LEMMA 5.7. *Let $u \in V_1^*$ and $v \neq v' \in V_2^*$ such that $(u, v^{(j)}), (u, v'^{(j')}) \in E(G^*)$ for some j and j', and the corresponding moves $W_i = \{u, v\}$ and $W_{i'} = \{u, v'\}$ in W are not consecutive*[7]*. Then, the graph G^* has a simple cycle C such that C has a witness $e = \{u, w\} \in E(W)$ with $w \in V_1^*$.*

Proof. We begin with a simple sufficient condition for a simple cycle of G^* to satisfy the above condition.

First, let $u \in V_1^*$ and $v \neq v' \in V_2^*$ such that $(u, v^{(j)}), (u, v'^{(j')}) \in E(G^*)$ for some j and j', and the corresponding moves $W_i = \{u, v\}$ and $W_{i'} = \{u, v'\}$ in W are not consecutive. Assume that $i < i'$ without loss of generality; then $i + 1 < i'$. We show:

CLAIM 5.1. *There is a node $w \notin \{u, v, v'\}$ that appears in an odd number of moves in $W_{i+1}, \ldots, W_{i'-1}$.*

Proof. See proof in the full version. ∎

We remark that Claim 5.1 holds even when W is a mixture of 1-moves and 2-moves. This will be important when we deal with the general case in Section 6.

We write $w^*(u, v^{(j)}, v'^{(j')}) \in V(W)$ to denote such a node w promised in the above claim (if more than one exist pick one arbitrarily). The next claim gives us a sufficient condition for a simple cycle C of G^* to satisfy the condition of the lemma:

CLAIM 5.2. *Let*

$$C = u_1 v_1^{(j_1)} u_2 v_2^{(j_2)} \cdots u_k v_k^{(j_k)} u_1$$

be a simple cycle of G^* for some nonnegative integers $j_1, \ldots, j_k$. Suppose for some $i \in [k]$ we have that $w := w^*(u_i, v_{i-1}^{(j_{i-1})}, v_i^{(j_i)}) \in V(W)$ does not appear in C (where $v_{i-1}^{(j_{i-1})}$ denotes $v_k^{(j_k)}$ if $i = 1$), i.e., $w \notin \{u_1, \ldots, u_k, v_1, \ldots, v_k\}$, then $(u_i, w) \in E(W)$ must be a witness of C.

Proof. See proof in the full version. ∎

Finally we prove the existence of a simple cycle C of G^* that satisfies the condition of the above claim. To this end, we first review a simple argument which shows that any bipartite graph with n nodes and minimum degree at least $100 \log n$ must have a simple cycle. Later we modify it to our needs.

The argument goes by picking an arbitrary node in the graph as the root and growing a binary tree of $\log n$ levels as follows:

1. In the first round we just add two distinct neighbors of the root in the graph as its children.
2. Then for each round, we grow the tree by one level by going through its current leaves one by one to add two children for each leaf. For each leaf u of the current tree we just pick two of its neighbors in the graph that do not appear in ancestors of u and add them as children of u. Such neighbors always exist since the tree will have no more than $\log n$ levels and each node has degree at least $100 \log n$ in the graph.

Given that there are only n nodes in the graph, there must be a node that appears more than once in the tree at the end. Let's consider the first moment when we grow a leaf by adding one child (labelled by u) and u already appeared in the tree. Note that the two nodes labelled by u are not related in the tree since we maintain the invariant that the label of a node does not appear in its ancestors. Combining paths from these two nodes to the first node at which the two paths diverge, we get a simple cycle of the graph.

We now adapt the above argument to prove the existence of a simple cycle C of G^* that satisfies the condition of Claim 5.2 by building a binary tree of $2 \log n$ levels as follows. We start with an arbitrary node $u_{\mathsf{root}} \in V_1^*$ as the root of the tree and expand the tree level by level, leaf by leaf, as follows:

[7] It is worth mentioning, that V_2^* always includes at least two vertices which are copies from different initial nodes v, v'. Indeed, if G^* was actually a star graph around $V_2^* = \{v^*\}$, then $\Omega(L/\log n) = E(G^*) = \Theta(V(G^*)) = O(L/\log^3 n)$, which leads to a contradiction. Additionally, notice that $v^{(j)}$ and $v'^{(j')}$ correspond to different nodes in the initial graph, otherwise the initial auxiliary graph H^* would have parallel edges.

