Algorithms and Lower Bounds for Cycles and Walks: Small Space and Sparse Graphs

Andrea Lincoln
MIT, Cambridge, MA, USA
andreali@mit.edu

Nikhil Vyas
MIT, Cambridge, MA, USA
nikhilv@mit.edu

Abstract

We consider space-efficient algorithms and conditional time lower bounds for finding cycles and walks in graphs. We give a reduction that connects the running time of undirected $2k$-cycle to finding directed odd cycles, s-t connectivity in directed graphs, and Max-3-SAT. For example, we show that if $2k$-cycle on $O(n)$-edge graphs can be solved in $O(n^{1.5-\epsilon})$ time for some $\epsilon > 0$ then, a $2^{n(1-\epsilon')}$ time algorithm exists for Max-3-SAT for some $\epsilon' > 0$. Additionally, we give a tight combinatorial lower bound for $2k$-cycle detection, specifically when k is odd, of $m^{2k/(k+1)+o(1)}$ given the Combinatorial k-Clique Hypothesis.

On the algorithms side, we present a randomized algorithm for directed s-t connectivity using $O(\lg(n)^2)$ space and $O(n^{\lg(n)/2+o(\lg(n))})$ expected time, giving a time improvement over Savitch's famous algorithm, which takes at least $n^{\lg(n)-o(\lg(n))}$ time. Under the conjecture that every $O(\lg(n)^2)$-space algorithm for directed s-t connectivity requires $n^{\Omega(\lg(n))}$ time, we show that undirected $2k$-cycle in $O(\lg(n))$ space requires $n^{\Omega(\lg(k))}$ time.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Graph algorithms analysis

Keywords and phrases k-cycle, Space, Savitch, Sparse Graphs, Max-3-SAT

Digital Object Identifier 10.4230/LIPIcs.ITCS.2020.11

Funding *Andrea Lincoln*: Supported by NSF Grants CCF-1417238, CCF-1528078 and CCF-1514339, and BSF Grant BSF:2012338.
Nikhil Vyas: Supported by an Akamai Presidential Fellowship, NSF Grant CCF-1552651 and CCF-1909429.

1 Introduction

The k-cycle problem, determining whether a graph on n nodes and m edges contains a cycle of length k, is a fundamental subgraph detection problem. Past work has explored efficient algorithms for detecting/counting, odd/even, directed/undirected cycles in both sparse and dense graphs (e.g. [17, 5, 2, 8, 18]). In this paper, we explore k-cycle algorithms using small space, e.g., poly$(\log n)$.

When $k \geq \Omega(n)$, for example, $k = n/2$, detecting a k-cycle is well-known to be NP-hard [9]. When k is constant and algorithms are given $O(n^2)$ space, odd cycles can be detected in matrix multiplication time [3, 5]. When k is constant and algorithms are given $O(n)$ space, even cycles can be detected in $O(n^2)$ time [17]. Given an undirected sparse graph, detecting a $2k$-cycle can be done faster, specifically $O(m^{2k/(k+1)})$ time [5]. However, this work leaves open the question of what happens in the very small space regime.

When algorithms are allowed $O(\lg(n) \cdot \lg(k))$ space, the Savitch algorithm can be used to solve k-cycle detection in time $n^{\lg(k)+o(\lg(k))}$ time. In particular, the Savitch algorithm can be used to detect if two nodes s and t are connected by a path of length at most k in a directed graph, using $n^{O(\lg(k))}$ time and $O(\lg(n) \cdot \lg(k))$ space. We explore the relationship between small space k-cycle detection and algorithms for s-t connectivity, and show that the complexities of the two problems are tightly connected. To the best of our knowledge, no s-t connectivity algorithms have been reported running in $O(\lg^2 n)$ space and $n^{0.999 \cdot \lg(n)}$ time.

Our Results

The primary contribution of this paper is a reduction from directed k'-cycle to undirected $2k$-cycle for constant k' and k such that $k = k' \operatorname{poly} \lg(k)$. We use this reduction to show hardness for even cycle detection in both small space and in sparse graphs.

We prove that, for small-space algorithms, the time complexities of k-cycle detection and k-path detection are equivalent up to constant factors in the exponent. We show that an $n^{o(\lg(k))}$ time and $O(\lg(n))$ space algorithm for the $2k$-cycle detection problem would imply an algorithm for s-t running in $n^{o(\lg(n))}$ time and $O(\lg^2(n))$ space, a significant improvement over Savitch's algorithm.

We propose two natural lower bound hypotheses, and derive interesting consequences from them. The first hypothesis is that s-t connectivity in $O(\lg^2(n))$ space requires $n^{\Omega(\lg(n))}$ time. The second (stronger) hypothesis is that s-t connectivity in $\operatorname{poly} \lg(n)$ space requires $n^{\Omega(\lg(n))}$ time. We call these hypotheses the Weak Savitch Hypothesis (WSH) and the Strong Savitch Hypothesis (SSH) respectively.

▶ **Definition 1.** *The Weak Savitch Hypothesis (WSH) states that given a directed graph G with n nodes and two specified nodes s and t determining if a path exists from s to t requires $n^{\Omega(\lg(n))}$ time if the algorithm can only use $O(\lg^2(n))$ space.*

▶ **Definition 2.** *The Strong Savitch Hypothesis (SSH) states that given a directed graph G with n nodes and two specified nodes s and t determining if a path exists from s to t requires $n^{\Omega(\lg(n))}$ time if the algorithm can only use $O(\operatorname{poly} \lg(n))$ space.*

Both of our Hypotheses are about the s-t connectivity problem, two well known algorithms for s-t connectivity are:

- Savitch's algorithm [12] running in time $\tilde{O}(n^{\lg n})$ and space $O(\lg^2 n)$.
- Depth-First search running in polynomial time and space $\tilde{O}(n)$.

The only improvement over these for any space greater than $\lg^2 n$ is the algorithm of Barnes, Buss, Ruzzo and Schieber [4] who gave a space $O(n/2^{\sqrt{\lg(n)}})$, polynomial time algorithm. They are also able to give a time space tradeoff of space s where $s > \lg^2 n$ and time $2^{O(\lg^2(n/s))}\operatorname{poly}(n)$. This time space tradeoff would require $n^{1-o(1)}$ space to achieve $n^{o(\lg n)}$ time. Our hypotheses are much weaker and only claim the non-existence of $n^{o(\lg n)}$ time algorithms for $O(\lg^2(n))$ space and $O(\operatorname{poly} \lg(n))$ space respectively.

We prove the following hardness for $2k$-cycle under Weak Savitch Hypothesis.

▶ **Theorem 3.** *Assuming Weak Savitch Hypothesis (WSH) solving k-cycle in undirected graphs in $O(\lg n)$ space requires $n^{\Omega(\lg k)}$ time.*

Could we strengthen the hypotheses (Weak Savitch Hypothesis (WSH) and the Strong Savitch Hypothesis (SSH)) to state that there are no $n^{(1-\epsilon)\lg(n)}$ time algorithms for any constant $\epsilon > 0$, instead of merely saying there is no $n^{o(\lg(n))}$ time algorithms? We prove that such a hypothesis would actually be false for randomized algorithms.

▶ **Theorem 4.** *There is a randomized algorithm for directed st-connectivity which runs in expected time $n^{\lg n/2+o(\lg n)}$ and $O(\lg^2(n))$ space.*

Our randomized algorithm works as follows. First, for sufficiently short path lengths (up to $\lg(n)$) we run Savitch's algorithm as usual. For longer path lengths, instead of trying to find the middle vertex of the path we look for any vertex which is approximately in the middle. We show that this minor modification can actually speed up Savitch's algorithm by a quadratic factor. Similar ideas were used by Zwick [19] to give algorithms for All pairs shortest paths problem in arbitrary space regime.

