Tight Hardness for Shortest Cycles and Paths in Sparse Graphs

Andrea Lincoln[*]   Virginia Vassilevska Williams[†]   Ryan Williams[‡]

Abstract

Fine-grained reductions have established equivalences between many core problems with $\tilde{O}(n^3)$-time algorithms on n-node weighted graphs, such as Shortest Cycle, All-Pairs Shortest Paths (APSP), Radius, Replacement Paths, Second Shortest Paths, and so on. These problems also have $\tilde{O}(mn)$-time algorithms on m-edge n-node weighted graphs, and such algorithms have wider applicability. Are these mn bounds optimal when $m \ll n^2$?

Starting from the hypothesis that the minimum weight $(2\ell+1)$-Clique problem in edge weighted graphs requires $n^{2\ell+1-o(1)}$ time, we prove that for all sparsities of the form $m = \Theta(n^{1+1/\ell})$, there is no $O(n^2 + mn^{1-\varepsilon})$ time algorithm for $\varepsilon > 0$ for *any* of the below problems

- Minimum Weight $(2\ell+1)$-Cycle in a directed weighted graph,
- Shortest Cycle in a directed weighted graph,
- APSP in a directed or undirected weighted graph,
- Radius (or Eccentricities) in a directed or undirected weighted graph,
- Wiener index of a directed or undirected weighted graph,
- Replacement Paths in a directed weighted graph,
- Second Shortest Path in a directed weighted graph,
- Betweenness Centrality of a given node in a directed weighted graph.

That is, we prove hardness for a variety of sparse graph problems from the hardness of a dense graph problem. Our results also lead to new conditional lower bounds from several related hypothesis for unweighted sparse graph problems including k-cycle, shortest cycle, Radius, Wiener index and APSP.

1 Introduction

The All-Pairs Shortest Paths (APSP) problem is among the most basic computational problems. A powerful primitive, APSP can be used to solve many other problems on graphs (e.g. graph parameters such as the girth or the radius), but also many non-graph problems such as finding a subarray of maximum sum [TT98] or parsing stochastic context free grammars (e.g. [Aku99]). Over the years, many APSP algorithms have been developed. For edge weighted n node, m edge graphs, the fastest known algorithms run in $n^3/2^{\Theta(\sqrt{\log n})}$ time [Wil14] for dense graphs, and in $O(mn + n^2 \log \log n)$ time [Pet02] for sparse graphs.

These running times are also essentially the best known for many of the problems that APSP can solve: Shortest Cycle, Radius, Median, Eccentricities, Second Shortest Paths, Replacement Paths, and so on[1]. For dense graphs, this was explained by Vassilevska Williams and Williams [VW10] and later Abboud et al. [AGV15] who showed that either all of {APSP, Minimum Weight Triangle, Shortest Cycle, Radius, Median, Eccentricities, Second Shortest Paths, Replacement Paths, Betweenness Centrality} have *truly subcubic* algorithms (with runtime $O(n^{3-\varepsilon})$ for constant $\varepsilon > 0$), or none of them do. Together with the popular hypothesis that APSP requires $n^{3-o(1)}$ time on a word-RAM (see e.g. [AV14, AGV15, Vas15, BGMW17]), these equivalences suggest that all these graph problems require $n^{3-o(1)}$ time to solve.

However, these equivalences no longer seem to hold for sparse graphs. The running times for these problems still match: $\tilde{O}(mn)$ is the best running time known for all of these problems. In recent work, Agarwal and Ramachandran [AR16] show that some reductions from prior work can be modified to preserve sparsity. Their main result is that if Shortest Cycle in directed weighted graphs requires $m^a n^{2-a-o(1)}$ time for some constant a, then so do Radius, Eccentricities, Second Shortest Paths, Replacement Paths, Betweenness Centrality and APSP in directed weighted graphs[2].

---

[1]For Shortest Cycle, an $O(mn)$ time algorithm was recently developed by Orlin and Sedeño-Noda [OS17]. For a full discussion of the best known running times of these problems see Appendix A.

[2]This is analogous to the dense graph regime of [VW10], where the main reductions went from Minimum Weight 3-Cycle (i.e. triangle). The

Unfortunately, there is no known reduction that preserves sparsity from APSP (or any of the other problems) back to Shortest Cycle, and there are no known reductions to Shortest Cycle from any other problems used as a basis for hardness within Fine-Grained Complexity, such as the Strong Exponential Time Hypothesis [IP01, IPZ01], 3SUM [GO95] or Orthogonal Vectors [Wil05, Vas15]. Without a convincing reduction, one might wonder:

Can Shortest Cycle in weighted directed graphs be solved in, say, $\tilde{O}(m^{3/2})$ time?
Can APSP be solved in $\tilde{O}(m^{3/2}+n^2)$ time?

Such runtimes are consistent with the dense regime of $m = \Theta(n^2)$. Minimum Weight Triangle, which is the basis of many reductions in the dense case, can be solved in $O(m^{3/2})$ time (e.g. [IR78]). What prevents us from having such running times for all the problems that are equivalent in the dense regime to Minimum Weight Triangle? Why do our best algorithms for these other problems take $\tilde{O}(mn)$ time, and no faster? In fact, we know of no $\varepsilon > 0$ for which problems like Shortest Cycle can be solved in $\tilde{O}(m^{1+\varepsilon}n^{1-2\varepsilon})$ time. Such a running time is $\Theta(n^3)$ for $m = \Theta(n^2)$ and $o(mn)$ for $m \leq o(n^2)$. Notice that $m^{3/2}$ is the special case for $\varepsilon = 1/2$. Is there a good reason why no $\tilde{O}(m^{1+\varepsilon}n^{1-2\varepsilon})$ time algorithms have been found?

Our results. We give compelling reasons for the difficulty of improving over $\tilde{O}(mn)$ for Shortest Cycle, APSP and other problems. We show for an infinite number of sparsities, any sparsity $m = n^{1+1/\ell}$ where $\ell \in \mathbb{N}$, obtaining an $O(n^2 + mn^{1-\varepsilon})$ time algorithm for Shortest Cycle (or any of the other fundamental problems) in weighted graphs for any constant $\varepsilon > 0$ would refute a popular hypothesis about the complexity of weighted k-Clique.

HYPOTHESIS 1.1. (MIN WEIGHT k-CLIQUE) *There is a constant $c > 1$ such that, on a Word-RAM with $O(\log n)$-bit words, finding a k-Clique of minimum total edge weight in an n-node graph with nonnegative integer edge weights in $[1, n^{ck}]$ requires $n^{k-o(1)}$ time.*

The Min Weight k-Clique Hypothesis has been considered for instance in [BT16] and [AVW14] to show hardness for improving upon the Viterbi algorithm, and for Local Sequence Alignment. The (unweighted) k-Clique problem is NP-Complete, but can be solved in $O(n^{\omega k/3})$ time when k is fixed [NP85][3] where $\omega < 2.373$ [Vas12, Gal14] is the matrix multiplication exponent. The problem is W[1]-complete and under the Exponential Time Hypothesis [IP01] it cannot be solved in $n^{o(k)}$ time. Finding a k-Clique of minimum

total weight (a Min Weight k-Clique) in an edge-weighted graph can also be solved in $O(n^{\omega k/3})$ if the edge weights are small enough. However, when the edge weights are integers larger than n^{ck} for large enough constant c, the fastest known algorithm for Min Weight k-Clique runs in essentially $O(n^k)$ time (ignoring $n^{o(1)}$ improvements). The special case $k = 3$, Min Weight 3-Clique is the aforementioned Minimum Weight Triangle problem which is equivalent to APSP under subcubic reductions and is believed to require $n^{3-o(1)}$ time.

Theorems 1.1 and F.4 of Vassilevska Williams and Williams [VW10], and Theorem 1.1 and Lemma 2.2 of Abboud et al. [AGV15] give sub-cubic dense reductions from APSP to many fundamental graph problems. Agarwal and Ramachandran [AR16] build on these reductions to show many sparsity-preserving reductions from Shortest Cycle to various fundamental graph problems.

They thus identify Shortest Cycle as a fundamental bottleneck to improving upon mn for many problems. However, so far there is no compelling reason why Shortest Cycle itself should need mn time.

THEOREM 1.1. ([AR16]) *Suppose that there is a constant $\varepsilon > 0$ such that one of the following problems on n-node, m-edge weighted graphs can be solved in $O(mn^{1-\varepsilon}+n^2)$ time:*

- *APSP in a directed weighted graph,*
- *Radius (or Eccentricities) in a directed weighted graph,*
- *Replacement Paths in a directed weighted graph,*
- *Second Shortest Path in a directed weighted graph,*
- *Betweenness Centrality of a given node in a directed weighted graph.*

Then, the Min Weight Cycle Problem is solvable in $O(mn^{1-\varepsilon'}+n^2)$ for some $\varepsilon' > 0$ time.

Our main technical contribution connects the complexity of small cliques in dense graphs to that of small cycles in sparse graphs:

THEOREM 1.2. *Suppose that there is an integer $\ell \geq 1$ and a constant $\varepsilon > 0$ such that one of the following problems on n-node, $m = \Theta(n^{1+1/\ell})$-edge weighted graphs can be solved in $O(mn^{1-\varepsilon}+n^2)$ time:*

- *Minimum Weight $(2\ell+1)$-Cycle in a directed weighted graph,*
- *Shortest Cycle in a directed weighted graph,*

Then, the Min Weight $(2\ell+1)$-Clique Hypothesis is false.

Combining our main Theorem 1.2 with the results from previous work in Theorem 1.1 gives us new conditional lower bounds for fundamental graph problems. We also create novel reductions from the k-Cycle problem, in Section B, and these give us novel hardness results for many new problems. The main new contributions are reductions to Radius

---

key point of [AR16] is that one can replace Minimum Weight 3-Cycle by Minimum Weight Cycle, and preserve the sparsity in the reduction.

[3] When k is divisible by 3; slightly slower otherwise.

in undirected graphs (the result in [AR16] is only for directed) and to the Wiener Index problem which asks for the sum of all distances in the graph. Together all these pieces give us the following theorem.

THEOREM 1.3. *Suppose that there is an integer $\ell \geq 1$ and a constant $\varepsilon > 0$ such that one of the following problems on n-node, $m = \Theta(n^{1+1/\ell})$-edge weighted graphs can be solved in $O(mn^{1-\varepsilon} + n^2)$ time:*

- *Minimum Weight $(2\ell+1)$-Cycle in a directed weighted graph,*
- *Shortest Cycle in a directed weighted graph,*
- *APSP in a directed or undirected weighted graph,*
- *Radius (or Eccentricities) in a directed or undirected weighted graph,*
- *Wiener index of a directed or undirected weighted graph,*
- *Replacement Paths in a directed weighted graph,*
- *Second Shortest Path in a directed weighted graph,*
- *Betweenness Centrality of a given node in a directed weighted graph.*

Then, the Min Weight $(2\ell+1)$-Clique Hypothesis is false.

So, either min weighted cliques can be found faster, or $\tilde{O}(mn + n^2)$ is the *optimal* running time for these problems, up to $n^{o(1)}$ factors, for an infinite family of edge sparsities $m(n)$. See Figure 7 in the Appendix for a pictorial representation of our conditional lower bounds.

Another intriguing consequence of Theorem 1.3 is that, assuming Min Weight Clique is hard, running times of the form $\tilde{O}(m^{1+\varepsilon}n^{1-2\varepsilon})$ for $\varepsilon > 0$ are impossible! If Shortest Cycle had such an algorithm for $\varepsilon > 0$, then for every integer $L > 1$ and $\delta = (1 - 1/L)\varepsilon > 0$ we have that for $m = \Theta(n^{1+1/L})$, $m^{1+\varepsilon}n^{1-2\varepsilon} \leq mn^{1-\delta}$ and hence the Min Weight $(2L+1)$-Clique Hypothesis is false.

Our reduction from Minimum Weight $(2L+1)$-Clique to Minimum Weight $(2L+1)$-Cycle produces a directed graph on n^L nodes and $m = O(n^{L+1})$ edges, and hence if directed Minimum Weight $(2L+1)$-Cycle can be solved in $O(m^{2-1/(L+1)-\varepsilon})$ time for some $\varepsilon > 0$, then the Min Weight $(2L+1)$-Clique Hypothesis is false. We present an extension for weighted cycles of even length as well, obtaining:

COROLLARY 1.1. *If Minimum Weight k-Cycle in directed m-edge graphs is solvable in $O(m^{2-2/(k+1)-\varepsilon})$ time for some $\varepsilon > 0$ for k odd, or in $O(m^{2-2/k-\varepsilon})$ time for k even, then the Minimum Weight ℓ-Clique Hypothesis is false for $\ell = 2\lceil k/2 \rceil - 1$.*

Directed k-cycles in unweighted graphs were studied by Alon, Yuster and Zwick [AYZ97] who gave algorithms with a runtime of $O(m^{2-2/(k+1)})$ for k odd, and $O(m^{2-2/k})$ for k even. We show that their algorithm can be extended to find Minimum Weight k-Cycles with only a polylogarithmic overhead, proving that the above conditional lower bound is tight.

THEOREM 1.4. *The Minimum Weight k-Cycle in directed m-edge graphs can be solved in $\tilde{O}(m^{2-2/(k+1)})$ time for k odd, and in $\tilde{O}(m^{2-2/k})$ time for k even.*

Our lower bound results compared to prior work are presented in Table 1. The upper bounds for the considered problems are as follows: Min k-Clique is easily solvable in $O(n^k)$ time. The best algorithms for all other problems in the table take $\tilde{O}(mn)$ time [AYZ16, OS17, Pet02, Dij59, GL09].

Sparse Unweighted Problems. We have proven tight conditional lower bounds for weighted graphs. However, for sparse enough ($m = O(n^{\omega-1})$) *unweighted* graphs, the best algorithms for APSP and its relatives also run in $\tilde{O}(mn)$ time (see Section A for the relevant prior work on APSP). We hence turn our attention to the unweighted versions of these problems.

Our reduction from Min Weight k-Clique to Min Weight k-Cycle still works for unweighted graphs just by disregarding the weights. We can get super-linear lower bounds for sparse unweighted problems from three different plausible assumptions.

As mentioned before, the fastest algorithm for k-Clique (for k divisible by 3) runs in $O(n^{\omega k/3})$ [NP85, EG04]. This algorithm has remained unchallenged for many decades and lead to the following hypothesis (see e.g. [ABV15]).

HYPOTHESIS 1.2. (THE k-CLIQUE HYPOTHESIS)
Detecting a k-Clique in a graph with n nodes requires $\Omega(n^{\omega k/3 - o(1)})$ time on a Word RAM.

From this we get super-linear lower bound for the shortest cycle problem. We get an analogous result to the one we had before:

THEOREM 1.5. *If the k-Clique Hypothesis is true, Shortest Cycle in undirected or directed graphs requires $m^{2\omega/3 - o(1)}$.*

We get super-linear lower bounds for various graph problems as a corollary of Theorem 1.5.

