Optimal Parallel Basis Finding in Graphic and Related Matroids

Sanjeev Khanna
Courant Institute, Warren Weaver Hall, New York University, NY, USA

Aaron Putterman
School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA

Junkai Song
Courant Institute, Warren Weaver Hall, New York University, NY, USA

Abstract

We study the parallel complexity of finding a basis of a *graphic* matroid under independence-oracle access. Karp, Upfal, and Wigderson (FOCS 1985, JCSS 1988) initiated the study of this problem and established two algorithms for finding a spanning forest: one running in $O(\log m)$ rounds with $m^{\Theta(\log m)}$ queries, and another, for any $d \in \mathbb{Z}^+$, running in $O(m^{2/d})$ rounds with $\Theta(m^d)$ queries. A key open question they posed was whether one could simultaneously achieve *polylogarithmic* rounds and *polynomially many* queries.

We give a *deterministic* algorithm that uses $O(\log m)$ adaptive rounds and $\text{poly}(m)$ non-adaptive queries per round to return a spanning forest on m edges, and complement this result with a matching $\Omega(\log m)$ lower bound for any (even randomized) algorithm with $\text{poly}(m)$ queries per round. Thus, the adaptive round complexity for graphic matroids is characterized *exactly*, settling this long-standing problem.

Beyond graphs, we show that our framework also yields an $O(\log m)$-round, $\text{poly}(m)$-query algorithm for any binary matroid satisfying a *smooth circuit counting* property, implying, among others, an optimal $O(\log m)$-round parallel algorithms for finding bases of *cographic* matroids. Finally, we conjecture a natural strengthening of known circuit-counting bounds for the much broader class of *regular* matroids and even an extension to so-called *max-flow min-cut matroids*; assuming it, our algorithm achieves the same $O(\log m)$ rounds and $\text{poly}(m)$ queries for all such matroids – which includes graphic and cographic matroids as special cases.

2012 ACM Subject Classification Theory of computation → Parallel algorithms

Keywords and phrases parallel algorithms, matroids

Digital Object Identifier 10.4230/LIPIcs.ICALP.2026.125

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2511.04826

Funding *Sanjeev Khanna*: Supported in part by NSF award CCF-2625203 and AFOSR award FA9550-25-1-0107.
Aaron Putterman: Supported in part by the Simons Investigator Awards of Madhu Sudan and Salil Vadhan and AFOSR award FA9550-25-1-0112.
Junkai Song: Supported in part by NSF award CCF-2625203.

1 Introduction

A key direction in combinatorial optimization is to understand how efficiently problems can be solved in parallel. This is often studied by analyzing the number of *adaptive* rounds an algorithm requires, where in each round the algorithm performs a polynomial amount of non-adaptive computation. An early and highly influential contribution in this direction was made by Karp, Upfal, and Wigderson [24, 26], who studied the role of adaptivity in

© Sanjeev Khanna, Aaron Putterman, and Junkai Song;
licensed under Creative Commons License CC-BY 4.0

computing a basis of a matroid given access to an *independence oracle*. Recall that a matroid $M = (E, I)$ consists of a *ground set* E (with $|E| = m$) and a family $I \subseteq 2^E$ of independent sets satisfying the following properties:

1. $\emptyset \in I$.
2. If $S \in I$ and $S' \subseteq S$, then $S' \in I$.
3. If $S \in I$, $S' \in I$, and $|S'| < |S|$, then there exists $x \in S \setminus S'$ such that $S' \cup \{x\} \in I$.

A *basis* of a matroid is an independent set of maximal size. The task of finding a basis of a matroid is a common generalization of many well-studied problems including finding a basis of a vector space and finding a spanning forest of a graph. However, since the number of matroids on m elements is super-exponential in m, it is generally infeasible to describe a matroid explicitly. Instead, access is typically provided via a suitable oracle. In the most basic setting (which we use in this paper), an algorithm can query an *independence oracle* that, given any $S \subseteq E$, returns whether $S \in I$ (i.e., whether S is independent).

In the decades since [24], the study of parallel algorithms has expanded into several related areas. For instance, Blikstad [7] and Blikstad and Tu [8] studied the related problem of *matroid intersection* under independence-oracle (or rank-oracle) access. In this problem, one is given two matroids M_1, M_2 on the same ground set and tasked with finding a largest set of elements that is independent in both M_1 and M_2. This framework generalizes other optimization problems like bipartite matching, arborescences, and tree packing, and this line of research has culminated in algorithms requiring a sublinear (in m) number of rounds. Understanding parallel complexity has also emerged as a fundamental topic in the study of submodular function minimization [5, 10, 11] and submodular function maximization [1–4, 9, 12–18, 27, 29] where, similar to our setting, function evaluations are given via oracle access. There has also been extensive study on parallel algorithms for graph matchings [19, 25, 30, 35], and parallel algorithms for finding maximal independent sets[1] in graphs [20, 31].

Despite significant progress on parallel algorithms for these various problems, many of the questions raised in [24] remain unresolved. For instance, in the setting of general matroids, [24] showed that any algorithm making $\text{poly}(m)$ queries per round must use at least $\Omega\left(\frac{m^{1/3}}{\log m}\right)$ adaptive rounds to find a basis. Conversely, they also showed that a basis of any matroid can be found in $O(\sqrt{m})$ rounds. Only recently have these bounds improved, with the work of Khanna, Putterman, and Song [28] presenting a new $\widetilde{O}(m^{7/15})$ round algorithm in this general case. Nevertheless, the true complexity of these problems is far from known, and as a step toward resolving the general case (as well as being a natural problem in its own right), [24] proposed focusing on the simpler setting of *graphic matroids*.

In a graphic matroid, the ground set E corresponds to the edge set of a graph. A set $S \subseteq E$ of edges is *independent* if no cycle is contained in S (so independent sets are acyclic subgraphs), and a *basis* of a graphic matroid is a spanning forest of the graph. Crucially, the algorithm sees only the oracle answers, and not the underlying graph (i.e., it can only query a set of edge labels, and the oracle returns whether that set contains a cycle). For this setting, [24] gave two incomparable algorithms for spanning forests via independence queries: one runs in $O(\log m)$ rounds with $m^{\Theta(\log m)}$ queries, and another, for any constant $d \in \mathbb{Z}^+$, runs in $O(m^{2/d})$ rounds with $\Theta(m^d)$ queries. Whether one can achieve *simultaneously* polylogarithmic rounds and polynomially many queries has remained unresolved for four decades, serving as the central motivation of our work.

[1] Note that these are independent sets in the sense that they are a set of vertices without edges between them, not in the matroid sense.

1.1 Our Contributions

Graphic Matroids. For the setting of graphic matroids, we provide a complete resolution of the complexity of finding a basis when given access to an independence oracle. We first give a *deterministic* algorithm with $O(\log m)$ rounds and $\text{poly}(m)$ queries per round:

▶ **Theorem 1.** *There is a* deterministic *parallel algorithm that, for any graphic matroid G with m elements, uses $O(\log(m))$-rounds, and at most* $\text{poly}(m)$ *non-adaptive queries to an independence oracle per round which returns a spanning forest of G.*

This result resolves an open question posed by [24]: it shows that the adaptive round complexity can be made logarithmic *without* blowing up the per-round query complexity to super-polynomial.

In fact, since the above theorem can be used to compute the rank of a graphic matroid in $O(\log(m))$ rounds, we can take advantage of [8]'s black-box recipe for finding *minimum* and *maximum* weight bases[2]:

▶ **Corollary 2.** *There is a* deterministic *parallel algorithm that uses $O(\log(m))$-rounds, and at most* $\text{poly}(m)$ *non-adaptive queries, which for any weighted, graphic matroid G, returns a minimum (or maximum) weight spanning forest.*

We complement our algorithmic results with a matching lower bound, ruling out even a randomized algorithm with asymptotically smaller number of rounds when queries per round are restricted to be polynomial.

▶ **Theorem 3.** *Let A be any randomized algorithm which, for graphic matroids on m elements, uses at most* $\text{poly}(m)$ *queries to an independence oracle per round. Then, in expectation, A must use $\Omega(\log(m))$ adaptive rounds to find a basis of a graphic matroid.*

Together, these results give a complete picture of the parallel complexity landscape for bases in graphic matroids, fully settling the question raised by [24] in this setting.

Beyond Graphic Matroids. Beyond the graphic case, we next show that the same cycle-structural framework extends to broader classes: namely binary ($\mathbb{F}_2$-representable) matroids that satisfy a *smooth circuit counting* bound.

