Smoothed Complexity of Local Max-Cut and Binary Max-CSP

Xi Chen*
xichen@cs.columbia.edu
Columbia University
New York, NY, USA

Chenghao Guo†
guoch16@mails.tsinghua.edu.cn
IIIS, Tsinghua University
Beijing, China

Emmanouil V. Vlatakis-Gkaragkounis‡
emvlatakis@cs.columbia.edu
Columbia University
New York, NY, USA

Mihalis Yannakakis§
mihalis@cs.columbia.edu
Columbia University
New York, NY, USA

Xinzhi Zhang¶
zhang-xz16@mails.tsinghua.edu.cn
IIIS, Tsinghua University
Beijing, China

ABSTRACT

We show that the smoothed complexity of the FLIP algorithm for local Max-Cut is at most $\phi n^{O(\sqrt{\log n})}$, 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. This improves the previously best upper bound of $\phi n^{O(\log n)}$ by Etscheid and Röglin [7]. Our result is based on an analysis of long sequences of flips, which shows that it is very unlikely for every flip in a long sequence to incur a positive but small improvement in the cut weight. We also extend the same upper bound on the smoothed complexity of FLIP to all binary Maximum Constraint Satisfaction Problems.

CCS CONCEPTS

• **Theory of computation → Graph algorithms analysis**.

KEYWORDS

Smoothed analysis, Local search, Max-cut, Max-CSP

ACM Reference Format:
Xi Chen, Chenghao Guo, Emmanouil V. Vlatakis-Gkaragkounis, Mihalis Yannakakis, and Xinzhi Zhang. 2020. Smoothed Complexity of Local Max-Cut and Binary Max-CSP. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC '20), June 22–26, 2020, Chicago, IL, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3357713.3384325

*Supported by NSF IIS-1838154 and NSF CCF-1703925.
†Part of this work was done while the author was visiting Columbia University.
‡Supported by NSF CCF-1703925, NSF CCF-1763970, NSF CCF-1814873 and NSF CCF-1563155.
§Supported by NSF CCF-1703925 and CCF-1763970.
¶Part of this work was done while the author was visiting Columbia University.

STOC '20, June 22–26, 2020, Chicago, IL, USA

ACM ISBN 978-1-4503-6979-4/20/06…$15.00
https://doi.org/10.1145/3357713.3384325

1 INTRODUCTION

Local search is one of the most prominent algorithm design paradigms for combinatorial optimization problems. 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 is reached. Many algorithms currently deployed in practice are based on local search, and all the empirical evidence suggests that they typically perform very well in practice, rarely running into long paths before reaching a local optimum.

However, despite their wide success in practice, the performance of many local search algorithms lacks rigorous justifications. A recurring phenomenon is that a local search algorithm is usually efficient in practice but analysis under the worst-case framework indicates the opposite — that the algorithm has exponential running time due to delicate pathological instances that one may never encounter in practice. A concrete (and probably one of the simplest) example of this phenomenon is the *FLIP* algorithm for the *local Max-Cut* problem.

Given an undirected graph $G = (V, E)$ with edge weights $(X_e : e \in E)$ (wlog in $[-1, 1]$), the local Max-Cut problem is to find a partition of V 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 one of the nodes to the other set. To find a local max-cut, the FLIP algorithm starts with an initial partition and keeps moving nodes to the other side, one by one, as long as the move increases the weight of the cut, until no local improvement can be made. Note that the FLIP algorithm, similar to the simplex algorithm, is really a family of algorithms since one can apply different rules, deterministic or randomized, to pick the next node when more than one nodes can improve the cut. The local Max-Cut problem is known to be PLS-complete [12], where PLS is a complexity class introduced by [10] to characterize local search problems. A consequence of the proof of the completeness result is that FLIP takes exponential time to solve local Max-Cut in the worst case, regardless of the pivoting rule used [12]. The local Max-Cut problem can be viewed equivalently as the problem of finding a pure Nash equilibrium in a *party affiliation game* [8]. In this case, the FLIP algorithm corresponds to the better response dynamics for the game. The local Max-Cut problem is also closely related to the problem of finding a stable configuration in a neural network in the Hopfield model

[9] (see Section 5 for the definition). In this case the FLIP algorithm corresponds to the natural asynchronous dynamics where in each step an unstable node flips its state, and the process repeats until the network converges to a stable configuration.

Max-Cut is an example of a Maximum Binary Constraint Satisfaction Problem (Max-2CSP). In a general *Max-2CSP*, the input consists of a set of Boolean variables and a set of constraints with weights over some pairs of variables. The problem is then to find an assignment to the variables that maximizes the sum of weights of satisfied constraints. So Max-Cut is the special case when all constraints are XOR of the two variables. Other well-studied special cases include Max-2SAT (Maximum Satisfiability when every clause has at most two literals), and Max-Directed Cut (the max-cut problem for weighted directed graphs); see Section 5 for their definitions. We can consider more generally the *Binary Function Optimization Problem* (or BFOP in short), where instead of constraints we have functions over some pairs of variables and the objective function is a weighted sum of these functions (again see Section 5 for the formal definition). The FLIP algorithm can be used to find local optima for general Max-2CSP and BFOP, where flipping the assignment of any single variable cannot improve the objective function.

In this paper we study the *smoothed complexity* of the FLIP algorithm for local Max-Cut, Max-2CSP and BFOP. The smoothed analysis framework was introduced by Spielman and Teng [14] to provide rigorous justifications for the observed good practical performance of the simplex algorithm (the standard local search algorithm for Linear Programming), even though the simplex algorithm is known to take exponential-time in the worst case for most common pivoting rules (e.g. [11]). Since then, smoothed analysis has been applied in a range of areas; see [13]. Specialized to the local Max-Cut problem, the edge weights of the given undirected graph $G = (V, E)$ are assumed to be drawn independently from a vector $\mathcal{X} = (\mathcal{X}_e : e \in E)$ of probability distributions, one for each edge. Each $\mathcal{X}_e$ is a distribution supported on $[-1, 1]$ and its density function is bounded from above by a parameter $\phi > 0$. Notice that as $\phi \to 1/2$, the model approaches the average-case analysis framework for uniform edge weights. A related alternative model for smoothed analysis is to allow an adversary to pick arbitrary weights w_e, which are then perturbed by adding a small random perturbation $\mathcal{Z}_e$, i.e. the edge weights are $\mathcal{X}_e = w_e + \mathcal{Z}_e$. In this case, ϕ corresponds to the maximum value of the pdf of $\mathcal{Z}_e$.

The question is to give an upper bound $T(n, \phi)$ such that for any G and $\mathcal{X}$, the FLIP algorithm terminates within $T(n, \phi)$ steps with high probability (say $1 - o_n(1)$) over the draw of edge weights $X \sim \mathcal{X}$ (where we use $X \sim \mathcal{X}$ to denote independent draws of $X_e \sim \mathcal{X}_e$).

The best result for $T(n, \phi)$ before our work is the quasipolynomial upper bound $\phi n^{O(\log n)}$ by Etscheid and Röglin [7], based on a rank-based approach which we review in Section 1.2. Before their work, polynomial upper bounds were obtained by Elsässer and Tscheuschner [5] and by Etscheid and Röglin [6] for special cases either when G has $O(\log n)$ degree or when G is a complete graph with edge weights given by Euclidean distances. After the work of [7], Angel et. al [1] obtained a polynomial upper bound for $T(n, \phi)$ when G is a complete graph. Their polynomial bound was further improved by Bibak et. al [2], again for complete graphs.

1.1 Our Results

We prove a $\phi n^{O(\sqrt{\log n})}$ upper bound for the smoothed complexity of FLIP for local Max-Cut:

Theorem 1.1. *Let $G = (V, E)$ be an undirected graph over n vertices, and let $\mathcal{X} = (\mathcal{X}_e : e \in E)$ be a sequence of probability distributions such that every $\mathcal{X}_e$ 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 $X \sim \mathcal{X}$, any implementation of the FLIP algorithm running on G and X takes at most $\phi n^{O(\sqrt{\log n})}$ many steps to terminate.*

Our proof of Theorem 1.1 can be strengthened to get the same bound for the expected number of steps needed to terminate:

Corollary 1.2. *Under the same setting of Theorem 1.1, any implementation of the FLIP algorithm takes at most $\phi n^{O(\sqrt{\log n})}$ many steps to terminate on expectation.*

Given G and edge weights X, we define the (directed) configuration graph they form as follows: vertices of the graph correspond to configurations (or partitions) $\gamma : V \to \{-1, 1\}$; there is an edge from γ to γ' if γ' can be obtained from γ by moving one node and the weight of γ' is strictly larger than that of γ under X, i.e., each edge is a move that strictly improves the cut weight. Theorem 1.1 is established by showing that, with probability at least $1 - o_n(1)$ over the draw of $X \sim \mathcal{X}$, there is no directed path longer than $\phi n^{O(\sqrt{\log n})}$ in the configuration graph formed by G and X.

We also extend Theorem 1.1 to obtain the same upper bound for the smoothed complexity of the FLIP algorithm running on Max-2CSP and BFOP.

Theorem 1.3. *Let I be an arbitrary instance of a Max-2CSP (or BFOP) problem with n variables and m constraints (or functions) with independent random weights in $[-1, 1]$ with density at most ϕ. Then with probability at least $1 - o_n(1)$ over the draw of weights, any implementation of the FLIP algorithm running on I takes at most $\phi mn^{O(\sqrt{\log n})}$ many steps to terminate.*

1.2 The Rank-based Approach

We briefly review the ranked-based approach of [7] and then discuss the main technical barrier to obtaining an upper bound that is asymptotically better than $n^{O(\log n)}$.

Since the maximum possible weight of a cut in the weighted graph is at most $O(n^2)$, if an execution of the FLIP algorithm is very long, then almost all the steps must have a very small gain, less than some small amount ϵ. Therefore, the execution must contain many long substrings (consecutive subsequences) of moves, all of which yield very small gain, in $(0, \epsilon]$. Let $B = (\sigma_1, \ldots, \sigma_k)$ be a sequence of moves, where the σ_i's are the nodes flipped in each step, and let $\gamma : V \to \{-1, 1\}$ be the configuration (partition) of the nodes at the beginning. The increase of the cut weight made by the i-th move is a linear combination of the weights of the edges incident to the node σ_i that is flipped in the i-th step with coefficients either -1 or 1; thus, the increase can be written as the inner product of

a $\{-1, 0, 1\}$-vector indexed by $e \in E$ and the edge weight vector X. We refer to the former as the improvement vector of the i-th move. From our assumption about the probability distributions of edge weights, it is easy to see that for any step, the probability that the increase lies in $(0, \epsilon]$ is at most $\phi\epsilon$. If these events for different steps were independent, then the probability that all the steps of the sequence have this property would be at most $(\phi\epsilon)^k$, i.e., it would go down rapidly to 0 with the length k of the sequence. Unfortunately these events may be quite correlated. However, a lemma of [7] (restated as Lemma 2.1 in Section 2) shows that if the improvement vectors in some steps are linearly independent then they behave like independent events in the sense that the probability that they all yield a gain in $(0, \epsilon]$ is at most $(\phi\epsilon)^r$, where r is the number of linearly independent steps. This suggests that a useful parameter for obtaining a bound is the rank of the set of improvement vectors for the steps of the sequence.

One problem is that the improvement vectors generally depend on the initial configuration γ of nodes that do not appear in the sequence B. Their number may be much larger than the rank r, and thus considering all their possible initial values will overwhelm the probability $(\phi\epsilon)^r$. For this reason, [7] (and we) combine consecutive occurrences of the same node in the sequence B of moves: for each pair (i, j), $i < j \in [k]$, such that σ_i and σ_j are two consecutive occurrences of the same node in B (we call such a pair an *arc*), we form the improvement vector of the arc by summing the improvement vectors of the two steps i and j. Thus, the total gain in cut weight from the two steps is given by the inner product of the improvement vector for the arc and X; if every step of B has gain at most $\epsilon/2$ then every arc has gain at most ϵ. We call such a sequence *ϵ-improving*. The improvement vectors of the arcs do not depend on the initial configuration of *inactive* nodes, those that do not appear in the sequence. The *rank* of the sequence B is defined as the rank of the matrix $M_{B,\gamma}$ whose rows correspond to edges of G and whose columns are improvement vectors of arcs of B. The aforementioned lemma (Lemma 2.1 in Section 2) then implies that if the rank of a sequence B is r then the probability that B is ϵ-improving is at most $(\phi\epsilon)^r$.