COROLLARY 1.2. *If the k-Clique Hypothesis is true, the following problems in unweighted graphs require $m^{2\omega/3 - o(1)}$ time:*

- *Betweenness Centrality in a directed graph,*
- *APSP in an undirected or directed graph,*
- *Radius in an undirected or directed graphs, n*
- *Wiener Index in an undirected or directed graph.*

Weighted Problem	Lower Bnd	LB from	LB source
Min k-clique	$n^{k-o(1)}$	k-clique conj.	By Def
Min k-cycle (k odd)	$mn^{1-o(1)}$	**Min k-clique**	**Thm 4.1**
Shortest cycle	$mn^{1-o(1)}$	**odd min k-cycle**	**Thm 6.1**
Directed Radius	$mn^{1-o(1)}$	Shortest Cycle	[AR16]
Undirected Radius	$mn^{1-o(1)}$	**odd min k-cycle**	**Thm B.1**
Directed APSP	$mn^{1-o(1)}$	Shortest Cycle	[AR16]
Undirected APSP	$mn^{1-o(1)}$	**odd min k-cycle**	**Cor B.2**
Undirected APSP	$mn^{1-o(1)}$	Und. Shortest Cycle	[AR16]
Und. Wiener Index	$mn^{1-o(1)}$	**odd min k-cycle**	**Thm B.2**
Dir. 2nd Shortest Path	$mn^{1-o(1)}$	Shortest Cycle	[AR16]
Dir. Repl. Paths	$mn^{1-o(1)}$	Shortest Cycle	[AR16]

Table 1: Weighted graph lower bounds. Our results are in bold. Und stands for undirected and Dir stands for directed. Repl stands for replacement.
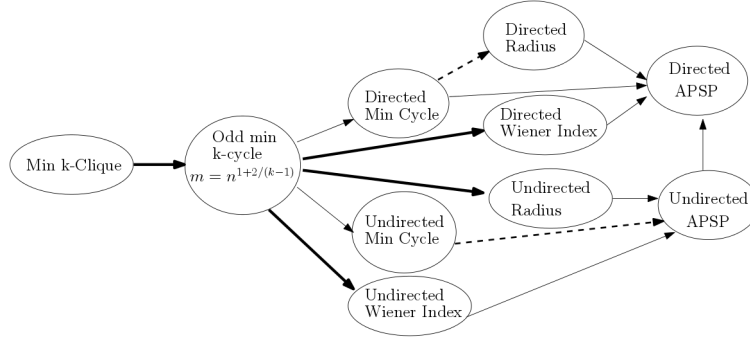

Figure 1: A depiction of a network of reductions related to sparse weighted graph problems and the dense Minimum k-clique problem. The bold edges represent a subset of the reductions in this paper. The green edges are reductions from Agarwal and Ramachandran [AR16].

The reader may notice that the matrix multiplication exponent shows up repeatedly in the unweighted cases of these problems. This is no coincidence. The best known combinatorial[4] algorithms for the unweighted k-clique algorithm take $\tilde{\Theta}(n^k)$ time. This has lead to a new hypothesis.

HYPOTHESIS 1.3. (COMBINATORIAL k-CLIQUE) *Any combinatorial algorithm to detect a k-Clique in a graph with n nodes requires $n^{k-o(1)}$ time on a Word RAM [ABV15].*

Our reduction from k-clique to k-cycle is combinatorial. Thus, an $O(mn^{1-\varepsilon})$ time (for $\varepsilon > 0$) combinatorial algorithm for the directed unweighted k-cycle problem for odd k and $m = n^{(k+1)/(k-1)}$ would imply a combinatorial algorithm for the k-clique problem with running time $O(n^{k-\varepsilon'})$ for $\varepsilon' > 0$. Any algorithm with a competitive running time must use fast matrix multiplication, or give an exciting new algorithm for k-clique.

Currently, the best bound on ω is $\omega < 2.373$ [Gal14, Vas12], and the $m^{2\omega/3-o(1)}$ lower bound for Shortest Cycle and related problems might conceivably be $m^{1.58}$ which is not far from the best known running time $O(m^{1.63})$ for 5-cycle [YZ04]. Yuster and Zwick gave an algorithm based on matrix multiplication for directed k-Cycle, however they were unable to analyze its running time for $k > 5$. They conjecture that if $\omega = 2$, their algorithm runs faster than the best combinatorial algorithms for every k, however even the conjectured runtime goes to m^2, as k grows. In contrast, for $\omega = 2$, our lower bound based on the k-Clique Hypothesis is only $m^{4/3-o(1)}$[5]. We thus search for higher conditional lower bounds based on different and at least as believable hypotheses.

To this end, we formalize a working hypothesis about the complexity of finding a hyperclique in a hypergraph. An ℓ-hyperclique in a k-uniform hypergraph G is composed of a

---

[4]Informally, combinatorial algorithms are algorithms that do not use fast matrix multiplication.

[5]Of course, the Yuster and Zwick running time for k-cycle might not be optimal, and it might be that $O(m^{2-\delta})$ time is possible for k-cycle for some $\delta > 0$ and all k.

set of ℓ nodes of G such that all k-tuples of them form a hyperedge in G.

HYPOTHESIS 1.4. ((ℓ,k)-HYPERCLIQUE HYPOTHESIS)
Let $\ell > k > 2$ be integers. On a Word-RAM with $O(\log n)$ bit words, finding an ℓ-hyperclique in a k-uniform hypergraph on n nodes requires $n^{\ell - o(1)}$ time.

Why should one believe the hyperclique hypothesis? There are many reasons: (1) When $k > 2$, there is no $O(n^{\ell - \varepsilon})$ time algorithm for any $\varepsilon > 0$ for ℓ-hyperclique in k uniform hypergraphs. (2) The natural extension of the techniques used to solve ℓ-clique in graphs will *NOT* solve ℓ-hyperclique in k-uniform hypergraphs in $O(n^{\ell - \varepsilon})$ time for any $\varepsilon > 0$ when $k > 2$. We prove this in Section 8. (3) There are known reductions from notoriously difficult problems such as Exact Weight k-Clique, Max k SAT and even harder Constrained Satisfaction Problems (CSPs) to (ℓ,k)-Hyperclique so that if the hypothesis is false, then all of these problems have exciting improved algorithms. For these and more, see the Discussion in Section 7.

Now, let us state our results for unweighted Shortest Cycle based on the (ℓ,k)-Hypothesis. The same lower bounds apply to the other problems of consideration (APSP, Radius etc.).

THEOREM 1.6. *Under the (ℓ,k)-Hypothesis, the Shortest Cycle problem in directed unweighted graphs requires $m^{k/(k-1)-o(1)}$ time on a Word RAM with $O(\log n)$ bit words.*

The theorem implies in particular that, Shortest Cycle in unweighted directed graphs requires (a) $m^{3/2 - o(1)}$ time, unless Max 3-SAT (and other CSPs) have faster than 2^n algorithms, (b) $m^{4/3 - o(1)}$ time, unless Exact Weight k-Clique has a significantly faster than n^k algorithm. The latter is the same lower bound as from k-Clique when $\omega = 2$ but it is from a different and potentially more believable hypothesis. Finally, Shortest Cycle and its relatives are not in linear time, unless the (ℓ,k)-Hypothesis is false for every constant k.

Our new lower bounds for sparse unweighted graph problems are summarized in Table 2. Odd k-cycle is conjectured to run in time $\tilde{O}(m^{(k+1)\omega/(2\omega+k+1)})$ by [YZ04]. The fastest algorithms for all other problems in the table run in time $O(\min\{mn, n^\omega\})$ [IR78, Sei95].

Overview See Figure 1 for a depiction of our core reductions.

In Sections 3 to 6 we cover the core reductions and show they are tight to the best known algorithms. The reduction from hyperclique to hypercycle is covered in Section 3. The reduction from hypercycle to directed cycle in Section 4. The algorithms for weighted minimum k-cycle which match the conditional lower bounds are discussed in Section 5. The

reduction from minimum weight clique to shortest cycle is in Section 6.

In Sections 7 to 9 we give justification for the hardness of the unweighted versions of these problems. In Section 7 we discuss the hyper-clique hypothesis and give justification for it. In Section 8 we show that the generalized matrix product related to finding hypercliques in k-uniform hypergraphs can not be sped up with a Strassen like technique. In Section 9 we reduce Max-k-SAT to Tight Hypercycle.

In Appendix A we discuss the prior work getting fast algorithms for the sparse graph problems we study. In Appendix B we present the reductions from minimum k-cycle and minimum cycle to Radius, Weiner Index and APSP. In Appendix C we reduce general CSP to the Hyperclique problem. Finally, in Appendix D we extend our lower bounds to make improved but non-matching lower bounds for graph densities between $n^{1+1/\ell}$ and $n^{1+1/(\bar{\ell}+1)}$.

2 Preliminaries

In this section we define various notions that we will be using and prove some simple lemmas.

Definitions and notation. Throughout this paper will be discussing problems indexed by k and ℓ. For example, k-cycle, k-clique, (ℓ,k)-Hyperclique. We will treat the k and ℓ values as being constant in these problems. A *hypergraph* $G = (V, E)$ is defined by its vertices V and its hyperedges E where each $e \in E$ is a subset of V. G is a k-uniform hypergraph if all its hyperedges are of size k.

Graphs are just 2-uniform hypergraphs. Unless otherwise stated, the variables m and n will refer to the number of hyperedges and vertices of the hypergraph in question. Unless otherwise stated, the graphs in this paper will be directed. Hypergraphs will not be directed. We will use node and vertex interchangeably.

An *ℓ-hypercycle* in a k-uniform hypergraph is an ordered ℓ-tuple of vertices $v_1, \ldots, v_\ell$ such that for every $i \in 1, \ldots, \ell$, $(v_i, v_{i+1}, \ldots, v_{i+k-1})$ is a hyperedge (where the indices are mod k).

We will be dealing with simple hypercycles, so that all v_i are distinct. These types of hypercycles are known as tight hypercycles. We will omit the term tight for conciseness.

An *ℓ-hyperclique* in a k-uniform hypergraph is a set of $\ell > k$ vertices $v_1, \ldots, v_\ell$ such that all subsets of k of them $v_{i1}, \ldots, v_{ik}$ form a hyperedge.

A *k-circle-layered graph* is a k-partite directed graph G where edges only exist between adjacent partitions. More formally the vertices of G can be partitioned into k groups such that $V = V_1 \cup \ldots \cup V_k$ and $V_i \cap V_j = \emptyset$ if $i \neq j$. The only edges from a partition V_i go to the partition $V_{i+1 \mod k}$.

Hardness Hypotheses. We will state several hardness hypotheses that we will be using.

Problem	Lower Bound	LB from	LB src
k-cycle odd	$m^{(k-\lceil (k+1)/3 \rceil +1)/k - o(1)}$	**Max 3-SAT**	**Thm 9.1**
k-cycle odd	$m^{(2\omega k)/(3(k+1)) - o(1)}$	**k-clique**	**Lem 4.1**
Shrt. cycle	$m^{2\omega/3 - o(1)}$	**k-clique**	**Lem 6.1**
Shrt. cycle	$m^{3/2 - o(1)}$	**Max 3-SAT**	**Lem 6.1**
U. Radius	$m^{2\omega/3 - o(1)}$	**k-clique**	**Lem B.2**
U. Radius	$m^{3/2 - o(1)}$	**Max 3-SAT**	**Lem B.2**
U. Wiener Ind.	$m^{2\omega/3 - o(1)}$	**k-clique**	**Lem B.3**
U. Wiener Ind.	$m^{3/2 - o(1)}$	**Max 3-SAT**	**Lem B.3**
U. APSP	$m^{2\omega/3 - o(1)}$	**k-clique**	**Cor B.2**
U. APSP	$m^{3/2 - o(1)}$	**Max 3-SAT**	**Cor B.2**

Table 2: Unweighted graph lower bounds. Our results are in bold. Upper bounds marked with $*$ are conjectured. U stands for undirected. Src stands for source. Shrt stands for shortest. Ind stands for index.

The first concerns the Min Weight k-Clique problem. Min Weight 3-Clique is known to be equivalent to APSP and other problems [VW10], and no truly subcubic algorithms are known for the problem. This issue extends to larger cliques: if the edge weights are large enough, no significantly faster algorithms than the brute-force algorithm are known. This motivates the following hypothesis used as the basis of hardness in prior work (see e.g. [BT16, AVW14]).

REMINDER OF HYPOTHESIS 1.1 (Min Weight k-Clique Hypothesis). *There is a constant c such that, on a Word-RAM with $O(\log n)$ bit words, finding a k-Clique of minimum total edge weight in an n-node graph with nonnegative integer edge weights bounded by n^{ck} requires $n^{k-o(1)}$ time.*

The exact weight version of the k-clique problem is at least as hard as Min Weight k-Clique [VW13], so that if the previous hypothesis is true, then so is the following one. For $k = 3$, the Exact 3-Clique problem is known to be at least as hard as both APSP and 3-SUM, making the following hypothesis even more believable.

HYPOTHESIS 2.1. (EXACT WEIGHT k-CLIQUE) *There is a constant c such that, on a Word-RAM with $O(\log n)$ bit words, finding a k-Clique of total edge weight exactly 0, in an n-node graph with integer edge weights bounded in $[-n^{ck}, n^{ck}]$ requires $n^{k-o(1)}$ time.*

Let $k \geq 3$ be an integer. The following hypothesis concerns the Max-k-SAT problem. The brute-force algorithm for Max-k-SAT on n variables and m clauses runs in $O(2^n m)$ time. There have been algorithmic improvements for the approximation of Max-k-SAT [AW02, ABZ05, FG95] and Max-2-SAT [Wil07]. No $O(2^{(1-\varepsilon)n})$ time algorithms are known for any $\varepsilon > 0$ for $k \geq 3$. Williams [Wil05, Wil07] showed that Max-2-SAT does have a faster algorithm running in $O(2^{\omega n/3} \cdot \text{poly}(mn))$ time, however the algorithm used can not extend to Max k SAT for $k > 2$ (see the discussion in Section 8).

HYPOTHESIS 2.2. (MAX-k-SAT HYPOTHESIS) *On a Word-RAM with $O(\log n)$ bit words, given a k-CNF formula on n variables, finding a Boolean assignment to the variables that satisfies a maximum number of clauses, requires $2^{n-o(n)}$ time.*

The Max-k-SAT hypothesis implies the following hypothesis about hyperclique detection, as shown by Williams [Wil07] for $k = 3$ (see Appendix for the generalization for $k > 3$). Williams [Wil07] in fact showed that hyperclique detection solves even more difficult problems such as Satisfiability of Constraint Satisfaction Problems, the constraints of which are given by degree k polynomials defining Boolean functions on the n variables. Thus if the following hypothesis is false, then more complex MAX-CSP problems than MAX-k-SAT can be solved in $O(2^{(1-\varepsilon)n})$ time for $\varepsilon > 0$.

REMINDER OF HYPOTHESIS 1.4 ((ℓ, k)-Hyperclique Hypothesis). *Let $\ell > k > 2$ be integers. On a Word-RAM with $O(\log n)$ bit words, finding an ℓ-hyperclique in a k-uniform hypergraph on n nodes requires $n^{\ell-o(1)}$ time.*

Abboud et al. [ABDN17] have shown (using techniques from [ALW14]) that if the ($\ell, 4$)-Hyperclique Hypothesis is false for some ℓ, then the Exact Weight ℓ-Clique Hypothesis is also false. Thus, the Hyperclique Hypothesis should be very believable even for $k = 4$. The hypergraphs we are considering are dense ($m = \Theta(n^k)$). Hyperclique can be solved faster in hypergraphs where $m = o(n^{\ell/(\ell-1)})$ [GIKW17].