Other Implications

Our k'-cycle to $2k$-cycle reduction ($k = k' \operatorname{poly}\lg(k)$) has many more implications. We give lower bounds for sparse even-cycle detection by reducing Max-3-SAT to sparse even-cycle detection and give a tight combinatorial lower bound for sparse even-cycle. We achieve this by combining our reduction and previous hardness results for directed k-cycle [10].

▶ **Theorem 5.** *If undirected $2k$-cycle, for any large enough constant k can be solved in a graph G with $m = O(n)$ in time $O(m^{1.5-\epsilon})$ then max-3-SAT can be solved in time $O^*(2^{(1-\epsilon')n})$.*

We additionally present a very simple reduction from detecting odd k-cycles to detecting $2k$-cycles. This reduction is inspired by a reduction from Dahlgaard et al. [5]. Our results here are tight, but only in the "combinatorial" regime. The notion of *combinatorial algorithms* is not formally defined, however, it is a commonly used notion (e.g. [10, 1]) which informally just excludes fast matrix multiplication as a subroutine (e.g. [6, 15]). The category of combinatorial algorithms considered generally assumes boolean matrix multiplication requires $n^{3-o(1)}$ time, and attempts to capture algorithms that are fast in practice [16]. We can obtain a combinatorial lower bound for the undirected $2k$-cycle detection of $\Theta(m^{2-o(1)})$ time, for all large enough constants k, if the Combinatorial k-Clique Hypothesis is true.

Additionally for *odd* k we show a tight bound for combinatorial $2k$-cycle detection. If you can detect $2k$-cycles combinatorially in time $m^{(1-\epsilon)2k/(k+1)}$ for some $\epsilon > 0$ then you violate the Combinatorial k-Clique Hypothesis. Dahlgaard et al. present an algorithm for all $2k$-cycles that runs in time $m^{2k/(k+1)}$. So for odd k, we have a tight combinatorial lower bound for $2k$-cycles.

Previous Work

Over the years, there have been many algorithmic results for the cycle detection problem. In arbitrary space, $2k$-cycle can be solved in $O(n^2)$ space as long as k is a constant [17]. This beats the matrix multiplication (n^ω) time algorithms for odd k-cycle detection assuming the matrix multiplication constant is larger than two ($\omega > 2$) [2]. Dahlgaard et al. give an algorithm for sparse even $2k$-cycle detection which runs in $O(m^{2k/(k+1)})$ time, where m is the number of edges in the input graph [5]. These algorithms require $\Omega(m)$ space to run. In very small space finding a k-cycle takes much longer, and as we show in this paper has a strong relationship to s-t connectivity in small space. Savitch's algorithm solves s-t connectivity in space $\lg^2(n)$ in time $n^{\lg(n)-o(\lg(n))}$ [12]. In this paper we present a randomized algorithm which runs in space $\lg^2(n)$ in time $n^{\lg(n)/2+o(\lg(n))}$, improving over the Savitch algorithm. The problem of solving s-t connectivity in less than $\lg^2(n)$ space has also been explored. Gopalan et al. present a randomized time space tradeoff for the s-t connectivity problem where given $\lg^2(n)/\lg(d)$ space they acheive a running time of $O(n^{d\lg(n)/\lg(d)})$ for $d \in [2, n]$ [7].

While we provide a reduction from directed k-cycle for arbitrarily large k to even cycle, a previous paper reduced 3-cycle specifically to even cycle [5]. Their reduction technique inspired ours. They give a lower bound of $m^{3/2-o(1)}$ for even cycle relying on a combinatorial conjecture for the running time of 3-cycle detection. Notably, they assume no combinatorial $n^{3-o(1)}$ algorithm exists for 3-cycle detection. We give a matching lower bound of $m^{3/2-o(1)}$ using a *non-combinatorial* hypothesis.

Furthermore, our reduction gives an improved lower bound of $m^{2-o(1)}$ using a combinatorial assumption. Giving a tight combinatorial lower bound for the even cycle problem. We give a combinatorial lower bound of $m^{\frac{2k}{k+1}-o(1)}$ for the undirected $2k$-cycle detection problem for odd k. This is closely related to the running time of $O(m^{2k/(k+1)})$ for the algorithm detecting *even* k cycles combinatorially by Dahlgaard et al. [5]. We can additionally use a simple reduction inspired by Dahlgaard et al. [5] and a reduction from k-clique to sparse k-cycle from previous work [10] to give a tight combinatorial lower bound for the $4k+2$-cycle problem. to give a lower bound of $m^{\frac{2k}{k+1}-o(1)}$ for detecting $2k = 4k' + 2$ cycles. This shows optimality of the Dahlgaard et al algorithm for half of all even k-cycle detection problems.

By giving a reduction from directed k'-cycle to $2k$-cycle we can show $n^{\Omega(\lg(k))}$ hardness for $2k$-cycle in $O(\lg(n))$ space. However, one can not show this hardness using the 3-cycle reduction of Dahlgaard et al., because 3-cycle has an $O(n^3)$-time logspace algorithm.

In the weaker model of AC_0 formulas a $n^{\Omega(\lg(k))}$ lower bound is known unconditionally for the $\leq k$ connectivity problem [11].

Improving low-space algorithms for directed st-connectivity is a longstanding open problem [14]. Barnes, Buss, Ruzzo and Schieber [4] gave a sublinear space $O(n/2^{\sqrt{\lg(n)}})$, polynomial time algorithm. Even their results only give $(\lg n)^{O(\lg n)}$ factor improvements over Savitch if we restrict to polylog space. Hence for polylog space they require $n^{\lg n(1-o(1))}$ time. Melkebeek and Prakriya [13] gave a unambiguous algorithm running in polynomial time and $O(\lg^{3/2} n)$ space. The time and space here are both better than the ones in our hypotheses but unambiguous Turing machines are possibly much more powerful then deterministic ones.

Organization

In Section 2 we cover definitions and preliminaries. In Section 3 we present a faster s-t connectivity algorithm. In Section 4 we give our reduction between directed k-cycle and even k-cycle. In Section 5 we cover the implications of our reduction.

2 Preliminaries

We will always assume k in the k-cycle/walk problems to be a constant unless stated otherwise.

Notation

We will use the notation $\tilde{O}(f(n))$ to suppress sub-polynomial factors $\left(f(n)^{o(1)}\right)$.

Definitions

▶ **Definition 6.** *A k-cycle in a graph G is a set of k distinct vertices, $x_1, \ldots, x_k$ in G such that the edges $(x_1, x_2), \ldots, (x_{k-1}, x_k), (x_k, x_1)$ all exist in G.*

A k-walk in a graph G is a set of k not necessarily distinct vertices, $x_1, \ldots, x_k$ in G such that the edges $(x_1, x_2), \ldots, (x_{k-1}, x_k)$ all exist in G.

Throughout the paper we will take k to be a large enough constant. We will have runtimes of the form $n^{\Theta(\lg(k))}$ where Θ hides a constant independent of k; such notation only makes sense for k that can grow unboundedly.

We will assume throughout that all graphs are presented in adjacency list format. This is done for simplicity and not assuming this has no effect on our results.

▶ **Reminder of Definition 1.** *The Weak Savitch Hypothesis (WSH) states that given a directed graph G with n nodes and two specified nodes s and t determining if a path exists from s to t requires $n^{\Omega(\lg(n))}$ time if the algorithm can only use $O(\lg^2(n))$ space.*

▶ **Reminder of Definition 2.** *The Strong Savitch Hypothesis (SSH) states that given a directed graph G with n nodes and two specified nodes s and t determining if a path exists from s to t requires $n^{\Omega(\lg(n))}$ time if the algorithm can only use $O(\operatorname{poly}\lg(n))$ space.*

Currently the best known algorithms for directed s t connectivity which run in time $n^{o(\lg(n))}$ require $n^{1-o(1)}$ space[4].