▶ **Definition 4** (Binary Matroid). *A matroid $M = (E, I)$ is said to be* binary *($\mathbb{F}_2$ representable) if there is a map from every element $e_i \in E$, to a vector v_i over $\mathbb{F}_2$ such that a set $S \subseteq E$ is independent if and only if the corresponding set of vectors $\{v_i : e_i \in S\}$ is linearly independent in $\mathbb{F}_2$.*

▶ **Definition 5** (Smooth Circuit Counting Bound). *For a matroid $M = (E, I)$, a* circuit *in M is any minimal dependent set of elements. For a set of elements $S \subseteq E$, $M|_S = (S, I \cap 2^S)$. We say that a matroid $M = (E, I)$ satisfies a smooth circuit counting bound if for every $S \subseteq E$, if $M|_S$ has minimum circuit length ℓ, then for any $\alpha \in \mathbb{Z}^+$, the number of circuits of length $\leq \alpha \cdot \ell$ in $M|_S$ is at most $m^{O(\alpha)}$.*

Indeed, with these definitions and by leveraging the toolkit from the graphic matroid setting, we have the following more general algorithmic statement:

[2] Note that in this setting, each element of the matroid is also given an associated weight. The weight of a basis is the sum of the weights of the constituent elements.

▶ **Theorem 6.** *There is a randomized $O(\log m)$-round, $\mathrm{poly}(m)$-query algorithm that, for any binary matroid $M = (E, I)$ on m elements satisfying the smooth circuit counting bound, finds a basis of M with high probability.*

In fact, this automatically implies such a bound for the class of *cographic* matroids. For a graph G, its cographic matroid is the matroid whose circuits are precisely the *cuts* of G. Cographic matroids satisfy the smooth circuit counting bound as shown by Karger's classic cut-counting bound [23], thus yielding the corollary below:

▶ **Corollary 7.** *There is an algorithm which, for any cographic matroid M on m elements, finds a basis of M in $O(\log(m))$-rounds and $\mathrm{poly}(m)$-independence queries with high probability.*

On the other hand, by considering the cographic matroids defined by the family of instances used in lower bound for graphic matroids (Theorem 3), we can also show that these $\Omega(\log(m))$ rounds are necessary for cographic matroids, thereby also settling the round complexity of basis finding in cographic matroids.

Beyond this, we can even consider the well-studied class of *regular* matroids (matroids that are representable over *every* field). For such matroids, the binary condition is true by definition and a weakened form of the smooth circuit counting bound is known to be true. Indeed, the work of Gurjar and Vishnoi [22] showed that in regular matroids, the number of circuits of size $\leq \alpha\ell$ (for ℓ being the minimum circuit size) is at most $m^{O(\alpha^2)}$. In fact, they even showed that this same bound holds for so-called "max-flow min-cut" matroids, a class that generalizes beyond regular matroids. Intuitively, these are matroids where certain circuit structures in the matroid (analogous to the max flow) are equal to an appropriately defined structure relating to complements of circuits in the *dual matroid* (analogous to the minimum cut). Such matroids have seen extensive study in their own right, and we refer the reader to [21, 32, 33] for further discussion.

Nevertheless, this raises the tantalizing possibility of whether the counting bound of [21] can be strengthened to match the asymptotics of Definition 5:

▶ **Conjecture 8.** *Let $M = (E, I)$ be a max-flow min-cut matroid on m elements with minimum circuit length ℓ. Then, for any $\alpha \in \mathbb{Z}^+$, M has at most $m^{O(\alpha)}$ circuits of length $\leq \alpha \cdot \ell$.*

As stated above, a proof of Conjecture 8 would immediately imply an $O(\log(m))$-round, $\mathrm{poly}(m)$-query algorithm for all max-flow min-cut matroids. Likewise a proof of Conjecture 8 only for the smaller class of *regular matroids* would also imply an algorithm of the same complexity for regular matroids. With the statement of our results provided, we now proceed to a discussion of the techniques we use to prove them.

A High Level Discussion of Techniques and Comparison With Prior Work

Note that despite the recent progress in parallel basis finding algorithms for the general matroid case [28], the techniques in this paper are entirely disjoint. At a high level, [28] relies on a polynomial round matroid decomposition technique, which is based on random permutations, prefix circuits, and so-called greedily-optimal sets. This decomposition then enables basis finding via contraction on large independent sets and deleting elements in the general matroid setting, and does not yield any new results for the graphic matroid setting.

Our work instead develops new cycle-structure tools specific to graphs and certain other classes of matroids. These provide bounds on cycle overlaps, introduce randomized sampling to isolate short cycles, and allow for derandomization yielding universal query sets. This avoids the need for contractions altogether and instead only iteratively deletes redundant edges (while preserving connectivity). On a more technical level, our results crucially rely on

the underlying matroid obeying smooth circuit counting bounds (see Definition 5). While such bounds hold for some classes of matroids, they are *not* true for all matroids (for instance, they even fail for *uniform matroids*).

This fundamental difference is also reflected in the results: general matroids admit only polynomial-round algorithms for basis finding, with a known $\widetilde{\Omega}(m^{1/3})$ round lower bound. In this work, we instead obtain tight logarithmic-round algorithms for graphic and similarly structured matroids. This constitutes the first progress in this specific setting, thereby settling the question of the complexity of the graphic case, as posed by [24].

1.2 Technical Overview

In this section, we describe the techniques that underlie our contributions in more detail. Before explaining our exact algorithms however, we provide a brief overview of how prior algorithms work, in order to give a better context for our contribution.

1.2.1 Prior Work

As part of the existing algorithmic literature for finding bases of graphic matroids, we discuss two broad paradigms for making progress towards recovering bases in general matroids (and therefore also for graphic matroids):

1.2.1.1 Deleting Redundant Elements

The first paradigm used in [24] for recovering bases of graphic matroids is the deletion of redundant elements. More specifically, given a graphic matroid M and a subset T of the elements in M, we can say that an element $b \in M$ is *redundant given* T if there is a cycle in $T \cup b$ that uses the element b. Beyond this, one can imagine recovering an entire set B, where for every element $b \in B$, the element b forms a cycle using elements in T.

Because these are graphic matroids (and therefore by definition model a graph), deleting these redundant elements *does not* alter the connected components of the underlying graph. In other words, if B is a set of redundant edges then the *sparser* graph encoded by $M \setminus B$ has the same connectivity structure as the graph encoded by M.

1.2.1.2 Contracting on Independent Sets

The second paradigm used in the literature is to contract on large independent sets. Given a graphic matroid M, recall that a set of edges T is *independent* if the subgraph defined by T has no cycles. *Contracting* on T means merging the endpoints of every edge in T (akin to standard graph edge contractions [6]). As long as T is independent, the contracted matroid M/T has smaller rank: in fact, $\text{rank}(M/T) = \text{rank}(M) - |T|$. Moreover, independence queries to M/T can be simulated by queries to M: any query $S \subseteq M/T$ (a set of edges in the contracted graph) corresponds to querying $T \cup S$ in the original matroid M. Thus, contracting on an independent set T reduces the problem of finding a basis of M to finding a basis of the smaller matroid M/T.

1.2.1.3 Using Deletion and Contraction on Graphic Matroids

The work of [24] provides a natural approach for finding bases of graphic matroids that alternates between deleting redundant elements and contracting on independent sets. To start, [24] fixes a parameter d. This can either be thought of as a fixed constant, or $O(\log(m))$, depending on which of their results one wishes to derive. Now, in a single round, [24] queries the independence oracle with all subsets of $\leq d$ edges from M.

In a single round, this requires $O(m^d)$ queries, but also *completely* enumerates all cycles of length $\leq d$. This is because for any cycle C in M of length $\ell \leq d$, deleting any element $x \in C$ yields an independent set, while querying the entire set C yields a dependent set. Thus, just by looking at the responses to these $O(m^d)$ queries, the algorithm can determine all cycles of length $\leq d$. Finally, after discovering all such cycles, [24] shows that there is a canonical way to *delete* a set of redundant edges, which yields a graphic matroid where there are *no* cycles of length $\leq d$, but without altering the connected components. We denote these deleted elements by S.