Simple k-Cycle Reductions. Note that throughout this paper we will use the fact that the k-cycle and k-clique problems we consider are as hard in k-partite graphs as they are in general graphs. Furthermore, the k-cycle problems we consider are as hard in k-circle layered graphs as they are in general graphs. Using the k-partite or k-circle layered versions often makes reductions more legible.

k-cycle has different behavior when k is even and odd. To get some results we will use a simple reduction from k cycle to $k+1$ cycle.

LEMMA 2.1. *Let $G = (V,E)$ be an n node m edge k-circle-layered graph. Suppose further that the edges have integer weights in $\{-M,\ldots,M\}$. Then in $O(m+n)$ time one can construct a $k+1$-partite directed graph G' on $\leq 2n$ nodes and $\leq n+m$ edges with weights in $\{-M,\ldots,M\}$, so that G' contains a directed $(k+1)$-cycle of weight Y if and only if G contains a directed k-cycle of weight Y.*

Proof. Take V_2, say, and split every node $v \in V_2$ into v_0 and v_1, placing a directed edge (v_0,v_1) of weight 0 and splitting the edges incident to v among v_0 and v_1, so that v_0 gets all edges incoming from V_1 and v_1 gets all edges outgoing to V_2.

An immediate corollary is:

COROLLARY 2.1. *Suppose that there is a $T(n,m)$-time algorithm that can detect a (min-weight/ 0-weight/ unweighted) $k+1$-cycle in a $k+1$-circle-layered directed n-node, m-edge graph, then there is a $O(m+n)+T(2n,m+n)$ time algorithm that can detect a (min-weight/ 0-weight/ unweighted) k-cycle in a k-circle-layered n-node, m-edge directed graph.*

The following Lemma allows us to assume that all graphs that we are dealing with are circle-layered.

LEMMA 2.2. *Suppose that a (min-weight/ 0-weight/ unweighted) k-cycle can be detected in $T(n,m)$ time in a k-circle-layered directed graph where the edges have integer weights in $\{-W,\ldots,W\}$. Then in $\tilde{O}(k^k(m+n+T(m,n)))$ time one can detect a(min-weight/ 0-weight/ unweighted) k-cycle in a directed graph G (not necessarily k-circle-layered) on n nodes and m edges with weights in $\{-W,\ldots,W\}$.*

Proof. We use the method of color-coding [AYZ16]. We present the randomized version, but this can all be derandomized using k-perfect families of hash functions, resulting in roughly the same runtime. Every node in the graph selects a color from $\{1,\ldots,k\}$ independently uniformly at random. We take the original graph and we only keep an edge (u,v) if $c(v) = c(u)+1 \mod k$ and we remove edges that do not satisfy this condition. The created subgraph G' is k-partite - there is a partition for each color, and by construction, the edges only go between adjacent colors, so that the graph is k-circle-layered.

Since G' is a subgraph of G, if G' has a k-cycle C, then C is also a k-cycle in G. Suppose now that G has a k-cycle $C = \{u_1,\ldots,u_k\}$. If for each i, $c(u_i) = i$, then C is preserved in G'. Thus, C is preserved with probability at least $1/k^k$, and repeating $O(k^k \log n)$ times, we will find C whp.

3 Reduction from Hyperclique to Hypercycle

In this section we will reduce the problem of finding an ℓ-hyperclique in a k-uniform hypergraph to finding an ℓ-hypercycle in a $\gamma(\ell,k)$-uniform hypergraph for some function γ which is roughly $(k-1)\ell/k$.

By a color-coding argument we can assume that the hypergraph is k-partite- the vertex set is partitioned into k parts $\{V_i\}$ so that no hyperedge contains two nodes in the same V_i. The color-coding approach reduces the hyperclique problem to $2^{O(k)} \log n$ instances of the k-partite hyperclique problem. A simple randomized approach assigns each vertex a random color from $\{1,\ldots,k\}$, and then part V_i includes the vertices colored i. One removes all hyperedges containing two vertices colored the same and argues that any particular k-hyperclique has all its vertices colored differently with probability $1/(2e)^k$. Thus $2^{O(k)} \log n$ instances of the k-partite hyperclique problem suffice with high probability. The approach can be derandomized with standard techniques.

In the following theorem an arc will refer to a valid partial list of nodes from a hyperclique or hypercycle. This usage is attempting to get across the intuition that a set of nodes in a hyperclique can be covered by a small number of overlapping sets if those sets are large enough. See Figure 2 for an image depiction.

We will hence prove the following theorem:

THEOREM 3.1. *Let G be a k-uniform hypergraph on n vertices V, partitioned into ℓ parts $V_1,\ldots,V_\ell$. Let $\gamma = \ell - \lceil \ell/k \rceil + 1$. In $O(n^\gamma)$ time we can create a γ-uniform hypergraph G' on the same node set V as G, so that G' contains an ℓ-hypercycle if and only if G contains an ℓ-hyperclique with one node from each V_i.*

If G has weights on its hyperedges in the range $[-W,W]$, then one can also assign weights to the hyperedges of G' so that a minimum weight ℓ-hypercycle in G' corresponds to a minimum weight ℓ-hyperclique in G and every edge in the hyperclique has weight between $[-\binom{\gamma}{k}W, \binom{\gamma}{k}W]$. Notably, $\binom{\gamma}{k} \leq O(\ell^k)$.

Proof. Consider the numbers $1,\ldots,\ell$ written in order around a circle and let $i_1 < i_2 < \ldots < i_k$ be any k of them. We are interested in covering all these k numbers by an arc of the circle. What is the least number of numbers from 1 to ℓ an arc covers if it covers all the i_j?

It's not hard to see that the arc starts at one of the i_j, goes clockwise and ends at i_{j-1} (indices mod k). Let $s(j)$ be the number of numbers strictly between i_{j-1} an i_j. The number of numbers that the arc contains is thus $\ell - s(j)$, and that the best arc picks the j that maximizes $s(j)$.

The sum $\sum_{j=1}^{k} s(j)$ equals $\ell - k$, and hence the maximum $s(j)$ is at least the average and is thus $\geq \lceil (\ell-k)/k \rceil$. Hence the best arc has at most $\ell - \lceil \ell/k \rceil + 1$ numbers. See Figure 2.
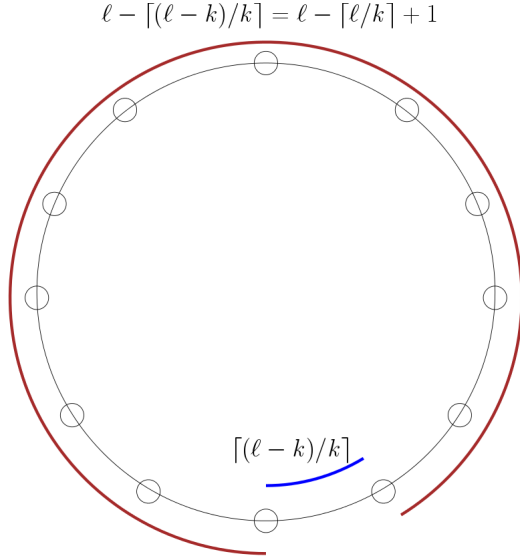
$$\ell - \lceil (\ell - k)/k \rceil = \ell - \lceil \ell/k \rceil + 1$$

$$\lceil (\ell - k)/k \rceil$$

Figure 2: A depiction of why hypercycle needs sets of size $\ell - \lceil \ell/k \rceil + 1$ to cover every choice of k elements.

Now, let G be the given ℓ-partite k-uniform hypergraph in which we want to find an ℓ-hyperclique. Let $V_1, \ldots, V_\ell$ be the vertex parts and let E be the set of k-hyperedges. We will build a new hypergraph on the same set of nodes but with hyperedges of size $\gamma = \ell - \lceil \ell/k \rceil + 1$ as follows.

Consider each $i \in [\ell]$ and every choice of nodes $u_i \in V_i, u_{i+1} \in V_{i+1}, \ldots, u_{i+\gamma-1} \in V_{i+\gamma-1}$ call the set of chosen nodes U, i.e. nodes in γ consecutive parts (mod ℓ). We need only consider the sets of γ consecutive parts because every subset of size k will be contained in one of these sets, by our choice of γ. We add a hyperedge between the nodes in U if every size k subset of U forms a hyperedge in G. That is, we create a big hyperedge in G' if all the k-tuples contained in it form a hyperedge in G. The runtime to create G' is $O(n^\gamma)$ as is the number of hyperedges created. Clearly G' is γ-uniform.

Now suppose that $a_1 \in V_1, \ldots, a_\ell \in V_\ell$ is an ℓ-hyperclique in G. All the hyperedges $(a_i, \ldots, a_{i+\gamma-1})$ are present in G' so $a_1, \ldots, a_\ell$ forms an ℓ-hypercycle in G'.

Now suppose that $a_1 \in V_1, \ldots, a_\ell \in V_\ell$ is an ℓ-hypercycle in G'. Consider $A = \{a_1, \ldots, a_\ell\}$ in G. We will show that it is an ℓ-hyperclique. Let $a_{i_1}, \ldots, a_{i_k}$ for $i_1 < i_2 < \ldots < i_k$ be any k nodes of A.

Let t be the index that maximizes $s(i_t)$ as in the beginning of the proof. Then, $\{i_t, i_t + 1, \ldots, i_{t-1}\}$ (which contains all $i_1, \ldots, i_k$) contains at most γ nodes and is thus contained in $\{i_t, i_t + 1, \ldots, i_{t+\gamma-1}\}$ which is a hyperedge in G' since $a_1 \in V_1, \ldots, a_\ell \in V_\ell$ is an ℓ-hypercycle in G'. However by the way we constructed the hyperedges, it must be that $(a_{i_1}, \ldots, a_{i_k})$ is a hyperedge of G. Thus all k-tuples are hyperedges in G and $\{a_1, \ldots, a_\ell\}$ is an ℓ-hyperclique in G.

So far we have shown that we can construct a hypergraph so that the ℓ-hypercliques in G correspond to the ℓ-hypercycles in G'. Suppose now that G is a hypergraph with weights on its hyperedges. We will define weights for the hyperedges of G' so that the weight of any ℓ-hypercycle in G' equals the weight of the ℓ-hyperclique in G that it corresponds to. To achieve this, we will assign each hyperedge y of G to some hyperedges $E(y)$ of G' and we will say that these hyperedges are responsible for y. Then we will set the weight of a hyperedge e of G' to be the sum of the weights of the hyperedges of G' that it is responsible for. We will guarantee that for any hypercycle of G', no two hyperedges in it are responsible for the same hyperedge of G, and that every hyperedge of the hyperclique that the hypercycle is representing is assigned to some of the hypercycle hyperedges.

Consider any hyperedge of G, $A = (a_{i_1}, \ldots, a_{i_k})$ with $a_{i_j} \in V_{i_j}$. Let i_t be the smallest index that maximizes $s(i_t)$. We assign A to every hyperedge of G' contained in $V_{i_t} \times V_{i_t+1} \times \ldots \times V_{i_t+\gamma-1}$ that intersects V_{i_j} exactly at a_{i_j}. Then notice that any ℓ-hypercycle that contains $a_{i_1}, \ldots, a_{i_k}$ contains exactly one of these hyperedges, so that the weight of the hypercycle is exactly the weight of the hyperclique that it corresponds to. Since every hyperedge of G' contains $\binom{\gamma}{k}$ hyperedges of G, the weights of the hyperedges lie in $[-W\binom{\gamma}{k}, W\binom{\gamma}{k}]$.

4 Reduction from Hypercycle to Cycle in Directed Graphs

We have shown hardness for hypercycle from hyperclique. However, in order to get results on cycles in normal graphs we have to show that hypercycle can be solved efficiently with cycles in graphs. We do so below.

LEMMA 4.1. *Given an n-node λ uniform hypergraph H with nodes partitioned into $V_1, \ldots, V_k$ in which one wants to find a k-hypercycle $(v_1, v_2, \ldots, v_k)$ with $v_j \in V_j$ for each j, one can in $\tilde{O}(n^\lambda)$ time create a k-circle-layered directed graph G on $O(n^{\lambda-1})$ nodes and $O(n^\lambda)$ edges, so that H contains a k-hypercycle with one node in each partition V_i if and only if G contains a directed k-cycle. Moreover, if H has integer weights on its edges bounded by M, then one can add integer edge weights to the edges of the graph G, bounded by M, so that the minimum weight k-cycle in G has the same weight as the minimum weight k-hypercycle in H.*

If k is odd, the G can be made undirected.

Proof. Recall that a k-hypercycle in a λ-uniform hypergraph is formed by having a list of k nodes $v_1, v_2, \ldots, v_k$ and having a hyperedge for all choices of $i \in [1, k]$ formed by the set $v_i, v_{i+1}, \ldots, v_{i+\lambda-1}$ where we consider indices mod k.

We describe the construction of the directed graph G. It will be k-circle-layered with node parts $U_1, \ldots, U_k$. For each $i \in \{1, \ldots, k\}$, we will add a node in part U_i of G for

every choice of $\lambda - 1$ nodes $v_i, \ldots, v_{i+\lambda-2}$ such that $v_i \in V_i$, $v_{i+1} \in V_{i+1}$, ..., $v_{i+\lambda-2} \in V_{i+\lambda-2}$. This totals $n^{\lambda-1}$ nodes. Call this node $((v_i, \ldots, v_{i+\lambda-2}))$.

We will add a directed edge in G between nodes $((v_i, \ldots, v_{i+\lambda-2}))$ and $((v'_{i+1}, \ldots, v'_{i+\lambda-1}))$ if $v'_j = v_j$ for $j \in [i+1, i+\lambda-2]$ and $\{v_i, \ldots, v_{i+\lambda-2}, v'_{i+\lambda-1}\}$ is a hyper-edge in H. Assign this edge the weight of the hyperedge $\{v_i, \ldots, v_{i+\lambda-2}, v'_{i+\lambda-1}\}$ in H. Every node in G can connect to a maximum of n other nodes giving us $|E| = O(n^{\lambda+1})$.

Now note that G is a k-circle-layered graph. Further note that if a k-cycle exists in G then each of its edges corresponds to a hyper-edge edge in H and the set of vertices represented in the k-cycle in G corresponds to a choice of k nodes $v_1, v_2, \ldots, v_k$. Further, every edge of G covers λ adjacent vertices from $v_1, v_2, \ldots, v_k$.

We also note that if k is odd, then the edges of G can be made undirected: any k cycle in G must have a node from each U_i, as removing any U_i from G makes it bipartite, and no odd cycles can exist in a bipartite graph.