3 Making Savitch run faster

3.1 Savitch's Algorithm

We begin by recalling Savitch's classical algorithm.

Algorithm 1 Savitch's Algorithm.

```
    Input: Graph $G = (V, E)$, vertices $s, t$, $k \geq 0$
    Output: If there exists a path of length at most $k$ from $s$ to $t$.
1:  function SA$(G, s, t, k)$
2:      if $k = 1$ then
3:          Output TRUE if $(s, t) \in E$, otherwise FALSE
4:      Let $|V| = n$.
5:      for $v \in [n]$ do
6:          if $SA(G, s, v, \lfloor k/2 \rfloor) \wedge SA(G, v, t, \lceil k/2 \rceil) =$ TRUE then
7:              Output TRUE
8:      Output $ret$
```

▶ **Lemma 7** ([12]). *Savitch's algorithm solves directed $\leq k$-reachability in time $T_S(n, k) = \tilde{O}(n^{\lg(k)})$ using space $O(\lg(k)\lg(n))$.*

Savitch's Algorithm [12] solves the directed $s - t$ connectivity problem as follows. It considers the more general problem of $\leq k$ connectivity. $k = n$ is equivalent to $s - t$ connectivity. For determining if there is a path of length at most k between s and t, we guess the middle vertex v in such a path, then recursively try to solve $\leq k/2$ connectivity for (s, v) and (v, t). This gives us the recurrence:

$$T(n, k) \leq n(2 \cdot T(n, k/2)) + \operatorname{poly}(n)$$

with the base case $T(n, 1) = O(\operatorname{poly}(n))$. This base case is unaffected by being in adjacency list format or adjacency matrix format. This solves to $T(n, n) = \tilde{O}(n^{\lg n})$.

We now turn improving the running time of s-t connectivity in $O(\lg^2(n))$ space.

Similarly to the Savich's algorithm we will be guessing a node in the middle of the hypothetical path. However, Savich's algorithm requires the node that is exactly in the middle of the path. In our algorithm we will accept a node that is approximately in the middle. There is a trade-off with approximating. By approximating we improve the probability of an accept. Consider if we accepted any node that was with $\epsilon_k \cdot k$ nodes of the true middle of a k length path, then we have a $2\epsilon_k/n$ probability of correctly guessing such a node. However, this approximation factor trades off how much progress we make with a correct guess. In Savich's Algorithm each correct guess halves the path length. In contrast, with an approximation factor of ϵ_k we divide the path into the less favorable $1/2 - \epsilon_k$ and $1/2 + \epsilon_k$ split. This trade-off is optimal when ϵ_k is approximately $1/\lg(n)$, so we accept any node that is in the middle $1/\lg(n)$ fraction of a k length path.

Our square-root speedup intuitively comes from improving the probability of correctly guessing from $1/n$ to $k/(\lg(n) \cdot n)$. This approaches n as k approaches $\lg(n)$, but achieves a huge savings when k is much larger than $\lg(n)$.

▶ **Reminder of Theorem 4.** *There is a randomized algorithm for directed st-connectivity which runs in expected time $n^{\lg n/2+o(\lg n)}$ and $O(\lg^2(n))$ space.*

Proof. For sufficiently short path lengths (up to $\lg(n)$) we run Savitch's algorithm as usual. For longer path lengths, instead of trying to find the middle vertex of the path we look for any vertex which is approximately in the middle. We describe the algorithm in full detail below.

Algorithm 2 Improved Savitch's Algorithm.

```
   Input: Graph $G = (V, E)$, vertices $s, t$
   Output: If there exists a path from $s$ to $t$ in $G$.
1: function ISA$(G, s, t)$
2:     Output $ISA_2(G, s, t, n)$
3: function $ISA_2(G, s, t, k)$
4:     if $k \leq \lg(n)$ then
5:         Output SA(G, s, t, k)
       let $V_r$ be a list of randomly sampled nodes from $V$.
6:     for $v \in V_r$ do
7:         if $ISA_2(G, s, v, k/2 + k/\lg(n))) \wedge ISA_2(G, v, t, k/2 + k/\lg(n)) =$ TRUE then
8:             Output TRUE
9:     Output FALSE
```

In Savitch's algorithm we try to guess the middle vertex in a path. Our improvement derives from the fact that we only look for a vertex which is approximately in the middle.

Savitch's algorithm considers the subproblem of $\leq k$ connectivity. We consider a more relaxed notion described below.

$ISA_2(G, s, t, k)$ returns YES if there exists a path of length at most k between s and t and returns NO if there is no path between s and t. ISA_2 as implemented may return either YES or NO if there is a path from s to t but no path of length at most k. Returning YES on larger length paths does not affect correctness as in st-connectivity we are only concerned about existence of a path, not a bounded length path.

Suppose there exists an at most k length path between s and t. We guess a vertex v hoping that v is one of the middle $2\delta_k = 2k/\lg(n)$ vertices in this path. We will guess v uniformly at random. The probability of a correct guess is $\epsilon_k = 2\delta_k/n$.

If our guess is correct distance between (s, v) and (v, t) are at most $k/2 + \delta_k$ hence we can recurse on $ISA_2(G, s, v, k/2 + \delta_k)$ and $ISA_2(G, v, t, k/2 + \delta_k)$.

Let $T(n, k, s, t)$ be the random variable which denotes the time for solving $ISA_2(G, s, t, k)$ using Algorithm 2.

Let $T_E(n,k) = \max_{s,t} \mathbb{E}[T(n,k,s,t)]$.
Setting $\delta_k = \max(k/(\lg n), 1)$ gives us the following recurrence for $k \geq \lg(n)$:

$$T_E(n,k) \leq \sum_{i=1}^{\infty} \Pr[i^{th} \text{ guess was the first correct guess}] \cdot i \cdot (2T_E(n, k/2+\delta_k)) + \text{poly}(n))$$
$$T_E(n,k) \leq \sum_{i=1}^{\infty} \left((1-\epsilon_k)^{i-1} \epsilon_k \cdot i \cdot (2T_E(n,k/2+\delta_k)) + \text{poly}(n)\right).$$

We can rewrite this equation as:

$$\begin{aligned} T_E(n,k) &\leq \left(\sum_{i=1}^{\infty} i\left((1-\epsilon_k)^{i-1}\epsilon_k\right)\right) \cdot (2T_E(n,k/2+\delta_k) + \text{poly}(n)) \\ &\leq (1/\epsilon_k) \cdot (2T_E\left(n, k/2+\delta_k\right) + \text{poly}(n)) \\ &\leq 2\left(1/\epsilon_k\right) \cdot T_E\left(n, k/2+\delta_k\right) + \text{poly}(n). \end{aligned}$$

Now we can plug in $\epsilon_k = 2\delta_k/n$ and $\delta_k = \max(k/(\lg n), 1)$ to get the following:

$$\begin{aligned} T_E(n,k) &\leq (n/\delta_k) \cdot T_E\left(n, k/2+\delta_k\right) + \text{poly}(n) \\ &\leq \frac{n\lg(n)}{k} \cdot T_E\left(n, k/2 + k/\lg(n)\right) + \text{poly}(n). \end{aligned}$$

For $k \geq \lg(n)$, at each recursion step k gets multiplied by $(1/2 + 1/\lg(n))$. Let k_j denote its value after j steps and k_0 denotes its initial value. For directed st connectivity $k_0 = n$.