1. Case 1: The leaf we would like to grow is labelled a node $u \in V_1^*$. In this case we add two children as follows. Let $u_1 v_1^{(j_1)} \cdots u_{k-1} v_{k-1}^{(j_{k-1})} u$ be the path from the root (u_1) to u in the tree, where $u_1, \ldots, u_{k-1}, u \in V_1^*$ and $v_1^{(j_1)}, \ldots, v_{k-1}^{(j_{k-1})} \in V_2^*$. We pick two neighbors $v^{(j)}, v'^{(j')}$ of u in G^* with distinct $v, v' \in V_2$ as its children in the tree. We would like v and v' to satisfy the following two properties: (1) v and v' do not lie in $\{v_1, v_2, \cdots, v_{k-1}\}$, (2) v and v' are different from $w^*(u_i, v_i^{(j_i)}, v_i'^{(j_i')})$ for every $i = 1, \ldots, k-1$, where $v_i'^{(j_i')}$ denotes the other child of v_i in the tree and (3) the move corresponding to $\{u, v^{(j)}\}$ in W and the move corresponding to $\{u, v'^{(j')}\}$ in W are not consecutive moves. The existence of $v^{(j)}$ and $v'^{(j')}$ that satisfy (1), (2) and (3) follows trivially from the fact that every node (in particular, u here) has degree at least $100 \log n$ in G^*. Indeed, to satisfy (2) and (3), for each time we may reject at most 2 possible leafs. Given that the tree will only grow for $2 \log n$ levels –the half of times with V_1^* leafs and the rest half with V_2^* –, we have $k \leq \log n$ and there are at most $2k \leq 2 \log n$ edges of u that need to be avoided. Moreover, no two edges from u go two the same $C(v)$ for some $v \in V_2$ (Because we don't allow parallel edges).

2. Case 2: The leaf we would like to grow is labelled a node $v^{(j)} \in V_2^*$. In this case we just add one neighbor $u \in V_1^*$ of $v^{(j)}$ as its only child. Let $u_1 v_1^{(j_1)} \cdots v_{k-1}^{(j_{k-1})} u_k v^{(j)}$ be the path from the root to $v^{(j)}$. We pick a neighbor $u \in V_1^*$ of $v^{(j)}$ in G^* that satisfies (1) $u \notin \{u_1, \cdots, u_k\}$ and (2) u is different from $w^*(u_i, v_i^{(j_i)}, v_i'^{(j_i')})$ for every $i = 1, \ldots, k-1$ and u is different from $w^*(u_k, v^{(j)}, v'^{(j')})$, where $v_i'^{(j_i')}$ denotes the other child of u_i and $v_i'^{(j_i')}$ denotes the other child of u_k in the tree. The existence of such u follows from the same argument as Case 1.

Given that the tree has $2 \log n$ levels and there are only n nodes, there must be a node that appears more than once in the tree at the end, and let's consider the first moment when we grow a leaf by adding a child and the same node already appeared in the tree. Similarly we trace the two paths and let $u \in V_1^*$ be the node where the two paths diverge; note that given the construction of the tree, this node must be a node in V_1^*, given that nodes in V_2^* only have one child in the tree. On the one hand, the way we construct the tree makes sure that combining the two paths leads to a simple cycle C of G^*. On the other hand, let $v^{(j)}, v'^{(j')} \in V_2^*$ be the two children of u (which are next to u on the cycle). Then it is easy to verify that $w^*(u, v^{(j)}, v'^{(j')})$ does not appear on the cycle we just found.