We immediately obtain the following corollaries:

COROLLARY 4.1. *Let $\lambda = \ell - \lceil \ell/k \rceil + 1$. Under the (ℓ, k)-Hyperclique Hypothesis, min weight ℓ-cycle in directed graphs (or in undirected graphs for ℓ odd) cannot be solved in $O(m^{\ell/\lambda - \varepsilon})$ time for any $\varepsilon > 0$ for $m = \Theta(n^{1+1/(\lambda-1)})$ edge, n node graphs.*

Proof. We start with a k-uniform hypergraph with n_{old} nodes. The number of edges in the graph produced by Lemma 4.1 when applied to this hypergraph is $(n_{old})^{\lambda}$. By the (ℓ, k)-Hyperclique Hypothesis any algorithm to find a (ℓ, k)-Hyperclique should take $(n_{old})^{\ell}$ time. Combining these facts we get a bound of $\Omega(m^{\ell/\lambda - o(1)})$.

The number of nodes produced by Lemma 4.1 is $n = (n_{old})^{\lambda-1}$, the number of edges is $m = (n_{old})^{\lambda}$. Thus, $m = n^{1+1/(\lambda-1)}$

COROLLARY 4.2. *Let $\lambda = k - \lceil k/2 \rceil + 1$. Under the Min Weight k-Clique Hypothesis, min weight k-cycle in directed graphs (or in undirected graphs for k odd) cannot be solved in $O(nm^{\lceil k/2 \rceil/\lambda - \varepsilon})$ time for any $\varepsilon > 0$ for $m = \Theta(n^{1+1/(\lambda-1)})$ edge, n node graphs.*

Proof. The Min Weight k-Clique Hypothesis is equivalent to the Min Weight $(k, 2)$-Hyperclique Hypothesis. We can plug in these numbers to get the result above.

When considering odd sizes of cliques and cycles, these results become show $O(mn)$ hardness for the cycle problems at certain densities.

COROLLARY 4.3. *Under the Min Weight $(2k+1)$-Clique Hypothesis, min weight $(2k+1)$-cycle in directed or undirected graphs cannot be solved in $O(mn^{1-\varepsilon})$ time for any $\varepsilon > 0$ for $m = \Theta(n^{1+1/k})$ edge, n node graphs.*

Proof. The Min Weight $(2k+1)$-Clique Hypothesis is equivalent to the Min Weight $((2k+1), 2)$-Hyperclique Hypothesis. We can plug in these numbers to get the result above. We then note that directed $(2k+1)$-cycle is solved by undirected $(2k+1)$-cycle because $(2k+1)$ is odd.

COROLLARY 4.4. *Under the Exact Weight $(2k+1)$-Clique Hypothesis, exact weight $(2k+1)$-cycle in directed and undirected graphs cannot be solved in $O(mn^{1-\varepsilon})$ time for any $\varepsilon > 0$ for $m = \Theta(n^{1+1/k})$ edge, n node graphs.*

Proof. The Exact Weight $(2k+1)$-Clique Hypothesis is equivalent to the Exact Weight $((2k+1), 2)$-Hyperclique Hypothesis. We can plug in these numbers to get the directed version of the above corollary. We then note that directed $(2k+1)$-cycle is solved by undirected $(2k+1)$-cycle because $(2k+1)$ is odd.

5 Probably Optimal Weighted k-Cycle Algorithms

The reductions from ℓ-hyperclique in k-uniform hypergraphs (through hypercycle) to directed ℓ-cycle produces graphs on $O(n^{\gamma-1})$ nodes and $O(n^{\gamma})$ edges where $\gamma = \ell - \lceil \ell/k \rceil + 1$.

For the special case of the reduction from Min Weight ℓ-Clique ($k = 2$), one obtains a graph on $O(n^{\lfloor \ell/2 \rfloor + 1})$ edges. Suppose that ℓ is odd. The number of edges in the graph is $O(n^{(\ell+1)/2})$, and solving the Shortest ℓ-Cycle problem in this graph in $O(n^{\ell-\varepsilon})$ time for any $\varepsilon > 0$ would refute the Min Weight ℓ-Clique Hypothesis. We immediately obtain that Min Weight ℓ-Cycle on m edge graphs requires $m^{2-2/(\ell+1)-o(1)}$ time.

Using Lemma 2.1, we can also conclude that if ℓ is even, then solving Min Weight ℓ-Cycle on m edge graphs requires $m^{2-2/\ell-o(1)}$ time.

THEOREM 5.1. *Assuming the Min Weight ℓ-Clique Hypothesis, on a Word RAM on $O(\log n)$ bit words, Min Weight ℓ-Cycle on m edge graphs requires $m^{2-2/\ell-o(1)}$ time if ℓ is even and $m^{2-2/(\ell+1)-o(1)}$ time if ℓ is odd.*

The rest of this section will show that the above runtime can be achieved:

THEOREM 5.2. *Min Weight ℓ-Cycle on m edge graphs can be solved in $\tilde{O}(m^{2-2/\ell})$ time if ℓ is even and $\tilde{O}(m^{2-2/(\ell+1)})$ time if ℓ is odd.*

The proof proceeds analogously to Alon, Yuster and Zwick's algorithm [AYZ97] for ℓ-Cycle in unweighted directed graphs. Let us review how their algorithm works and

see how to modify it to handle weighted graphs. First, pick a parameter Δ and take all $O(m/\Delta)$ nodes of degree $\geq \Delta$. Call the set of these nodes H. For every $v \in H$, Alon, Yuster and Zwick use an $\tilde{O}(m)$ time algorithm by Monien [Mon85] to check whether there is an ℓ-cycle going through v. If no ℓ-cycle is found, they consider the subgraph with all nodes of H removed and enumerate all $\lceil \ell/2 \rceil$-paths X and all $\lfloor \ell/2 \rfloor$-paths Y in it. The number of paths in $X \cup Y$ is $\leq m\Delta^{\lceil \ell/2 \rceil - 1}$. Then one sorts X and Y in lexicographic order of the path endpoints and searches in linear time in $|X| + |Y|$ for a path in X from a to b and a path in Y from b to a. To make sure that the cycle closed by these paths is simple, one can first start by color coding in two colors red and blue and let X contain only paths with red internal nodes and Y only paths with blue internal nodes, or one can just go through all paths that share the same end points. Either way, the total runtime is asymptotically $m^2/\Delta + m\Delta^{\lceil \ell/2 \rceil - 1}$, and setting $\Delta = m^{1/\lceil \ell/2 \rceil}$ gives a runtime of $\tilde{O}(m^{2-1/(\lceil \ell/2 \rceil)})$.

One can modify the algorithm to give a Shortest ℓ-cycle in an edge-weighted graph, as follows. First, we replace Monien's algorithm with an algorithm that given a weighted graph and a source s can in $\tilde{O}(m)$ time determine a shortest ℓ-cycle C containing s. To this end, we use color-coding: we give every node of G a random color from 1 to ℓ and note that with probability at least $1/\ell^\ell$, the ith node of C is colored i, for all i, whp; as s is the first node of C, we can assume that s is colored 1. As usual, this can be derandomized using ℓ-perfect hash families. Now, in G, only keep the edges (u, v) such that $c(v) = c(u) + 1$ (not mod ℓ, so there are no edges between nodes colored ℓ and nodes colored 1). This makes the obtained subgraph G' ℓ-partite and acyclic. Now, run Dijkstra's algorithm from s, computing the distances $d(s, v)$ for each $v \in V$. Then for every in-neighbor u of s in G colored ℓ, compute $d(s, u) + w(u, s)$ and take the minimum of these, $W = \min_u d(s, u) + w(u, s)$. If the nodes of C are colored properly (the ith node is colored i), then W is the weight of the shortest ℓ-cycle through s since the shortest path from s to any u colored k, if the distance is finite, must have ℓ nodes colored from 1 to ℓ. Dijkstra's algorithm runs in $\tilde{O}(m)$ time, and one would want to repeat $O(\ell^\ell \log n)$ times to get the correct answer with high probability (the same cost is obtained in the derandomization).

Now that we have a counterpart of Monien's algorithm, let's see how to handle the case when the shortest k-cycle in the graph only contains nodes of low degree. Similar to the original algorithm, we again compute the set of paths X and Y, but we only consider shortest paths together with their weights. Then one is looking for two paths (one between a and b and the other between b and a) so that their sum of weights is minimized. This can also be found in linear time in $|X|$ and $|Y|$ when they are sorted by end points (a, b) and by weight. The total runtime is again $\tilde{O}(m^{2-1/(\lceil \ell/2 \rceil)})$.

6 Hardness Results for Shortest Cycle

THEOREM 6.1. *If Shortest cycle in an N node, M edge directed graph can be solved in $T(N, M)$ time, then the Minimum Weight k-cycle in an n node, m edge directed graph is solvable in $\tilde{O}(T(n, m))$ time.*

Proof. Let the weights of the k-cycle instance range between $-W$ and W. Use Lemma 2.2 to reduce the Min Weight k-cycle problem to one in a k-circle-layered graph G with partitions $A_1, A_2, \ldots, A_k$. Add the value $4W$ to each edge, which adds $4kW$ to the value of every k-cycle. Every cycle in a directed k-circle-layered graph is a ck-cycle when c is a positive integer since every cycle must go around the graph circle some number of times. Due to the added weight $4W$, the Shortest cycle in the new graph will minimize the number of edges: Any ck-cycle C for $c \geq 2$ will have weight $\geq 4ckW + w(C) \geq 3ckW \geq 6kW$, where $w(C) \geq -Wck$ is the weight of C in G. The weight of a k cycle, however is at most $4kW + kW = 5kW < 6kW$. Thus, the weight of the Shortest Cycle in the new graph is exactly the weight of the Min Weight k-Cycle in G, plus $4kW$, and the Shortest Cycle will exactly correspond to the Min Weight k-Cycle in G.

LEMMA 6.1. *If Shortest Cycle can be solved in $T(n, m)$ time in an n-node, m-edge directed unweighted graph, then k-cycle in a directed unweighted n-node, m-edge graph is solvable in $\tilde{O}(T(n, m))$ time.*

Proof. The proof is similar but simpler than that of Theorem 6.1. We first reduce to k-cycle in a k-circle-layered graph, and then just find the Shortest Cycle in it. Since the graph obtained is directed and k-circle-layered, if it contains a k-cycle, then that cycle is its shortest cycle.

COROLLARY 6.1. *If Min Weight $(2L+1)$-clique requires $n^{2L+1-o(1)}$ time, then Shortest Cycle in directed weighted graphs requires $mn^{1-o(1)}$ time whenever $m = \Theta(n^{1+1/L})$.*

Directed Shortest Cycle in unweighted graphs requires $m^{3/2-o(1)}$ time under the Max 3 SAT Hypothesis, $m^{4/3-o(1)}$ time under the Exact Weight K Clique Hypothesis, and $m^{2\omega/3-o(1)}$ time under the K-Clique Hypothesis.

Proof. The first statement follows immediately from Lemma 6.1 and Corollary 4.2. We will focus on the second part of the corollary.

The reduction in Corollary 9.1 from Max 3 SAT on N variables to ℓ-cycle (for any $\ell > 3$) produces a $(n^{\ell - \lceil \ell/3 \rceil})$-node, $(n^{\ell - \lceil \ell/3 \rceil + 1})$-edge graph (for $n = 2^{N/\ell}$, so that solving ℓ-cycle in it in $O(n^{\ell - \varepsilon})$ time for any $\varepsilon > 0$, then the Max 3 SAT Hypothesis is false. Now suppose that Shortest cycle in a directed graph can be solved in $O(m^{3/2-\varepsilon})$ time for some $\varepsilon > 0$. Set ℓ to be any integer greater than $3/\varepsilon$ and

divisible by 3. Consider the ℓ-cycle problem in $n^{\ell-\lceil\ell/3\rceil+1}$-edge graphs obtained via the reduction from Max 3 SAT. Reduce it to Shortest Cycle as in Lemma 6.1. As ℓ is divisible by 3, the number of edges in consideration is $O(n^{2\ell/3+1})$. Then, applying the $O(m^{3/2-\varepsilon})$ time algorithm, we can solve the ℓ-cycle instance in $O(n^{(2\ell/3+1)(3/2-\varepsilon)})$ time. As we set $\ell \geq 3/\varepsilon$, the exponent in the running time is $\ell+3/2-\varepsilon(2\ell/3+1) \leq \ell+3/2-\varepsilon(2/\varepsilon+1) \leq \ell-1/2-\varepsilon$, and hence we obtain a faster algorithm for ℓ-cycle and contradict the Max 3-SAT hypothesis.

A similar argument applies to show that $m^{4/3-o(1)}$ time is needed under the Exact Weight K Clique Hypothesis, and $m^{2\omega/3-o(1)}$ time is needed under the K-Clique Hypothesis.

7 Discussion of the Hyperclique Hypothesis

In this section we discuss why the (ℓ,k)-Hyperclique hypothesis is believable.

First, when $k > 2$, the fastest algorithms for the ℓ-hyperclique problem run in $n^{\ell-o(1)}$ time, and this is not for lack of trying. Many researchers [WBK$^+$] have attempted to design a faster algorithm, for instance by mimicking the matrix multiplication approach for k-Clique. However, in doing this, one needs to design a nontrivial algorithm for a generalized version of matrix multiplication. Unfortunately, in Section 8, we show that the rank and even the border rank of the tensor associated with this generalized product is as large as possible, thus ruling out the arithmetic circuit approach for the problem. Thus, if a faster than n^ℓ algorithm exists for k-uniform hypergraphs with $k > 2$, then it must use radically different techniques than the Strassen-like approach to regular matrix multiplication.

Another reason to believe the Hyperclique hypothesis is due to its relationship to Maximum Constraint Satisfaction Problems (CSPs). R. Williams [Wil07] showed that Max-3-SAT can be reduced to finding a 4-Hyperclique in a 3-uniform hypergraph, so that if the latter can be solved in $O(n^{4-\varepsilon})$ time for n node graphs and $\varepsilon > 0$, then Max-3-SAT can be solved in $O(2^{(1-\delta)n})$ time for formulas on n variables.

Max-3-SAT has long resisted attempts to improve upon the brute-force 2^n runtime. Recent results (e.g. [ACW16]) obtained $2^{n-o(n)}$ time improvements, but there is still no $O((2-\varepsilon)^n)$ time algorithm. Generalizing the reduction from [Wil07] (see Section 9), one can reduce Max-k-SAT to ℓ-hyperclique in a k-uniform hypergraph for any $\ell > k$, so that if the latter problem can be solved in $O(n^{\ell-\varepsilon})$ time for n node graphs and $\varepsilon > 0$, then Max-k-SAT can be solved in $O(2^{(1-\delta)n})$ time for formulas on n variables. In fact, R. Williams [Wil07] showed that even harder Constraint Satisfaction Problems (CSPs) can be reduced to hyperclique. CSPs where the constraints are degree 3 polynomials representing Boolean functions over the n variables. In Section C we generalize this to CSPs where the constraints are

degree k polynomials. Such CSPs include Max-k-SAT and also include some CSPs with constraints involving more than k variables. In any case, the (ℓ,k)-Hypothesis captures the difficulty of this very general class of CSPs.