For $k_0 = n$ and for $j \leq b = \lceil -\frac{\lg(n/\lg(n))}{\lg(1/2+1/\lg(n))} \rceil$ we have $k_j = k_0(1/2+1/\lg(n))^j = n(1/2 + 1/\lg(n))^j$ as for $j < b$, $n(1/2+1/\lg(n))^j > \lg(n)$. Note that b is positive ($\lg(1/2+1/\lg(n))$ is negative). Note that $k_b = n(1/2+1/\lg(n))^j \leq \lg(n)$.

$$T_E(n,n) \leq \left(\prod_{j=0}^{b-1} \frac{n\lg(n)}{k_j}\right) \cdot (T_E(n,k_b) + \text{poly}(n)).$$

As $k_j = n(1/2+1/\lg(n))^j$ for $j \leq b$ and $k_b \leq \lg(n)$

$$T_E(n,n) \leq \left(\prod_{j=0}^{b-1} \frac{n\lg(n)}{n(1/2+1/\lg(n))^j}\right) \cdot (T_E(n,\lg(n)) + \text{poly}(n)).$$

When we are looking for paths of length at most $\lg(n)$ we will call the original Savitch's algorithm. So, $T_E(n,\lg(n)) = T_S(n,\lg(n)) = \tilde{O}(n^{\lg\lg n}) = n^{o(\lg(n))}$ by Lemma 7. Recalling that b is positive and $b = \lceil -\frac{\lg(n/\lg(n))}{\lg(1/2+1/\lg(n))} \rceil$:

$$\begin{aligned} T_E(n,n) &\leq \left(\prod_{j=0}^{b-1} \frac{\lg(n)}{(1/2+1/\lg(n))^j}\right) \cdot (n^{o(\lg(n))} + \text{poly}(n)) \\ &\leq \left(\frac{\lg^b(n)}{(1/2+1/\lg(n))^{b^2/2}}\right) \cdot (n^{o(\lg(n))}) \end{aligned}$$

As $b = \lceil -\frac{\lg(n/\lg(n))}{\lg(1/2+1/\lg(n))} \rceil$, $\lg^b(n) = n^{o(\lg(n))}$ we have

$$\begin{aligned} T_E(n,n) &\leq (1/2 + 1/\lg(n))^{-b^2/2} \cdot n^{o(\lg(n))} \\ &\leq 2^{-\frac{\lg(1/2+1/\lg(n))\lg^2(n)}{2\lg(1/2+1/\lg(n))^2}} \cdot n^{o(\lg(n))} \\ &\leq 2^{-\frac{\lg^2(n)}{2\lg(1/2+1/\lg(n))}} \cdot n^{o(\lg(n))} \\ &\leq 2^{\frac{\lg^2(n)}{2(1-\lg(1+2/\lg(n)))}} \cdot n^{o(\lg(n))} \\ &\leq 2^{\frac{\lg^2(n)}{2(1-2/\lg(n)))}} \cdot n^{o(\lg(n))} \\ &\leq 2^{\frac{\lg^2(n)}{2}} \cdot n^{o(\lg(n))} \\ &\leq n^{(\lg(n)/2+o(\lg(n)))}. \end{aligned}$$

So $T_E(n,n) \leq n^{(\lg(n)/2+o(\lg(n)))}$, giving the desired result. ◀

4 Reductions Between k-cycle Problems

In this section we will give a reduction from directed k' cycle to $2k$ cycle, where k' is odd and $k' \leq k/\lg^{O(1)}(k)$. This will allow us to give lower bounds for even cycle as well as odd cycle in the sparse setting. First, we will need a helper lemma about the existence of primes with appropriate properties.

▶ **Lemma 8.** *There exists a constant $k^\star$ such that $\forall k \geq k^\star$ we can write $2k = \sum_{i=0}^{4} z_i r_i$ where $z_i \in \mathbb{N}, r_i \leq \lg^{O(1)}(k), \sum |z_i r_i - 2k/5| \leq \lg^{O(1)}(k)$ and any four of r_i's together have a common prime factor which does not divide k.*

Proof. Let p_i for $0 \leq i \leq 4$ be distinct prime numbers which do not divide k. By the prime number theorem we can take $p_i \leq \lg^2(k)$ for large enough constant k. Define $r_i = \prod_{j \neq i} p_j$. This allows us to satisfy the requirements that $r_i \leq \lg^{O(1)}(k)$ and any four of r_i's together have a common prime factor which does not divide k.

As the $gcd(r_i) = 1$, by Eulid's gcd algorithm there exists integers y_i such that $1 = \sum y_i r_i$ hence there exists integers z_i for which $2k = \sum z_i r_i$. Choose z_i's such that $\sum |z_i r_i - 2k/5|$ is minimized. Assume $\sum |z_i r_i - 2k/5| > 10 \lg^{11}(k)$. This implies that there exists a j, j' such that $z_j r_j - 2k/5 > \lg^{11}(k)$ and $z_{j'} r_{j'} - 2k/5 < -\lg^{11}(k)$. We can substitute z_j with $z_j - p_j$ and $z_{j'}$ with $z_{j'} + p_{j'}$. This maintains the sum $\sum z_i r_i = 2k$ as $r_j p_j = r_{j'} p_{j'}$. The substitution decreases $z_j r_j - 2k/5$ by $\prod p_i < \lg^{10}(k) < \lg^{11}(k)$ and increases $z_{j'} r_{j'} - 2k/5$ by $\prod p_i < \lg^{10}(k) < \lg^{11}(k)$. Hence both $|z_j r_j - 2k/5|$ and $|z_{j'} r_{j'} - 2k/5|$ decrease while $|z_i r_i - 2k/5|$ for $i \neq j, j'$ remain the same. This gives a contradiction as we chose z_i's to minimize $\sum |z_i r_i - 2k/5|$. So we can assume that $\sum |z_i r_i - 2k/5| \leq 10 \lg^{11}(k)$. This also implies that for all i, $z_i r_i > 2k/5 - \lg^{O(1)}(k)$ hence $z_i > 0$ for large enough k. ◀

Given a sparse graph G there are many possible ways for the input to be formatted. Given an adjacency list format there exist two different efficient algorithms for returning if (u, v) is an edge in the graph. If one is worried about space and not time (as we are in our $O(\lg(n))$ space lower bounds) there is a $O(\lg(n))$ space $O(\lg(n))$ time algorithm. Simply binary search through the adjacency list input to find (u, v). If one is worried about time and not space then one can hash the adjacency list giving a $O(n)$ space data structure that has lookup time $O(1)$. We don't care about $\lg(n)$ factors in our time and thus default to the $O(\lg(n))$ space and time algorithm.

▶ **Theorem 9.** *There exists a constant $k^\star$ such that $\forall k \geq k^\star$, there exists $k' \geq k/(\lg^{O(1)} k)$ such that a n-node, m edge instance of directed k' cycle can be reduced to $\tilde{O}(m)$ node, $\tilde{O}(m)$ edge instance of $2k$ cycle on graph G. Further any bit in the description of G can be produced by a $O(\lg(n))$ space and $O(\lg(n))$ time.*

Proof. Take z_i, r_i's satisfying Lemma 8 for k. Set $k' = \sum z_i$. Let G be a instance of directed k' cycle with n nodes and m edges. We will first apply color coding using Lemma 20, to produce a k' partite graph. This graph will only have edges between "adjacent" partitions. As explained in Lemma 20.

Let $s_i = \sum_{j<i} z_j$. We create a new graph G' by replacing each directed edge in between the s_i to the $s_{(i+1 \mod 5)}$(cyclically) parts by a r_i length path with undirected edges. As we are replacing z_i partitions of edges by an r_i length path this gives a natural partition of G' with $\sum z_i r_i = 2k$ parts.

$$\begin{aligned}\text{Number of nodes and edges in } G' &\leq \max(r_i) \cdot m \\ &\leq \lg^{O(1)}(k) \cdot m \quad \text{[By Lemma 8]} \\ &\leq O(m) \qquad \text{[As } k \text{ is a constant]}\end{aligned}$$

We will now show that the new graph G' has a $2k$-cycle iff G had a directed k'-cycle.