However, as performed so far, this algorithm is not repeatable, as finding the cycles of length $d+1$ will require even more queries than finding those of length $\leq d$. Thus, the key observation from [24] is to now find a *large independent set*, conditioned on the fact that there are no cycles of length $\leq d$ in the graph. In particular, the authors show that subsampling the elements at rate approximately $\frac{1}{m^{1/d}}$ yields an independent set (set of edges with no cycles) with high probability (and can in fact be derandomized). Once such an independent set is found (denote this by T), the algorithm then contracts on T. Thus in only 2 rounds, and using only $O(m^d)$ queries, the authors reduce the problem of finding a basis in M, to the problem of finding a basis in $(M \setminus S)/T$. The key invariant is that the *rank* of the matroid decreases in each round. While initially as large as m, after 2 rounds, the rank of $(M \setminus S)/T$ is now bounded by $m - m^{1-1/d}$. After repeating this $O(m^{1/d}\log(m))$ times, the algorithm then recovers a basis of the matroid.

Note that in order to get $O(\log(m))$ rounds, d *must* be set to be $\widetilde{\Omega}(\log(m))$, as otherwise the number of elements we contract on is not sufficiently large to terminate within $O(\log(m))$ rounds.

1.2.2 Our New Algorithm for Graphic Matroids

The key intuition behind our improved algorithm is, perhaps surprisingly, to *avoid* contracting on independent sets altogether. Instead, our algorithm relies entirely on deleting redundant elements. In fact, the entire algorithm can be viewed as maintaining the following two invariants (while using only $\text{poly}(m)$ queries per round):

1. After ℓ rounds, there are no cycles of length $\leq 1.01^\ell$ in the current graph.
2. After ℓ rounds, the connected components of the graph are *the same* as the starting graph.

Once these invariants hold, it is clear that the algorithm terminates in $O(\log m)$ rounds (here m is the number of edges). Indeed, after $O(\log m)$ rounds, the minimum cycle length in the graph will exceed m, and the connected components remain unchanged from the start. In other words, we have exactly recovered a spanning forest of the original graph.

Below, we explain how we can capture these invariants, *without* blowing up the query complexity. We first present our algorithm as a randomized procedure, and then discuss the intuition for how it can be derandomized.

1.2.2.1 Cycle Counting Bounds and Cycle Structure

To start, we will require several useful facts about the structure of cycles in graphs. First, we recall the following "cycle counting bound", which has appeared in many works (see, for instance, [19, 34]):

▶ **Theorem 9.** *Let G be a graph with $\leq m$ edges, and let ℓ denote the length of the shortest cycle in G. Then, for any $\alpha \in \mathbb{Z}^+$, G has at most $(2m)^{2\alpha}$ cycles of length $\leq \alpha \cdot \ell$.*

Ultimately, given a graphic matroid with minimum cycle length ℓ, our goal will be to *enumerate* all cycles of length $[\ell, 1.01\ell]$. The above theorem shows that, at the very least, *the number* of such cycles is polynomially bounded. However, this cycle counting bound alone is not enough for us. In order to argue that we can enumerate cycles, our goal is ultimately to understand the *overlap* that cycles can have with one another, as we shall see.

Towards this direction, we establish some additional structural properties of cycles:

▷ Claim 10. Let C, C' be two distinct cycles in a graph. Then the symmetric difference of C, C' (denoted by $C \oplus C'$) contains a cycle.

We omit the formal proof, but one can see this as follows: since C and C' are cycles, every vertex in $C \oplus C'$ has even degree (each vertex is incident to 0, 2, or 4 edges in the union), so $C \oplus C'$ must decompose into one or more cycles. With this fact in hand, our goal is to show that for any fixed short cycle C, no other cycle C' can overlap too much with C. We formalize this in the following lemma:

▶ **Lemma 11.** *Let G be a graph such that there are no cycles of length $\leq \ell$. Let C be a cycle of length $\leq 1.01\ell$ in G and let C' be any other cycle (and denote its length by k'). Then, $|C' \setminus C| \geq \frac{k'}{4}$.*

The lemma follows from a simple case analysis: when $|C'|$ is large, the overlap is inherently large due to $|C|$ being small; and when $|C'|$ is small, we instead use Claim 10 to lower bound $|C \setminus C'|$.

Together, these results show that short cycles are not only few but also well-separated, which is precisely what enables efficiently isolating them via random sampling as we explain next.

1.2.2.2 Random Sampling to Isolate Short Cycles

Now, our key algorithmic tool will be *random sampling* of the edges in our graphic matroid. Recall that given a graphic matroid M, with minimum cycle length ℓ, our goal is to enumerate *all cycles* of length $[\ell, 1.01\ell]$. We will show that there is a sampling rate p such that *for any cycle C* of length $[\ell, 1.01\ell]$, C will be the *unique* surviving cycle with probability $\geq \frac{1}{\text{poly}(m)}$ when we sample the edges from M at rate p. By this, we mean that every edge in the cycle C is chosen during sampling, and *for every other* cycle C', at least one edge is *not* chosen during sampling.

To see why this is the case, let us re-visit Lemma 11 and Theorem 9: together, these imply the following bound, which we call the *cycle overlap counting bound*:

▶ **Lemma 12** (Cycle Overlap Counting Bound)**.** *Let G be a graph with minimum cycle length ℓ and $\leq m$ edges, and let C be any cycle in G of length $\leq 1.01\ell$. Then, for any $\alpha \in \mathbb{Z}^+$, the number of cycles $C' \neq C$ for which $|C' \setminus C| \leq \frac{\alpha\ell}{4}$ is at most $(2m)^{2\alpha}$.*

Next, we choose our value of p carefully. In particular, if we fix a cycle C, our goal is for C to survive the sampling of elements from M at rate p, but *for no other* cycle to survive. To achieve this, we set p such that

$$p^\ell = \frac{1}{m^{100}}.$$

Immediately, we can observe that for a cycle C of length $\in [\ell, 1.01\ell]$, C will survive sampling *if and only if* every edge in C is selected, and thus C survives sampling with probability $p^{|C|}$. Since $|C| \leq 1.01\ell$, we get that

$$\Pr[C \text{ survives sampling }] \geq p^{1.01\ell} \geq \frac{1}{m^{101}}.$$

Next, we have to bound the probability that *any other cycle* C' survives sampling *conditioned* on C surviving sampling. Note that the fact that we condition on C surviving sampling is exactly why we needed counting bounds for the *cycle overlap* sizes, as opposed to simply the cycle sizes themselves.

The remainder of the proof is slightly Karger-esque [23]: let us fix a cycle C' such that $|C' \setminus C| \in \left[\frac{\alpha\ell}{8}, \frac{\alpha\ell}{4}\right]$. Our goal is to understand

$$\Pr[C' \text{ survives sampling} | C \text{ survives sampling}].$$

To do this, we can observe that conditioned on C surviving sampling, C' survives sampling if and only if all the edges in $C' \setminus C$ survive sampling. Further, because these edges are disjoint from the edges in C, their survival probability is *independent* of the edges in C. That is to say

$$\Pr[C' \text{ survives sampling} | C \text{ survives sampling}] = \Pr[C' \setminus C \text{ survives sampling}],$$

which we can now evaluate to be exactly $p^{|C' \setminus C|}$, which is $\leq p^{\frac{\alpha\ell}{8}} = \left(\frac{1}{m^{100}}\right)^{\alpha/8} \leq \frac{1}{m^{12\alpha}}$.

However, now we use the fact that by Lemma 12, there are at most $(2m)^{2\alpha}$ cycles C' for which $|C' \setminus C| \in \left[\frac{\alpha\ell}{8}, \frac{\alpha\ell}{4}\right]$. In particular, we can simply take a union bound over all such cycles and see that

$$\Pr\left[\exists C' \neq C : |C' \setminus C| \in \left[\frac{\alpha\ell}{8}, \frac{\alpha\ell}{4}\right] \wedge C' \text{ survives sampling} | C \text{ survives sampling}\right] \leq \frac{(2m)^{2\alpha}}{m^{12\alpha}}.$$

Now, integrating over α, we see that the probability that *any cycle* $C' \neq C$ survives sampling conditioned on C surviving sampling is bounded above by $\frac{1}{\text{poly}(m)}$. In particular, conditioned on C surviving, this means C is the *unique* surviving cycle with probability $\geq 1/2$. Thus,

$$\begin{aligned}
&\Pr[C \text{ is the unique surviving cycle}] \\
&\quad = \Pr[C \text{ survives}] \cdot \Pr[\forall C' \neq C, C' \text{ does not survive} | C \text{ survives}] \\
&\quad \geq \Pr[C \text{ survives}] \left(1 - \frac{1}{\text{poly}(m)}\right) \geq \frac{1}{2 \cdot m^{101}}.
\end{aligned}$$

Thus, if we repeat this sampling procedure some large polynomial number of times (say, m^{102}), then with extremely high probability, every cycle of length $[\ell, 1.01\ell]$ is the *unique* surviving cycle in some sub-sampled graph. All that remains is to find a way to identify and remove these cycles.