Another reason to believe the Hypothesis is due to its relationship to the Exact Weight k-Clique Conjecture [VW13] which states that finding a k-Clique of total edge weight exactly 0 in an n node graph with large integer weights requires $n^{k-o(1)}$ time. The Exact Weight k-Clique conjecture is implied by the Min Weight k-Clique conjecture, so it is at least as believable. Furthermore, for the special case $k = 3$, both 3SUM and APSP can be reduced to Exact Weight 3-Clique, so that a truly subcubic algorithm for the latter problem would refute both the APSP and the 3SUM conjectures [Pat10, VW13, VW10]. Exact Weight k Clique is thus a very difficult problem. Recent work by Abboud et al. [ABDN17] shows how to use the techniques in [ALW14] to reduce the Exact Weight k-Clique problem to (unweighted) k-Clique in a 4-uniform hypergraph. Thus, if one believes the Exact Weight k-Clique conjecture, then one should definitely believe the $(\ell,4)$-Hyperclique Hypothesis. (A generalization of this approach also shows that Exact Weight ℓ-Hyperclique in a k-uniform hypergraph can be tightly reduced to (unweighted) ℓ-hyperclique in a $2k$-uniform hypergraph.)

We note that the hypothesis concerns dense hypergraphs. For hyperclique in sparse hypergraphs, faster algorithms are known: the results of Gao et al. [GIKW17] imply that an ℓ-hyperclique in an m-hyperedge, n-node k-uniform hypergraph (for $\ell > k$) can be solved in $m^{\ell-1}/2^{\Theta(\sqrt{\log m})}$.

8 No Generalized Matrix Multiplication for k>2

The fastest known algorithm for ℓ-clique reduces ℓ-clique to triangle detection in a graph and then uses matrix multiplication to find a triangle [NP85]. One might ask, is there a similar approach to finding an ℓ-hyperclique in a k-uniform hypergraph faster than $O(n^\ell)$ time?

The first step would be to reduce ℓ-hyperclique problem in a k-uniform hypergraph to $k+1$-hyperclique in a k-uniform hypergraph. This step works fine: Assume for simplicity that ℓ is divisible by $(k+1)$ so that $\ell = c(k+1)$. We will build a new graph G'. Take all c-tuples of vertices of G and create a vertex in G' corresponding to the tuple if it forms an c-hyperclique in G (if $c < k$, any c-tuple is a hyperclique, and if $c \geq k$, it is a hyperclique if all of its k-subsets are hyperedges). For every choice of k distinct c-tuples, create a hyperedge in G' on them if every choice of k nodes from their union forms a hyperedge in G. Now, $k+1$-hypercliques of G' correspond to ℓ-hypercliques of G. G' is formed in $O(n^{ck})$ time and has $O(n^c)$ nodes. Hence if a $(k+1)$-hyperclique in a k-uniform hypergraph on N nodes can be found in $O(N^{(k+1)-\varepsilon})$ time for some $\varepsilon > 0$,

then an ℓ-hyperclique in a k-uniform hypergraph on n nodes can be found in $O(n^{ck} + n^{c(k+1)-\varepsilon c}) = O(n^{\ell-\delta})$ time for $\delta = \min\{\ell/(k+1), c\varepsilon\} > 0$.

Thus it suffices to just find $k+1$-hypercliques in k-uniform hypergraphs. Following the approach for finding triangles (the case $k = 2$), we want to define a suitable matrix product.

In the matrix multiplication problem we are given two matrices and we are asked to compute a third. Matrices are just tensors of order 2. The new product we will define is for tensors of order k. We will call these k-tensors for brevity. The natural generalization of matrix multiplication for k-tensors of dimensions $n \times \ldots \times n$ (k times) is as follows.

k-wise matrix product Given k k-tensors of dimensions $n \times \ldots \times n$, $A^1, \ldots, A^k$, compute the k-tensor C given by

$$C[i_1, \ldots, i_k] =$$
$$\prod_{\ell \in [n]} A^1[i_1, \cdots, i_{k-1}, \ell] \cdot A^2[i_2, \cdots, i_{k-1}, \ell, i_k] \cdots$$
$$\cdots A^k[\ell, i_k, i_1, \cdots, i_{k-2}].$$

The special case of $k = 3$ was defined in 1980 by Mesner et al. [MB90]: Given three 3-tensors A^1, A^2, A^3 with indices in $[n] \times [n] \times [n]$ compute the product C defined as $C[i, j, k] = \prod_{\ell \in [n]} A^1[i, j, \ell] \cdot A^2[j, \ell, k] \cdot A^3[\ell, k, i]$. The more general definition as above was defined later by [GER11] and its properties have been studied within algebra and combinatorics, e.g. [Gna15].

Now, if one can compute the k-wise matrix product in $T(n)$ time, then one can also find a $k+1$-hyperclique in a k-uniform hypergraph in the same time: define A to be the adjacency tensor of the hypergraph – it is of order k and has a 1 for every k-tuple that forms a hyperedge; if the k-wise product of k copies of A has a nonzero for some k-tuple that is also a hyperedge, then the hypergraph contains a $k+1$-hyperclique.

Now the question is: *"Is there an $O(n^{k-\varepsilon})$ time algorithm for $(k-1)$-wise matrix product for $k \geq 4$ and $\varepsilon > 0$?"*

A priori, it seems quite plausible that a faster than n^k-time algorithm exists for generalized matrix product, as all of the techniques developed for matrix multiplication carry over to the general case: tensor products, rank, border rank etc. The only thing missing is a suitable base case algorithm. Many researchers have searched for such an algorithm (e.g. [WBK+]). Williams asked in [Wil10] whether an $O(n^{4-\varepsilon})$ time algorithm exists for $k = 3$ and $\varepsilon > 0$, and this question has remained unanswered.

We will show that there is *no* smaller arithmetic circuit than the trivial one for generalized matrix product. In other words, there can be no analogue of the fast algorithms for clique that carry over to hyperclique.

THEOREM 8.1. *For every $k \geq 3$, the border rank of the tensor of k-wise matrix multiplication is exactly n^{k+1}.*

Proof. [Sketch.] We give a sketch of the proof for $k = 3$. The general case is similar but requires more indices. Consider the tensor t which is such that $t_{i,j,k,j',k',l',k'',l'',i'',l''',i''',j'''}$ is 1 whenever $i = i'' = i''', j = j' = j''', k = k' = k''$ and 0 otherwise. Let t be in $A^* \otimes B^* \otimes C^* \otimes D^*$. Define the flattening of t which is a linear transformation $T : A^* \otimes B^* \to C \otimes D$ that takes $a(i, j, k)^* \otimes b(j', k', l)^*$ to $c(k, l, i) \otimes d(l, i, j)$ if $j = j', k = k'$ and to 0 otherwise. The vectors $c(k, l, i) \otimes d(l, i, j)$ are n^4 independent vectors that span the image of T, and hence the map has rank n^4. Since T is a linear transformation it must also have border rank n^4. Because T is a flattening of t, t must have border rank at least n^4.

Theorem 8.1 completely rules out the arithmetic circuit approach for $k > 2$. Nevertheless, there might still be an $O(n^{k+1-\varepsilon})$ time algorithm (at least for the Boolean version of the product) that uses operations beyond $+, \cdot, /$, for instance by working at the bit level. Obtaining such an algorithm is quite a challenging but very interesting endeavor.

9 Max-k-SAT to Tight Hypercycle

We are dealing with graphs and SAT instances. For ease of reading we will capitalize the variables associated with graphs (i.e. $M = |E|$ and $N = |V|$) and leave the variables associated with SAT instances in lower case (i.e. m for number of clauses and n for number of variables). The current fastest known algorithms for Max-k-SAT come from Alman, Chan and Williams [ACW16].

LEMMA 9.1. *For all integers $L \geq 3$ if we can detect L-hyper cliques in a k uniform hypergraph, G, with N nodes in time $T(N)$ then we can solve max-k-SAT in $\tilde{O}((mn^k)^{\binom{L}{k}} T(L2^{n/L}) + mn^k 2^{nk/L})$ time.*

Proof. Let n be the number of variables and m the number of clauses in our max-3-SAT instance.

Notice that the clauses of k-SAT can be represented by a k degree polynomial. Each clause only depends on k variables and $x_i^2 = x_i$.

So we can apply Theorem C.1.

If the unweighted L-hyperclique problem on a k-uniform hypergraph can be solved in time $T(n)$ then the maximal degree-k-CSP problem can be solved in $O((mn^k)^{\binom{L}{k}} T(2^{n/L}) + mn^k 2^{kn/L})$ time. When L and k are constants

COROLLARY 9.1. *For all integers $L \geq 3$ if we can detect a tight L-hyper cycle in a $d = L - \lceil L/3 \rceil + 1$ uniform hypergraph, G, with $N = L2^{n/L}$ nodes in time $T(N)$ then we can*

solve max-3-SAT in $\tilde{O}(m^{\binom{L}{3}}T(2^{n/L}) + 2^{n(d/L)} + m^{\binom{L}{3}}2^{3n/L})$ time.

Proof. Take Lemma 9.1 in the case of $k = 3$ to get a reduction from max-3-SAT to $m^{\binom{L}{k}}T(N)$ instances of L-hyperclique in a k regular graph. let $d = L - \lceil L/3 \rceil + 1$. We then use Theorem 3.1 to generate a d-uniform hypergraph G' on $N = L2^{n/L}$ nodes such that G has a L-hypercycle only if the associated L-hyperclique instance has a clique. The overhead of this reduction is $O(2^{n(d/L)} + m^{\binom{L}{3}}2^{3n/L})$ time to create these graphs. It takes $m^{\binom{L}{3}}T(N)$ time to run L-hypercycle on the $m^{\binom{L}{3}}$ instances. Giving us a total time of $\tilde{O}(m^{\binom{L}{3}}T(2^{n/L}) + 2^{n(d/L)} + m^{\binom{L}{3}}2^{3n/L})$.

COROLLARY 9.2. *If directed L-cycle can be solved in a graph G with $N = 2^{n(d-1)/L}$ nodes and $M = 2^{nd/L}$, where $d = L - \lceil L/3 \rceil + 1$, in time $T(N, M)$ then max-3-SAT can be solved in time $\tilde{O}(m^{\binom{L}{3}}T(N, M) + M + mn^3 2^{3n/L})$.*

Proof. If directed L-cycle can be solved in a graph G with $N = 2^{n(d-1)/L}$ nodes and $M = 2^{nd/L}$ in time $T(N, M)$ then by Lemma 4.1 we can solve tight L-hyper cycle in a d regular graph in time $O(T(N, M) + M)$.

If we could detect a tight L-cycle in a d regular hypergraph, G, with $N = L2^{n(d-1)/L}$ nodes in time $O(T(N, M) + M)$ then we can solve max-3-SAT in $\tilde{O}(m^{\binom{L}{3}}T(N, M) + m^{\binom{L}{3}}M + 2^{3n/L})$ time.

THEOREM 9.1. *If directed L-cycle, for $L > 3$, can be solved in a graph G in time $O(M^{c_L - \varepsilon})$, where $c_L = L/d = L/(L - \lceil L/3 \rceil + 1)$ then max-3-SAT can be solved in time $\tilde{O}(2^{(1-\varepsilon')n})$.*

Proof. We can plug in the running time $O(M^{c_L - \varepsilon})$ into Corollary 9.2. On a graph with $N = 2^{n(d-1)/L}$ nodes and $M = 2^{nd/L}$ this running time gives

$$T(N, M) = 2^{(c_L - \varepsilon)nd/L} = 2^{(L/d - \varepsilon)nd/L} = 2^{(1 - \varepsilon d/L)n}.$$

We get that max 3-SAT is solved in time $\tilde{O}((mn^k)^{\binom{L}{k}}2^{(1-\varepsilon d/L)n} + m^{\binom{L}{k}}2^{nd/L} + 2^{n/L})$. Note that $d/L < 1$ for $L > 3$ and d and L are constants. Further note that $(mn^k)^{\binom{L}{k}}$ is a polynomial factor. Thus, max 3-SAT is solved in $\tilde{O}(2^{(1-\varepsilon')n})$.

This theorem gives us a lower bound for detecting L-cycles for large constant values of L of $\tilde{\Omega}(M^{3/2})$

COROLLARY 9.3. *If we can solve directed L-cycle in time $O(M^{3/2-\varepsilon})$ for all constant L then we can solve max-3-SAT in time $O(2^{1-\varepsilon'n})$.*

Proof. We will use Theorem 9.1 and set L large enough that $c_L > 3/2 - \varepsilon$.

As $L/(L - \lceil L/3 \rceil + 1) > L/(2L/3 + 1) = 3/2 - 9/(4L + 6)$. If $L > (9/\varepsilon - 6)/4$ then $c_L > 3/2 - \varepsilon$. We can now invoke Theorem 9.1. If we can solve L-cycle in time $O(M^{3/2-\varepsilon})$ then we can L-cycle in time $O(M^{c_L - \delta})$ for some $\delta > 0$. Thus we can solve max-3-SAT in time $O(2^{(1-\varepsilon')n})$.

What if you care about specifically solving directed cycle for a particular constant L? Our result will garner improvements over the previous result of $\tilde{\Omega}(M^{4/3})$ for many small cycle lengths. Notably for 7-cycle we get the bound of $\tilde{\Omega}(M^{7/5})$ (and $7/5 > 4/3$).

COROLLARY 9.4. *If we can solve directed 7-cycle in a graph with $M = N^{5/4}$ in time $O(M^{7/5-\varepsilon})$ then we can solve max-3-SAT in time $O(2^{(1-\varepsilon')n})$.*

Proof. Once again we will use Theorem 9.1 and we will plug in $L = 7$ note that $c_L = 7/5$ giving us the bound.

References

[ABDN17] Amir Abboud, Karl Bringmann, Holger Dell, and Jesper Nederlof. Personal communication, 2017.

[ABV15] 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.

[ABZ05] Adi Avidor, Ido Berkovitch, and Uri Zwick. Improved approximation algorithms for MAX NAE-SAT and MAX SAT. In *Approximation and Online Algorithms, Third International Workshop, WAOA 2005, Palma de Mallorca, Spain, October 6-7, 2005, Revised Papers*, pages 27–40, 2005.

[ACW16] Josh Alman, Timothy M. Chan, and R. Ryan Williams. Polynomial representations of threshold functions and algorithmic applications. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 467–476, 2016.

[AGV15] Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1681–1697, 2015.

[Aku99] Tatsuya Akutsu. Approximation and exact algorithms for rna secondary structure prediction and recognition of stochastic context-free languages. *Journal of Combinatorial Optimization*, 3(2):321–336, 1999.

[ALW14] Amir Abboud, Kevin Lewi, and Ryan Williams. Losing weight by gaining edges. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, pages 1–12, 2014.

[AR16] Udit Agarwal and Vijaya Ramachandran. Fine-grained complexity and conditional hardness for sparse graphs. *arXiv preprint arXiv:1611.07008v3*, 2016.

[AV14] Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 434–443. IEEE, 2014.

[AVW14] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *International Colloquium on Automata, Languages, and Programming*, pages 39–51. Springer, 2014.

[AW02] Takao Asano and David P. Williamson. Improved approximation algorithms for MAX SAT. *J. Algorithms*, 42(1):173–202, 2002.

[AYZ97] Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.

[AYZ16] Noga Alon, Raphael Yuster, and Uri Zwick. Color coding. In *Encyclopedia of Algorithms*, pages 335–338. 2016.

[BGMW17] Karl Bringmann, Pawel Gawrychowski, Shay Mozes, and Oren Weimann. Tree edit distance cannot be computed in strongly subcubic time (unless APSP can). *CoRR*, abs/1703.08940, 2017.