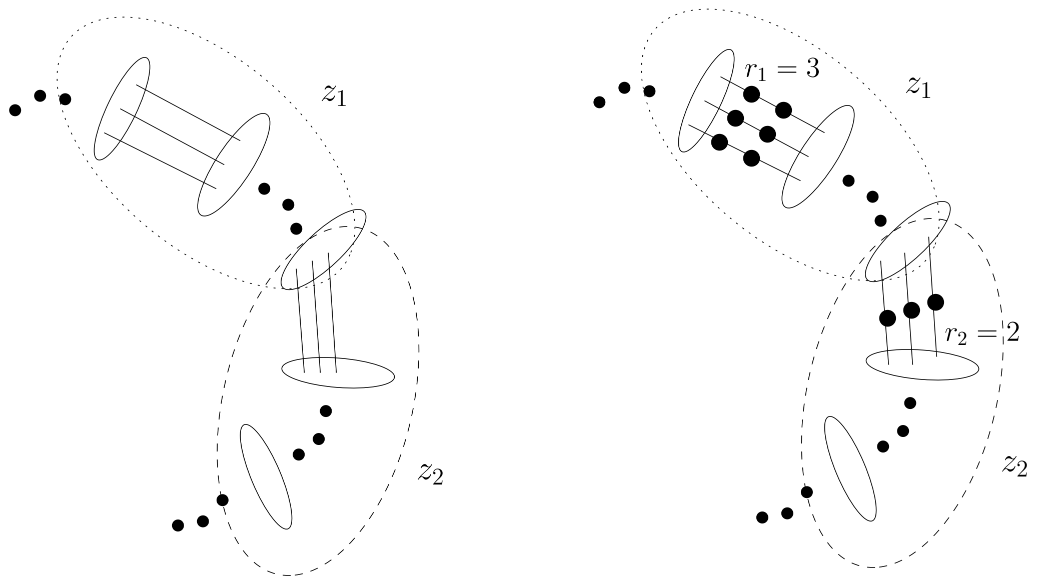

Figure 1 Depicting the edge transformation from G to G'. The dotted oval contains z_1 edge sets. The dashed oval contains z_2 edge sets. Every edge in the dotted oval is turned into r_1 edges by replacing the edge with a path. Every edge in the dashed oval is turned into r_2 edges by replacing the edge with a path.

Each edge in the graph G' is part of one of the introduced r_i length paths. If one of the edges from an r_i length path is used for a $2k$ cycle then all the edges in the path must be used as intermediate vertices in the path have no other edges that the edges in the path itself. Hence we can think of our $2k$ cycle as composed of meta-edges of lengths r_i. Suppose for constructing the $2k$ cycle we only use at most 4 out of the 5 possible r_i lengths. Then we cannot construct a $2k$ cycle as these four r_i's share a common prime factor which does not occur in $2k$. Hence we need to use all 5 possible r_i lengths. Wlog we can assume that we start from 0th partition. There are 2 approaches to hitting all 5 possible r_i length paths:

1. Go around all the $2k$ partitions.
2. Do not go around all partitions.

For type 1, note that we can never back-track as otherwise we will have more than $2k$ edges. If we find a $2k$ cycle using type 1 without backtracking this exactly corresponds to a directed k' cycle in G by replacing all meta-edges by the corresponding edge of G.

For type 2 paths note that we must hit all of the 5 possible lengths.

Let group i be the set of edges produced in G' from the edges between the s_i and $s_{i+1 \mod 5}^{th}$ partitions in G. To pass from one side of group i to the next requires passing through at least z_i paths of length r_i. So passing from one side of group i to the other requires a path of length at least $z_i r_i$.

To hit all possible lengths we must have at least one edge from all five groups. To pass through all five groups at minimum requires doubling back through three groups. To double back through a group requires two paths each of length $z_i r_i$. See Figure2 for a depiction.

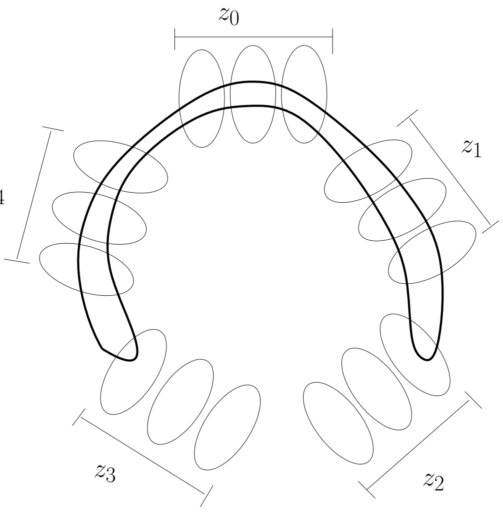

Figure 2 In bold we depict a path passing through partitions, reaching at least one r_i length path of each of the 5 possible lengths. Each oval represents a partition of s_j. Between each partition is a path of one of the lengths r_i. Note that to reach all five the path must "double back" through three partitions.

For simplicity understand r_{i+c} and z_{i+c} in the next equation to be $r_{(i+c \mod 5)}$ and $z_{(i+c \mod 5)}$ respectively. So the minimum length of a type two cycle is

$$\min_i \left(2r_i + 2z_{i+1}r_{i+1} + 2z_{i+2}r_{i+2} + 2z_{i+3}r_{i+3} + 2r_{i+4}\right).$$

By Lemma 8 $\sum |z_j r_j - 2k/5| \leq \lg^{O(1)}(k)$, so $\sum |2z_j r_j - 4k/5| \leq 2\lg^{O(1)}(k)$. Thus, $\sum_{j=i+1}^{i+3} |2z_j r_j - 4k/5| \leq 2\lg^{O(1)}(k)$. Finally giving

$$\sum_{j=i+1}^{i+3} 2z_j r_j \geq \frac{12}{5}k - 2\lg^{O(1)}(k) = 2.4k - 2\lg^{O(1)}(k) > 2k.$$

A type 2 $2k$-cycle, which doesn't form a cycle through all partitions, must have length greater than $2k$. So, no such cycle exists.

Thus, the only $2k$-cycles in this graph correspond exactly to a directed k'-cycle in the original graph. ◀

5 Implications of k-Cycle Reductions

We now discuss the implications of the reduction from directed k'-cycle to undirected $2k$-cycle. We will start by showing that the algorithms for $2k$-cycle in $\lg(n)$ space are tight up to constant factors in the exponent. Next, we will give lower bounds for $2k$-cycle from Max-3-SAT.

5.1 2k-Cycle $\lg(n)$ Space Algorithms are Tight

Given the Weak Savitch Hypothesis we can show that undirected $2k$-cycle problems have nearly optimal algorithms.

In this section we give a reduction that, for all large enough k, reduces directed k'-cycle to undirected $2k$-cycle. That is, there is some constant $k^\star$ such that for all $k > k^\star$ we can reduce directed k'-cycle to undirected $2k$-cycle. Furthermore, in our reduction $k' = \Theta(k/\lg(k))$. So, for all $k > k^\star$ we have that $2k$-cycle reduces from directed $\Theta(k/\lg(k))$-cycle.

Following Lemmas 10 and 11 are very similar to lemmas in previous work and are roughly folklore. We present their proofs for completeness.