The main technical lemma of [7], which we will refer to as *the rank lemma*, shows that

Given any sequence H of length $5n$, there always exists a substring B of H, such that the rank[1] of B is at least $\Omega(\mathsf{len}(B)/\log n)$.

With this lemma, one can apply a union bound to upper bound the probability that there exists an initial configuration γ and a sequence B with $\mathsf{len}(B) \le 5n$ and rank $\Omega(\mathsf{len}(B)/\log n)$ such that B is ϵ-improving with respect to γ and $X \sim \mathcal{X}$ as follows:

$$\sum_{\ell \in [5n]} 2^\ell \cdot n^\ell \cdot (\phi\epsilon)^{\Omega(\ell/\log n)}. \tag{1}$$

Here n^ℓ is a trivial bound for the number of sequences of length ℓ and $(\phi\epsilon)^{\Omega(\ell/\log n)}$ is the probability that a sequence with rank $\Omega(\ell/\log n)$ is ϵ-improving. A crucial observation is that, by the definition of ranks (based on arcs instead of individual moves), we do not need to apply the union bound on the 2^n configurations over all nodes but only on configurations of nodes that appear in the sequence. In other words, initial configurations that only differ on non-active nodes can be treated as the same. This is why we can use 2^ℓ instead of 2^n in (1) since ℓ is a trivial upper bound for the number of active nodes. By setting $\epsilon = 1/(\phi n^{O(\log n)})$, (1) becomes $1 - o_n(1)$. It follows from the rank lemma that, with high probability, no sequence H of length $5n$ can be ϵ-improving and thus, the cut weight must go up by at least ϵ for every $5n$ moves. The $\phi n^{O(\log n)}$ upper bound of [7] then follows since the maximum possible weight of a cut is $O(n^2)$.

A natural question for further improvements is whether the $\log n$-factor lost in the rank lemma of [7] is necessary. Taking a closer look, the proof of [7] consists of two steps. First it is shown that given any sequence H of length $5n$, there is a substring B such that the number of repeating nodes in B (i.e., those that appear at least twice in B) is $\Omega(\mathsf{len}(B)/\log n)$. The rank lemma then follows by showing that the rank of B is at least proportional to the number of repeating nodes in it (which we include as Lemma 4.5 in Section 4.2). On the one hand, the first step of the proof turns out to be tight given an example constructed in [1]. Furthermore, we give a construction in the appendix of the full version [4] to show that, not only the proof approach of [7] is tight, but the rank lemma itself is indeed tight, by giving a graph G and a sequence H of length $5n$ such that every substring B of H has rank at most $O(\mathsf{len}(B)/\log n)$. Therefore, one cannot hope to obtain a bound better than $n^{O(\log n)}$ based on an improved version of this rank lemma.

1.3 A New Rank Lemma

We overcome the $\log n$-barrier to the rank-based approach of [7] on general graphs by considering not only substrings of H but also its *subsequences*. Recall that a subsequence of H is of the form $(\sigma_{i_1}, \ldots, \sigma_{i_k})$ with $i_1 < \cdots < i_k$. We use the same arc-based rank notion defined above. The main technical component (Lemma 3.1) is a new rank lemma that can be stated informally as follows:

If H is a sequence of moves of length $5n$, there is a subsequence B of H, such that the rank of B is at least $\Omega(\mathsf{len}(B)/\sqrt{\log n})$.

While the $\sqrt{\log n}$ in the statement naturally leads to the improvement from $\log n$ to $\sqrt{\log n}$ in our smoothed complexity bound, one needs to be careful when working with subsequences B of H. An advantage of using substrings of H is that improvement vectors of arcs are trivially preserved, which is not necessarily the case for subsequences of H. More formally, let $B = (\sigma_\ell, \ldots, \sigma_r)$ be a substring of H and $\alpha = (i, j)$ be an arc of H such that $\ell \le i < j \le r$. Then the corresponding arc $\beta = (i - \ell + 1, r - \ell + 1)$ of B has the same improvement vector as that of α in H. Therefore, B being not ϵ-improving trivially implies that H is not ϵ-improving. However, when $B = (\sigma_{i_1}, \ldots, \sigma_{i_k})$ is a subsequence of H, it is not necessarily the case that every arc β of B can be mapped back to an arc α of H and even if this is the case, it is in general not true that α and β share the same improvement vector and thus, B being not ϵ-improving does not necessarily imply that H is not ϵ-improving.

[1] Note that the rank is defined earlier using both B and the initial configuration γ. An observation from [1] shows that the rank actually does not depend on γ but only B.

Despite this limitation, we prove a *subsequence rank lemma* in Section 4 of the following form (still an informal[2] version; see Lemma 3.1):

> *If H is a sequence of moves of length $5n$, then there exists a subsequence B of H and a set of arcs Q of B such that the rank of Q (i.e., the rank of the matrix where we only include improvement vectors of arcs in Q) is $\Omega(\mathrm{len}(B)/\sqrt{\log n})$ and the improvement vector of every arc in Q is the same as that of its corresponding arc in H.*

Theorem 1.1 then follows quickly from the new rank lemma by a similar union bound.

The technical challenge for proving our new rank lemma is to balance the following trade-off. On the one hand, we would like to keep as many arcs of H in Q as possible so that they together give us a high rank compared to the length of B. On the other hand, the more arcs we want to keep the less aggressively we can delete moves from H, in order to have their improvement vectors preserved. To achieve this for an arc $\alpha = (i, j)$ of H, we need to make sure that the parity of the number of occurrences inside the arc of any node adjacent to the node $\sigma_i = \sigma_j$ in G remains the same after deletions.

We now give a sketch of the proof of our Main Lemma (Lemma 3.1). Let H be a sequence of moves of length $5n$. Given that it is much longer than the number n of vertices, it is easy to show that H has many arcs (actually at least $4n$; see Lemma 4.1). We first partition all arcs of H into $\log n$ many chunks according to their lengths (the length of an arc (i, j) is defined to be $j - i + 1$): chunk C_j contains all arcs of length between 2^j and 2^{j+1}. Then there must be a j^* such that $|C^*_j|$ is at least $\Omega(n/\log n)$. Focusing on arcs in C_{j^*} and letting $\ell = 2^{j^*+1}$, one can show (Lemma 4.6 in Section 4.2) that there is a substring $H' = (\sigma_i, \ldots, \sigma_{i+2\ell-1})$ of length 2ℓ such that the number of C_{j^*}-arcs contained in H' is $\Omega(\ell/\log n)$ (this should not come as a surprise because this is basically the expected number of C_{j^*}-arcs when we pick the window uniformly at random). Let C be the set of C_{j^*}-arcs in H'. If we take B to be H' and Q to be arcs that correspond to C in B, then the rank of Q can be shown to be $\Omega(|Q|)$ (by applying Lemma 4.5 discussed earlier and using the fact that all arcs in Q are almost as long as B up to a constant). However, the ratio $|Q|/\mathrm{len}(B) = |C|/(2\ell)$ is only $\Omega(1/\log n)$, too weak for our goal. Instead our proof uses the following new ideas.

The first idea is to group the $\log n$ chunks $C_1, \ldots, C_{\log n}$ into $\sqrt{\log n}$ groups $\mathcal{D}_1, \ldots, \mathcal{D}_{\sqrt{\log n}}$, each being the union of $\sqrt{\log n}$ consecutive chunks. In Case 1 and Case 2 of the proof, we pick a group $\mathcal{D}_{i^*}$, with ℓ'' set to be the maximum length of arcs in $\mathcal{D}_{i^*}$, and then pick a substring H'' of H of length $2\ell''$ by Lemma 4.6 so that the number of $\mathcal{D}_{i^*}$-arcs in H'' is $\Omega(\ell''/\sqrt{\log n})$. We show that when these $\mathcal{D}_{i^*}$-arcs satisfy certain additional properties (see more discussion about these properties below), then their rank is almost full and Lemma 3.1 for these two cases follows by setting B to be H'' and Q to be arcs of B that correspond to these $\mathcal{D}_{i^*}$-arcs in H''.

[2]The lemma stated here is still not in its formal version since we ignore the involvement of the initial configuration γ; see Lemma 3.1 for details. Fortunately the initial configuration will play a minimal role in the proof and we find it easier to gain intuition about the proof without considering it in the picture.

The second idea is to continue using the substring H' and the set C of C_{j^*}-arcs in it, with the rank of C being $\Omega(\mathrm{len}(H')/\log n)$, but now we try to *delete* as many moves from H' as possible to obtain the desired subsequence B and at the same time *preserve* improvement vectors of arcs in C.

We make two key observations about which moves can or cannot be deleted. First let σ_k be a move in H' such that node σ_k only appears once in H'. Then we cannot delete σ_k if $i < k < j$ for some arc $\alpha = (i, j) \in C$ and (σ_i, σ_k) is an edge in G; otherwise the improvement vector of α will not remain the same at the entry indexed by edge (σ_i, σ_k). As a result, if there are many such moves in H' then we cannot hope to preserve arcs in C and at the same time increase the ratio $|C|/\mathrm{len}(B)$ up to $1/\sqrt{\log n}$. To handle this situation, our first key observation is that having many such σ_k is indeed a good case: it would imply that many arcs $\alpha = (i, j)$ in H have a σ_k (referred to as a *witness* for α) such that $i < k < j$, (σ_i, σ_k) is an edge in G, σ_k only appears once inside α, and *both the previous and next occurrences of σ_k are pretty far away from k*. We handle this case in Case 2 of our proof. As discussed earlier, we pick a group $\mathcal{D}_{i^*}$ and a substring H'' of H. Assuming that most $\mathcal{D}_{i^*}$-arcs in H'' satisfy this additional property now, their witnesses can be used to certify the linear independence of their improvement vectors; this implies that these $\mathcal{D}_{i^*}$-arcs in H'' have almost full rank.

The next observation is about repeating nodes in H'. Let $\beta = (k, r)$ be an arc that shares no endpoint with arcs in C. We say β *overlaps* with an arc $\alpha = (i, j) \in C$ if (σ_k, σ_i) is an edge in G and either $k < i < r < j$ or $i < k < j < r$. If β does *not* overlap with any arc in C then it is not difficult to show that the deletion of both moves k and r of β will have no effect on improvement vectors of arcs in C. Therefore, we can keep deleting until no such arc exists in H' anymore. But, what if many arcs in H' overlap with arcs in C? Our second observation is that this is again a good case for us. Assuming that there are $\Omega(\ell/\sqrt{\log n})$ arcs in H' that overlap with arcs in C, we show that the rank of these arcs is almost full and thus, the ratio of the rank and the length of H' is $\Omega(1/\sqrt{\log n})$; this is our Case 3.1. (Note that the discussion here is very informal. In the actual proof, we need to impose an extra condition (see Definition 4.4) on arcs in C in order to show that the rank of arcs overlapping with arcs in C is almost full. We handle the case when most arcs of H violate this condition in Case 1 of the proof, by working with a group $\mathcal{D}_{i^*}$ as discussed earlier.)

Now we can assume that all moves in H' can be deleted except those that are endpoints of arcs in C and endpoints of arcs that overlap with at least one arc in C (the number of which is at most $O(\ell/\sqrt{\log n})$). Recall from the discussion at the beginning that the rank of C is almost full. Given that the length of the subsequence B obtained after deletions is $O(\sqrt{\log n}) \cdot |C|$, the rank lemma follows (since we made sure that the deletion of moves does not affect improvement vectors of arcs in C). This is handled as the last case, Case 3.2, in the proof of the Main Lemma.

With the proof sketch given above, the choice of $\sqrt{\log n}$ in the statement of the Main Lemma is clearly the result of balancing these delicate cases. At a high level, the proof of the Main Lemma relies on a detailed classification of arcs based on a number of their attributes that we can take advantage in the analysis of their ranks. The proof involves an intricate analysis of sequences and their

properties and uses very little from the structure of the graph itself and the Max-Cut problem. As a consequence, the proof readily extends to all other local Max-2CSP problems with the same bound on their smooth complexity.

Organization. The structure of the rest of the paper is as follows. Section 2 gives basic definitions and background. Section 3 states the Main Lemma and uses it to prove Theorem 1.1. Section 4, which is technically the heart of the paper, proves the Main Lemma. Section 5 presents the extension to general binary Max-CSP and Function problems, and Section 6 offers concluding remarks and open problems.