[Bra01] U. Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(2):163–177, 2001.

[BT16] Arturs Backurs and Christos Tzamos. Improving viterbi is hard: Better runtimes imply faster clique algorithms. *CoRR*, abs/1607.04229, 2016.

[CGS15] Marek Cygan, Harold N. Gabow, and Piotr Sankowski. Algorithmic applications of baur-strassen's theorem: Shortest cycles, diameter, and matchings. *J. ACM*, 62(4):28:1–28:30, 2015.

[Cha12] Timothy M Chan. All-pairs shortest paths for unweighted undirected graphs in o (mn) time. *ACM Transactions on Algorithms (TALG)*, 8(4):34, 2012.

[Dij59] E. W. Dijkstra. A note on two problems in connection with graphs. *Numer. Math.*, pages 269–271, 1959.

[DKS17] Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Morten Stöckel. Finding even cycles faster via capped k-walks. *CoRR*, abs/1703.10380, 2017.

[EG04] Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theor. Comput. Sci.*, 326(1-3):57–67, 2004.

[FG95] Uriel Feige and Michel X. Goemans. Aproximating the value of two prover proof systems, with applications to MAX 2sat and MAX DICUT. In *Third Israel Symposium on Theory of Computing and Systems, ISTCS 1995, Tel Aviv, Israel, January 4-6, 1995, Proceedings*, pages 182–189, 1995.

[Fre77] L. Freeman. A set of measures of centrality based upon betweenness. *Sociometry*, 40:35–40, 1977.

[Gal14] 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.

[GER11] E. K. Gnang, A. Elgammal, and V. Retakh. A spectral theory for tensors. *Annales de la faculte des sciences de Toulouse Mathematiques*, 20(4):801–841, 7 2011.

[GIKW17] Jiawei Gao, Russell Impagliazzo, Antonina Kolokolova, and R. Ryan Williams. Completeness for first-order properties on sparse structures with algorithmic applications. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2162–2181, 2017.

[GL09] Zvi Gotthilf and Moshe Lewenstein. Improved algorithms for the k simple shortest paths and the replacement paths problems. *Information Processing Letters*, 109(7):352–355, 2009.

[Gna15] E. K. Gnang. A combinatorial approach to the algebra of hypermatrices. Technical report, arXiv:1403.3134 [math.CO], 2015.

[GO95] A. Gajentaan and M. Overmars. On a class of $o(n^2)$ problems in computational geometry. *Computational Geometry*, 5(3):165–185, 1995.

[IP01] R. Impagliazzo and R. Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

[IPZ01] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

[IR78] Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM Journal on Computing*, 7(4):413–423, 1978.

[MB90] Dale M. Mesner and Prabir Bhattacharya. Association schemes on triples and a ternary algebra. *Journal of Combinatorial Theory, Series A*, 55(2):204 – 234, 1990.

[MMG89] Kavindra Malik, Ashok K Mittal, and Santosh K Gupta. The k most vital arcs in the shortest path problem. *Operations Research Letters*, 8(4):223–227, 1989.

[Mon85] B. Monien. How to find long paths efficiently. *Annals of Discrete Mathematics*, 25:239—254, 1985.

[MP88] Bojan Mohar and Tomaž Pisanski. How to compute the wiener index of a graph. *Journal of Mathematical Chemistry*, 2(3):267–277, 1988.

[NP85] J. Nešetřil and S. Poljak. On the complexity of the subgraph problem. *Commentationes Math. Universitatis Carolinae*, 26(2):415–419, 1985.

[NPW03] Enrico Nardelli, Guido Proietti, and Peter Widmayer. Swapping a failing edge of a single source shortest paths tree is good and fast. *Algorithmica*, 35(1):56–74, 2003.

[OS17] James B. Orlin and Antonio Sedeño-Noda. An $O(nm)$ time algorithm for finding the min length directed cycle in a graph. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1866–1879, 2017.

[Pat10] Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 603–610. ACM, 2010.

[Pet02] Seth Pettie. A faster all-pairs shortest path algorithm for real-weighted sparse graphs. In *International Colloquium on Automata, Languages, and Programming*, pages 85–97. Springer, 2002.

[PR05] Seth Pettie and Vijaya Ramachandran. A shortest path algorithm for real-weighted undirected graphs. *SIAM J. Comput.*, 34(6):1398–1431, 2005.

[PS14] Rasmus Pagh and Francesco Silvestri. The input/output complexity of triangle enumeration. In *Proceedings of the*

33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pages 224–233. ACM, 2014.

[RV11] Liam Roditty and Virginia Vassilevska Williams. Minimum weight cycles and triangles: Equivalences and algorithms. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 180–189. IEEE, 2011.

[RZ12] Liam Roditty and Uri Zwick. Replacement paths and k simple shortest paths in unweighted directed graphs. *ACM Trans. Algorithms*, 8(4):33:1–33:11, 2012.

[Sei95] Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of computer and system sciences*, 51(3):400–403, 1995.

[SZ99] Avi Shoshan and Uri Zwick. All pairs shortest paths in undirected graphs with integer weights. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 605–615, 1999.

[TT98] Hisao Tamaki and Takeshi Tokuyama. Algorithms for the maxium subarray problem based on matrix multiplication. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, 25-27 January 1998, San Francisco, California.*, pages 446–452, 1998.

[Vas08] Virginia Vassilevska. *Efficient algorithms for path problems in weighted graphs*. PhD thesis, Carnegie Mellon University, 2008.

[Vas12] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *STOC*, pages 887–898, 2012.

[Vas15] Virginia Vassilevska Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, pages 17–29, 2015.

[VW10] Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 645–654. IEEE, 2010.

[VW13] Virginia Vassilevska Williams and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. *SIAM Journal on Computing*, 42(3):831–854, 2013.

[WBK+] Ryan Williams, Arnab Bhattacharyya, Mikko Koivisto, Thore Husfeldt, and Michael Forbes. Personal communication.

[Wil05] Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2–3):357–365, 2005.

[Wil07] Ryan Williams. *Algorithms and Resource Requirements for Fundamental Problems*. PhD thesis, Carnegie Mellon University, Computer Science Department, 8 2007. CMU-CS-07-147.

[Wil10] R. Williams. A generalization of boolean matrix multiplication for order 3 tensors, 2010.

[Wil11] Virginia Vassilevska Williams. Faster replacement paths. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1337–1346, 2011.

[Wil14] Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 664–673. ACM, 2014.

[YZ97] Raphael Yuster and Uri Zwick. Finding even cycles even faster. *SIAM J. Discrete Math.*, 10(2):209–222, 1997.

[YZ04] 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*, pages 254–260. Society for Industrial and Applied Mathematics, 2004.

[Zwi02] U. Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *JACM*, 49(3):289–317, 2002.

A Prior Work On Improved Running Times

In this section we provide some of the prior algorithmic work on the various problems that we study. We limit our scope to algorithms that compute the problems below exactly. We note that faster approximation algorithms are known for many of them.

k-Cycle. The complexity of finding a k-Cycle in an n node, m edge graph depends heavily on whether the graph is directed and whether k is even or odd. In directed graphs, the fastest algorithm for finding k-cycles run in $\tilde{O}(n^\omega)$ time [AYZ16]. For sparse graphs, [AYZ97] obtained faster algorithms using matrix multiplicatin. When k is odd, the k-cycle problem in directed graphs is equivalent to the k-cycle problem in undirected graphs (see e.g. [Vas08]). When k is even, however, the problem is much easier in undirected graphs. Yuster and Zwick [YZ97] showed that a k-cycle in an undirected graph can be found in $\tilde{O}(n^2)$ time for any even constant k. Dahlgaard et al. [DKS17] recently extended this result, giving an $\tilde{O}(m^{2k/(k+2)})$ time algorithm for all even k.

Shortest Cycle. The Shortest Cycle problem in weighted graphs was recently shown to be solvable in $O(mn)$ time by Orlin and Sedeño-Noda [OS17]. In directed or undirected graphs with small integer weights in the interval $\{1, \ldots, M\}$, Roditty and Vassilevska Williams [RV11] showed how to find a shortest cycle in $\tilde{O}(Mn^\omega)$ time, generalizing a result by Itai and Rodeh [IR78] that obtained $O(n^\omega)$ for unweighted graphs.

APSP. In directed or undirected weighted graphs, the fastest algorithm for APSP runs in $n^3/2^{\Theta(\sqrt{\log n})}$ time [Wil14] for dense graphs. For APSP in sparse the fastest algorithm is $O(mn + n^2 \log \log n)$ time [Pet02] for directed graphs, and $O(mn \lg(\alpha(m,n)))$ in undirected graphs [PR05].

For undirected graphs with integer weights in $\{-M, \ldots, M\}$, Shoshan and Zwick [SZ99] provided an $\tilde{O}(Mn^\omega)$ time algorithm, generalizing Seidel's $O(n^\omega \log n)$ time algorithm for unweighted undirected graphs [Sei95]. Zwick [Zwi02] obtained an $O(M^{0.681}n^{2.532})$ time algorithm for APSP in directed graphs with integer weights in $\{-M, \ldots, M\}$; his algorithm for $M = 1$ gives the best known running time for APSP in directed unweighted graphs. Chan [Cha12] obtained $O(mn \log \log n/\log n)$ time for sparse undirected graphs, while for APSP in sparse directed graphs the $O(mn)$ BFS-based algorithm is currently the fastest.

Radius. The fastest known algorithm for Radius in directed or undirected graphs with large edge weights just uses the best known APSP algorithms above. For graphs with integer edge weights in $\{-M, \ldots, M\}$, Cygan et al. [CGS15] provide an $\tilde{O}(Mn^\omega)$ time algorithm; when setting $M = 1$ one gets the fastest known runtime of $\tilde{O}(n^\omega)$ for unweighted graphs.

Wiener Index. The best algorithms for Wiener index compute APSP and sum up all the pairwise distances. (See Mohar and Pisanski [MP88] for versions of the Floyd-Warshall and other algorithms for Wiener index.)

Replacement Paths (RP) and Second Shortest Path (SSP). RP and SSP for undirected graphs can be solved very efficiently: Malik et al. [MMG89] presented an $\tilde{O}(m)$ time algorithm. Nardelli et al. [NPW03] improved this runtime to $O(m\alpha(n))$ in the word-RAM model. The best runtime for both problems in directed graphs with arbitrary edge weights is $O(mn + n^2 \log \log n)$ by Gotthilf and Lewenstein [GL09]. For dense weighted graphs, the best algorithm use the best algorithms for APSP. Vassilevska Williams [Wil11] obtained an $\tilde{O}(Mn^\omega)$ time algorithm for RP and SSP for directed graphs with edge weights in $\{-M, \ldots, M\}$. For unweighted directed graphs, Roditty and Zwick [RZ12] gave a randomized combinatorial algorithm which computes replacement paths $\tilde{O}(m\sqrt{n})$ time.

Betweenness Centrality. The notion of betweenness centrality was introduced by Freeman [Fre77] in the context of social networks, and since then became one of the most important graph centrality measures in the applications. Brandes' algorithm [Bra01] computes the betweenness centrality of all nodes in time $O(mn + n^2 \log n)$ using a counting variant of Dijkstras algorithm. Similar to other papers in the area, [Bra01] neglects the bit complexity of the counters storing the number of pairwise shortest paths. This is reasonable in practice since the maximum number N of alternative shortest paths between two nodes tends to be small. By also considering N, the running time grows by a factor $O(\log N) = O(n \log n)$.

B Beating O(mn) for Many Graph Problems is Probably Hard

B.1 Undirected k-cycle reduces to radius
First we will demonstrate a reduction from k-cycle to Radius in the undirected weighted case. Radius will solve k-cycle at any graph density and thus we will get a lower bound of $mn^{1-o(1)}$ for all densities $m = n^{1+1/L}$.

To make the reduction more readable we will first prove a lemma that gives a reduction from minimum k-cycle to negative k-cycle in a k-circle-layered graph. The latter problem is defined as follows: given a directed weighted graph, find a k-cycle of negative total edge weight.

LEMMA B.1. *If the negative directed k-cycle problem on a k-circle-layered graph, G with weights in the range $[-R,R]$ can be solved in time $T(n,m,R)$ time then the directed minimum k-cycle problem with weights in the range $[-R,R]$ can be solved in $\tilde{O}(\lg(R)T(n,m,R))$ time.*

Proof. We use the color coding lemma (Lemma 2.2) to reduce the directed minimum k-cycle problem to directed min-

imum k-cycle in a k-circle-layered graph. Let the partitions of this graph G be $U_1, U_2, \ldots, U_k$.

Now, given any weight $T \in [-Rk, Rk]$, we can add $-T$ to all edges between U_1 and U_2, creating a new graph $G(T)$, and ask for a negative k-cycle in $G(T)$. If such a cycle C exists, then since C must go around the circle of the k-circle-layered graph, then the weight of C in $G(T)$ is its weight $w(C)$ in G, minus T. Notice that $w(C) - T < 0$ if and only if $w(C) < T$. Hence, for any T, using one negative k cycle query we can check whether the Min Weight k Cycle in G has weight $< T$ or not.

We can now binary search over the possible cycle lengths $O(\lg(R))$ times to find the minimum cycle length in G.

THEOREM B.1. *If Radius in an undirected weighted N node M edge graph can be computed in $f(N, M)$ time then the minimum weight directed k-cycle problem in n node, m edge graphs with weights in $[-R, R]$ can be solved in $\tilde{O}(f(O(n), O(m)) \log R + m)$ time.*

Proof. We take the minimum weight k-cycle problem with edge weights in the range $[-R, R]$ and use our previous reduction (from Lemma B.1) to the negative weight k-cycle problem in a k-circle-layered graph, G, with partitions U_1, U_2, $\ldots$, U_k. We will refer to the jth node in U_i by u_i^j. We will create a new graph G' on partitions $V_1, \ldots, V_k$ where V_i corresponds to the nodes in U_i of G. In particular, v_i^j corresponds to node u_i^j of G. See Figure 3 for an illustration of the construction.

Let $F = 20kR$. Let V_1' be a copy of the nodes in V_1.

Let the edges from $v_i^{(j)} \in V_i$ to $v_{i+1}^{(p)} \in V_{i+1}$ for $i \in [1, k-1]$ exist if there is an edge between $u_i^{(j)} \in U_i$ and $u_{i+1}^{(p)} \in U_{i+1}$. The weight of the edge is given by $w(v_i^{(j)}, v_{i+1}^{(p)}) = w(u_i^{(j)}, u_{i+1}^{(p)}) + F$.

We will introduce edges between $v_k^{(j)} \in V_k$ and $v'^{(p)}_1 \in V_1'$ if there is an edge between $u_k^{(j)} \in U_k$ and $u_1^{(p)} \in U_1$. The weight of the edge is given by $w(v_k^{(j)}, v'^{(p)}_1) = w(u_k^{(j)}, u_1^{(p)}) + F$.

Let $\hat{V}_i$ be $V_i \cup \{u_i\}$ where u_i is a new node connected to all nodes in V_i with edges of weight $3F/4 = 15kR$.