▶ **Lemma 10.** *If the Weak Savitch Hypothesis is true then $\leq k-$connectivity in an directed graph G with n nodes requires $n^{\Omega(\lg k)}$ time if we are restricted to $O(\lg n)$ space.*

Proof. We can consider the Savitch algorithm for determining if there is a path of length $\leq \ell$ between two nodes in a graph with n nodes. The Savitch algorithm [12] as described in Section 3 has the following recurrence :

$$T(n, \ell) \leq 2nT(n, \ell/2) + \text{poly}(n).$$

Where the standard base case is $T(n, 1) = O(\lg(n))$. This gives an algorithm running in time $O(n^{(\lg \ell)}\text{poly}(n))$.

If Weak Savitch Hypothesis is true then there exists some constant $\epsilon \geq 0$ such that no algorithm solving st connectivity (equivalent to $\leq n$ connectivity) in $\lg^2(n)$ space runs in time $n^{\epsilon \lg(ell)}$.

If an algorithm for k-path runs in $n^{o(\lg(k))}$ and $O(\lg n)$ space then we can repeatedly use it to find an $\leq \ell$ length path. Specifically, we will run the k-path algorithm where we treat an edge as existing if there is an ℓ/k walk between two nodes. To determine if such an edge exists we will recurse. This gives us the following recurrence:

$$T(n, \ell) \leq n^{o(\lg(k))}T(n, \ell/k) + \tilde{O}(1).$$

For $\ell = n$, this recurrence gives us $T(n, n) \leq n^{o(\lg(k)) \lg_k(n)}$ This running time can be simplified to $O(n^{o(\lg(k)) \lg(n)/\lg(k)})$.

For all ϵ there exists some sufficiently large but constant k such that $o(\lg(k))/\lg(k) \leq \epsilon$ which contradicts the Weak Savitch Hypothesis. ◀

▶ **Lemma 11.** *Assuming Weak Savitch Hypothesis (WSH) directed, k-cycle cannot be solved in $O(\lg n)$ space and $n^{o(\lg k)}$ time.*

Proof. Suppose directed k-cycle is in $O(\lg n)$ space and $n^{o(\lg k)}$ time. Suppose we are given an instance of $k-1$-walk $(G = (V, E), a, b)$. Construct a new graph G' with k copies of each vertex v named $v_1, v_2, \ldots, v_k$. Add u_i, u_{i+1} as an edge. Add (u_i, v_{i+1}) as an edge in G' if (u, v) was an edge in G. Also add (a_k, b_1) as an edge. Now there is a k-cycle in G' iff there was a $k-1$-walk from a to b in G. As the number of vertices in G' is kn and k

is a constant hence we have a $O(\lg n)$ space and $n^{o(\lg k)}$ time algorithm for (un)directed k-walk. By Lemma 10 this contradicts Weak Savitch Hypothesis (WSH). Note that we will not actually construct G' instead we use $O(\lg n)$ space and have a $O(\lg n)$ time algorithm that serves as an oracle for G'. ◀

▶ **Reminder of Theorem 3.** *Assuming Weak Savitch Hypothesis (WSH) solving k-cycle in undirected graphs in $O(\lg n)$ space requires $n^{\Omega(\lg k)}$ time.*

Proof. By Lemma 11 we know that k'-cycle in directed graphs cannot be solved in $O(\lg n)$ space and $n^{o(\lg k')}$.

Suppose we have a $O(\lg n)$ space and $n^{o(\lg k)}$ time algorithm for k-cycle in undirected sparse graphs for even k. Then let k' be the largest integer satisfying the definition in Theorem 9. Then by Theorem 9 we can reduce directed k'-cycle to k-cycle and solve it in time $n^{o(\lg k)} = n^{o(\lg k')}$ because $k' \geq k/\lg^{O(1)}(k)$. Note that as in the reduction we can produce each bit in $O(\lg(n))$ space and time $O(\lg(n))$ our space and running time bounds are not affected. The reduction only increases the number of vertices by a constant factor and k is large enough. Using Lemma 11 gives a contradiction. ◀

▶ **Corollary 12.** *Assuming Weak Savitch Hypothesis (WSH) detecting k-cycles in undirected sparse graphs in $O(\lg(n))$ space takes $n^{\Theta(\lg k)}$ time.*

Proof. By Theorem 3 we have that k-cycle in log space requires $n^{\Omega(\lg k)}$ time. By Lemma 21 we have that k-cycle can be solved in log space and $n^{O(\lg k)}$ time. Thus, the k-cycle problem requires $n^{\Theta(\lg k)}$ time, conditioned on WSH.

◀

Note that we can replace the Weak Savitch Hypothesis with the Strong Savitch Hypothesis in these proofs. They will in fact make the statements stronger. If the Strong Savitch Hypothesis is true then k-cycle in undirected graphs cannot be solved in $O(\text{poly}\lg(n))$ space and $n^{o(\lg k)}$ time.

5.2 Even Cycle Lower Bounds From Max-CSP

Previous work [10] has connected efficiently detecting directed k-cycles in sparse graphs with solving the Max-L-SAT problem [10]. In fact, this work connects directed k-cycle detection to Max-L-CSP problems, where every clause is representable by a L degree Boolean polynomial. In this section we will limit our focus to the Max-3-SAT and Max-L-SAT problems.

If, for some particular constant L, one believes that the Max-L-SAT problem requires $2^{n(1-o(1))}$ time, then from this we get a super linear lower bound for constant length even cycle. In fact, we get a super linear lower bound for constant length undirected even cycle in sparse graphs where $m = O(n)$. Such a super linear linear lower bound for undirected even cycle has not previously been obtained.

Our reduction provides a lower bound for sparse undirected even cycle from assumptions made about difficult exponential time Constraint Satisfaction Problems (CSPs). Notably, we are able to provide super linear lower bounds for even cycles from assumptions stated in the paper of Lincoln, Vassilevska Williams and Williams [10].

▶ **Theorem 13** (Sparse Directed k-Cycle is Hard [10]). *If directed k-cycle, for any large enough k, can be solved in a graph G in time $O(m^{c_k-\epsilon})$, where $c_k = k/(k - \lceil k/3 \rceil + 1)$ then max-3-SAT can be solved in time $O^*(2^{(1-\epsilon')n})$.*

Using our reduction and the above theorem we can provide a lower bound for undirected sparse cycle.

▶ **Reminder of Theorem 5.** *If undirected $2k$-cycle, for any large enough constant k can be solved in a graph G with $m = O(n)$ in time $O(m^{1.5-\epsilon})$ then max-3-SAT can be solved in time $O^*(2^{(1-\epsilon')n})$.*

Proof. By applying Theorem 9 we transform an input graph into one that has $m = O(n)$. Furthermore, it shows that if undirected $2k$-cycle, for all constant k, can be solved in time $O(m^{1.5-\epsilon})$ then, there exists an infinite sequence of increasing odd cycle lengths solved in time $O(m^{1.5-\epsilon})$.

Given an infinite increasing sequence of odd cycle lengths they solve the same length of directed cycle. Then, there exists some $k \geq \frac{1}{\epsilon}3 \geq \frac{1}{\epsilon}\frac{9}{4}$ which, if solved in $O(m^{1.5-\epsilon})$ time is solved faster than $O(m^{c_k-\epsilon})$, where $c_k = k/(k - \lceil k/3 \rceil + 1)$ time.

Thus, undirected even cycle can not be solved in time $O(m^{1.5-\epsilon})$ ◀

We can give a more precise result due to our reduction's tight relationship between $k' = \lg^{O(1)}(k)k$. Note that $2k$-cycle can be reduced to k'-cycle where $k' = \Theta(2k/\lg^{O(1)}(2k))$ then. Given this $2k$-cycle requires $\Omega(m^{c_{k'}-o(1)})$, where $c_{k'} = k'/(k' - \lceil k'/3 \rceil + 1)$. Thus we have that $2k$- cycle requires at least $\Omega(n^{1.5-O(\lg^{O(1)}(k)/k)})$ time.