2 PRELIMINARIES

Given a positive integer n we use $[n]$ to denote $\{1, \ldots, n\}$. Given two integers $i \le j$, we write $[i : j]$ to denote the interval of integers $\{i, \ldots, j\}$. Given an interval $I = [i : j]$, we write $\mathsf{len}(I) = j - i + 1$ to denote the *length* of the interval I.

Let $G = (V, E)$ be a weighted undirected graph with a weight vector $X = (X_e : e \in E)$, where $X_e \in [-1, 1]$ is the weight of edge $e \in E$. Under the smoothed complexity model, there is a family $\mathcal{X} = (\mathcal{X}_e : e \in E)$ of probability distributions, one for each edge; the edge weights X_e are drawn independently from the corresponding distributions $\mathcal{X}_e$. We assume that each $\mathcal{X}_e$ is a 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.) A *configuration* γ of a set of nodes $S \subseteq V$ is a map from S to $\{-1, 1\}$. A configuration γ of V corresponds to a partition of the nodes into two parts: the left part $\{u \in V : \gamma(u) = -1\}$ and the right part $\{u \in V : \gamma(u) = 1\}$. The *weight* of a configuration (partition) γ of V with respect to a weight vector X is the weight of the corresponding cut, i.e., the sum of weights of edges that connect a left node with a right node.

Formally, it is given by

$$\mathrm{obj}_{G,X}(\gamma) = \frac{1}{2} \sum_{(u,v) \in E} X_{(u,v)} \cdot \big(1 - \gamma(u)\gamma(v)\big).$$

The problem of finding a configuration of V that maximizes the cut weight is the well-known Max-Cut problem. We are interested in the Local Max-Cut problem, where the goal is to find a configuration γ of V that is a local optimum, i.e., $\mathrm{obj}_{G,X}(\gamma) \ge \mathrm{obj}_{G,X}(\gamma^{(v)})$ for all $v \in V$, where $\gamma^{(v)}$ is the configuration obtained from γ by flipping the sign of $\gamma(v)$.

A simple algorithm for Local Max-Cut is the FLIP algorithm:

Start from some initial configuration $\gamma = \gamma_0$ of V. While there exists a node $v \in V$ such that flipping the sign of $\gamma(v)$ would increase the cut weight, select such a node v (according to some pivoting criterion) and execute the flip, i.e., set $\gamma_{i+1} = \gamma_i^{(v)}$ and repeat.

The algorithm terminates with a configuration of V that cannot be improved by flipping any single node. The execution of FLIP for a given graph G and edge weights X depends on both the initial configuration γ_0 and the pivoting criterion used to select a node to flip in each iteration, when there are multiple nodes which can be profitably moved. Each execution of FLIP generates a sequence of nodes that are moved during the execution.

Given $G = (V, E)$ we denote a *sequence of moves* as a sequence $H = (\sigma_1, \ldots, \sigma_k)$ of nodes from V, where we write $\mathsf{len}(H) = k$ to denote its *length*. We say a node $v \in V$ is *active* in H if it appears in H, and is *repeating* if it appears at least twice in H. We write $S(H)$ to denote the set of active nodes in H, and use $S_1(H)$ (resp. $S_2(H)$) to denote the set of nodes that appear only once (resp. two or more times) in H. As usual, a *substring* of H is a sequence of the form $(\sigma_i, \sigma_{i+1}, \ldots, \sigma_j)$ for some $1 \le i < j \le k$, and a *subsequence* of H is a sequence of the form $(\sigma_{i_1}, \ldots, \sigma_{i_\ell})$ for some $1 \le i_1 < \cdots < i_\ell \le k$. Given a set $P \subseteq [k]$, we write H_P to denote the subsequence of H obtained by restricting to indices in P. When P is an interval $[i : j] \subseteq [k]$, H_P is a substring of H.

Next we introduce the notion of arcs and define their improvement vectors. An *arc* $\alpha = (i, j)$ of $H = (\sigma_1, \ldots, \sigma_k)$ is a pair of indices $i < j \in [k]$ such that $\sigma_i = \sigma_j$ and $\sigma_i \ne \sigma_\ell$ for all $i < \ell < j$ (i.e., σ_i and σ_j are two consecutive occurrences of the same node in H). We let $\mathsf{node}_H(\alpha) = \sigma_i = \sigma_j \in V$ and refer to it as the node of α. We will sometimes omit the subscript H when it is clear from the context. We also refer to i as the *left endpoint* and j as the *right endpoint* of α, and write $\mathsf{left}(\alpha) = i$ and $\mathsf{right}(\alpha) = j$. We write $\mathsf{len}(\alpha) = j - i + 1$ to denote the length of α.

Given a sequence $H = (\sigma_1, \ldots, \sigma_k)$ of moves (nodes) and an initial configuration $\gamma = \gamma_0$ before the first move of H, let γ_i denote the configuration after the i-th move of H. The gain in the cut weight from the i-th move is a linear combination of the weights of the edges incident to node σ_i that is flipped, where some edges have coefficient 1 and the rest have coefficient -1. Note that if H is part of an execution of the FLIP algorithm, then the gain is positive at every move.

For each arc $\alpha = (i, j)$ of H, we define the *improvement vector* of α with respect to γ and H, denoted by $\mathsf{impvm}_{\gamma,H}(\alpha)$, as follows: $\mathsf{impvm}_{\gamma,H}(\alpha)$ is a vector in $\{-2, 0, 2\}^E$ indexed by edges $e \in E$ (just like the weight vector X); its entry indexed by $e \in E$ is nonzero iff $e = (\mathsf{node}_H(\alpha), v) \in E$ for some node v that appears an odd number of times in $\sigma_{i+1}, \ldots, \sigma_{j-1}$. When this is the case, its value is set to be $2\gamma_{i-1}(\mathsf{node}_H(\alpha))\gamma_{i-1}(v)$. Note that for this definition we do not need to have the full configuration γ of V but only of the active nodes in $S(H)$. It also follows from the definition of improvement vectors that, if γ is the initial configuration of $S(H)$ and we move nodes one by one according to H, then the *total* gain in the cut weight obj from the i-th move and the j-th move is given by the inner product of $\mathsf{impvm}_{\gamma,H}(\alpha)$ and X. Indeed, letting $u = \mathsf{node}_H(\alpha)$, the total gain from these two moves equals

$$\underbrace{\sum_{v:(u,v)\in E} X_{(u,v)} \cdot \gamma_{i-1}(u)\gamma_{i-1}(v)}_{i\text{-th move}} + \underbrace{\sum_{v:(u,v)\in E} X_{(u,v)} \cdot \gamma_{j-1}(u)\gamma_{j-1}(v)}_{j\text{-th move}}$$

Given that $\gamma_{i-1}(u) = -\gamma_{j-1}(u)$, only those neighbors v of u that flipped an odd number of times in $\sigma_{i+1}, \ldots, \sigma_{j-1}$, i.e., $\gamma_{i-1}(v) \ne \gamma_{j-1}(v)$, contribute in the *total* gain of the two moves.

Inspired by the definition of improvement vectors above, we define the *interior* of α, denoted by $\mathsf{interior}_H(\alpha)$: $\mathsf{interior}_H(\alpha)$ contains all $k \in [i + 1 : j - 1]$ such that node σ_k appears an odd number of times in $\sigma_{i+1}, \ldots, \sigma_{j-1}$ and σ_k is adjacent to $\sigma_i = \mathsf{node}_H(\alpha)$ in the graph G.

We say an arc α of H is *improving* with respect to γ and X if the inner product of $\mathsf{impvm}_{\gamma,H}(\alpha)$ and X is positive. We say it is ϵ-*improving* for some parameter $\epsilon > 0$ if the inner product is in $(0, \epsilon]$. Furthermore, we say a set C of arcs of H is improving (or ϵ-improving) with respect to γ and X if every arc in C is improving (or ϵ-improving). A sequence H of moves is improving (or ϵ-improving) with respect to γ and X if every arc of H is improving (or ϵ-improving). Note that if H is part of the sequence of moves generated by an execution of the FLIP algorithm then H must be improving, because every move must increase the weight of the cut and therefore every arc is improving. On the other hand, if some move in H increases the cut weight by more than ϵ then the same is true for the arc that has it as an endpoint and thus, H is not ϵ-improving.

Let C be a set of arcs of H. A key parameter of C that will be used to bound the probability of C being ϵ-improving (over the draw of the edge weights $X \sim \mathcal{X}$) is the *rank* of C with respect to γ and H, denoted by $\mathsf{rank}_{\gamma,H}(C)$: this is the rank of the $|E| \times |C|$ matrix that contains improvement vectors $\mathsf{impvm}_{\gamma,H}(\alpha)$ as its column vectors, one for each arc $\alpha \in C$. To give some intuition for this parameter, one may hope that for a fixed sequence of moves H with K arcs

$$\mathbf{Pr}_{X\sim\mathcal{X}}\Big[\, H \text{ is } \epsilon\text{-improving} \,\Big] = \prod_{\alpha\in C} \mathbf{Pr}_{X\sim\mathcal{X}}\Big[\, \alpha \text{ is } \epsilon\text{-improving} \,\Big].$$

However, since there could be improving steps that are strongly correlated (as an extreme situation there could be two arcs with exactly the same improvement vector), one may expect the product on the right hand side to hold only for linearly independent $\mathsf{impvm}_{\gamma,H}(\alpha)$'s, introducing the necessity of analysis of the rank.

An observation from [7] is that $\mathsf{rank}_{\gamma,H}(C)$ is *independent* of the choice of γ. Indeed, a change of a node in the initial configuration would result in a change of sign on every row of the matrix that is incident with this node. So from now on we write it as $\mathsf{rank}_H(C)$. To simplify our discussion on $\mathsf{rank}_H(C)$ later, we use $\mathsf{impvm}_H(\alpha)$ to denote $\mathsf{impvm}_{\gamma_0,H}(\alpha)$ where γ_0 is the default initial configuration of $S(H)$ that maps every node in $S(H)$ to -1. Then $\mathsf{rank}_H(C)$ is the rank of the matrix that consists of $\mathsf{impvm}_H(\alpha)$, $\alpha \in C$. The next tool from [7] shows that the higher the rank is, the less likely for C to be ϵ-improving.

Lemma 2.1 (Lemma A.1 from [7]). *Let $\mathcal{X} = (\mathcal{X}_i : i \in [m])$ be a sequence of probability distributions in which each $\mathcal{X}_i$ has density bounded from above by a parameter $\phi > 0$. Let $r_1, \ldots, r_k \in \mathbb{Z}^m$ be k vectors that are linearly independent. Then for any $\epsilon > 0$, we have*

$$\mathbf{Pr}_{X\sim\mathcal{X}}\Big[\, \forall\, i \in [k] : \langle r_i, X\rangle \in [0,\epsilon] \,\Big] \le (\phi\epsilon)^k.$$

Corollary 2.2. *Let $G = (V, E)$ be an undirected graph and let $\mathcal{X} = (\mathcal{X}_e : e \in E)$ be a sequence of distributions such that each $\mathcal{X}_e$ has density bounded from above by a parameter $\phi > 0$. Let H be a sequence of moves, γ be a configuration of $S(H)$, and C be a set of arcs of H. Then for any $\epsilon > 0$,*

$$\mathbf{Pr}_{X\sim\mathcal{X}}\Big[\, C \text{ is } \epsilon\text{-improving with respect to } \gamma \text{ and } X \,\Big] \le (\phi\epsilon)^{\mathsf{rank}_H(C)}.$$

Finally we say a sequence H of moves is *nontrivial* if the interior of every arc in H is nonempty; H is *trivial* if at least one of its arcs has $\mathsf{interior}_H(\alpha) = \emptyset$. It follows from definitions that H cannot be improving (with respect to any γ and any X) if it is trivial. Since every sequence resulting from an execution of the FLIP algorithm is improving, it follows that it is also nontrivial. We will consider henceforth only nontrivial sequences (see Lemma 3.1).