Let B be a set of $\lg(n) + 1$ nodes $b_0, b_1, \ldots, b_{\lg(n)}$ where each b_i is connected to every node $v_1^{(j)}$ where the ith bit of j is a 1 and connect b_i to every node $v'^{(j)}_1$ where the ith bit of j is a 0. Here the weights of these edges are $kF/2$ between V_1 and B and the weights are $kF/2 - kR$ between V_1' and B. Connect the node u_1' node to all nodes in B.

We add a node x which has edges of weight $kF - 1$ to all nodes in V_1, so only nodes in V_1 or x could possibly have radius $< kF$. Note that if $k > 1$ then the shortest path from x to nodes in V_2 is at least $kF - 1 + 20kR - R$, making it impossible for it to be the center if the radius is less than kF.

We add a node y which has edges of weight $kF/2$ to all nodes in V_1 and edges of weight $kF/2 - F/2$ to every node in V_k.

So the shortest path between $v_1^{(i)}$ and $v'^{(j)}_1$ when $i \neq j$ is $kF - kR$.

The shortest path between $v_1^{(i)}$ and $v'^{(i)}_1$ has a length of the shortest k-cycle, if one exists, through node $u_1^{(i)}$ plus kF. Because, all other choices of paths use more than k edges (thus giving a weight of at least $(k + 1/2)F$) or go though y or B which will result in a weight of $(k + 1/2)F$ or more. So, the shortest path will be the shortest k-cycle through node $u_1^{(i)}$ plus kF.

Any node v_1^t in V_1 whose copy is part of any k-cycle in G will have a path of length $< kF$ to all nodes in V_i where $i < k$. To see this, first note that v_1^t can reach some node in V_i with a path of length at most $(i-1)F + (i-1)R = F(i-1)(1 + 1/(20k))$. Now, since every two node in V_i are connected with a path of length $1.5F$ through node u_i, v_1^t can reach any node in V_i using a path of length at most $(i - 1 + 3/2 + (i-1)/(20k))F$ which for $i < k$ is at max $(k - 1/2 + (k-2)/(20k))F < (k - 0.48)F < kF$.

We need to ensure that any node in V_1 involved in a cycle has a path of length $< Fk$ to all nodes in V_k. The node y solves this problem by providing a path of length $(k - 0.5)F$ between all nodes in V_1 and V_k.

So every node $v_1^{(i)}$ in V_1 has shortest paths of length $< kF$ to every node in the graph except, possibly, to the node $v'^{(i)}_1$. If $v_1^{(i)}$ is involved in a negative k-cycle then there is a shortest path between $v'^{(i)}_1$ and $v_1^{(i)}$ of length $< Fk$. Thus, if there is a negative k-cycle then the radius of the graph will be $< Fk$.

We would like to show that radius in unweighted graphs is also hard from (unweighted) k-cycle.

LEMMA B.2. *If Radius in an undirected unweighted N node M edge graph can be computed in $f(N, M)$ time then a directed k-cycle problem in n node, m edge graphs can be found in $\tilde{O}(f(O(n), O(m)) + m)$ time.*

Proof. If there is a k-cycle then the radius will be k and otherwise the radius will be larger. The reduction is similar to before: we start with a k-circle-layered graph G with partitions $U_1, \ldots, U_k$ and reduce it to a new graph G' with roughly the same number of nodes. See Figure 4 for a depiction of this graph. In G', we have partitions $V_1, \ldots, V_k$ where V_i corresponds to U_i in G.

As before, let V_1' be a copy of the nodes in V_1.

Let the edges from $v_i^{(j)} \in V_i$ to $v_{i+1}^{(p)} \in V_{i+1}$ for $i \in [1, k-1]$ exist if there is an edge between $u_i^{(j)} \in U_i$ and $u_{i+1}^{(p)} \in U_{i+1}$.

We will introduce edges between $v_k^{(j)} \in V_k$ and $v'^{(p)}_1 \in V_1'$ if there is an edge between $u_k^{(j)} \in U_k$ and $u_1^{(p)} \in U_1$.
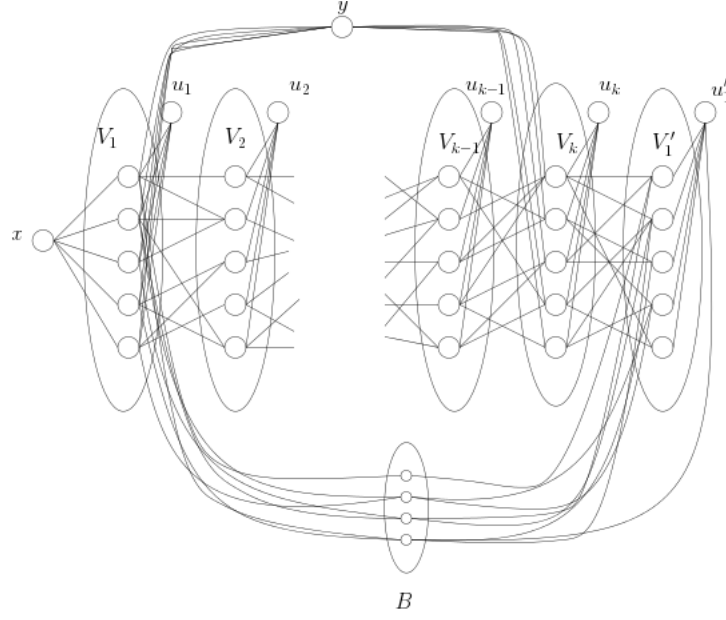
Figure 3: The Radius gadget.

Let $\hat{V}_i$ be $V_i \cup u_i$ where u_i is a node connected to all nodes in V_i. We connect u_i to u_{i+1} for all i $in[1, k-1]$ and connect u_k and u'_1. We additionally add edges between u_2 and every node in V_1.

Let B be a set of $\lg(n)+1$ nodes $b_0, b_1, \ldots, b_{\lg(n)}$ where each b_i is connected to every node $v_1^{(j)}$ where the ith bit of j is a 1. Let B' be a set of $\lg(n)+1$ nodes $b'_0, b'_1, \ldots, b'_{\lg(n)}$ where each b'_i is connected to every node $v'_1^{(j)}$ where the ith bit of j is a 0. We connect b_i and b'_i with a path of $k-2$ nodes.

We add a node x which to all nodes in V_1. We connect to x a path P containing nodes $p_1, \ldots, p_{k-1}$, so that nodes in V_1 are distance k from p_{k-1}. So only nodes in V_1 or x could possibly have radius $< k$.

The shortest path between $v_1^{(i)}$ and $v'_1^{(i)}$ has length k through B and B'. There is a path to u_i from any node in V_1 is $\leq k$. The shortest path from nodes in V_1 to nodes in $x \cup P$ is $\leq k$.

The shortest path between a node in V_1 and a node in V_i has length at most i for $i > 1$. Thus, every node in $v_1^{(i)} \in V_1$ is distance $\leq k$ from every node except, possibly $v_1^{(i)'}$.

If a k-cycle exists then that k-cycle corresponds to a node such that $v_1^{(i)'}$ and $v_1^{(i)}$ are at distance $\leq k$. If a k-cycle does not exist then there is no path from V_1 to V_2 to ... to V_k and back to the corresponding node in V'_1, thus the shortest path between the two nodes has length $> k$. Paths through the nodes u require extra edges and thus require at least $k+1$ edges to get to $v_1^{(i)'}$ from $v_1^{(i)}$.

We can detect a k-cycle by running radius and returning true if the radius is k or less and false otherwise.

B.2 Undirected k-cycle reduces to Wiener index Informally, The Wiener index in a graph is the sum of all pairwise distances in the graph. Formally:

Definition Let $d(v, u)$ be the (shortest path) distance between v and u in G. Then the Wiener index is

$$\mathbb{W}(G) = \sum_{v \in V} \sum_{u \in V} d(v, u).$$

THEOREM B.2. *If the Wiener index in an undirected weighted graph on N nodes and M edges can be computed in $f(N, M)$ time then the minimum weight k-cycle problem in N node, M-edge directed graphs can be solved in $\tilde{O}(f(N, M) + M)$ time.*

Proof. We take the minimum weight k-cycle problem with edge weights in the range $[-R, R]$ and use our previous reduction (from Lemma B.1) to the negative weight k-cycle problem on a k-partite graph, G, with partitions $U_1, U_2, \ldots, U_k$ such that edges only exist between partitions U_i and $U_{(i+1 \mod k)}$. We will describe the reduction graph G', see Figure 5 for a diagram of the gadget. G' contains a part V_i for each part U_i of G, for each $i \in \{1, \ldots, k\}$. In addition, there is a part V'_1 which contains copies of the nodes in V_1. For each i, call the jth node in V_i, v_i^j.

Let $F = 20kR$. Let the edges from $v_i^{(j)} \in V_i$ to $v_{i+1}^{(p)} \in V_{i+1}$ for $i \in [1, k-1]$ exist if there is an edge between $u_i^{(j)} \in$
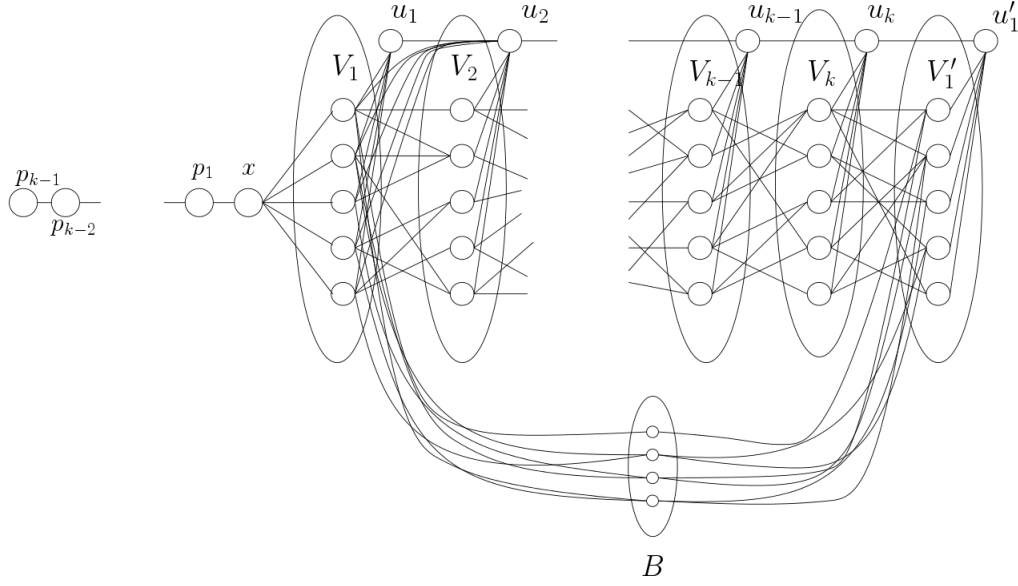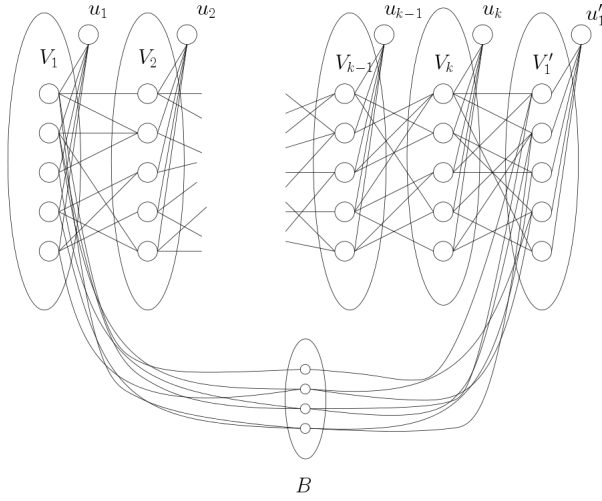
Figure 4: The unweighted radius gadget.

Figure 5: The Wiener Index gadget.

U_i and $u_{i+1}^{(p)} \in U_{i+1}$. The weight of the edge is given by $w(v_i^{(j)}, v_{i+1}^{(p)}) = w(u_i^{(j)}, u_{i+1}^{(p)}) + F$.

We will introduce edges between $v_k^{(j)} \in V_k$ and $v_1'^{(p)} \in V_1'$ if there is an edge between $u_k^{(j)} \in U_k$ and $u_1^{(p)} \in U_1$. The weight of the edge is given by $w(v_k^{(j)}, v_1'^{(p)}) = w(u_k^{(j)}, u_1^{(p)}) + F$.

Let $\hat{V}_i$ be $V_i \cup \{u_i\}$ where u_i is a node connected to all nodes in V_i with edges of weight $3F/4 = 15kR$. Let B be a set of $\lg(n) + 1$ nodes $b_0, b_1, \dots, b_{\lg(n)}$ where each b_i is connected to every node $v_1^{(j)}$ where the ith bit of j is a 1 and connect b_i to every node $v_1'^{(j)}$ where the ith bit of j is a 0. Where the

weights of these edges are $kF/2$ between V_1 and B and the weights are $kF/2 - kR$ between V_1' and B. By construction, the shortest path between $v_1^{(i)}$ and $v_1'^{(j)}$ when $i \neq j$ is $kF - kR$.

Let H be the graph composed of $\hat{V}_2 \cup \dots \cup \hat{V}_k$. Let H' be the graph composed of $\hat{V}_1 \cup H$. Let H'' be the graph composed of $H \cup \hat{V}_1'$. Let G' be the graph composed of $H \cup \hat{V}_1' \cup B$ where we add edges $\forall i \in [1, |V_1|]$ with weight $w(v_1^{(i)}, v_1'^{(i)}) = kF$.

Let the minimum k-cycle length in G going through $u_1^{(i)}$ be of length c_i. Note that $d_{G'}(v_1^{(i)}, v_1'^{(i)}) = kF + \min(0, c_i)$ by construction. Note that $d_{G'}(v_1^{(i)}, v_1'^{(j)}) = kF - kR$ for $i \neq j$. If we could access the sum of all of the $|V_1|^2$ pairwise distances between the nodes in V_1 and V_1' and see if those distances are, in sum, $(kF - kR)|V_1|^2 + |V_1|kR$ then we would know if a negative k-cycle exists.

We need to remove all the other distances first. Most easily we can calculate the distances to and from nodes in B. Let $w_B = \sum_{b \in B} \sum_{v \in G'} \delta_{G'}(b, v)$, we compute this in time $O(|B||G'|) = O(\lg(N)|G'|)$.

Next we can get the sum of the distances between any nodes in H by computing $\mathbb{W}(H)$. The graph has $M + N = O(M)$ edges in total. After this, we want to find the distances to and from the nodes in $\hat{V}_1$ to $G'/(\hat{V}_1' \cup B)$. We can get this from $\mathbb{W}(H') - \mathbb{W}(H)$. Next we want to find the distances too and from the nodes in $\hat{V}_1'$ to $G'/(\hat{V}_1 \cup B)$, we can get this from $\mathbb{W}(H'') - \mathbb{W}(H)$. Finally we need to account for the distance between u_1 and u_1', call this distance w_u we can run Dijstra's algorithm and find this distance in $\tilde{O}(M + N) = \tilde{O}(M)$ time.

Therefore the sum of all the pairwise distances between

nodes in V_1 and V_1' is equal to $\mathbb{W}(G') - (\mathbb{W}(H') - \mathbb{W}(H)) - (\mathbb{W}(H'') - \mathbb{W}(H)) - \mathbb{W}(H) - 2w_B - 2w_u$.