This ends the proof of the lemma. ∎

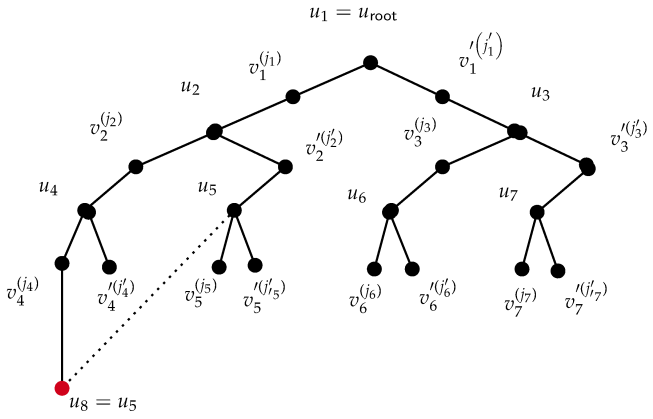

Figure 5: Example of Finding-Cycle Process.

6 General Case

We prove Lemma 3.2 for the general case. Let $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_N)$ be a valid move sequence of length $N = n \log^{10} n$ that consists of both 1-moves and 2-moves. We will consider two cases and deal with them separately: (1) the number of 1-moves in $\mathcal{S}$ is at least $N/\log^5 n$; and (2) the number of 1-moves is at most $N/\log^5 n$.

6.1 Case 1 We consider the case when there are at least $N/\log^5 n$ many 1-moves. In this case we show that there is a window W of $\mathcal{S}$ such that $\mathsf{rank}_{\mathsf{arcs}}(W)$ is large. The arguments used in this case are similar to those used in [22, 7, 10]. Given a window W of $\mathcal{S}$, we write $V_2(W)$ to denote the set of nodes $u \in V(W)$ such that at least two 1-moves in W are $\{u\}$.

Lemma 6.1. *There is a window W of $\mathcal{S}$ such that*

$$|V_2(W)| = \Omega\left(\frac{\mathsf{len}(W)}{\log^6 n}\right).$$

Proof. See proof in the full version. □

Lemma 6.2. *We have $\mathsf{rank}_{\mathsf{arcs}}(W) \geq \Omega(|V_2(W)|)$.*

Proof. See proof in the full version. □

6.2 Case 2 Let $\mathcal{S}$ be a valid move sequence of length N with no more than $N/\log^5 n$ many 1-moves. Let W be a window of $\mathcal{S}$. We write $\#_W(u)$ to denote the number of moves (including both 1-moves and 2-moves) that u appears in W, and write $\#^2_W(u)$ to denote the number of 2-moves that u appears in W.

We start by showing a lemma similar to Lemma 4.2 in Section 4.

Lemma 6.3. *Let $\mathcal{S}$ be a valid move sequence of length $N = n \log^{10} n$ with no more than $N/\log^5 n$ many 1-moves. Then there exists a window W of $\mathcal{S}$ such that at least $\Omega(\mathsf{len}(W)/\log n)$ many moves of W are 2-moves $W_i = \{u, v\}$ that satisfy*

$$\log^3 n \leq \#^2_W(u) \leq \#_W(u) \leq 2\log^7 n \quad \text{and} \quad \#^2_W(v) \geq \log^3 n \tag{6.16}$$

Proof. See proof in the full version. □

So we now have a valid move sequence W of length L, as a window of the original valid sequence $\mathcal{S}$, such that the number of 2-moves in W that satisfy (6.16) is at least $\Omega(L/\log n)$. The rest of the proof follows the same arguments used in Section 5. We give a sketch below.

First we define the same auxiliary graph $H = (V(W), E)$ such that there is a one-to-one correspondence between E and 2-moves in W. Note that the degree of a node u in H is the same as $\#^2_W(u)$.

We then show that there are disjoint sets of nodes $V_1, V_2 \subset V(W)$ and a subset of edges $E' \subseteq E$ that satisfy conditions similar to those of Lemma 5.1:

Lemma 6.4. *There are two disjoint sets of nodes $V_1, V_2 \subset V(W)$ and a subset of edges $E' \subseteq E$ such that*

1. *Every edge in E' has one node in V_1 and the other node in V_2;*
2. *$|V_1 \cup V_2| = O(L/\log^3 n)$ and $|E'| = \Omega(L/\log n)$;*
3. *$\#_W(u) \leq 2\log^7 n$ for every node $u \in V_1$.*

The proof is exactly the same as that of Lemma 5.1, except that we define V to be the set of nodex v with $\#^2_W(v) \geq \log^3 n$ and V_h to be the set of nodes v with $\#_W(v) \geq 2\log^7 n$.