3 MAIN LEMMA AND THE PROOF OF THEOREM 1.1

We prove the following main technical lemma in Section 4:

Lemma 3.1. *Let $G = (V, E)$ be an undirected graph over n vertices. Given any nontrivial sequence H of moves of length $5n$ and any configuration γ of $S(H)$, there exist (i) a sequence B of moves of length at most $5n$, (ii) a configuration τ of $S(B)$, and (iii) a set of arcs Q of B such that*

(1) The rank of Q in B satisfies

$$\frac{\mathsf{rank}_B(Q)}{\mathsf{len}(B)} \ge \Omega\left(\frac{1}{\sqrt{\log n}}\right) \quad (\textit{High-rank property}); \tag{2}$$

(2) For every arc $\alpha \in Q$, there exists an arc α' of H such that

$$\mathsf{impvm}_{\tau,B}(\alpha) = \mathsf{impvm}_{\gamma,H}(\alpha'). \quad (\textit{Vector-Preservation property}).$$

As discussed earlier in Section 1.3, the new sequence B in Lemma 3.1 is either a substring or a subsequence of H. When we pick B to be a substring of H, say a substring that starts with the i-th move of H, the natural choice of Q is the set of all arcs in B (since we would like $\mathsf{rank}_B(Q)$ to be as large as possible in (2)) and that of τ is the configuration γ_{i-1} of $S(B)$ derived from γ after making the first $i-1$ moves of H. With these choices, the second condition of Lemma 3.1 is trivially satisfied and the main goal is to lowerbound the rank of arcs in B. This is indeed the proof strategy followed in all previous works [1, 2, 7]. The key new idea of the paper is the use of subsequences of H as B instead of substrings of H. While this gives us more flexibility in the choice of B to overcome the $(\log n)$-barrier of [7] as sketched earlier in Section 1.3, one needs to be very careful when deleting moves and picking arcs to be included in Q in order to satisfy the second condition.

We delay the proof of Lemma 3.1 to Section 4. Instead, below we use it to prove Theorem 1.1 and Theorem 1.2.

Proof of Theorem 1.1 assuming Lemma 3.1. Let $c_1 > 0$ be a constant to be specified later and let

$$\epsilon = \frac{1}{\phi \cdot n^{c_1\sqrt{\log n}}}.$$

We write F to denote the following event on the draw of the weight vector $X \sim \mathcal{X}$:

> Event F: For every sequence B of length at most $5n$, every configuration τ of $S(B)$, and every set Q of arcs of B satisfying (where $a > 0$ is the constant in Lemma 3.1)
>
> $$\frac{\mathsf{rank}_B(Q)}{\mathsf{len}(B)} \ge \frac{a}{\sqrt{\log n}}, \tag{3}$$
>
> Q is *not* ϵ-improving with respect to τ and X.

We break the proof into two steps. First we prove that F occurs with probability at least $1 - o_n(1)$ over the draw of the weight vector $X \sim \mathcal{X}$. Next we show that when F occurs, any implementation of the FLIP algorithm must terminate within $\phi \cdot n^{O(\sqrt{\log n})}$ many steps.

For the first step, we fix an $\ell \in [5n]$, a sequence B of length ℓ, a configuration τ of $S(B)$ and a set Q of arcs of B that satisfies (3) (so the rank is at least $a\ell/\sqrt{\log n}$). It follows from Corollary 2.2 that the probability of Q being ϵ-improving with respect to τ and $X \sim \mathcal{X}$ is at most $(\phi\epsilon)^{a\ell/\sqrt{\log n}}$. Applying a union bound (on ℓ, B, τ and Q), F does not occur with probability at most

$$\begin{aligned}\mathbf{Pr}[\neg F] &\le \sum_{\ell\in[5n]} n^\ell \cdot 2^\ell \cdot 2^{\ell-1} \cdot (\phi\epsilon)^{\frac{a\ell}{\sqrt{\log n}}} \\ &\le \sum_{\ell\in[5n]} \left((4n)^{\frac{\sqrt{\log n}}{a}} \cdot \phi\epsilon\right)^{\frac{a\ell}{\sqrt{\log n}}} = o_n(1),\end{aligned}$$

where the factor $n^\ell 2^\ell$ is an upper bound for the number of choices for B of length ℓ and the initial configuration τ of $S(B)$, and the factor $2^{\ell-1}$ is because there can be no more than $\ell - 1$ arcs in a sequence of length ℓ. The last equation follows by setting c_1 in the choice of ϵ sufficiently large.

For the second step, first notice that when F occurs, it follows from Lemma 3.1 that there exists no sequence H of length $5n$ together with a configuration γ of $S(H)$ so that H is ϵ-improving with respect to γ and X. Taking any implementation of the FLIP algorithm running on G with weights X, this implies that the cut weight obj must go up by at least ϵ for every $5n$ consecutive moves. As the weight of any cut lies in $[-n^2, n^2]$, the number of steps it takes to terminate is at most

$$5n \cdot \frac{2n^2}{\epsilon} = \phi \cdot n^{O(\sqrt{\log n})}.$$

This finishes the proof of Theorem 1.1. □

Proof of Theorem 1.2 assuming Lemma 3.1. We let F_ϵ denote the event F in the proof of Theorem 1.1 with a specified $\epsilon > 0$. Let

$$\epsilon_0 = \frac{1}{\phi \cdot n^{c_1\sqrt{\log n}}},$$

where $c_1 > 0$ is a constant to be fixed shortly.

For $\epsilon \le \epsilon_0$, we have

$$\begin{aligned}\mathbf{Pr}[\neg F_\epsilon] &\le \sum_{\ell\in[5n]} n^\ell \cdot 2^\ell \cdot 2^{\ell-1} \cdot (\phi\epsilon_0)^{\left\lceil \frac{a\ell}{\sqrt{\log n}} \right\rceil} \cdot \left(\frac{\epsilon}{\epsilon_0}\right)^{\left\lceil \frac{a\ell}{\sqrt{\log n}} \right\rceil} \\ &\le \sum_{\ell\in[5n]} \frac{1}{5n^2} \cdot \left(\frac{\epsilon}{\epsilon_0}\right)^{\left\lceil \frac{a\ell}{\sqrt{\log n}} \right\rceil} \le \frac{\epsilon}{n\epsilon_0},\end{aligned}$$

where c_1 is large enough so that the second inequality holds.

From the previous proof, we know that F_ϵ implies the number of steps (denote as L) is at most $10n^3/\epsilon$. By combining the inequality above, one can bound the tail probability of L. Let

$$L_0 = \frac{10n^3}{\epsilon_0} = \phi \cdot n^{O(\sqrt{\log n})}.$$

Then the probability that L is larger than cL_0 for any $c \ge 1$ is

$$\mathbf{Pr}[L > cL_0] \le \mathbf{Pr}[\neg F_{\epsilon_0/c}] \le \frac{1}{nc}.$$

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

$$\mathbb{E}[L] \le L_0 + \sum_{s=\lceil L_0\rceil}^{2^n} \mathbf{Pr}[L \ge s] \le L_0 + \sum_{s=\lceil L_0\rceil}^{2^n} \frac{L_0}{ns} = O(L_0).$$

This finishes the proof of Theorem 1.2. □

4 PROOF OF THE MAIN LEMMA

We proceed now to the proof of Lemma 3.1. The plan of the section is as follows. Given a nontrivial sequence $H = (\sigma_1, \ldots, \sigma_m)$ of moves of length $m = 5n$, we classify in Section 4.1 its arcs into good ones and bad ones and introduce the notion of the radius of an arc. In Section 4.2 we prove a few basic lemmas that will be used in the proof of Lemma 3.1. Next we partition the set of all arcs of H into chunks according to their lengths in Section 4.3 and present an overview of cases of the proof of Lemma 3.1. There will be three cases and they will be covered in Section 4.4, 4.5 and 4.6, respectively. For each case we choose B to be either a substring or a subsequence of H. Among all cases, there is only one occasion where we choose B to be a subsequence of H. As discussed earlier in Section 3, the second condition of Lemma 3.1 is trivially satisfied when B is a substring of H (since we don't change the interior of any arc). Therefore, there is no need to specify the configuration τ in cases when B is chosen to be a substring of H.

4.1 Classification of Arcs

We start with a quick proof that there are many arcs in a long sequence of moves.

Lemma 4.1. *For any sequence B of moves, the number of arcs in B is at least* $\mathsf{len}(B) - n$.

Proof. Denote by χ_v the number of occurrences of node v in B. Then the number of arcs in B is $\sum_{v\in V}(\chi_v - 1) \ge \sum_{v\in V}\chi_v - n = \mathsf{len}(B) - n$. □

Corollary 4.2. *If H is a sequence of moves of length $5n$, then it contains at least $4n$ arcs.*

Next we give the definition of good and bad arcs in a sequence. We start with some notation. Let $H = (\sigma_1, \ldots, \sigma_m)$ be a sequence of moves of length $m = 5n$. We use $\mathcal{A}$ to denote the set of all arcs in H; by Corollary 4.2 we have $|\mathcal{A}| \ge 4n$.

For each $k \in [m]$, we define the *predecessor* $\mathrm{pred}_H(k)$ of the k-th move to be the largest index $i < k$ such that $\sigma_i = \sigma_k$ and set $\mathrm{pred}_H(k)$ to be $-\infty$ if no such index i exists (i.e., $\mathrm{pred}_H(k)$ is the index of the previous occurrence of the same node σ_k). Similarly we define the *successor* $\mathrm{succ}_H(k)$ of the k-th move in H to be the smallest index $j > k$ such that $\sigma_j = \sigma_k$ and set $\mathrm{succ}_H(k)$ to be $+\infty$ if no such j exists. Next we define the radius of a move and an arc:

Definition 4.3 (Radius). For each $k \in [m]$ we define the *radius* of the k-th move in H as

$$\mathrm{radius}_H(k) = \max\left\{\text{L-radius}_H(k), \text{R-radius}_H(k)\right\},$$

where

$$\text{L-radius}_H(k) = k - \text{pred}_H(k) + 1, \quad \text{and}$$
$$\text{R-radius}_H(k) = \text{succ}_H(k) - k + 1.$$

Given an arc $\alpha = (i, j) \in \mathcal{A}$ of H, we define its radius as[3]

$$\text{radius}_H(\alpha) = \max\left\{\text{radius}_H(k) \,\middle|\, k \in \text{interior}_H(\alpha)\right\}.$$

It follows from the definition that if $\text{radius}_H(k)$ is not $+\infty$ then both $\text{pred}_H(k)$ and $\text{succ}_H(k)$ are defined and then both $(\text{pred}_H(k), k)$ and $(k, \text{succ}_H(k))$ are arcs of H. As an example of the radius of an arc $\alpha = (i, j)$, if there is a $k \in [i + 1 : j - 1]$ such that node σ_k is adjacent to $\text{node}_H(\alpha)$ in G and σ_k does not appear anywhere else in H (i.e., $\sigma_k \in S_1(H)$) then $\text{radius}_H(\alpha) = +\infty$. Another example is shown in Figure 1. Here $\text{radius}_H(\alpha) = \text{radius}_H(k) = 5$ assuming that (σ_k, σ_i) is in G.

Finally we define good arcs and bad arcs.

Definition 4.4 (Good and bad arcs). We say an arc $\alpha = (i, j)$ of H is *good* if it satisfies

$$\min\left\{\text{L-radius}_H(i),\ \text{R-radius}_H(j)\right\} \geq \frac{\text{len}(\alpha)}{2^{\lceil\sqrt{\log n}\rceil}}.$$

Otherwise we say that α is *bad*. Given a set of arcs $C \subseteq \mathcal{A}$ we write $\text{good}_H(C)$ to denote the set of good arcs in C and $\text{bad}_H(C)$ to denote the set of bad arcs in C.

Given a nonempty set of arcs $C \subseteq \mathcal{A}$, we write

$$\text{maxlen}(C) = \max_{\alpha \in C}\left\{\text{len}(\alpha)\right\}$$

and use $\text{endpoints}(C)$ to denote

$$\left\{k \in [m] : k = \text{left}(\alpha) \text{ or } k = \text{right}(\alpha) \text{ for } \alpha \in C\right\}.$$

4.2 Basic Lemmas

We start with a lemma that will be used to bound the rank of a set of arcs in a sequence. It is essentially the same as Lemma 3.2 in [7], which connects the rank of a set C of arcs with the number of distinct nodes that appear as endpoints of arcs in C. The proof can be found in the full version [4].

Lemma 4.5. *Let C be a set of arcs of a nontrivial sequence H such that the nodes of arcs in C are all distinct. Then* $\text{rank}_H(C) \geq |C|/2$.