1.2.2.3 Removing Short Cycles

Sampling ensures that each short cycle is uniquely exposed in some sampled subgraph. We now need a procedure to identify these cycles using only independence-oracle queries. Given a sampled edge set E', we can determine whether it contains exactly one cycle as follows:

1. Query $\text{Ind}(E')$ to test whether E' is dependent.
2. Assuming E' is dependent, we next need to ensure it has a unique cycle. If $\text{Ind}(E' \setminus \{x\})$ remains dependent for all x, E' has multiple cycles; otherwise, the edges whose removal restores independence belong to the unique cycle in E'.

Repeating over polynomially many samples in parallel recovers all short cycles. We can then invoke the deletion procedure of [24] to remove these cycles without changing the connectivity structure of the graph, thus ensuring both invariants discussed above.

1.2.2.4 Derandomization via Universal Query Sets

The randomized procedure above already yields an $O(\log m)$-round algorithm; to make it *deterministic*, we transform the sampling process into a fixed collection of universal query sets, leveraging strong probabilistic guarantees for success. Specifically, recall that we showed that for any graphic matroid M with $\leq m$ elements and shortest cycle length ℓ, there is a random sampling procedure which for any cycle C of length $\in [\ell, 1.01\ell]$, ensures that

$$\Pr[C \text{ uniquely survives sampling}] \geq \frac{1}{2 \cdot m^{101}}.$$

If we repeating this sampling procedure m^{200} times (say), and denote these random samples by $B_1, \ldots B_{m^{200}}$, then for a fixed M, and a fixed C in M of length $[\ell, 1.01\ell]$,

$$\Pr[\exists B_i : C \text{ uniquely survives sampling in } B_i] \geq 1 - 2^{-m^{98}}.$$

Importantly, we can now take a union bound (1) over all cycles C in M (which is only $\text{poly}(m)$), and (2) over *all* possible graphic matroids M on m elements, since there are $\leq 2^{m^2}$ possible such matroids, establishing that there exists some choice of $B_1, \ldots B_{m^{200}}$ that simultaneously recovers short cycles in *all possible graphic matroids* with minimum circuit length ℓ.

In other words, we have identified a polynomial-size *universal query set* for all m-element graphic matroids with minimum cycle length ℓ. Since the number of edges and minimum circuit length change over time, we provide the *deterministic* algorithm with a set $B_1^{a,\ell}, \ldots B_{m^{200}}^{a,\ell}$ of queries for every $a \in [m], \ell \in [m]$. This gives a (non-uniform) derandomization of our basis-finding algorithm.

1.2.2.5 Conceptual Framework and Generalization Beyond Graphic Matroids

Conceptually, our algorithm departs from contraction-based methods and develops a *cycle-isolation framework*: by iteratively isolating and removing short dependent structures, we gradually increase the minimum circuit length while preserving rank. This framework can thus be extended to matroids that satisfy the following two properties:

1. A smooth circuit counting bound on the *total number* of cycles of length $\leq \alpha\ell$ (with ℓ being the minimum).
2. Closure of circuits under symmetric difference (Lemma 11).

To generalize beyond graphic matroids, we work with *circuits* instead of cycles, where circuits are the minimal dependent sets, and observe that these conditions hold for binary ($\mathbb{F}_2$-representable) matroids satisfying a smooth circuit-counting bound. This yields a unified framework that extends our results beyond graphic matroids to *cographic*, and possibly to *regular* or even *max-flow min-cut* matroids assuming Conjecture 8 holds.

1.2.3 Matching Lower Bounds for Graphic Matroids

We show that any (randomized) algorithm limited to $\text{poly}(m)$ independence queries per round needs $\Omega(\log m)$ adaptive rounds. For the overview, assume each round permits at most m^c independence queries (for a fixed constant c). We construct a hard instance family and give some intuition as to why it forces $\Omega(\log m)$ rounds; the formal proof is a bit delicate, and appears in the full version.

1.2.3.1 The Lower Bound Instance

Fix the constant c and set $L \coloneqq \Theta(m^{1/2})$ and $\gamma \coloneqq (1000c)^2$. Build a graph G containing, for each $i = 0, 1, \ldots, \lfloor \log_\gamma(\sqrt{L}) \rfloor$, L vertex-disjoint cycles of length $\sqrt{L}\,\gamma^i$ (and no other edges sharing vertices with these cycles). Equivalently,

$$G = \bigcup_{i=0}^{\lfloor \log_\gamma(\sqrt{L}) \rfloor} L \cdot C_{\sqrt{L}\,\gamma^i},$$

where C_ℓ denotes a cycle of length ℓ. This gives $|E(G)| = \Theta(L^2) = \Theta(m)$; we pad with isolated edges if needed so that the total is exactly m.

Crucially, the algorithm never sees the graph; it sees only the answers to independence queries. We therefore hide the structure by assigning uniformly random labels to the edges. An independence query on a label set reveals only whether a cycle is present, not which edges form it.

1.2.3.2 Deriving the Lower Bound

Let G denote a random instance from the above class, and let us consider the first round of queries that any algorithm makes, say, $B_1, \ldots B_{m^c}$. The key observation is that if $\mathrm{Ind}(B_i) = 0$ (that is, there is a cycle in B_i), then with very high probability, it is one of the cycles of *shortest length* (i.e., $\sqrt{L}$) that is in B_i.

Since the labels are uniform, a fixed query B_i is effectively a random set of $|B_i|$ edges. Let $\beta \coloneqq |B_i|/|E(G)|$ denote its sampling rate. A particular cycle of length ℓ is fully contained in B_i with probability β^ℓ. For the shortest cycles (length $\sqrt{L}$), the expected number contained in B_i is $L \cdot \beta^{\sqrt{L}}$. If *none* of these shortest cycles is hit, then necessarily $\beta^{\sqrt{L}} \le 1/\sqrt{L}$; otherwise, a Chernoff bound would imply that one survives w.h.p. But then for longer cycles (length $\gamma\sqrt{L}$),

$$\beta^{\gamma\sqrt{L}} \le \left(\tfrac{1}{\sqrt{L}}\right)^\gamma \le \tfrac{1}{L^{1000c}},$$

so even across m^c queries, the chance to capture such a long cycle is negligible.

Hence, in the first round either (i) no cycle is captured, or (ii) with high probability at least one captured cycle has length $\sqrt{L}$.

In summary, first-round answers are driven almost entirely by the shortest cycles:

$$\mathrm{Ind}(B) = \mathbf{1}\big[\exists \text{ cycle of length } \sqrt{L} \subseteq B\big] \quad \text{w.h.p.}$$

Thus the algorithm's information gain is essentially the identity of edges in the shortest cycles. (For the argument, we will even *reveal* those edges at the end of round one.)

The same reasoning iterates: in round two, the "effective" shortest length becomes $\sqrt{L}\gamma$, and so on. After i rounds the algorithm has essentially learned only the edges belonging to cycles of length at most $\sqrt{L}\,\gamma^i$.

To eliminate all cycles (and hence recover a spanning forest), the algorithm must "reach" cycles of length $\Theta(L)$, i.e., require $\sqrt{L}\,\gamma^i \approx L$. This means we need $i \approx \log_\gamma(\sqrt{L}) = \Omega(\log L) = \Omega(\log m)$ rounds. Hence any algorithm restricted to $\mathrm{poly}(m)$ queries per round needs $\Omega(\log m)$ rounds on this instance family.

For the formal proof, one must also account for information leaked by *independent* queries (which certify that many edge sets contain no cycles, including longer ones). We bound this information carefully, showing it does not accelerate progress beyond the shortest-cycle layer in each round; chaining these bounds yields the claimed $\Omega(\log m)$ lower bound.

1.3 Organization

In Section 2, we present the formal analysis of our cycle overlap counting bound, our sampling procedure for enumerating cycles, and our procedure for removing cycles, culminating in a poly(m) query, $O(\log(m))$ round randomized algorithm for finding spanning forests. Due to space constraints, we defer the following to the full version of the paper: derandomization of the cycle enumeration algorithm, formal analysis for our lower bound, and generalizations of the basis-finding algorithm to wider classes of matroids.