Next we focus on E'', which contains all edges in E' of the same sign (or edges in E' of different signs, whichever contains more edges). Similarly every cycle in E'' corresponds to a useful cycle of W. The case when E'' contains many parallel edges can be handled exactly the same way as in Lemma 5.3. So we may delete all parallel edges from E'', and finish the proof using Lemma 5.4.

The proof of Lemma 5.4 for the general case is very similar, with the following changes:

1. In the definition of k_v for each $v \in \mathsf{wit}_2(\mathcal{E}_s)$, we need it to be the number of moves (including both 1-moves and 2-moves) in W that involve at least one node in $\mathsf{wit}_1(v)$. This can still be bounded from above by $|\mathsf{wit}_1(v)| \cdot 2\log^7 n$ since we have $\#_W(u) \leq \log^7 n$ for all $u \in V_1$ as promised in Lemma 6.4 above.

2. As we commented earlier, Claim 5.1 works even when W consists of both 1-moves and 2-moves.

This finishes the proof of Lemma 3.2 for the general case.

7 Binary Max-CSP and Function Optimization Problems

We recall the definition of binary maximum constraint satisfaction problems, and more generally function optimization problems.

DEFINITION 7.1. *An instance of Binary Max-CSP (Constraint Satisfaction Problem), or MAX 2-CSP, consists of a set $V = \{x_1, \ldots, x_n\}$ of variables that can take values over $\{0, 1\}$ and a set $C = \{c_1, \ldots, c_m\}$ of constraints with given respective weights $w_1, \ldots, w_m$, where each constraint is a predicate on a pair of variables. The MAX 2-CSP problem is: given an instance, find an assignment that maximizes the sum of the weights of the satisfied constraints.*

Several problems can be viewed as special cases of Binary Max-CSP where the predicates of the constraints are restricted to belong to a fixed family $\mathcal{P}$ of predicates; this restricted version is denoted Max-CSP($\mathcal{P}$). For example, the Max Cut problem in graphs is equivalent to Max-CSP($\mathcal{P}$) where $\mathcal{P}$ contains only the "not-equal" predicate ($x \neq y$, where x, y are the two variables). The Max Directed Cut problem, where the input graph is directed and we seek a partition of the nodes into two parts N_1, N_2 that maximizes the total weight of the edges directed from N_1 to N_2, corresponds to the case that $\mathcal{P}$ contains only the $<$ predicate (i.e. $x < y$). MAX 2SAT corresponds to the case that $\mathcal{P}$ consists of all 4 possible clauses on two variables.

A generalization of MAX 2-CSP is the class of *Binary function optimization problems* (BFOP) where instead of constraints (predicates) we have functions on two arguments that take values in $\{0, 1, \ldots, d\}$ instead of $\{0, 1\}$, where d is a fixed constant (or even is polynomially bounded). For convenience and consistency with the notation of configurations in the Max Cut problem, we will use in the following $\{-1, 1\}$ as the domain of the variables instead of $\{0, 1\}$. That is, the problem is: Given a set $V = \{x_1, \ldots, x_n\}$ of variables with domain $D = \{-1, 1\}$, a set $F = \{f_1, \ldots, f_m\}$ of functions, where each f_i is a function of a pair (x_{i_1}, x_{i_2}) of variables, and given respective weights $w_1, \ldots, w_m$, find an assignment $\tau : V \to D$ to the variables that maximizes $\sum_{i=1}^{m} w_i \cdot f_i(\tau(x_{i_1}), \tau(x_{i_2}))$.

Even though a function in BFOP (or a constraint in Max-2CSP) has two arguments, its value may depend on only one of them, i.e. it may be essentially a unary function (or constraint). More generally, it may be that the two arguments of the function can be decoupled and the function can be separated into two unary functions. We say that a binary function $f(x, y)$ is *separable* if there are unary functions f_1, f_2 such that $f(x, y) = f_1(x) + f_2(y)$ for all values of x, y; otherwise f is *nonseparable*. For binary domains there is a simple criterion for separability: a function $f(x, y)$ is separable if and only if $f(-1, -1) + f(1, 1) = f(-1, 1) + f(1, -1)$ [11]. If in a given BFOP instance some binary functions are separable, then we can decompose them into the equivalent unary functions. Thus, we may assume, without loss of generality, that a given BFOP instance has unary and binary functions, where all the binary functions are nonseparable. We say that an instance is *complete*, if every pair of variables appear as the arguments of a (nonseparable) binary function in the instance.