We need some notation for the next lemma. We say an arc $\alpha \in \mathcal{A}$ is *contained* in an interval $I \subseteq [m]$ if both $\text{left}(\alpha)$ and $\text{right}(\alpha)$ lie in I. For a set of arcs $C \subseteq \mathcal{A}$ of H, we write $C\lfloor_I$ to denote the set of $\alpha \in C$ contained in I. Intuitively $C\lfloor_I$ is the set of arcs in C that can be inherited by the substring H_I of H. The next lemma follows from (essentially) an averaging argument; the proof can be found in the appendix of [4].

Lemma 4.6. *Let H be a sequence of moves of length m, $P \subseteq [m]$ be a nonempty set of indices and $C \subseteq \mathcal{A}$ be a nonempty set of arcs of H such that* $\text{len}(\alpha) \leq \ell$ *for all $\alpha \in C$ for some positive integer parameter $\ell \leq m/2$. Then there exists an interval $I = [i : i + 2\ell - 1] \subseteq [m]$ of length 2ℓ such that*

$$\frac{|C\lfloor_I|}{|C|} \geq \max\left\{\frac{2\ell}{16m}, \frac{|P \cap I|}{4|P|}\right\}. \tag{4}$$

Thus, it holds that

$$\frac{|C\lfloor_I|}{\text{len}(I)} \geq \Omega\left(\frac{|C|}{\text{len}(H)}\right) \quad \text{and} \quad \frac{|C\lfloor_I|}{|P \cap I|} \geq \Omega\left(\frac{|C|}{|P|}\right). \tag{5}$$

4.3 Overview of Cases

We now begin to prove Lemma 3.1. Let $G = (V, E)$ be a graph over n vertices, $H = (\sigma_1, \ldots, \sigma_m)$ be a nontrivial sequence of moves of length $m = 5n$, and γ be a configuration of $S(H)$. Let $\mathcal{A}$ be the set of all arcs of H (with $|\mathcal{A}| \geq 4n$ by Corollary 2.2).

We first partition $\mathcal{A}$ into $s = \lceil \log m \rceil = \Theta(\log n)$ *chunks* $C_1, \ldots, C_s$ according to lengths:

$$C_i = \left\{\alpha \in \mathcal{A} : \text{len}(\alpha) \in [2^{i-1} + 1 : 2^i]\right\}.$$

Letting $w = \lceil\sqrt{\log n}\rceil$ and $t = \lceil s/w \rceil$ so both w and t are $\Theta(\sqrt{\log n})$, next we assign these chunks to t *groups* $\mathcal{D}_1, \ldots, \mathcal{D}_t$:

$$\mathcal{D}_i = C_{(i-1)w+1} \cup \cdots \cup C_{iw},$$

where the last group $\mathcal{D}_t$ may contain less than w chunks. From the definition of *chunks* and *groups* we have the following fact:

Fact 4.1. *If P is a set of arcs from the same chunk C_j, then for any arc $\alpha \in P$, it holds that*

$$\text{len}(\alpha) \leq \text{maxlen}(P) \leq 2 \cdot \text{len}(\alpha)$$

If Q is a set of arcs from the same group $\mathcal{D}_i$, then for any arc $\beta \in Q$,

$$\text{len}(\beta) \leq \text{maxlen}(Q) \leq 2^w \cdot \text{len}(\beta).$$

For each chunk C_j in $\mathcal{D}_i$, we further partition it into two sets $C_j = \mathcal{L}_j \cup \mathcal{S}_j$ based on the radius (when $\mathcal{D}_i = \emptyset$, $\mathcal{L}_j$ and $\mathcal{S}_j$ are trivially empty even though $\text{maxlen}(\mathcal{D}_i)$ below is not defined):

$$\mathcal{L}_j = \left\{\alpha \in C_j : \text{radius}_H(\alpha) > 2 \cdot \text{maxlen}(\mathcal{D}_i)\right\} \quad \text{and}$$
$$\mathcal{S}_j = \left\{\alpha \in C_j : \text{radius}_H(\alpha) \leq 2 \cdot \text{maxlen}(\mathcal{D}_i)\right\}.$$

Here $\mathcal{L}_j$ (the long-radius arcs in C_j) contains all arcs $\alpha \in C_j$ such that there exists a $k \in \text{interior}_H(\alpha)$ (where σ_k is adjacent to $\text{node}_H(\alpha)$ in G and σ_k occurs an odd number of times inside α) such that $\text{radius}_H(k)$ is larger[4] than $2 \cdot \text{maxlen}(\mathcal{D}_i)$. The set $\mathcal{S}_j$ (the short-radius arcs in C_j) contains all arcs $\alpha \in C_j$ such that every $k \in \text{interior}_H(\alpha)$ has predecessor and successor within $2 \cdot \text{maxlen}(\mathcal{D}_i)$. See Figure 2 for an example

Recall that our goal is to find a sequence B, a set Q of arcs of B, and a configuration τ of $S(B)$ that satisfy both conditions of Lemma 3.1. The first case we handle in Section 4.4 is when

$$\textbf{Case 1:} \quad \left|\text{bad}_H(\mathcal{A})\right| \geq 0.01 \cdot |\mathcal{A}|. \tag{6}$$

Otherwise, $|\text{good}_H(\mathcal{A})| \geq 0.99 \cdot |\mathcal{A}|$ and we pick a group $\mathcal{D}_{i^*}$ according to the following lemma; see the proof in [4].

[3] Note that this is well defined because when H is nontrivial, the $\text{interior}_H(\alpha)$ of every arc α is nonempty.

[4] Remember that this could be the case when for example either $\text{pred}_H(k) = -\infty$ or $\text{succ}_H(k) = +\infty$.

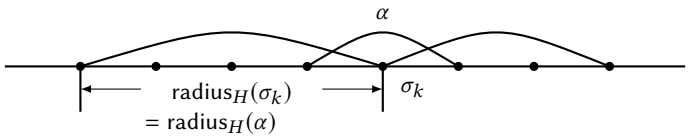

Figure 1: An example of the radius of an arc α

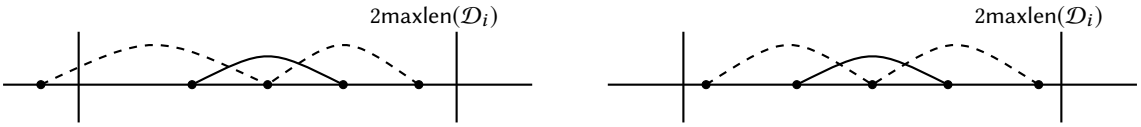

Figure 2: An example of long-radius **(left) and** short-radius **(right) arcs with only one node in their interior.**

Lemma 4.7. *Assume that* $|\text{good}_H(\mathcal{A})| \geq 0.99 \cdot |\mathcal{A}|$. *Then there exists a group* $\mathcal{D}_{i^*}$, *for some* $i^* \in [t]$, *that satisfies the following two conditions (below we assume* $\mathcal{D}_0 = \mathcal{D}_{t+1} = \emptyset$ *by default):*

$$\frac{|\text{good}_H(\mathcal{D}_{i^*})|}{|\mathcal{A}|} \geq \frac{1}{2t} \quad \textit{and} \quad \frac{|\text{good}_H(\mathcal{D}_{i^*})|}{|\mathcal{D}_{i^*-1} \cup \mathcal{D}_{i^*} \cup \mathcal{D}_{i^*+1}|} \geq \frac{1}{7}.$$

Fixing a group $\mathcal{D}_{i^*}$ that satisfies Lemma 4.7 and letting

$$\mathcal{L} = \cup_{C_j \subseteq \mathcal{D}_{i^*}} \mathcal{L}_j \quad \text{and} \quad \mathcal{S} = \cup_{C_j \subseteq \mathcal{D}_{i^*}} \mathcal{S}_j,$$

we next split into two cases, depending on whether the majority of good arcs in $\mathcal{D}_{i^*}$ are long-radius or short-radius arcs. We handle the second case in Section 4.5 when

$$\textbf{Case 2:} \quad \left|\text{good}_H(\mathcal{L})\right| \geq 0.5 \cdot \left|\text{good}_H(\mathcal{D}_{i^*})\right| \tag{7}$$

and we handle the third case in Section 4.6 when

$$\textbf{Case 3:} \quad \left|\text{good}_H(\mathcal{S})\right| \geq 0.5 \cdot \left|\text{good}_H(\mathcal{D}_{i^*})\right|. \tag{8}$$

4.4 Case 1: Many Arcs in $\mathcal{A}$ Are Bad

We say an arc $\alpha = (i, j)$ of H is *dual-bad* if either $\text{pred}_H(i)$ exists and $\text{L-radius}_H(i) > \text{len}(\alpha) \cdot 2^w$, or $\text{succ}_H(j)$ exists and $\text{R-radius}_H(j) > \text{len}(\alpha) \cdot 2^w$ where we recall that $w = \lceil\sqrt{\log n}\rceil$. Figure 3 gives an example of this definition. Given $C \subseteq \mathcal{A}$, we write $\text{dual-bad}_H(C)$ to denote the set of dual-bad arcs in C.

We prove the following lemma in the full version [4] which implies Lemma 3.1 for Case 1, by setting B to be H_I and Q to be the set of arcs of B induced by C, with C and I from the below statement.

Lemma 4.8 (Lemma 3.1 for Case 1). *Assume that* $|\text{bad}_H(\mathcal{A})| \geq 0.01 \cdot |\mathcal{A}|$. *Then there exists an interval* $I \subseteq [m]$ *and a set* C *of arcs of* H *contained in* I *such that*

$$\frac{\text{rank}_H(C)}{\text{len}(I)} \geq \Omega\left(\frac{1}{\sqrt{\log n}}\right).$$

4.5 Case 2: Most Arcs in $\text{good}_H(\mathcal{D}_{i^*})$ Are Long-radius

We start with some intuition for the second case.

In this case we work with a group $\mathcal{D}_{i^*}$ that has many long-radius[5] arcs. Recall that $\mathcal{L}$ is the set of long-radius arcs in $\mathcal{D}_{i^*}$. Let $\ell = \text{maxlen}(\mathcal{L})$. It follows from Lemma 4.7 that there is an interval I of length 2ℓ such that $\mathcal{L}|_I$ is large. Furthermore, given that arcs in $\mathcal{L}|_I$ are long-radius, one can show that every $\alpha \in \mathcal{L}|_I$ has a $k \in \text{interior}_H(\alpha)$ such that $\text{pred}_H(k)$ or $\text{succ}_H(k)$ is outside of I.

Now if it is the oversimplified scenario that *both* $\text{pred}_H(k)$ and $\text{succ}_H(k)$ are outside of I, then one can conclude that $\mathcal{L}|_I$ has full rank since only the improvement vector of α has a nonzero entry indexed by $(\text{node}_H(\alpha), \sigma_k)$. To see this is the case, note that there is no arc with node σ_k contained in I and no other arc with node the same as α can have σ_k in its interior. Our goal is to show for the general scenario that $\mathcal{L}|_I$ has almost full rank.

The following lemma implies Lemma 3.1 for the second case, by setting the subsequence B to be H_I and the set Q to be arcs of B induced by C, using I and C of the below statement.

Lemma 4.9 (Lemma 3.1 for Case 2). *Assume that there exists a group* $\mathcal{D}_{i^*}$ *that satisfies Lemma 4.7 and (7). Then there exists an interval* $I \subseteq [m]$ *and a set* C *of arcs of* H *contained in* I *such that*

$$\frac{\text{rank}_H(C)}{\text{len}(I)} \geq \Omega\left(\frac{1}{\sqrt{\log n}}\right).$$

Proof. Recall that we are in the case where we have a group $\mathcal{D}_{i^*}$ that satisfies Lemma 4.7 and (7). Given that $\mathcal{L} = \cup_{C_j \subseteq \mathcal{D}_{i^*}} \mathcal{L}_j$, they together imply that

$$|\mathcal{L}| \geq \left|\text{good}_H(\mathcal{L})\right| \geq 0.5 \cdot \left|\text{good}_H(\mathcal{D}_{i^*})\right| = \Omega\left(\frac{|\mathcal{A}|}{\sqrt{\log n}}\right).$$

Let $\ell = \text{maxlen}(\mathcal{L})$. If $\ell \leq m/2$ then, by Lemma 4.6, there is an interval $I \subseteq [m]$ of length 2ℓ such that

$$\frac{|\mathcal{L}|_I|}{\text{len}(I)} = \Omega\left(\frac{|\mathcal{L}|}{m}\right) = \Omega\left(\frac{1}{\sqrt{\log n}}\right).$$

If $\ell > m/2$, then $\mathcal{D}_{i^*}$ is the last group ($i^* = t$). In this case, let $I = [m]$, and $\mathcal{L}|_I = \mathcal{L}$ satisfies the same property. In either case, let

[5] Notice that the assumption of Case 2 is actually stronger, that there are many arcs in $\mathcal{D}_{i^*}$ that are both good and long-radius. It turns out that we will only use their long-radius property in the proof of this case.