2 Randomized Algorithm for Graphic Matroids

We start by covering some auxiliary lemmas that will be helpful in the analysis of our algorithm.

2.1 Cycle Counting Bounds

To start, recall the statement from the work of [34][3]:

▷ Claim 13 ([34]). Let G be a graph on n vertices, and suppose that G has no cycles of length $\leq \ell$. Then, G has at most n^4 cycles of length $\leq 2\ell$.

In fact, we will need a stronger version of this statement (which has appeared in the literature, see [21] for instance).

▷ Claim 14. Let G be a graph with $\leq m$ edges, and minimum cycle length ℓ. Then, for any $\alpha \in \mathbb{Z}^+$, the number of cycles of length $\leq \alpha \cdot \ell$ is at most $(2m)^{2\alpha}$.

Next, we use this counting bound to bound the *overlap* between cycles.

2.2 Cycle Overlap Properties

To better understand properties of cycle overlaps, we need the following claims and lemmas. To start, we have the following:

▷ Claim 15 (Symmetric Difference of Cycles). Let C, C' be two distinct cycles in a graph. Then the symmetric difference of C, C' (denoted by $C \oplus C'$) contains a cycle.

Proof. Consider the set of edges given by $C \oplus C'$, and let us consider the degree of any vertex v. We claim that v will always have degree $0, 2$ or 4 in $C \oplus C'$, and thus there must be a cycle in the graph $C \oplus C'$ (as it is a non-empty graph with all degrees being even).

To see why, if v has degree 0 in both C, C', then v also has degree 0 in $C \oplus C'$. If v has degree 2 in C, but degree 0 in C' (or vice versa), then v will have degree 2 in $C \oplus C'$, as no edges incident to v are removed. Finally, if v has degree 2 in both C and C', then we must look at how many edges are in common. If v shares 0 edges between C, C' then its degree in the symmetric difference is 4, if v shares 1 edge between C, C', its degree is 2, and if it shares both edges between C, C', then its degree is 0 in the symmetric difference. This concludes the proof. ◁

[3] This work only shows the bound for a constant multiple of the minimum cycle length. A simple adaptation holds for arbitrary values times the minimum cycle length.

Next, we establish the following lemma, which lower bounds the overlap size between any cycles:

▶ **Lemma 16.** *Let E be a set of edges such that there is no cycle of length $\leq \ell$. Let C be a cycle of length $k \leq 1.01\ell$ in E and let C' be a cycle of length k' in E. Then, $|C' \setminus C| \geq \frac{k'}{4}$.*

Proof. Note that if k' is sufficiently large relative to ℓ, the above lemma is trivial. For instance, if $k' \geq 1.5\ell$, then $|C' \setminus C| \geq k' - 1.01\ell \geq k' - (3/4)k' = \frac{k'}{4}$ (using $1.01\ell \leq (3/4)k'$ for $k' \geq 1.5\ell$). The interesting case is when $k' \leq 1.5\ell$. Here we use Claim 15: since $C \oplus C'$ must contain a cycle and ℓ is the shortest cycle length in G, we have $|C \oplus C'| \geq \ell$. We can rewrite

$$|C \oplus C'| = |C| + |C'| - 2|C \cap C'| = |C| + |C'| - 2(|C'| - |C' \setminus C|).$$

Plugging in $|C \oplus C'| \geq \ell$ and using $|C| \leq 1.01\ell$ and $|C'| \geq \ell$, we obtain

$$|C| + |C'| - 2(|C'| - |C' \setminus C|) \geq \ell,$$

which simplifies to

$$2|C' \setminus C| \geq \ell + |C'| - |C| \geq 0.99\ell.$$

Thus $|C' \setminus C| \geq 0.495\ell$, which in this case is at least $k'/4$. ◀

With these lemmas, we are now ready to start presenting our algorithms. We begin by studying algorithms for recovering unique cycles in graphs using only an independence oracle.

2.3 Detecting a Single Cycle

As mentioned in the introduction, our algorithm proceeds by removing cycles in an iterative manner, gradually eliminating cycles of increasing lengths. Suppose in some iteration of the algorithm we have the promise that there are no cycles of length $\leq \ell$ in the graph. Then our goal for the iteration is to (1) eliminate all cycles of length $\leq 1.01\ell$ and (2) to do this *without* altering the connectivity of the graph. We will accomplish this task by repeatedly sampling the edges in the graph to create the following *good event*: that there is a unique cycle that survives among the sampled edges and moreover, its length is $\leq 1.01\ell$.

Conditioned on this good even, we must then identify exactly the edges that are participating in this unique surviving cycle, and then repeat this process many times until we have enumerated all cycles of length $\leq 1.01\ell$ in the graph. Note that because of Claim 14, as a sanity check we can see that the number of potential cycles we must recover is bounded by some polynomial in m (although there is no guarantee that these cycles are easy to find).

As a first step towards identifying these cycles, we present a simple algorithm for detecting whether or not there is a single cycle in a graph (and if there is a single cycle, the algorithm returns exactly the edges in the cycle):

Algorithm 1 DetectSingleCycle(E').

```
Initialize the set of critical edges S = ∅.
if Query Ind(E') = 1 then
    return ⊥, No cycles.
end
for e ∈ E' do
    if Query Ind(E' \ {e}) = 1 then
        S ← S ∪ {e}.
    end
end
if S = ∅ then
    return ⊥, ≥ 2 cycles.
end
return S.
```

We summarize the performance of the algorithm in several claims:

▷ Claim 17. Let E' be a set of edges which has no cycles, then Algorithm 1 correctly returns that there are no cycles.

Proof. The independence query to E' will be 1 if and only if there are no cycles. ◁

▷ Claim 18. Suppose E' has ≥ 2 cycles, then Algorithm 1 returns that there are ≥ 2 cycles.

Proof. To prove this, we use an auxiliary claim: namely that if a graph G has at least 2 distinct cycles, then there is no single edge whose removal kills both cycles. To see why, let us denote two cycles by C_1, C_2. If an edge e is in both C_1 and C_2, then by Claim 15, $C_1 \oplus C_2$ contains a cycle, *and* $e \notin C_1 \oplus C_2$. Thus, there must be some cycle in the graph which does not include the edge e, so e's removal does not disconnect the graph. Otherwise, if e is not in both of C_1, C_2, one of C_1, C_2 will also remain intact after e's removal. This yields the claim. ◁

▷ Claim 19. Suppose E' has exactly 1 cycle, then Algorithm 1 returns exactly the constituent edges of this cycle.

Proof. Let the cycle be denoted by C. Observe that removing any edge $e \in C$ will disconnect the cycle, and hence the independence queries will now return 1. Likewise, deleting any edge $e \notin C$ will not disconnect the cycle, and the cycle will still be present, so the independence queries will return 0. ◁

Finally, we also observe that the above algorithm can be implemented in 1 round of adaptivity:

▷ Claim 20. Algorithm 1 can be implemented in 1 round of adaptivity and makes $|E'| + 1$ queries to Ind.

Proof. Notice that the algorithm queries $\text{Ind}(E')$, and $\text{Ind}(E' \setminus \{e\}) : \forall e \in E'$. All these queries are made without reference to the results from previous queries. ◁

Thus, we get the following lemma to summarize Algorithm 1:

▶ **Lemma 21.** *For a set of edges E', Algorithm 1 makes $|E'|+1$ queries to* Ind *in only a single round, and returns $\perp$ if E' has no cycles or ≥ 2 cycles, and otherwise returns a set $S \subseteq E'$ which is exactly the edges involved in the single cycle in E'.*

Proof. This follows from Claim 17, Claim 18, Claim 19, and Claim 20. ◀

Thus, we have a simple algorithm for identifying when there is a single cycle in a graph. In the coming sections, we will show how to use this procedure to identify and remove all of the short cycles in a given graph.

2.4 Sampling

Motivated by the previous subsection, our goal will now be to sub-sample the graph at a specific rate such that only *one* cycle will survive the sampling process with high probability. When only one cycle survives, we can then identify this cycle exactly by Algorithm 1. The sampling algorithm is provided below, which takes in a number of edges m, the edge set E, as well as a parameter ℓ corresponding to the minimum cycle length in E.