We would like to note that no super-linear hardness was known for even cycles prior to this. And, here, we have connected the hardness of even cycles to that of Max-3-SAT.

We would further like to note that we can combine our reduction with the more general result of [10], which gets super linear hardness for directed cycle from Max-L-SAT. Max-L-SAT is more believably $2^{n-o(n)}$ hard the larger L is. From Max-L-SAT we continue to show super-linear hardness for undirected $2k$-cycle problems. However, instead of showing $O(m^{1.5-O(1)})$ hardness the result is instead $O(m^{\frac{L}{L-1}-o(1)})$ hardness.

5.3 Combinatorial Even Cycle Lower Bounds From the Combinatorial K-Clique Conjecture

Abboud, Backurs and, Vassilevska Williams present the Combinatorial k-Clique Hypothesis (CKCH) [1]. First, what is a *combinatorial* algorithm? In her survey Vassilevska Williams defines it as follows: *"The desire for more practical algorithms motivates the notion of "combinatorial" algorithms. This notion is not well-defined, however it roughly means that the runtime should have a small constant in the big-O, and that the algorithm is feasibly implementable.[16]"* Generally, combinatorial assumptions assume that matrix multiplication requires $n^{3-o(1)}$ time, that is there is no fast matrix multiplication allowed. Given this context, we now re-produce the Combinatorial k-Clique Hypothesis from [1].

▶ **Definition 14** (Combinatorial k-Clique Hypothesis [1])**.** *Let C be the smallest constant such that a* combinatorial *algorithm running in $O(n^{Ck/3})$ time exists for detecting a k-clique in a n node $O(n^2)$ edge graph, for all sufficiently large constants k.*

The Combinatorial k-Clique Hypothesis states that $C = 3$[1]*.*

More informally, no $O(n^{k(1-\epsilon})$ *combinatorial* algorithm exists for the k-clique problem.

We can once again use the k-clique to k-cycle reduction of Lincoln, Vassilevska Williams and Williams [10]. Their reduction is itself combinatorial.

[1] See the discussion on page 2 of Abboud, Backurs and, Vassilevska Williams [1]. For a short formal statement see Hypothesis 1.3 in Lincoln, Vassilevska Williams and Williams [10].

▶ **Theorem 15** (Combinatorial Sparse k-Cycle Lower Bound [10]). *Detecting a directed odd k-cycle in a graph with n nodes and $m = n^{(k+1)/(k-1)}$ in time $O((nm)^{(1-\epsilon)}) = O(m^{(1-\epsilon)2k/(k+1)}) = O(n^{(1-\epsilon)2k/(k-1)})$ for any constant $\epsilon > 0$ violates the Combinatorial k-Clique Hypothesis* [2].

▶ **Corollary 16.** *Detecting a directed k-cycle in a graph with n nodes and $m = n^{k/(k-2)}$ in time $O(nm^{(1-\epsilon)}) = O(m^{(2k-2)/k})$ for any constant $\epsilon > 0$ violates the Combinatorial k-Clique Hypothesis.*

Proof. There is an immediate reduction from directed k-cycle to directed $k+1$-cycle that adds $O(n)$ edges to the graph to get a lower bound for even k. For odd k we have a better lower bound. ◀

Recall that Dahlgaard et al. have an algorithm for even $2k$-cycle detection that runs in $\tilde{O}\left(m^{2k/(k+1)}\right)$ time [5]. We will show that this algorithm is optimal for even $k > 3$ that are twice an odd number. That is, the half of even numbers represented by $4k+2$ for integer $k \geq 1$.

▶ **Theorem 17.** *Let $2k = (4k'+2)$ for some constant integer k'.*
Detecting $2k$-cycles combinatorially requires time $m^{(1-o_k(1))2k/(k+1)}$ if the Combinatorial k-Clique Hypothesis is true.

Proof. We will start with Theorem 15. Note that $2k'+1$ is odd. Take the directed instance and use color coding to produce a $2k'+1$-partite undirected instance. Furthermore, this graph is a $2k'+1$-cycle graph. That is, partition $P_{(i)}$ only has edges to partitions $P_{(i-1 \mod (k'+1))}$ and $P_{(i+1 \mod (2k'+1))}$.

To solve an undirected odd $2k'+1$-cycle problem in a $2k'+1$-cycle graph we will create an instance of $4k'+2$-cycle by replacing every edge (u,v) with a node, x_{uv} and two edges $(u, x_{u,v})$ and $(x_{u,v}, v)$.

Any $4k'+2$-cycle in this new graph with contain $2k'+1$ x_{uv} type nodes. In fact these x_{uv} type nodes will be every other node. If an x_{uv} type node is in a cycle so are u and v. So, a $4k'+2$ length path will look like $v_1, x_{v_1v_2}, v_2, x_{v_2v_3}, v_3 \ldots, v_{2k'}, x_{v_{2k'}v_{2k'+1}}, v_{2k'+1}$. The existence of a $4k'+2$ length path in our new graph corresponds exactly to there being a $2k'+1$ length path in the original graph.

Theorem 15 tells us that there is a $m^{(4k'+2)/(2k'+2)-o(1)}$ combinatorial lower bound if Combinatorial k-Clique Hypothesis is true. In our new graph m has grown by a factor of only two. If we plug in $2k = 4k'+2$ we get the desired lower bound of $m^{2k/(k+1)-o(1)}$ from Combinatorial k-Clique Hypothesis. ◀

The above result works for specific even cycles i.e. when $2k = 4k'+2$, next we prove a lower bound which holds for $2k$ cycles for all large enough k.

▶ **Corollary 18.** *If there exists a constant $\epsilon > 0$ such that for some $k > \frac{2}{\epsilon}$ directed k-cycle can be detected in $O(m^{2-\epsilon})$ time using a combinatorial algorithm then the Combinatorial k-Clique Hypothesis is violated.*

Proof. For $k > \frac{2}{\epsilon}$, $m^{2-\epsilon} = O(m^{(2k-2)/(k)(1-\delta)})$ for some constant δ. So, by Corollary 16 such an algorithm would violate the Combinatorial k-Clique Hypothesis. ◀

[2] See the discussion after Hypothesis 1.3 in the introduction of Lincoln, Vassilevska Williams and Williams [10].

▶ **Theorem 19.** *Assume the Combinatorial k-Clique Hypothesis. Then, for all constant $\epsilon > 0$ there exits a k_0 such that for all $k \geq k_0$ even $2k$-cycles in an undirected graph with n nodes and $m = O(n)$ edges requires $\Omega(m^{2-\epsilon})$ time.*

Proof. By Theorem 9 there exits a $k' \geq k/(\lg^{O(1)} k)$ such that if directed k' cycle requires $\Omega(m^{2-\epsilon})$ then so does undirected $2k$ cycle on a graph with $O(m)$ vertices and $O(m)$ edges.

As long as $k' = k/(\lg^{O(1)} k) > 2/\epsilon$ we know that directed k' cycle is $\Omega(m^{2-\epsilon})$ hard. As for a large enough k, $k/(\lg^{O(1)} k) > 2/\epsilon$ we have that for large enough k undirected $2k$ cycle on graphs with $m = O(n)$ requires $\Omega(m^{2-\epsilon})$ time. ◀

Note that for all constant k there exist $O(n^2)$ combinatorial algorithms for undirected $2k$-cycle [17]. So, this suggests that these algorithms are optimal, for *combinatorial* algorithms for all large enough k.

6 Conclusion

We present a faster algorithm for small space cycle detection. We also present lower bounds for the efficiency of k-cycle in this small space regime. We show the running time of odd as well as even k-cycle must grow as $n^{\Omega(\lg(k))}$ when given only $O(\lg^2(n))$ space.