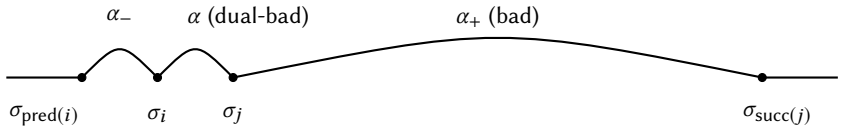

Figure 3: The arc α is dual-bad if either the adjacent $\mathsf{len}(\alpha_-)$ or $\mathsf{len}(\alpha_+)$ is longer than $\mathsf{len}(\alpha)\cdot 2^w$. We can also interpret it as either α_- or α_+ exists and is a bad arc.

$C = \mathcal{L}\lfloor_I$. We finish our proof by showing that $\mathsf{rank}_H(C) \geq |C|/2$, and the rest of the proof can be found in the full version [4]. □

4.6 Case 3: Most Arcs in $\mathrm{good}_H(\mathcal{D}_{i^*})$ Are Short-radius

Combining Lemma 4.7 and (8), we are left with the case that

$$\frac{|\mathrm{good}_H(\mathcal{S})|}{|\mathcal{A}|} \geq \Omega\left(\frac{1}{\sqrt{\log n}}\right) \quad \text{and} \tag{9}$$
$$\frac{|\mathrm{good}_H(\mathcal{S})|}{|\mathcal{D}_{i^*-1} \cup \mathcal{D}_{i^*} \cup \mathcal{D}_{i^*+1}|} \geq \Omega(1).$$

In Section 4.6.1, we handle the easy case when $i^* = t$ is the last group and its last two chunks contain the majority of good arcs:

$$\left|\mathrm{good}_H(\mathcal{S}_{s-1} \cup \mathcal{S}_s)\right| \geq 0.5 \cdot \left|\mathrm{good}_H(\mathcal{S})\right|. \tag{10}$$

4.6.1 *Case 3.0: $i^* = t$ and (10) holds.*

We prove the following lemma in [4]:

Lemma 4.10 (Lemma 3.1 for Case 3.0). *Assume that $\mathcal{D}_{i^*} = \mathcal{D}_t$ and $\left|\mathrm{good}_H(\mathcal{S}_{s-1} \cup \mathcal{S}_s)\right| \geq 0.5 \cdot \left|\mathrm{good}_H(\mathcal{S})\right|$. Then there exists an interval $I \subseteq [m]$ and a set C of arcs of H contained in I such that*

$$\frac{\mathsf{rank}_H(C)}{\mathsf{len}(I)} \geq \Omega\left(\frac{1}{\sqrt{\log n}}\right).$$

Ruling out Case 3.0, we have from (9) that there is a C_{j^*} in $\mathcal{D}_{i^*}$ with $j^* \leq s-2$ such that

$$\frac{|\mathrm{good}_H(\mathcal{S}_{j^*})|}{|\mathcal{A}|} \geq \Omega\left(\frac{1}{\log n}\right) \quad \text{and}$$
$$\frac{|\mathrm{good}_H(\mathcal{S}_{j^*})|}{|\mathcal{D}_{i^*+1} \cup \mathcal{D}_{i^*} \cup \mathcal{D}_{i^*-1}|} \geq \Omega\left(\frac{1}{\sqrt{\log n}}\right).$$

Let $C^* = \mathrm{good}_H(\mathcal{S}_{j^*})$, $\ell = \mathsf{maxlen}(C^*) \leq m/2$ since $j^* \leq s-2$ and

$$P = \mathsf{endpoints}\left(\mathcal{D}_{i^*-1} \cup \mathcal{D}_{i^*} \cup \mathcal{D}_{i^*+1}\right) \subseteq [m]$$

(so $|P| = \Theta(|\mathcal{D}_{i^*-1} \cup \mathcal{D}_{i^*} \cup \mathcal{D}_{i^*+1}|)$). Using Lemma 4.6 there is an interval I of length 2ℓ such that

$$\frac{|C^*\lfloor_I|}{\mathsf{len}(I)} = \Omega\left(\frac{1}{\log n}\right) \quad \text{and} \quad \frac{|C^*\lfloor_I|}{|P \cap I|} = \Omega\left(\frac{1}{\sqrt{\log n}}\right). \tag{11}$$

For convenience we write $C = C^*\lfloor_I$ in the rest of the proof. Every arc $\alpha \in C$ lies in $\mathcal{S}_{j^*}$, has length between $\ell/2$ and ℓ, is good and short-radius, and is contained in I (which has length 2ℓ).

We need the following definition of two arcs overlapping with each other:

Definition 4.11. We say that two arcs $\alpha = (i,j)$ and $\beta = (k,\ell)$ of H *overlap* if $\mathsf{node}(\alpha) \neq \mathsf{node}(\beta)$ are adjacent in the graph and the endpoints of the arcs satisfy $i < k < j < \ell$ or $k < i < \ell < j$.

Given C, we say an arc $\beta = (i,j)$ is *endpoint-disjoint* from C if $i, j \notin \mathsf{endpoints}(C)$, i.e. β shares no endpoint with any arc $\alpha \in C$. We write Overlap to denote the set of arcs $\beta = (i,j) \in \mathcal{A}$ that are contained in I, are endpoint-disjoint from C, and overlap with at least one arc in C. On the other hand, we write NonOverlap to denote the set of arcs $\beta \in \mathcal{A}$ that are contained in I, endpoint-disjoint from C, and do not overlap with any arc in C.

We distinguish two cases depending on the size of the set Overlap. Section 4.6.2 handles Case 3.1 when Overlap is "large", specifically,

$$\frac{|\mathsf{Overlap}|}{\mathsf{len}(I)} \geq \frac{\left|(\mathcal{D}_{i^*+1} \cup \mathcal{D}_{i^*} \cup \mathcal{D}_{i^*-1})\lfloor_I\right|}{\mathsf{len}(I)} + \frac{1}{\sqrt{\log n}}, \tag{12}$$

and Section 4.6.3 handles the opposite Case 3.2, when Overlap is "small".

4.6.2 *Case 3.1:* Overlap *is large.*

Lemma 4.12 (Lemma 3.1 for Case 3.1). *Assume that condition (12) holds, i.e* Overlap *is large. Then for the interval $I \subseteq [m]$ of Equation (11) there exists a set F of arcs of H contained in I such that*

$$\frac{\mathsf{rank}_H(F)}{\mathsf{len}(I)} \geq \Omega\left(\frac{1}{\sqrt{\log n}}\right).$$

Proof. Let $F = \mathsf{Overlap}\backslash((\mathcal{D}_{i^*+1} \cup \mathcal{D}_{i^*} \cup \mathcal{D}_{i^*-1})\lfloor_I)$. In this case we have

$$\frac{|F|}{\mathsf{len}(I)} \geq \frac{1}{\sqrt{\log n}}. \tag{13}$$

For every $\alpha \in F$ we pick arbitrarily an arc $\beta \in C$ such that α and β overlap; we call β a *witness arc* for α. (Figure 4 gives an example of Overlap and witness.) Assume without loss of generality that for the majority of $\alpha \in F$, its witness arc $\beta \in C$ is on the left of α, meaning that $\mathsf{left}(\beta) < \mathsf{left}(\alpha)$; the argument is symmetric otherwise. We write F' to denote the subset of such arcs in F; we have $|F'| \geq |F|/2$.

Next we partition the interval I into five quantiles so that each one is of length $\lfloor \mathsf{len}(I)/5 \rfloor$ or $\lceil \mathsf{len}(I)/5 \rceil$. We also assume that $\mathsf{len}(I)$ is sufficiently large so that $\lceil \mathsf{len}(I)/5 \rceil < \mathsf{len}(I)/4$; otherwise it is bounded by a constant and our goal is trivially met. (Note that C is nonempty. So I is an interval of constant length and contains at least one arc. We can thus prove Lemma 3.1 directly just by taking $B = H_I$ and one single arc in it; the ratio in (2) is $\Omega(1)$.) Let β be the witness arc of an $\alpha \in F'$. Given that $\beta \in C \subseteq C_{j^*}$, we have

$$\mathsf{len}(\beta) > \frac{\mathsf{maxlen}(C_{j^*})}{2} \geq \frac{\ell}{2} = \frac{\mathsf{len}(I)}{4} > \left\lceil \frac{\mathsf{len}(I)}{5} \right\rceil.$$

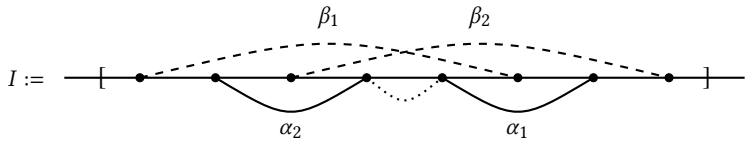

Figure 4: An example of Overlap **and witness. Here** node(β_1) **and** node(β_2) **are adjacent to** node(α_1) = node(α_2) **in** G**. The dashed arcs are in** C**, the solid arcs are in** Overlap**, and the dotted arc is in** NonOverlap**.** β_1 **is the witness of** $\alpha_1 \in F'$ **(i.e.** β_1 **is on the left of** α_1**), and** β_2 **is the witness of** $\alpha_2 \in F \setminus F'$**.**

Since β is contained in I, right(β) cannot lie in the first quantile of I. Partitioning F' into F'_i for $i = 2, 3, 4, 5$ so that F'_i contains all arcs $\alpha \in F'$ such that right(β) of its witness arc β lies in the i-th quantile of I and letting F'_q denote the largest set, we have

$$|F'_q| \geq \frac{|F'|}{4} \geq \frac{|F|}{8} = \Omega\left(\frac{\mathsf{len}(I)}{\sqrt{\log n}}\right).$$

We finish the proof by showing that $\mathsf{rank}_H(F'_q)$ is full. The proof that $\mathsf{rank}_H(F'_q)$ is full is similar to the proof of Case 2. We order arcs $\alpha_1, \ldots, \alpha_{|F'_q|}$ in F'_q by the right endpoints of their witness arcs: β_i is the witness arc of α_i and they satisfy

$$\mathsf{right}(\beta_1) \leq \cdots \leq \mathsf{right}(\beta_{|F'_q|}).$$

Note that we used $\leq$ instead of $<$ because some of the witness arcs might be the same.

We prove below that for each $i \in [|F'_q|]$,

(1) the entry of $\mathsf{impvm}_H(\alpha_i)$ indexed by edge (node(α_i), node(β_i)) is nonzero, and

(2) the entry of $\mathsf{impvm}_H(\alpha_j)$ indexed by the same edge is 0 for every $j < i$.

It follows from this triangular structure in the matrix that $\mathsf{rank}_H(F'_q)$ is full. The proof of (1) and (2) can be found in [4]. □

Lemma 3.1 follows by taking $B = H_I$ and Q to be the set of arcs of B induced by the quantile F'_q of F.

4.6.3 Case 3.2: Overlap *is small.* We are in the last case when

$$\frac{|\mathsf{Overlap}|}{\mathsf{len}(I)} < \frac{\left|(\mathcal{D}_{i^*+1} \cup \mathcal{D}_{i^*} \cup \mathcal{D}_{i^*-1})\lfloor_I\right|}{\mathsf{len}(I)} + \frac{1}{\sqrt{\log n}}.$$

Given that $|(\mathcal{D}_{i^*+1} \cup \mathcal{D}_{i^*} \cup \mathcal{D}_{i^*-1})\lfloor_I| \leq |P \cap I|$, we have from (11)

$$|\mathsf{Overlap}| < |P \cap I| + \frac{|\mathsf{len}(I)|}{\sqrt{\log n}} = O\left(\sqrt{\log n} \cdot |C|\right). \tag{14}$$

Let γ be the configuration of $S(H)$ in the statement of Lemma 3.1, and γ' be the configuration of $S(H)$ before the first move of I. We start with a sketch of the proof for this case. First we note that we can apply Lemma 4.5 to conclude that $\mathsf{rank}_H(C) = \Omega(|C|)$. This is again because the length of I is 2ℓ, all arcs in C are contained in I, and have length at least $\ell/2$. Hence we can pick a subset C' of C of size $|C'| \geq |C|/4$ such that the arcs of C' have distinct nodes. Lemma 4.5 then implies that $\mathsf{rank}_H(C) \geq \mathsf{rank}_H(C') = \Omega(|C|)$.