The 2-FLIP local search algorithm can be applied to a MAX 2-CSP or BFOP problem to compute a locally optimal assignment that cannot be improved by flipping the value of any one or two variables. We will show that the smoothed complexity of 2-FLIP for any complete MAX 2-CSP or BFOP instance is (at most) quasipolynomial.

THEOREM 7.1. *Let I be an arbitrary complete instance of MAX 2-CSP (or BFOP) with n variables and m constraints (functions) with independent random weights in $[-1, 1]$ with density at most $\phi > 0$. With probability at least $1 - o_n(1)$ over the draw of weights, every implementation of 2-FLIP terminates within $m\phi n^{O(\log^{10} n)}$ steps.*

Proof. Consider a (complete) instance I of a BFOP problem with n variables and m functions, and a sequence $\mathcal{S}$ of moves of 2-FLIP starting from an initial configuration. The proof follows the same structure as the proof for

Max Cut. The only thing that changes is the improvement vector in each step, which depends on the specific functions of the instance: the vector has one coordinate for each function f_i in the instance and the entry is equal to the change in the value of the function resulting from the move. Arcs and cycles of $\mathcal{S}$ are defined in the same way as in Max Cut, and the improvement vectors of arcs and cycles are defined in an analogous way from the improvement vectors of the moves.

The heart of the proof for Max Cut is Lemma 3.2 which showed that there is a window $\mathcal{W}$ and a set of arcs or a set of cycles of $\mathcal{W}$ whose improvement vectors have rank $\Omega(\frac{\mathsf{len}(\mathcal{W})}{\log^{10} n})$. We will show that the lemma holds for any BFOP problem.

We associate with the BFOP instance I the graph G where the nodes correspond to the variables of I and the edges correspond to the binary functions of I; since I is a complete instance, the graph G is the complete graph, possibly with multiple edges connecting the same pair of nodes (if there are multiple functions with the same pair of arguments). We will identify the variables of I with the nodes of G and the functions of I with the edges of G.

In the general case of the Max Cut problem, in Case 1 where there is a large number of 1-moves, we identified a window $\mathcal{W}$ and a large set A' of arcs in the window whose set of improvement vectors are linearly independent. The argument relied only on the zero-nonzero structure of the improvement vectors: it showed that the matrix M formed by these vectors and a set of rows corresponding to a certain set E' of witness edges is a lower triangular matrix with nonzero diagonal. Take a set F' of functions of I that contains for each edge $\{u, v\} \in E'$ a function $f_k(u, v)$ with this pair as arguments (it exists because the instance I is complete), and form the matrix M' with the set F' as rows and the set A' of arcs as columns. We will show that the matrix M' has the same zero-nonzero structure as M, thus it also has full rank.