There are many future directions of research. We leave open several algorithms questions as well as questions about lower bounds.

While we get tight upper and lower bounds in the regime we consider, there are many open questions about k-cycle outside of this regime. What is the space-time trade-off for these k-cycle algorithms when the space available is polynomial in n? For example, how efficient are algorithms when given $n^{1/8}$ space? Additionally, our results apply to large constant k, but do not imply statements for specific fixed k. What are the fastest small space algorithms for $k = 3, 4, 5\ldots$? Finally, our results are not very *fine-grained*, we are agnostic to the specific constants in the exponents of these algorithms. So, what is the specific constant? Can k-cycle algorithms be solved in small space in $n^{\lg(k)/2}$ time? $n^{\lg(k)/100}$ time?

We also leave open related algorithmic questions about s t connectivity in small space. We present a s t connectivity algorithm in small space which runs in $n^{\lg(n)/2}$ expected time. Can a deterministic algorithm have the same running time? What are the fastest randomized and deterministic algorithms for the s t connectivity problem?

We leave open lower bounds questions as well. Can we find matching, or at least non-trivial, lower bounds for k-cycle algorithms in small polynomial space. Relatedly, we get combinatorial lower bounds for even cycle in arbitrary space. Can one find tight *non-combinatorial* lower bounds for even cycle in arbitrary space?

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the Current Clique Algorithms are Optimal, So is Valiant's Parser. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 98–117, 2015. doi:10.1109/FOCS.2015.16.

2 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and Counting Given Length Cycles. *Algorithmica*, 17(3):209–223, 1997. doi:10.1007/BF02523189.

3 Noga Alon, Raphael Yuster, and Uri Zwick. Color Coding. In *Encyclopedia of Algorithms*, pages 335–338. Springer, 2016. doi:10.1007/978-1-4939-2864-4_76.

4 Greg Barnes, Jonathan F. Buss, Walter L. Ruzzo, and Baruch Schieber. A Sublinear Space, Polynomial Time Algorithm for Directed s-t Connectivity. *SIAM J. Comput.*, 27(5):1273–1282, 1998. doi:10.1137/S0097539793283151.

5 Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Morten Stöckel. Finding even cycles faster via capped k-walks. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 112–120, 2017. doi:10.1145/3055399.3055459.

6 François Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303, 2014. doi:10.1145/2608628.2608664.

7 Parikshit Gopalan, Richard J. Lipton, and Aranyak Mehta. Randomized Time-Space Tradeoffs for Directed Graph Connectivity. In Paritosh K. Pandya and Jaikumar Radhakrishnan, editors, *FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science, 23rd Conference, Mumbai, India, December 15-17, 2003, Proceedings*, volume 2914 of *Lecture Notes in Computer Science*, pages 208–216. Springer, 2003. doi:10.1007/978-3-540-24597-1_18.

8 Richard C. Holt and Edward M. Reingold. On the time required to detect cycles and connectivity in graphs. *Mathematical systems theory*, 6(1):103–106, March 1972. doi:10.1007/BF01706081.

9 Richard M. Karp. Reducibility Among Combinatorial Problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, pages 85–103, 1972. doi:10.1007/978-1-4684-2001-2_9.

10 Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. Tight Hardness for Shortest Cycles and Paths in Sparse Graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1236–1252, 2018.

11 Benjamin Rossman. Formulas vs. circuits for small distance connectivity. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 203–212, 2014.

12 Walter J. Savitch. Relationships Between Nondeterministic and Deterministic Tape Complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970. doi:10.1016/S0022-0000(70)80006-X.

13 Dieter van Melkebeek and Gautam Prakriya. Derandomizing Isolation in Space-Bounded Settings. *SIAM J. Comput.*, 48(3):979–1021, 2019. doi:10.1137/17M1130538.

14 Avi Wigderson. The Complexity of Graph Connectivity. In *Mathematical Foundations of Computer Science 1992, 17th International Symposium, MFCS'92, Prague, Czechoslovakia, August 24-28, 1992, Proceedings*, pages 112–132, 1992.

15 Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 887–898, 2012. doi:10.1145/2213977.2214056.

16 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the International Congress of Mathematicians, page to appear*, 2018.

17 Raphael Yuster and Uri Zwick. Finding Even Cycles Even Faster. *SIAM J. Discrete Math.*, 10(2):209–222, 1997. doi:10.1137/S0895480194274133.

18 Raphael Yuster and Uri Zwick. Detecting Short Directed Cycles Using Rectangular Matrix Multiplication and Dynamic Programming. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, pages 254–260, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=982792.982828.

19 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002. doi:10.1145/567112.567114.

A Basic Lemmas for the Paper

The ideas in various lemmas below are not novel. These are just important working lemmas for the rest of the paper. We cite lemmas we use directly. For those lemmas that are variations on previous work we attempt to reference where the ideas came from.

We will be using color coding to simplify many proofs. To see an example of the output graphs from color coding look at Figure 3.

▶ **Lemma 20** (Color Coding [3]). *Let E be the edge set of the graph G. Given a graph G one can produce $c = \lg(n)^{k^2}$ graphs $G_1, \ldots, G_c$ in $\tilde{O}(|E|)$ time such that:*

- *each graph has k partitions $P_1, \ldots, P_k$ and edges exist only between P_i and $P_{i+1 \mod k}$,*
- *each graph G_i has at most $|E|$ edges and $|V|$ vertices,*
- *if and only if G contains a k-cycle, at least one of the graphs G_i will contains a k-cycle,*
- *and finally if G contains a k-cycle in at least one of the graphs G_i the k cycle will have one node from every partition.*

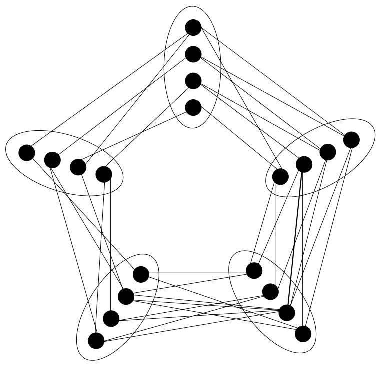

Figure 3 An example of the output of color coding with five colors. There are five partitions and the partions are themselves in a cycle.

▶ **Lemma 21.** *An algorithm exists to detect if a directed k-cycle exists using the edge (s, t) in a graph G with n nodes and $O(n^2)$ edges in time $n^{O(\lg(k))}$ using space $O(\lg(k)\lg(n))$.*

Proof. If a non-colorful cycle we start by using the color coding Lemma 20, to produce graphs with k partitions such that our k-cycle, if it exists, uses one node from each partition in one of these graphs. We consider only the color coded graphs where s and t are in partitions V_1 and V_k respectively. If we start with colorful k-cycle then we simply use the colors themselves for color coding.

Now, if we could detect if a k-path exists between s and t which uses one node from every partition we are done. To do this we will guess, in sequence, each of the n possible choices of nodes that could be in the middle partition. Then, we will recursively solve this same colorful path problem on both shorter paths. The running time of this algorithm is given by the following recurrence with base case $T(n, 2) = n^{O(1)}$:

$$T(n, k) = n2T(n, k/2) + O(1).$$

Solving this recurrence gives us $T(n, k) = 2^{\lg(k)} n^{\lg(k)} n^{O(1)}$. When simplified we find $T(n, k) = n^{\lg(k)+o(\lg(k))} = n^{O(\lg(k))}$. This gives us the desired result.

Keeping these guesses in memory requires $\lg(n)$ bits per guess of a node and we must track $O(\lg(k))$ guesses at a time. This gives us $O(\lg(k)\lg(n))$ space. ◀

The above lemma is using the same ideas as Savitch [12].