Algorithm 2 RecoverCycleSuperset(E, m, ℓ).

```
Let p be such that p^ℓ = 1/m^100.
for i ∈ [m^102], in parallel do
    Let E^(i) be the result of independently keeping each e ∈ E with probability p.
end
return {E^(i) : i ∈ [m^102]}
```

To understand the sampling procedure, we focus on a single cycle C in E of length k, for $k \in (\ell, 1.01\ell]$, and start by showing the following:

▷ Claim 22. Let E be a set of m edges such that there is no cycle of length $\leq \ell$ and let C be a cycle in E of length k, for $k \in (\ell, 1.01\ell]$. Let E' be the result of independently keeping each edge in E with probability p, where $p^\ell = \frac{1}{m^{100}}$. Then, C is the unique cycle in E' with probability $\frac{1}{2 \cdot m^{101}}$.

Proof. First, we calculate the probability that C survives the sampling procedure. This is straightforward, as C has $\leq 1.01\ell$ edges. So, after sampling at rate p, the probability all $\leq 1.01\ell$ edges survive is:

$$\Pr[C \text{ survives sampling}] \geq p^{1.01\ell} = \left(p^\ell\right)^{1.01} = \left(\frac{1}{m^{100}}\right)^{1.01} = \frac{1}{m^{101}}.$$

Thus, it remains only to bound the probability that *some other cycle* C' also survives the sampling at rate p, conditioned on C surviving the sampling.

Indeed, to bound this, let us suppose that C' is of length k'. By Lemma 16, we know that $|C' \setminus C| \geq \frac{k'}{4}$. Thus,

$$\Pr[C' \text{ survives sampling} | C \text{ survives sampling}] \leq p^{k'/4},$$

as there will be at least $k'/4$ edges in C' which are not in C (and thus these edges surviving the sampling process is independent of C surviving).

Next, it remains to take a union bound over all possible cycles C'. For this, let α be the power of 2 such that $k' \in [\alpha\ell, 2\alpha\ell]$, and then we use Claim 14. Specifically, for cycles C' of length $[\alpha\ell, 2\alpha\ell]$, we know that there are at most $(2m)^{2\alpha}$ such cycles. So, for a fixed α, this means we get the following bound:

$$\begin{aligned}
&\Pr[\exists C' \neq C \text{ of length } [\alpha\ell, 2\alpha\ell] \text{ that survives sampling} | C \text{ survives sampling}] \\
&\quad \leq \sum_{C' \neq C: \text{ cycle of length } [\alpha\ell, 2\alpha\ell]} \Pr[C' \text{ survives sampling} | C \text{ survives sampling}] \\
&\quad \leq \sum_{C': \text{ cycle of length } [\alpha\ell, 2\alpha\ell]} p^{\alpha\ell/4} \leq p^{\alpha\ell/4} \cdot (2m)^{2\alpha} = \left(\frac{1}{m^{100}}\right)^{\frac{\alpha}{4}} \cdot (2m)^{2\alpha} \leq \left(\frac{2}{m^{23}}\right)^{\alpha}.
\end{aligned}$$

To conclude, we can then take a union bound over $\alpha \in \{1, 2, 4, 8, \ldots m\}$. Thus, we see that:

$$\begin{aligned}
&\Pr[\exists C' \neq C \text{ that survives sampling} | C \text{ survives sampling}] \\
&\quad \leq \sum_{\alpha \in \{1,2,4,8,\ldots m\}} \Pr[\exists C' \neq C \text{ of length } [\alpha\ell, 2\alpha\ell] \text{ that survives sampling} | C \text{ survives sampling}] \\
&\quad \leq \sum_{\alpha \in \{1,2,4,8,\ldots m\}} \left(\frac{2}{m^{23}}\right)^{\alpha} \leq \sum_{\alpha=1}^{\infty} \left(\frac{2}{m^{23}}\right)^{\alpha} \\
&\quad \leq \frac{4}{m^{23}}
\end{aligned}$$

where the final inequality follows because the expression is a geometric series with ratio $< 1/2$.

Finally, recall that our goal was to show that C is the unique cycle which survives sampling with non-negligible probability. For this, observe that

$$\Pr[C \text{ survives sampling}] = \Pr[C \text{ uniquely survives sampling}] + \Pr[\exists C' \neq C : C \wedge C' \text{ survive sampling}].$$

Now, this second term we can bound by our above work. I.e.,

$$\begin{aligned}
&\Pr[\exists C' \neq C : C \wedge C' \text{ survive sampling}] \\
&\quad = \Pr[\exists C' \neq C \text{ that survives sampling} | C \text{ survives sampling}] \cdot \Pr[C \text{ survives sampling}] \\
&\quad \leq \frac{4}{m^{23}} \cdot \Pr[C \text{ survives sampling}].
\end{aligned}$$

Thus, we see that

$$\Pr[C \text{ uniquely survives sampling}] \geq \Pr[C \text{ survives sampling}] \cdot \left(1 - \frac{4}{m^{23}}\right) \geq \frac{1}{2 \cdot m^{101}},$$

as we desire. ◁

Using the above claim, we now show that (with high probability) every cycle C of length $(\ell, 1.01\ell]$ is the unique surviving cycle for some $E^{(i)}$ produced by Algorithm 2.

▶ **Lemma 23.** *Let E be a set of m edges such that there is no cycle of length $\leq \ell$. Then, with probability $1 - 2^{-\Omega(m)}$, for every cycle C in E of length $(\ell, 1.01\ell]$, there is an index $i \in [m^{102}]$ such that C is the unique cycle in $E^{(i)}$.*

Proof. Fix any cycle C in E of length $(\ell, 1.01\ell]$. By Claim 22, we know that over the randomness of the sampling procedure, C will be the unique cycle present in $E^{(i)}$ with probability $\frac{1}{2m^{101}}$. Thus, by repeating this procedure m^{102} times, we know that there is at least one index i for which C is the unique cycle with probability $1 - 2^{-\Omega(m)}$.

Now, because there are at most $(2m)^5$ cycles of length $(\ell, 1.01\ell]$ in a graph with no cycles of length $\leq \ell$, we can take a union bound over all these cycles. This means that with probability $1 - 2^{-\Omega(m)}$, for every cycle C in E of length $(\ell, 1.01\ell]$, there is an index $i \in [m^{102}]$ such that C is the unique cycle in $E^{(i)}$. ◀

2.5 Final Cycle Recovery Algorithm

Now, it remains only to piece together these algorithms. We present this below as an algorithm:

Algorithm 3 TotalCycleRecovery(E, m, ℓ).

```
Let p be such that p^ℓ = 1/m^100.
Cycles = {}.
for i ∈ [m^102] in parallel do
    Let E^(i) be the result of independently keeping each e ∈ E with probability p.
    Let S = DetectSingleCycle(E^(i)).
    if S ≠ ⊥ then
        Cycles ← Cycles ∪ {S}.
    end
end
return Cycles.
```

We can combine our results from the previous subsections to understand what Algorithm 3 achieves:

▶ **Lemma 24.** *Let E be a set of m edges such that there is no cycle of length $\leq \ell$. Then, with probability $1 - 2^{-\Omega(m)}$, the output of Algorithm 3 is a set of cycles which contains every cycle C in E of length $(\ell, 1.01\ell]$.*

Proof. First, recall that by Lemma 21, the algorithm DetectSingleCycle does not return $\perp$ if and only if there is a single cycle in the input graph. By this same lemma, when the output is not $\perp$, the algorithm recovers exactly the constituent edges of the cycle. Hence the output of Algorithm 3 is necessarily a set of cycles (as the cycles in the subsampled graphs will also be cycles in the original graph). It remains only to show that *every* cycle of length $(\ell, 1.01\ell]$ is included in the output. This follows exactly from Lemma 23. Indeed, every cycle of length $(\ell, 1.01\ell]$ will be the unique cycle surviving the sampling procedure with probability $1 - 2^{-\Omega(m)}$, and thus will be included in the output as well. This yields the lemma at hand. ◀

Likewise, we can observe that Algorithm 3 is implementable in parallel without any adaptivity:

▷ Claim 25. On input E, with m edges, and parameter ℓ, Algorithm 3 is implementable with $\text{poly}(m)$ queries to the independence oracle in a single round.

Proof. By Lemma 21, each invocation of DetectSingleCycle requires $O(m)$ queries to the independence oracle. Each of the sub-sampled graphs is checked in parallel, and thus the total number of queries is $O(m \cdot m^{102}) = \text{poly}(m)$, without any adaptivity. ◁

2.6 Cycle Removal

After recovering all of the cycles of length $(\ell, 1.01\ell]$, we must find a way to delete these cycles without altering the connectivity of the graph. To do this, we take advantage of a basic operation from the work of [24]. Although this statement is included in [24], there was no proof provided there, so we re-prove the result here.