Consider an arc of the move sequence $\mathcal{S}$ corresponding to two moves $\mathcal{S}_i = \{u\}$, $\mathcal{S}_j = \{u\}$, $i < j$, and a function f_k of I. If u is not one of the arguments of the function, then the corresponding entry of the improvement vector of the arc is obviously 0. If u is one of the argument, i.e. the k-th function is $f_k(u, v)$ (similarly if it is $f_k(v, u)$), then the corresponding entry of the improvement vector of the arc is $\gamma_i(u)[f_k(\gamma_i(u), \gamma_i(v)) - f_k(-\gamma_i(u), \gamma_i(v))] - \gamma_j(u)[f_k(\gamma_j(u), \gamma_j(v)) - f_k(-\gamma_j(u), \gamma_j(v))]$. If v moves an even number of times between $\mathcal{S}_i$ and $\mathcal{S}_j$, then $\gamma_i(v) = \gamma_j(v)$ and it follows that the entry is 0, both in the case that $\gamma_i(u) = \gamma_j(u)$ and in the case that $\gamma_i(u) = -\gamma_j(u)$. On the other hand, if v moves an odd number of times between $\mathcal{S}_i$ and $\mathcal{S}_j$, then $\gamma_i(v) = -\gamma_j(v)$ and it follows that the k-th entry of the improvement vector is $\gamma_i(u)[f_k(\gamma_i(u), \gamma_i(v)) - f_k(-\gamma_i(u), \gamma_i(v))] - \gamma_j(u)[f_k(\gamma_j(u), -\gamma_i(v)) - f_k(-\gamma_j(u), -\gamma_i(v))]$. Letting $\gamma_i(u) = a, \gamma_i(v) = b$, the entry is $a[f_k(a, b) + f_k(-a, -b) - f_k(-a, b) - f_k(a, -b)]$ (both when $\gamma_i(u) = \gamma_j(u)$ and when $\gamma_i(u) = -\gamma_j(u)$); this quantity is nonzero because f_k is nonseparable. Thus, the entry for $f_k(u, v)$ of the improvement vector of the arc is nonzero exactly when the entry of the arc in the Max Cut problem for the edge (u, v) is nonzero. It follows that the matrix M' has the same zero-nonzero structure as M, thus it also has full rank.

In Case 2 of the Max Cut problem, where the number of 2-moves is very large, there were two subcases. In the first subcase, where there are many parallel edges in the graph that we associated with the window of the move sequence, we found a large set of 2-cycles whose improvement vectors were linearly independent. In the other case, where there are "few" parallel edges, we constructed a large set of cycles (of length $O(\log n)$), again with linearly independent improvement vectors. In both cases, the proof of linear independence relied again only on the zero-nonzero structure of the vectors, and not on the precise value of the entries. We will argue that in both cases, the corresponding vectors of these cycles in the BFOP instance I have the same zero-nonzero structure.

In the first subcase we found many 2-cycles $(u_1, v_1), \ldots, (u_k, v_k)$, and corresponding "witness" edges $(u_1, z_1), \ldots, (u_k, z_k)$ such that the matrix M with rows corresponding to the witness edges and columns corresponding to the 2-cycles in the Max Cut problem is lower triangular with nonzero diagonal. The nodes u_i are distinct (the v_i and the z_i may not be distinct) and $z_i \neq u_i, v_i$ for all i. For each witness pair (u_i, z_i) pick a function f_{r_i} of instance I with this pair of variables as arguments, in either order, say wlog the function is $f_{r_i}(u_i, z_i)$. Consider the matrix M' with rows corresponding to the functions $f_{r_i}(u_i, z_i)$ and columns corresponding to the 2-cycles (u_i, v_i). Note that the entry $M(j, i)$ is nonzero if one of the nodes u_j, z_j is in $\{u_i, v_i\}$ and the other node appears an odd number of times between the two moves $\{u_i, v_i\}$, and it is 0 otherwise,

i.e. if $\{u_j, z_j\} \cap \{u_i, v_i\} = \emptyset$, or if one of u_j, z_j is in $\{u_i, v_i\}$ and the other node appears an even number of times between the two moves $\{u_i, v_i\}$. Importantly it cannot be that $\{u_j, z_j\} = \{u_i, v_i\}$ because $u_j \neq u_i, v_i$. Examining the value $M'(j,i)$ in the same way as in the case of arcs above, we observe that if $M(j,i) = 0$ then also $M'(j,i) = 0$, and if $M(j,i) \neq 0$ then also $M'(j,i) \neq 0$. Thus, M' has the same zero-nonzero structure as M and hence it has also full rank.

In the second subcase of Case 2, we found many cycles $C_1, \ldots C_k$ and corresponding witness edges $\{u_i, v_i\}$ such that for every i, (1) C_i does not contain any u_j for $j < i$, nor v_i, (2) C_i has exactly two edges incident to u_i and node v_i appears an odd number of times between the two moves corresponding to these two edges, (3) if C_i contains v_j for some $j < i$ (the cycle C_i may go more than once through v_j), then u_j does not appear between any pair of moves that correspond to consecutive edges of the cycle C_i incident to v_i. We used these properties in Max Cut to show that the matrix M whose rows correspond to the witness edges and the columns correspond to the cycles C_i is lower triangular with nonzero diagonal. As before, for each witness pair (u_i, v_i) pick a function f_{r_i} of instance I with this pair of variables as arguments, and let M' be the matrix with these functions as rows and the cycles C_i as columns. We can use the above properties to show that the matrix M' is also lower triangular with nonzero diagonal. Property (2) and the fact that $v_i \notin C_i$ (from property (1)) imply that $M'(i,i) \neq 0$ for all i. Properties (1) and (3) can be used to show that $M'(j,i) = 0$ for all $j < i$. Therefore, M' has full rank.