The main step of the proof is to construct a subset $R \subset I$ that satisfies $R \cap \mathsf{endpoints}(C) = \emptyset$. We remove the moves in R to obtain the desired subsequence B of H: $B = H_{I \setminus R}$, and let τ be the restriction of configuration γ' on $S(B)$. For each $i \in I \setminus R$, we use $\rho(i) \in [|B|]$ to denote its corresponding index in B. Then each arc $\alpha = (i, j) \in C$ corresponds to an arc $\rho(\alpha) = (\rho(i), \rho(j))$ in B (since $R \cap \mathsf{endpoints}(C) = \emptyset$, both i and j survive), and we write Q to denote the set of $|C|$ arcs of B that consists of $\rho(\alpha)$, $\alpha \in C$.

The key property we need from the set R is that the removal of moves in R does not change the improvement vector of any arc $\alpha \in C$. More formally, we prove in Lemma 4.14 that $\mathsf{impvm}_{\gamma,H}(\alpha) = \mathsf{impvm}_{\tau,B}(\rho(\alpha))$ for all $\alpha \in C$. It follows that (1) B, Q and τ satisfy the second condition of Lemma 3.1, and (2) $\mathsf{rank}_B(Q) = \mathsf{rank}_H(C) = \Omega(|C|)$. To finish the proof of Lemma 3.1, we prove in Lemma 4.13 that the length of B is small: $|B| = |I \setminus R| \leq O(\sqrt{\log n}) \cdot |C|$.

We now construct R. To help the analysis in Lemma 4.14 we will consider R as being composed of three parts, $R = R_1 \cup R_2 \cup R_3$. Given the plan above we would like to add as many indices $i \in I \setminus \mathsf{endpoints}(C)$ to R as possible since the smaller $I \setminus R$ is, the larger the ratio $|C|/|I \setminus R|$ becomes. At the same time we need to maintain the key property that the removal of R does not change the improvement vector of any arc $\alpha \in C$.

For each node $u \in S(H_I)$, we consider the following cases: (**a**) $u \in S_1(H_I)$, i.e the node u appears exactly once in the interval, (**b**) $u \in S_2(H_I)$ and u appears an even number of times and (**c**) $u \in S_2(H_I)$ and u appears an odd number of times.

Case a: $u \in S_1(H_I)$. Let $k \in I$ with $\sigma_k = u$ be the unique occurrence of u in I. If the radius of k is long-radius: $\mathsf{radius}_H(k) > 2 \cdot \mathsf{maxlen}(\mathcal{D}_{i^*})$, we add k to R_1; we leave k in $I \setminus R$ otherwise. The idea here is that if the radius of k is long-radius, then given that every arc $\alpha \in C$ is short-radius, we have $k \notin \mathsf{interior}_H(\alpha)$ and thus, the removal of k does not affect the improvement vector of α. On the other hand, if the radius of k is small, then it is an endpoint of two arcs $(\mathsf{pred}_H(k), k)$ and $(k, \mathsf{succ}_H(k))$ and both have length at most $2 \cdot \mathsf{maxlen}(\mathcal{D}_{i^*})$. At the same time, given that u only appears once in I and that I has length 2ℓ, at least one of them has length at least $\ell \geq 2^{j^*-1} + 1$. As a result, we have $k \in P$ when k is not added to R_1.

Case b: $u \in S_2(H_I)$ and u appears an even number of times in H_I. Let $k_1 < k_2 < \cdots < k_{2q}$ be the occurrences of u in I for some $q \geq 1$. Then for each $i \in [q]$, we add both k_{2i-1} and k_{2i} to R_2 if $(k_{2i-1}, k_{2i}) \in$ NonOverlap, and keep both in $I \setminus R$ otherwise. Note that if $(k_{2i-1}, k_{2i}) \notin$ NonOverlap, then either $(k_{2i-1}, k_{2i}) \in$ Overlap or at least one of the two

endpoints is in endpoints(C). As a result, we can conclude that the number of these $2q$ indices that do not get added to R_2 can be bounded from above by

$$O\Big(\text{number of } \beta \in \mathsf{Overlap} \cup C \text{ with } \mathsf{node}_H(\beta) = u\Big).$$

See Figure 5 for an illustration of case a and case b.

Case c: $u \in S_2(H_I)$ and u appears an odd number of times in H_I. Let $k_1 < \cdots < k_{2q+1}$ be the occurrences of u in I for some $q \geq 0$. See Figure 6 for an illustration of case c.

Case $\mathbf{c}_1$: If the number of $\beta \in \mathsf{Overlap} \cup C$ with $\mathsf{node}_H(\beta) = u$ is at least 1, then we can handle this case similarly as Case 2: For each $i \in [q]$ we add k_{2i-1} and k_{2i} to R_2 if the arc (k_{2i-1}, k_{2i}) is in NonOverlap, and we always keep k_{2q+1} in $I \setminus R$. In this case, the number of these $2q+1$ indices that do not get added to R_3 is

$$1 + O\Big(\text{number of } \beta \in \mathsf{Overlap} \cup C \text{ with } \mathsf{node}_H(\beta) = u\Big)$$

which remains an $O(\cdot)$ of the same quantity given that the latter is at least 1.

Case $\mathbf{c}_2$: Consider the case when there is no $\beta \in \mathsf{Overlap} \cup C$ with $\mathsf{node}_H(\beta) = u$. We start with the easier situation when there is no $k \in I$ such that $\sigma_k = u$ and $k \in \mathsf{interior}_H(\alpha)$ for some $\alpha \in C$. In this case we add all $k \in I$ with $\sigma_k = u$ to R_3. Note that the $(\mathsf{node}_H(\alpha), u)$-th entry of the improvement vector of every $\alpha \in C$ is 0. So removing all occurrences of u has no effect.

Case $\mathbf{c}_3$: We are left with the case when there is no $\beta \in \mathsf{Overlap} \cup C$ with $\mathsf{node}_H(\beta) = u$ and at the same time, there is an arc $\alpha \in C$ such that $k_i \in \mathsf{interior}_H(\alpha)$ for some i. Combining these two assumptions we must have that $k_i \in \mathsf{interior}_H(\alpha)$ for all $i \in [2q+1]$. Given that α is a short-radius arc, we have that the radius of both k_1 and k_{2q+1} is at most $2 \cdot \mathsf{maxlen}(\mathcal{D}_{i^*})$. On the other hand, given that $\mathsf{len}(I) = 2\ell$ and $\mathsf{len}(\alpha) \leq \ell$, the radius of either k_1 or k_{2q+1} is at least $\ell/2 \geq 2^{j^*-2}$. If this holds for k_1, we add $k_2, \ldots, k_{2q+1}$ to R_2 and keep k_1 in $I \setminus R$; otherwise we add $k_1, \ldots, k_{2q}$ to R_2 and keep k_{2q+1} in $I \setminus R$. In both cases the index left in $I \setminus R$ lies in P.

Summarizing Case b and Case c, we have that R_2 consists of endpoints of a collection of endpoint-disjoint arcs in NonOverlap. Moreover, the number of indices left in $I \setminus R$ can be bounded by

$$|P \cap I| + O(|\mathsf{Overlap} \cup C|). \tag{15}$$

This gives us the following bound on $|I \setminus R|$:

Lemma 4.13. $|I \setminus R| \leq O(\sqrt{\log n}) \cdot |C|$.

Proof. This follows by combining (15), (11) and (14). □

Finally we show in the full version [4] that there is no change in the improvement vectors of $\alpha \in C$ after removing R.

Lemma 4.14. *For every arc $\alpha \in C$ of H, its corresponding arc $\beta = \rho(\alpha) \in Q$ of B satisfies*

$$\mathsf{impvm}_{\tau,B}(\beta) = \mathsf{impvm}_{\gamma,H}(\alpha).$$

Lemma 4.15 (Lemma 3.1 for Case 3.2). *Assume that condition (14) holds, i.e.,* Overlap *is small. Then there exist (i) a sequence B of moves of length at most $5n$, (ii) a configuration τ of $S(B)$, and (iii) a set of arcs Q of B such that*

(1) The rank of Q in B satisfies

$$\frac{\mathsf{rank}_B(Q)}{\mathsf{len}(B)} \geq \Omega\left(\frac{1}{\sqrt{\log n}}\right) \qquad (A);$$

(2) For every arc $\alpha \in Q$, there exists an arc α' of H such that

$$\mathsf{impvm}_{\tau,B}(\alpha) = \mathsf{impvm}_{\gamma,H}(\alpha') \qquad (B).$$

Proof. Indeed using the interval $I \subseteq [m]$ of Equation (11), we set $B = H_{I \setminus R}$ and τ be the restriction of configuration γ' on $S(B)$. We set also Q to the arcs of B which are induced by collection $C = C^*\downarrow_I$ of (11). Lemma 4.14 shows that (B)—Vector-Preservation— property holds for B, Q, τ. Finally the aforementioned analysis shows that (i) $\mathsf{rank}(C)$ is almost full or equivalently that $\mathsf{rank}_B(Q) \geq \Omega(|Q|)$ and (ii) using Lemma 4.13, $\mathsf{len}(B) = \mathsf{len}(H_{I \setminus R}) \leq O(\sqrt{\log n}) \cdot |Q|$ implying that (A)—High-rank property— holds too. □

5 BINARY MAX-CSP AND FUNCTION PROBLEMS

Definition 5.1. An instance of Max-CSP (Constraint Satisfaction Problem) consists of a set $V = \{x_1, \ldots, x_n\}$ of variables that can take values over a domain D, and a set $C = \{c_1, \ldots, c_m\}$ of constraints with given respective weights $w_1, \ldots, w_m$. A constraint c_i is a pair (R_i, t_i) consisting of a relation R_i over D of some arity r_i (i.e. $R_i \subseteq D^{r_i}$), and an r_i-tuple of variables (i.e., $t_i \in V^{r_i}$). An assignment $\tau : V \to D$ satisfies the constraint c_i if $\tau(t_i) \in R_i$. The MAX CSP problem is: given an instance, find an assignment that maximizes the sum of the weights of the satisfied constraints.

We will focus here on the case of binary domains D, which wlog we can take to be $\{0, 1\}$, and binary relations ($r_i = 2$); we refer to this as Binary Max-CSP, or Max-2CSP. Several problems can be viewed as special cases of Binary Max-CSP where the relations of the constraints are restricted to belong to a fixed family $\mathcal{R}$ of relations; this restricted version is denoted Max-CSP($\mathcal{R}$). For example, the Max Cut problem in graphs is equivalent to Max-CSP($\mathcal{R}$) where $\mathcal{R}$ contains only the "not-equal" (binary) relation $\neq$ (i.e., the relation $\{(0,1),(1,0)\}$). Other examples include:

- *Directed Max Cut.* Given a directed graph with weights on its edges, partition the set of nodes into two sets V_0, V_1 to maximize the weight of the edges that are directed from V_0 to V_1. This problem is equivalent to Max-CSP($\mathcal{R}$) where $\mathcal{R}$ consists of the relation $\{(0,1)\}$.
- *Max 2SAT.* Given a weighted set of clauses with two literals in each clause, find a truth assignment that maximizes the weight of the satisfied clauses. This is equivalent to Max-CSP($\mathcal{R}$), where $\mathcal{R}$ contains 4 relations, one for each of the 4 possible clauses with two literals $a \vee b, \bar{a} \vee b, a \vee \bar{b}, \bar{a} \vee \bar{b}$; the relation for a clause contains the three assignments that satisfy the clause. If we allow unary clauses in the 2SAT instance, then we include in $\mathcal{R}$ also the two unary constraints for a and $\neg a$.