▶ **Lemma 26.** *Let E be a set of edges with some fixed ordering of the edges $e_1, \dots e_m$, and let* Cycles *be an arbitrary subset of the cycles in E. For each cycle $C \in$ Cycles, let $C = (e_{i_{C,1}}, \dots e_{i_{C,|C|}})$ denote the ordered set of the edges that are in the cycle C. Let E' be the result of simultaneously deleting from E the edge with the* largest *index from every cycle in* Cycles*. Then,*

1. *Every cycle in* Cycles *has at least one edge removed.*
2. *The connected components of E' are the same as the connected components of E.*

Proof. The first item is essentially trivial. Every cycle in Cycles has some constituent edge deleted.

The second item is less immediate. To see why the connectivity of E and E' is the same, let us consider adding the edges in $E \setminus E'$ back in to E', in increasing order (i.e., starting with the edge which has the smallest label). Let us denote these edges by $e_1^*, \dots e_g^*$. We claim that the connectivity of $E' \cup \{e_1^*, \dots e_i^*\}$ is the same as the connectivity of $E' \cup \{e_1^*, \dots e_{i+1}^*\}$. For the base case, we show that the connected components of E' are the same as the connected components of $E' \cup \{e_1^*\}$. This is because e_1^* was the smallest labeled edge which was removed. Because it was removed though, this means that there is some cycle C_1 for which e_1^* has the largest label, and hence all other edges in $C_1 \setminus \{e_1^*\}$ are still present in E'. But, adding e_1^* then does not alter the connected components, as the vertices connected by e_1^* are already connected by the edges in $C_1 \setminus \{e_1^*\}$.

Now, we show the general case. I.e., that the connectivity of $E' \cup \{e_1^*, \dots e_i^*\}$ is the same as the connectivity of $E' \cup \{e_1^*, \dots e_{i+1}^*\}$. For this, observe that because the e_j^*'s are ordered in terms of their labels, $E' \cup \{e_1^*, \dots e_i^*\}$ contains every edge whose label is smaller than e_{i+1}^*. In particular, because e_{i+1}^* was removed, this means that e_{i+1}^* was the largest labeled edge in some cycle C_{i+1}. This means that every edge in $C_{i+1} \setminus \{e_{i+1}^*\}$ is still in $E' \cup \{e_1^*, \dots e_i^*\}$. But, adding e_{i+1}^* then does not alter the connected components, as the vertices connected by e_{i+1}^* are already connected by the edges in $C_{i+1} \setminus \{e_{i+1}^*\}$.

To conclude, we simply observe that by induction, this means that the connected components of E' are the same as the connected components of $E' \cup \{e_1^*, \dots\} = E$. ◀

2.7 Spanning Forest Computation

With all of our building blocks now established, we present our final algorithm for finding a spanning forest below. Note that because we make progress on the order of $\ell \to 1.01\ell$ in each round, we need to initialize our algorithm on a graph with no cycles of length ≤ 100. To do this, we enumerate all cycles of length ≤ 100 explicitly in the first round:

Algorithm 4 FindSpanningForest(E, m).

```
1  Let e_1, ... e_m be a fixed ordering of the edges of E.
2  ℓ = 100.
3  Cycles = ∅.
4  for E' ⊆ E : |E'| ≤ ℓ in parallel do
5  |  S = DetectSingleCycle(E').
6  |  if S ≠ ⊥ then
7  |  |  Cycles ← Cycles ∪ {S}
8  |  end
9  end
10 for C ∈ Cycles in parallel do
11 |  Let e* be the edge in C with the largest index with respect to the ordering.
12 |  E ← E \ {e*}.
13 end
14 while |E| ≥ ℓ do
15 |  Cycles = TotalCycleRecovery(E, |E|, ℓ).
16 |  for C ∈ Cycles in parallel do
17 |  |  Let e* be the edge in C with the largest index with respect to the ordering.
18 |  |  E ← E \ {e*}.
19 |  end
20 |  ℓ ← ⌊1.01 · ℓ⌋.
21 end
22 return E.
```

We now present a sequence of claims analyzing the above algorithm.

▷ Claim 27. After invoking Algorithm 4 on a set of edges E with m edges, after Line 13, the connected components of E have not changed and there are no cycles left of length ≤ 100.

Proof. Observe that the set of cycles in line 10 is indeed a set of cycles as per Lemma 21. In particular, every cycle of length ≤ 100 will be recovered, as the algorithm queries all subsets of ≤ 100 edges, and so there will be some query for each cycle. The cycle deletion procedure is exactly that of Lemma 26, and thus all cycles of length ≤ 100 are removed, without altering the connected components of E. ◁

▷ Claim 28. Let E be a set of m edges with no cycles of length $\leq \ell$. For each iteration of the while loop (Line 14) in Algorithm 4, the algorithm removes all cycles of length $\leq 1.01\ell$ without altering the connected components of E, with probability $1 - 2^{-\Omega(m)}$.

Proof. This set of cycles recovered includes all cycles of length $\leq 1.01\ell$ with probability $1 - 2^{-\Omega(m)}$ by Lemma 24. These cycles are then deleted without altering the connected components as per Lemma 26. ◁

Next, we also bound the number of iterations of the while loop:

▷ Claim 29. After invoking Algorithm 4 on a set of edges E with m edges, the while loop in line 14 runs for at most $O(\log(m))$ iterations, with probability $1 - 2^{-\Omega(\sqrt{m})}$, and returns a set of edges with no cycles of length $\leq m + 1$.

Proof. First, recall that any connected component on k vertices has $\leq k^2$ edges. Thus, for a set E of m edges, any spanning forest must have $\geq \sqrt{m}$ edges. Thus, as long as the set of connected components defined by E has not changed under deleting edges, the number of remaining edges must be $\Omega(\sqrt{m})$.

Next, observe that the set of edges E in the input to line 14 has no cycles of length ≤ 100 as per Claim 27 (but still has $\Omega(\sqrt{m})$ edges). Now, by Claim 28, with probability $1 - 2^{-\Omega(\sqrt{m})}$, all cycles of length $\leq 100 \cdot 1.01$ are removed.

In general, observe that for an integer $k \geq 100$, it must be the case that

$$\lfloor k \cdot 1.01 \rfloor \geq k \cdot 1.005.$$

This is because

$$\lfloor k \cdot 1.01 \rfloor = k + \lfloor k \cdot 0.01 \rfloor,$$

and for $k \geq 100$, we can write $k = \alpha \cdot 100 + k \mod 100$ for $\alpha \geq 1$. This means

$$\lfloor k \cdot 0.01 \rfloor = \alpha,$$

while

$$0.005k = \alpha/2 + (k \mod 100) \cdot 0.005 < \alpha/2 + 1/2.$$

Thus,

$$\lfloor k \cdot 1.01 \rfloor = k + \lfloor k \cdot 0.01 \rfloor = k + \alpha \geq k + \alpha/2 + 1/2 > k + 0.005k = 1.005k.$$

Thus, if we let ℓ denote the minimum cycle length for an iteration of the while loop in line 14, after this iteration, the new minimum cycle length will be $\geq 1.005\ell$. After $O(\log(m))$ iterations, the minimum cycle length will be $\geq m + 1$, and thus there will be no cycles remaining. In particular, this also means that the number of edges remaining (denoted by $|E|$) is less than the minimum cycle length, and so the stopping condition of the while loop is met.

Note that the probability bound holds because in each iteration there are $\Omega(\sqrt{m})$ edges remaining (where m denotes the initial number of edges), and each iteration increases the cycle length by a factor of ≥ 1.005 with probability $1 - 2^{-\Omega(\sqrt{m})}$. Thus, we can afford to take a union bound over each iteration failing. This yields the claim. ◁

We conclude with our primary theorem:

▶ **Theorem 30.** *Algorithm 4 outputs a spanning forest of a graphic matroid on m edges with probability $1 - 2^{-\Omega(\sqrt{m})}$ using $\text{poly}(m)$ queries to an independence oracle per round and $O(\log(m))$ rounds of adaptivity.*

Proof. To see the bound on the number of queries, and rounds of adaptivity, recall that each invocation of DetectSingleCycle requires $O(m)$ queries to the oracle by Lemma 21. In each round, there are $\text{poly}(m)$ invocations made to Lemma 21 (in parallel), and thus the total number of queries per round is $\text{poly}(m)$. Likewise, by Claim 29, the number of rounds required is $O(\log(m))$, and returns a spanning forest of E with probability $1 - 2^{-\Omega(\sqrt{m})}$. This yields the theorem. ◀

3 Conclusions

We have resolved a longstanding open question regarding the parallel complexity of computing bases in graphic matroids. Specifically, we presented a deterministic parallel algorithm that, given access to an independence oracle, computes a spanning forest in $O(\log m)$ adaptive rounds using only $\text{poly}(m)$ non-adaptive queries per round. This matches our lower bound, which shows that any (even randomized) algorithm using $\text{poly}(m)$ queries per round must require $\Omega(\log m)$ adaptive rounds in expectation. Together, these results provide a tight and complete characterization of the round-query tradeoff for graphic matroids.