Once we have Lemma 3.2 for the BFOP instance I, the rest of the proof is the same as for Max Cut. The only difference is that, if the maximum value of a function in I is d (a constant, or even polynomial in n), then the maximum absolute value of the objective function is md instead of n^2 that it was in Max Cut. ∎

8 Conclusions

We analyzed the smoothed complexity of the SWAP algorithm for Graph Partitioning and the 2-FLIP algorithm for Max Cut and showed that with high probability the algorithms terminate in quasi-polynomial time for any pivoting rule. The same result holds more generally for the class of maximum binary constraint satisfaction problems (like Max-2SAT, Max Directed Cut, and others). We have not made any attempt currently to optimize the exponent of $\log n$ in the bound, but we believe that with a more careful analysis the true exponent will be low. There are several interesting open questions raised by this work. We list some of them below.

1. Can our bounds be improved to polynomial? In the case of the 1-FLIP algorithm in the full perturbation model (i.e. when all edges of K_n are perturbed) a polynomial bound was proved in [1]. Can a similar result be shown for 2-FLIP and SWAP?

2. Can our results be extended to the structured smoothed model, i.e., when we are given a graph G and only the edges of G are perturbed? In the case of 1-FLIP we know that this holds [15, 11], but 2-FLIP is much more challenging. The additional technical difficulty in the structured model arises as there are fewer edges that can serve as witnesses for useful cycles, making it harder to find many useful cycles that are linearly independent.

3. We saw in this paper how to analyze local search when one move flips simultaneously two nodes. This is a qualitative step up from the case of single flips, that creates a number of obstacles which had to be addressed. This involved the introduction of nontrivial new techniques in the analysis of the sequence of moves, going from sets in the case of 1-moves to graphs for the case of 2-moves. Dealing with local search that flips 3 or more nodes will require extending the methods further to deal with hypergraphs. We hope that our techniques will form the basis for handling local search algorithms that flip multiple nodes in one move, e.g. k-FLIP for higher k, and even more ambitiously powerful methods like Kernighan-Lin that perform a deep search in each iteration and flip/swap an unbounded number of nodes.

4. Can our results be extended to Max k-Cut or k-Graph Partitioning where the graph is partitioned into $k > 2$ parts? In the case of 1-FLIP for Max k-Cut quasi-polynomial bounds were shown in [7].

5. Can similar results be shown for Max-CSP with constraints of higher arities, for example Max 3SAT? No bounds are known even for 1-FLIP. In fact, analyzing 1-FLIP for Max 3SAT seems to present challenges that have similarities with those encountered in the analysis of 2-FLIP for Max 2SAT and Max Cut, so it is possible that the techniques developed in this paper will be useful also in addressing this problem.

References

[1] O. Angel, S. Bubeck, Y. Peres, and F. Wei, *Local max-cut in smoothed polynomial time*, in Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, ACM, 2017, pp. 429–437.

[2] D. Arthur, B. Manthey, and H. Röglin, *Smoothed analysis of the k-means method*, Journal of the ACM (JACM), 58 (2011), pp. 1–31.

[3] D. Arthur and S. Vassilvitskii, *Worst-case and smoothed analysis of the icp algorithm, with an application to the k-means method*, in 2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06), IEEE, 2006, pp. 153–164.

[4] R. Beier, H. Röglin, C. Rösner, and B. Vöcking, *The smoothed number of pareto-optimal solutions in bicriteria integer optimization*, Mathematical Programming, (2022), pp. 1–37.