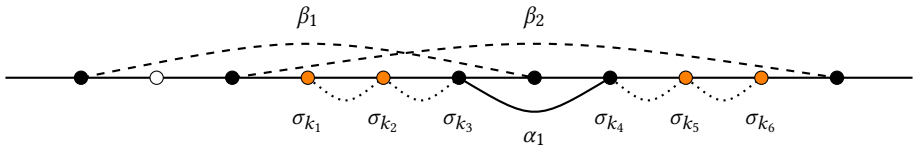

Figure 5: Illustration of Case a and Case b. Here The dashed arcs are in C, the solid arc is in Overlap**, and the dotted arcs are in** NonOverlap**. The circle is in B_1, and the orange nodes are in B_2.**

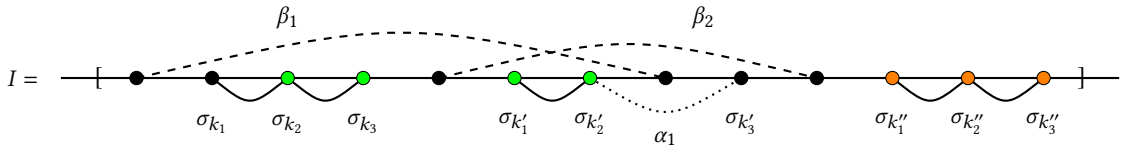

Figure 6: Illustration of Case c. Here node(β_1) **and** node(β_2) **are adjacent to** node(α_1) **in G. The dashed arcs are in C, the dotted arc is in** Overlap**, and the solid arcs are in** NonOverlap**. The green nodes are in B_3, and the orange nodes are in B_2.**

- *Stable Neural Network.* A neural network in the Hopfield model [9] is an undirected graph $G = (V, E)$ (the nodes correspond to the neurons, the edges to the synapses) with a given weight w_e for each edge $e \in E$ and a given threshold t_u for each node $u \in V$; the weights and thresholds are not restricted in sign. A configuration γ is an assignment of a value (its 'state') -1 or 1 to each node. A node u is stable in a configuration γ if $\gamma(u) = -1$ and $t_u + \sum_{v:(u,v)\in E} w_{(u,v)}\gamma_v \leq 0$, or $\gamma(u) = 1$ and $t_u + \sum_{v:(u,v)\in E} w_{(u,v)}\gamma_v \geq 0$. A configuration is *stable* if all the nodes are stable in it. Hopfield showed that every neural network has one or more stable configurations, using a potential function argument: a node u is unstable in a configuration γ iff flipping its state increases the value of the potential function $p(\gamma) = \sum_{u\in V} t_u \cdot \gamma(u) + \sum_{(u,v)\in E} w_{(u,v)} \cdot \gamma(u)\gamma(v)$. Hence, γ is stable iff $p(\gamma)$ cannot be increased by flipping the state of any node. Thus, the problem of finding a stable configuration is the same as the local Max-CSP($\mathcal{R}$) problem when $\mathcal{R}$ contains the unary constraint a and the binary constraint $a = b$. The natural greedy algorithm, in which unstable nodes asynchronously (one-at-a-time) flip their state (in any order) monotonically increases the potential function and converges to a stable configuration[6]. Our results apply to the smoothed analysis of this natural dynamics for neural networks.
- *Network coordination game* with 2 strategies per player. We are given a graph $G = (V, E)$ where each node corresponds to a player with 2 strategies, and each edge (u, v) corresponds to a game $\Gamma_{u,v}$ between players u, v with a given payoff matrix (both players get the same payoff in all cases). The total payoff of each player for any strategy profile is the sum of the payoffs from all the edges of the player. The problem is to find a pure equilibrium (there always exists one as these are potential games). This problem can be viewed as a special case of local Max-CSP($\mathcal{R}$) where $\mathcal{R}$ contains the 4 singleton relations $\{(0, 0)\}, \{(0, 1)\}, \{(1, 0)\}, \{(1, 1)\}$, and we want to find a locally optimal assignment that cannot be improved by flipping any single variable. The FLIP algorithm in this case is the better response dynamics of the game. Boodaghians et. al. [3] studied the smoothed complexity of the better response algorithm for network coordination games, where each entry of every payoff matrix is independently drawn from a probability distribution supported on $[-1, 1]$ with density at most ϕ. They showed that for general graphs and k strategies per player the complexity is at most $\phi \cdot (nk)^{O(k \log(nk)}$ with probability $1 - o(1)$, and in the case of complete graphs it is polynomial.

A constraint can be viewed as a function that maps each assignment for the variables of the constraint to a value 1 or 0, depending on whether the assignment satisfies the constraint or not. We can consider more generally the *Binary function optimization problem* (BFOP), where instead of constraints we have functions (of two arguments) with more general range than $\{0, 1\}$, for example $\{0, 1, \ldots, k\}$, for some k fixed (or even polynomially bounded): Given a set $V = \{x_1, \ldots, x_n\}$ of variables with domain $D = \{0, 1\}$, a set $F = \{f_1, \ldots, f_m\}$ of functions, where each f_i is a function of a pair t_i of variables, and given respective weights $w_1, \ldots, w_m$, find an assignment $\tau : V \to D$ to the variables that maximizes $\sum_{i=1}^{m} w_i \cdot f_i(\tau(t_i))$. In the local search version, we want to find an assignment that cannot be improved by flipping the value of any variable. For smoothed analysis, the weights w_i are drawn independently from given bounded distributions as in Max Cut. We will show that the bounds for Max Cut extend to the general Binary Max-CSP and Function optimization problems with arbitrary (binary) constraints and functions.

Consider an instance of BFOP. Even though a function (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 the case that the function depends on both variables but the two variables 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', f'' such that $f(x, y) = f'(x) + f''(y)$ for all values

[6]Note that if unstable nodes flip their state simultaneously then the algorithm may oscillate and not converge to a stable configuration.

of x, y; otherwise f is *nonseparable*. For binary domains there is a simple easy criterion; see the proof in [4].

Lemma 5.2. *A binary function f of two arguments with domain $\{0,1\}$ is separable iff $f(0,0) + f(1,1) = f(0,1) + f(1,0)$.*

If our given instance of BFOP has some separable functions, then we can replace them with the equivalent unary functions. After this, we have a set of unary and binary functions, where all the binary functions are nonseparable.

Consider a sequence H of variable flips starting from an initial assignment (configuration) $\gamma \in \{0,1\}^n$. When we flip a variable x_j in some step, the change in the value of the objective function can be expressed as $\langle \delta, X \rangle$, where the coordinates of the vectors δ, X correspond to the functions of the given instance, the vector δ gives the changes in the function values and X is the vector of random weights of the functions. Define the matrix $M_{H,\gamma}$ which has a row corresponding to each nonseparable function f_i and a column for each pair of closest flips of the same variable in the sequence H, where the column is the sum of the vectors δ_1, δ_2 for the two flips, restricted to the nonseparable functions. Note that for separable functions, the corresponding coordinate of $\delta_1 + \delta_2 = 0$. Thus, the sum of the changes in the value of the objective function in the two closest flips of the same variable is equal to the inner product of the column δ with the vector of random weights of the nonseparable functions. The proof of the following lemma can be found in [4].

Lemma 5.3. *The entry of the matrix $M_{H,\gamma}$ at the row for the (nonseparable) function f_i and the column corresponding to an arc α of a variable x_j is nonzero iff x_j is one of the variables of the function f_i and the other variable x_k of f_i appears an odd number of times in the interior of α.*

Thus, the zero-nonzero structure of the matrix $M_{H,\gamma}$ is the same as that of the matrix for the Max Cut problem on the graph G which has the variables as nodes and has edges corresponding to the nonseparable functions with respect to the same initial configuration γ and sequence of flips H. The arguments in the proof for the Max Cut problem that choose a subsequence and bound the rank of the corresponding submatrix depend only on the zero-nonzero structure of the matrix and not on the precise values: In every case, we identify a diagonal submatrix or a triangular submatrix of the appropriate size. Therefore, we can apply the same analysis for the general Max-2CSP and BFOP problems with arbitrary binary constraints or functions, proving Theorem 1.3.

6 CONCLUSIONS

We analyzed the smoothed complexity of the FLIP algorithm for local Max-Cut, and more generally, for binary maximum constraint satisfaction problems (like Max-2SAT, Max-Directed Cut, Stable Neural Network etc.). We showed that with high probability, every execution of the FLIP algorithm for these problems, under any pivoting rule, takes at most $\phi n^{O(\sqrt{\log n})}$ steps to terminate. The proof techniques involve a sophisticated analysis of the execution sequences of flips that are potentially generated by the FLIP algorithm, with the goal of identifying suitable subsequences (including non-contiguous subsequences) that contain many steps with linearly independent improvement vectors, which are preserved from the full execution sequence. We do not know at this point whether the $\sqrt{\log n}$ in the exponent, which is due to the ratio between the length and the rank of the subsequence, can be improved or is best possible for this approach, i.e. whether our new rank lemma for subsequences is tight.

There are several other interesting open questions raised by this work. One question concerns the extension to non-binary constraints. For example, does a similar result hold for Max-3SAT? Does it hold generally for all Max-CSP with binary domains? There are several new challenges in addressing these questions.

Another question concerns the extension to domains of higher cardinality k. Simple examples of Max-2CSP with larger domain include Max-k-Cut, where the nodes are partitioned into k parts instead of 2 as in the usual Max-Cut, and the Network Coordination Game with k strategies per player. Bibak et. al. studied Max-k-Cut and showed that the FLIP algorithm converges with high probability in $\phi n^{O(\log n)}$ steps for general graphs for fixed k (and polynomial time for complete graphs if $k = 3$) [2]. Boodaghians et. al. studied the network coordination game and showed a similar bound $\phi n^{O(\log n)}$ for general graphs for fixed k (and in the case of complete graphs, polynomial time for all fixed k) [3]. Can the $\log n$ in the exponent be improved to $\sqrt{\log n}$ for these problems using a combination of the techniques in these papers and the present paper, and more generally does it hold for all Max-2CSP problems with non-binary domains?

Ultimately, is the true smoothed complexity of Local Max-CSP problems polynomial or are there bad examples of instances and distributions that force super-polynomial behavior?

REFERENCES

[1] Omer Angel, Sébastien Bubeck, Yuval Peres, and Fan Wei. 2017. Local max-cut in smoothed polynomial time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*. ACM, 429–437.

[2] Ali Bibak, Charles Carlson, and Karthekeyan Chandrasekaran. 2019. Improving the smoothed complexity of FLIP for max cut problems. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 897–916.

[3] Shant Boodaghians, Rucha Kulkarni, and Ruta Mehta. 2019. Smoothed Efficient Algorithms and Reductions for Network Coordination Games. *CoRR* abs/1809.02280v4 (2019). arXiv:1809.02280 http://arxiv.org/abs/1809.02280v4

[4] Xi Chen, Chenghao Guo, Emmanouil-Vasileios Vlatakis-Gkaragkounis, Mihalis Yannakakis, and Xinzhi Zhang. 2019. Smoothed complexity of local Max-Cut and binary Max-CSP. arXiv:cs.DS/1911.10381

[5] Robert Elsässer and Tobias Tscheuschner. 2011. Settling the complexity of local max-cut (almost) completely. In *International Colloquium on Automata, Languages, and Programming*. Springer, 171–182.

[6] Michael Etscheid and Heiko Röglin. 2015. Smoothed analysis of the squared Euclidean maximum-cut problem. In *Algorithms-ESA 2015*. Springer, 509–520.

[7] Michael Etscheid and Heiko Röglin. 2017. Smoothed Analysis of Local Search for the Maximum-Cut Problem. *ACM Trans. Algorithms* 13, 2, Article 25 (March 2017), 12 pages. https://doi.org/10.1145/3011870

[8] Alex Fabrikant, Christos Papadimitriou, and Kunal Talwar. 2004. The complexity of pure Nash equilibria. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*. ACM, 604–612.

[9] J. J. Hopfield. 1982. Neural networks and physical systems with emergent collective computational abilities. *Proc. Nat. Acad. Sci. USA* 79 (1982), 2554–2558.

[10] David S Johnson, Christos H Papadimitriou, and Mihalis Yannakakis. 1988. How easy is local search? *J. Comput. System Sci.* 37, 1 (1988), 79–100.

[11] Victor Klee and George J Minty. 1970. *How good is the simplex algorithm.* Technical Report. Washighton University, Seattle, Department of Mathematics.

[12] Alejandro A Schäffer and Mihalis Yannakakis. 1991. Simple local search problems that are hard to solve. *SIAM journal on Computing* 20, 1 (1991), 56–87.

[13] Daniel A Spielman and Shanghua Teng. 2009. Smoothed analysis: an attempt to explain the behavior of algorithms in practice. *parallel computing* 52, 10 (2009), 76–84.

[14] Daniel A Spielman and Shang-Hua Teng. 2004. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM (JACM)* 51, 3 (2004), 385–463.