Our approach introduces new structural insights into the behavior of cycles in graphs under random sampling, as well as techniques for efficiently enumerating and eliminating short cycles while preserving graph connectivity. These tools may find applications in development of parallel algorithms for other graph problems.

References

1 Eric Balkanski, Aviad Rubinstein, and Yaron Singer. An exponential speedup in parallel running time for submodular maximization without loss in approximation. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 283–302. SIAM, 2019. doi:10.1137/1.9781611975482.19.

2 Eric Balkanski, Aviad Rubinstein, and Yaron Singer. An optimal approximation for submodular maximization under a matroid constraint in the adaptive complexity model. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 66–77. ACM, 2019. doi:10.1145/3313276.3316304.

3 Eric Balkanski and Yaron Singer. The adaptive complexity of maximizing a submodular function. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1138–1151. ACM, 2018. doi:10.1145/3188745.3188752.

4 Eric Balkanski and Yaron Singer. Approximation guarantees for adaptive sampling. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 393–402. PMLR, 2018. URL: http://proceedings.mlr.press/v80/balkanski18a.html.

5 Eric Balkanski and Yaron Singer. A lower bound for parallel submodular minimization. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 130–139. ACM, 2020. doi:10.1145/3357713.3384287.

6 András A. Benczúr and David R. Karger. Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 47–55. ACM, 1996. doi:10.1145/237814.237827.

7 Joakim Blikstad. Sublinear-round parallel matroid intersection. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 25:1–25:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.25.

8 Joakim Blikstad and Ta-Wei Tu. Efficient matroid intersection via a batch-update auction algorithm. In Ioana Oriana Bercea and Rasmus Pagh, editors, *2025 Symposium on Simplicity in Algorithms, SOSA 2025, New Orleans, LA, USA, January 13-15, 2025*, pages 226–237. SIAM, 2025. doi:10.1137/1.9781611978315.18.

9 Adam Breuer, Eric Balkanski, and Yaron Singer. The FAST algorithm for submodular maximization. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 1134–1143. PMLR, 2020. URL: http://proceedings.mlr.press/v119/breuer20a.html.

10 Deeparnab Chakrabarty, Yu Chen, and Sanjeev Khanna. A polynomial lower bound on the number of rounds for parallel submodular function minimization. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 37–48. IEEE, 2021. doi:10.1109/FOCS52979.2021.00013.

11 Deeparnab Chakrabarty, Andrei Graur, Haotian Jiang, and Aaron Sidford. Improved lower bounds for submodular function minimization. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 245–254. IEEE, 2022. doi:10.1109/FOCS54457.2022.00030.

12 Chandra Chekuri and Kent Quanrud. Parallelizing greedy for submodular set function maximization in matroids and beyond. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 78–89, 2019. doi:10.1145/3313276.3316406.

13 Chandra Chekuri and Kent Quanrud. Submodular function maximization in parallel via the multilinear relaxation. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 303–322. SIAM, 2019. doi:10.1137/1.9781611975482.20.

14 Lin Chen, Moran Feldman, and Amin Karbasi. Unconstrained submodular maximization with constant adaptive complexity. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 102–113. ACM, 2019. doi:10.1145/3313276.3316327.

15 Alina Ene and Huy L. Nguyen. Submodular maximization with nearly-optimal approximation and adaptivity in nearly-linear time. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 274–282. SIAM, 2019. doi:10.1137/1.9781611975482.18.

16 Alina Ene, Huy L. Nguyen, and Adrian Vladu. Submodular maximization with matroid and packing constraints in parallel. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 90–101. ACM, 2019. doi:10.1145/3313276.3316389.

17 Matthew Fahrbach, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Non-monotone submodular maximization with nearly optimal adaptivity and query complexity. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 1833–1842. PMLR, 2019. URL: http://proceedings.mlr.press/v97/fahrbach19a.html.

18 Matthew Fahrbach, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Submodular maximization with nearly optimal approximation, adaptivity and query complexity. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 255–273. SIAM, 2019. doi:10.1137/1.9781611975482.17.

19 Stephen A. Fenner, Rohit Gurjar, and Thomas Thierauf. Bipartite perfect matching is in quasi-nc. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 754–763. ACM, 2016. doi:10.1145/2897518.2897564.

20 Mohsen Ghaffari and Christoph Grunau. Near-optimal deterministic network decomposition and ruling set, and improved MIS. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 2148–2179. IEEE, 2024. doi:10.1109/FOCS61266.2024.00007.

21 Rohit Gurjar and Nisheeth K. Vishnoi. On the number of circuits in regular matroids (with connections to lattices and codes). In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 861–880. SIAM, 2019. doi:10.1137/1.9781611975482.53.

22 Rohit Gurjar and Nisheeth K. Vishnoi. On the number of circuits in regular matroids (with connections to lattices and codes). *SIAM J. Discret. Math.*, 35(3):1688–1705, 2021. doi:10.1137/20M1338642.

23 David R. Karger. Global min-cuts in **RNC**, and other ramifications of a simple min-cut algorithm. In Vijaya Ramachandran, editor, *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA*, pages 21–30. ACM/SIAM, 1993. URL: http://dl.acm.org/citation.cfm?id=313559.313605.

24 Richard M. Karp, Eli Upfal, and Avi Wigderson. The complexity of parallel computation on matroids. In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 541–550. IEEE Computer Society, 1985. doi:10.1109/SFCS.1985.57.

25 Richard M. Karp, Eli Upfal, and Avi Wigderson. Constructing a perfect matching is in random NC. *Comb.*, 6(1):35–48, 1986. doi:10.1007/BF02579407.

26 Richard M. Karp, Eli Upfal, and Avi Wigderson. The complexity of parallel search. *J. Comput. Syst. Sci.*, 36(2):225–253, 1988. doi:10.1016/0022-0000(88)90027-X.

27 Ehsan Kazemi, Marko Mitrovic, Morteza Zadimoghaddam, Silvio Lattanzi, and Amin Karbasi. Submodular streaming in all its glory: Tight approximation, minimum memory and low adaptive complexity. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3311–3320. PMLR, 2019. URL: http://proceedings.mlr.press/v97/kazemi19a.html.

28 Sanjeev Khanna, Aaron Putterman, and Junkai Song. On the parallel complexity of finding a matroid basis. *arXiv preprint arXiv:2507.08194*, 2025. doi:10.48550/arXiv.2507.08194.

29 Wenzheng Li, Paul Liu, and Jan Vondrák. A polynomial lower bound on adaptive complexity of submodular maximization. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 140–152. ACM, 2020. doi:10.1145/3357713.3384311.

30 László Lovász. On determinants, matchings, and random algorithms. In Lothar Budach, editor, *Fundamentals of Computation Theory, FCT 1979, Proceedings of the Conference on Algebraic, Arithmetic, and Categorial Methods in Computation Theory, Berlin/Wendisch-Rietz, Germany, September 17-21, 1979*, pages 565–574. Akademie-Verlag, Berlin, 1979.

31 Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15(4):1036–1053, 1986. doi:10.1137/0215074.

32 Paul D Seymour. The matroids with the max-flow min-cut property. *Journal of Combinatorial Theory, Series B*, 23(2-3):189–222, 1977. doi:10.1016/0095-8956(77)90031-4.

33 Paul D Seymour. Decomposition of regular matroids. *Journal of combinatorial theory, Series B*, 28(3):305–359, 1980. doi:10.1016/0095-8956(80)90075-1.

34 Ashok Subramanian. A polynomial bound on the number of light cycles in an undirected graph. *Inf. Process. Lett.*, 53(4):173–176, 1995. doi:10.1016/0020-0190(94)00202-A.

35 Ola Svensson and Jakub Tarnawski. The matching problem in general graphs is in quasi-nc. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 696–707. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.70.