[5] R. Beier and B. Vöcking, *Random knapsack in expected polynomial time*, in Proceedings of the thirty-fifth annual ACM symposium on Theory of computing, 2003, pp. 232–241.

[6] A. Bhaskara, M. Charikar, A. Moitra, and A. Vijayaraghavan, *Smoothed analysis of tensor decompositions*, in Proceedings of the forty-sixth annual ACM symposium on Theory of computing, 2014, pp. 594–603.

[7] A. Bibak, C. Carlson, and K. Chandrasekaran, *Improving the smoothed complexity of flip for max cut problems*, in Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2019, pp. 897–916.

[8] A. Blum and J. Dunagan, *Smoothed analysis of the perceptron algorithm for linear programming*, (2002).

[9] S. Boodaghians, J. Brakensiek, S. B. Hopkins, and A. Rubinstein, *Smoothed complexity of 2-player nash equilibria*, in 2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS), IEEE, 2020, pp. 271–282.

[10] S. Boodaghians, R. Kulkarni, and R. Mehta, *Smoothed efficient algorithms and reductions for network coordination games*, in 11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA, T. Vidick, ed., vol. 151 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 73:1–73:15.

[11] X. Chen, C. Guo, E. V. Vlatakis-Gkaragkounis, M. Yannakakis, and X. Zhang, *Smoothed complexity of local max-cut and binary max-csp*, in Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, New York, NY, USA, 2020, Association for Computing Machinery, p. 1052–1065.

[12] D. Dadush and S. Huiberts, *A friendly smoothed analysis of the simplex method*, in Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, 2018, pp. 390–403.

[13] R. Elsässer and T. Tscheuschner, *Settling the complexity of local max-cut (almost) completely*, in International Colloquium on Automata, Languages, and Programming, Springer, 2011, pp. 171–182.

[14] M. Englert, H. Röglin, and B. Vöcking, *Smoothed analysis of the 2-opt algorithm for the general tsp*, ACM Transactions on Algorithms (TALG), 13 (2016), pp. 1–15.

[15] M. Etscheid and H. Röglin, *Smoothed analysis of local search for the maximum-cut problem*, ACM Trans. Algorithms, 13 (2017), pp. 25:1–25:12.

[16] B. Farrell and R. Vershynin, *Smoothed analysis of symmetric random matrices with continuous distributions*, Proceedings of the American Mathematical Society, 144 (2016), pp. 2257–2261.

[17] C. M. Fiduccia and R. M. Mattheyses, *A linear-time heuristic for improving network partitions*, in Proceedings of the 19th Design Automation Conference, ACM/IEEE, 1982, pp. 175–181.

[18] M. Garey, D. Johnson, and L. Stockmeyer, *Some simplified np-complete graph problems*, Theor. Comput. Sci., (1976), pp. 237–267.

[19] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon, *Optimization by simulated annealing: An experimental evaluation; part i, graph partitioning*, Oper. Res., 37 (1989), pp. 865–892.

[20] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis, *How easy is local search?*, Journal of Computer and System Sciences, 37 (1988), pp. 79–100.

[21] B. W. Kernighan and S. Lin, *An efficient heuristic procedure for partitioning graphs*, Bell Syst. Tech. J., 49 (1970), pp. 291–307.

[22] H. Röglin, *The complexity of Nash equilibria, local optima, and Pareto optimal solutions*, PhD thesis, Aachen, Techn. Hochsch., Diss., 2008, 2008.

[23] H. Röglin and B. Vöcking, *Smoothed analysis of integer programming*, Mathematical programming, 110 (2007), pp. 21–56.

[24] A. A. Schäffer and M. Yannakakis, *Simple local search problems that are hard to solve*, SIAM journal on Computing, 20 (1991), pp. 56–87.

[25] V. Sivakumar, S. Wu, and A. Banerjee, *Structured linear contextual bandits: A sharp and geometric smoothed analysis*, in International Conference on Machine Learning, PMLR, 2020, pp. 9026–9035.

[26] D. A. Spielman and S.-H. Teng, *Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time*, Journal of the ACM (JACM), 51 (2004), pp. 385–463.
[27] L. Xia, *The smoothed possibility of social choice*, Advances in Neural Information Processing Systems, 33 (2020), pp. 11044–11055.