Thus, if $\mathbb{W}(G') - \mathbb{W}(H') - \mathbb{W}(H'') + \mathbb{W}(H) - 2w_B - 2w_u < (kF - kR)|V_1|^2 + |V_1|kR$ then there is a negative k-cycle. If $\mathbb{W}(G') - \mathbb{W}(H') - \mathbb{W}(H'') + \mathbb{W}(H) - 2w_B - 2w_u \geq (kF - kR)|V_1|^2 + |V_1|kR$ then there is no negative k-cycle.

The total time to get all the sums we need is $O(f(N, M + N) + M\lg(N)) = \tilde{O}(f(N, M) + M\lg(N))$.

We would like to show that *unweighted* Wiener Index is also hard from k-cycle.

LEMMA B.3. *If the Wiener Index can be computed in $f(N, M)$ time in an undirected unweighted graph then the directed k-cycle problem can be solved in $\tilde{O}(f(N, M) + M)$ time.*

Proof. If there is a k-cycle then the Wiener Index will be lower and otherwise the Wiener Index will be larger. We will form the graph G' similar to before. See Figure 6 for a depiction of this graph. Let the U_i and V_i and V_1' be as in the previous proof.

Let the edges from $v_i^{(j)} \in V_i$ to $v_{i+1}^{(p)} \in V_{i+1}$ for $i \in [1, k-1]$ exist if there is an edge between $u_i^{(j)} \in U_i$ and $u_{i+1}^{(p)} \in U_{i+1}$. We will introduce edges between $v_k^{(j)} \in V_k$ and $v_1'^{(p)} \in V_1'$ if there is an edge between $u_k^{(j)} \in U_k$ and $u_1^{(p)} \in U_1$.
Let $\hat{V}_i$ be $V_i \cup u_i$ where u_i is a node connected to all nodes in V_i. We connect u_i to u_{i+1} for all i in $[1, k-1]$ and connect u_k and u_1'. We additionally add edges between u_2 and every node in V_1.
Let B be a set of $\lg(n) + 1$ nodes $b_0, b_1, \ldots, b_{\lg(n)}$ where each b_i is connected to every node $v_1^{(j)}$ where the ith bit of j is a 1. Let B' be a set of $\lg(n) + 1$ nodes $b_0', b_1', \ldots, b_{\lg(n)}'$ where each b_i' is connected to every node $v_1'^{(j)}$ where the ith bit of j is a 0. We connect b_i and b_i' with a path of $k - 2$ nodes.

The shortest path between $v_1^{(i)}$ and $v_1'^{(i)}$ has length k through B and B'. There is a path to u_i from any node in V_1 is $\leq k$. The shortest path from nodes in V_1 to nodes in $x \cup P$ is $\leq k$.
A the shortest path between a node in V_1 and a node in V_i has length at most i for $i > 1$. Thus, every node in $v_1^{(i)} \in V_1$ is distance $\leq k$ from every node except, possibly $v_1^{(i)'}$.

If a k-cycle exists then that k-cycle corresponds to a node such that $v_1^{(i)'}$ and $v_1^{(i)}$ are at distance $\leq k$. If a k-cycle does not exist then there is no path from V_1 to V_2 to $\ldots$ to V_k and back to the corresponding node in V_1', thus the shortest path between the two nodes has length $> k$. Paths through the
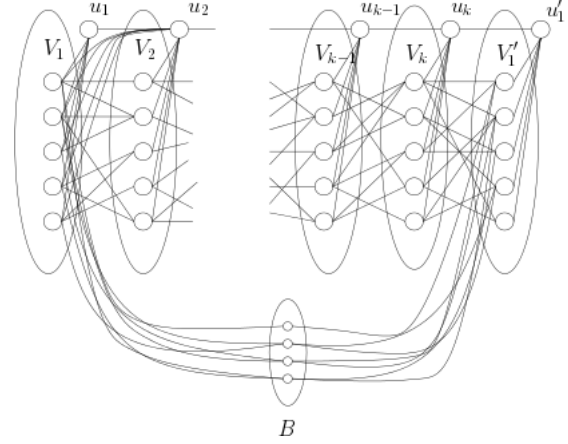

Figure 6: The unweighted Wiener Index gadget.

nodes u require extra edges and thus require at least $k + 1$ edges to get to $v_1^{(i)'}$ from $v_1^{(i)}$.

Let $H' = G'/(\hat{V}_1' \cup \hat{V}_1 \cup B \cup B')$, $H'' = G'/(\hat{V}_1' \cup B \cup B')$ and $H''' = G'/(\hat{V}_1 \cup B \cup B')$.

There are $O(k\lg(n))$ nodes in B, B' and the path between them. We can calculate these distance in $O(\lg(n)m)$ time. Call these distances d_B.

Similarly calculate the distance from nodes u_1 to $\hat{V}_1'$ and u_1' to $\hat{V}_1$ graph and call the sum of these two values d_u.

Note that $\mathbb{W}(H')$ captures the distances between nodes in H', the shortest paths between them never use nodes outside of H'. Similarly $\mathbb{W}(H'')$ and $\mathbb{W}(H''')$ each capture the distances between the nodes contained in them.

$\mathbb{W}(G') - \mathbb{W}(H'') - \mathbb{W}(H''') + \mathbb{W}(H') - 2d_B - 2d_u$ is equal to the sum of $\sum_{u \in V_1} \sum_{v \in V_1'} d(u, v)$.

Now let $d_x = \sum_{v_1^{(i)} \in V_1} \sum_{v_1^{(j)'} \in V_1' \text{ and } i \neq j} d(v_1^{(i)}, v_1^{(j)'})$. And note that $d_x = |V_1|(|V_1| - 1)k$.

Next note that $d(v_1^{(i)}, v_1^{(i)'}) = k + 1$ if there is no k-cycle through $v_1^{(i)}$.

So if there is no k-cycle then $\sum_{u \in V_1} \sum_{v \in V_1'} d(u, v) = |V_1|^2 k + |V_1|$ and if there is a k-cycle the sum is less.

There is a k-cycle if $\mathbb{W}(G') - \mathbb{W}(H'') - \mathbb{W}(H''') + \mathbb{W}(H') - 2d_B - 2d_u < |V_1|^2 k + |V_1|$ and there is no k-cycle if $\mathbb{W}(G') - \mathbb{W}(H'') - \mathbb{W}(H''') + \mathbb{W}(H') - 2d_B - 2d_u = |V_1|^2 k + |V_1|$.

COROLLARY B.1. *If undirected (unweighted/weighted) Wiener Index can be computed in $f(N, M)$ time on a graph of density $M = \tilde{\Theta}(N^{1+1/L})$ then the minimum weight (unweighted/weighted) k-cycle problem can be solved in $\tilde{O}(f(N, M) + M)$.*

B.3 Undirected k-cycle reduces to APSP A folklore reduction reduces Shortest Cycle in directed graphs to APSP in directed graphs as follows. Let G be the graph in which we want to find the Shortest Cycle. Compute APSP in G, and then for every edge add $d(u,v)+w(v,u)$ and take the minimum - this is the weight of the Shortest Cycle in G. Reducing directed Shortest Cycle to APSP in undirected graphs seems more problematic, as noted by Agarwal and Ramachandran [AR16]. In our paper, however, we were able to reduce Min Weight k-Cycle in a directed graph to Radius in undirected graphs. Radius in an undirected graph of course can easily be reduced to APSP: compute APSP and then set the radius to $\min_u \max_v d(u,v)$. Because of this, we immediately get the same lower bounds for APSP as for Radius and Min Weight k-Cycle. This reduction also works for unweighted graphs.

COROLLARY B.2. *If there is an $O(n^2 + mn^{1-\varepsilon})$ time algorithm for $\varepsilon > 0$ for APSP in m edge n node (directed/undirected, weighted/unweighted) graphs, then there is also an $O(n^2 + mn^{1-\varepsilon})$ time algorithm for Radius in m edge n node (directed/undirected, weighted/unweighted) graphs.*

C General CSP to Hyperclique

We will give the definitions of two versions of the degree-k-CSP problem. The maximization version and the exact weight version. Then we will show how maximum or exact weight ℓ-hyperclique finding solves these problems using a generalization of the argument from Williams [Wil07]. Finanlly, we will give a reduction between the unweighted (ℓ,k)-hyperclique problem and the degree-k-CSP problems.

Definition The maximum degree-k-CSP problem has a formula with n variables and m clauses. Each clause is a formula on n variables (though it may of course depend on fewer). For each clause c_i $i \in [1,m]$ there exists a d degree polynomial $p_i : \{0,1\}^n \to \{0,1\}$ such that c_i is satisfied on an assignment $\vec{a}$ when $p_i(\vec{a}) = 1$ and unsatisfied when $p_i(\vec{a}) = 0$.

The solution to a maximum k-CSP problem is a an assignment of the n variables, $\vec{a} = a_1, \ldots, a_n$ such that $\sum_{i=1}^m p_i(\vec{a})$ is maximized.

Definition A weighted degree-k-CSP problem has a formula with n variables and m clauses and two special constants $K_v, K_p \in [poly(n)]$. Each clause is a formula on n variables (though it may of course depend on fewer). For each clause c_i $i \in [1,m]$ there exists a k degree polynomial $p_i : \{0,1\}^n \to \{0,1\}$ such that c_i is satisfied on an assignment $\vec{a}$ when $p_i(\vec{a}) = 1$ and unsatisfied when $p_i(\vec{a}) = 0$.

The solution to a weighted k-CSP is a an assignment of the n variables, $\vec{a} = a_1, \ldots, a_n$ such that $\sum_{i=1}^n a_i = K_v$ and $\sum_{i=1}^m p_i(\vec{a}) = K_p$. [Wil07]

We will prove that finding a fast algorithm which breaks the (ℓ,k)-hyperclique hypothesis (for ℓ and k constant) results in algorithms for the maximum degree-k-CSP problem and weighted degree-k-CSP problem that run in $2^{(1-\varepsilon)n}$ for some $\varepsilon > 0$.

C.1 Degree-k-CSP to Weighted Hyperclique We will start by proving that the degree g coefficients of the polynomials p_i are bounded by 2^{g-1}. We will generalize the argument of R. Williams [Wil07].

LEMMA C.1. *Let p be a degree k polynomial that computes a boolean function. That is $p : \{0,1\}^n \to \{0,1\}$. The degree g coefficients of p are bounded by $[-2^{g-1}, 2^{g-1}]$ if $g > 0$ and the degree zero coefficient must be in $\{0,1\}$.*

Proof. R. Williams shows that this is true for $g = \{0,1,2\}$ for all n [Wil07]. We will prove this by induction.

If we have proven the statement for all coefficients of degree $g-1$ or less for all polynomials with $n < h+1$ and for coefficients of degree g for all polynomials with $n < h$ then we will show that the statement is true for coefficients of degree g and $n = h$. Let $q = p(x_1, \ldots, x_{h-1}, 1)$ and $r = p(x_1, \ldots, x_{h-1}, 0)$. Note $x_h \in \{0,1\}$ so $p = q$ when $x_h = 1$ and $p = r$ when $1 - x_h = 1$. Thus, the polynomial can be split into $p(x_1, \ldots, x_h) = x_h q(x_1, \ldots, x_{h-1}) + (1 - x_h) r(x_1, \ldots, x_{h-1})$.

Let $q_i =$ the sum of all the ith degree terms of q. Let $r_i =$ the sum of all the ith degree terms of r. Let $p_i =$ the sum of all the ith degree terms of p. Note that both r and q have only $h-1$ variables, and thus the condition has been proven for them by assumption.

We have that $p_i = x_h(q_{i-1} - r_{i-1}) + r_d$. Note that if a g degree term in p does not include variable x_h then the term comes from r_d, thus, the condition holds because r is degree g and has fewer then h variables. If a g degree term includes x_h then it comes from $x_h(q_{g-1} - r_{g-1})$, thus its coefficient is formed by the subtraction of two coefficients bounded by $[-2^{g-2}, 2^{g-2}]$ because q_{g-1} and r_{g-1} have degree less than g and less than h variables. The value of this coefficient is thus bounded by $[-2^{g-1}, 2^{g-1}]$. This covers all the g degree terms in p. Thus, the condition holds for g degree terms in polynomials with h variables.

Our base case is built from $g = \{0,1,2\}$ where this has been proven for all n and the fact that for all $g > 2$ and $h < 3$ the result is true because with at most 2 variables the maximum degree is 2 when p is multi-linear. Note that p must be multi-linear because $x_i \in \{0,1\}$ and thus $x_i^2 = x_i$.

By induction over g and h we prove the statement.

LEMMA C.2. *A degree-k-CSP formula can be transformed into a k-uniform hypergraph, H, on $\ell 2^{n/\ell}$ nodes ($\ell > k$) in $O(mn^k 2^{kn/\ell})$ time with two weights per edge $W_1(e)$ and*

$W_2(e)$. *Such that for every ℓ-hyperclique, h, the W_1 weight (sum over all edges of $W_1(e)$) is $\sum_{i=1}^{m} p_i(\vec{a})$ and the W_2 weight (sum over all edges of $W_2(e)$) is $\sum_{i=1}^{n} a_i$ for some assignment $\vec{a}$. Furthermore, for every assignment $\vec{a}$ there exists some ℓ-hyperclique whose weight corresponds to it. The weights $W_1(e)$ are bounded by $[-mn^k 2^k, mn^k 2^k]$. The weights $W_2(e)$ are bounded by n. When ℓ and k are constants.*

Proof. We will split the variables $x_1, \ldots, x_n$ into ℓ groups $X_i = [x_{1+in/\ell}, \ldots, x_{(i+1)n/\ell}]$. We create a node for all $2^{n/\ell}$ assignments for each X_i call this set V_i. This gives a total of $\ell 2^{n/\ell}$ nodes.

Let $S[i_1, \ldots, i_k]$ be the set of monomials from $p_1, \ldots, p_m$ that sets $X_{i_1}, \ldots, X_{i_k}$ are responsible for. We will consider set $S[i_1, \ldots, i_k]$ to be responsible for a monomial c if $S[i_1, \ldots, i_k]$ is the first set in lexicographical order that contains all the variables from c. Note that $S[i_1, \ldots, i_k]$ may contain variables from all m p_is. Every monomial will be in some $S[i_1, \ldots, i_k]$ because every monomial involves at most k variables.

Let $T[i_1, \ldots, i_k]$ be the set of indices of the sets of variables that an edge covering $X_{i_1}, \ldots, X_{i_k}$ are responsible for. We will have $j \in T[i_1, \ldots, i_k]$ if $j \in \{i_1, \ldots, i_k\}$ and $T[i_1, \ldots, i_k]$ is the first T in lexicographic order where $j \in \{i_1, \ldots, i_k\}$.

We create a hyper-edge for each choice of k nodes with at most 1 node from each V_i (we can not give two assignments to the same variable). We give an edge $e = (v_{i_1}, \ldots, v_{i_k})$ weight

$$W_1(e) = \sum_{c \in S[i_1, \ldots, i_k]} c(v_{i_1}, \ldots, v_{i_k})$$

and

$$W_2(e) = \sum_{j \in T[i_1, \ldots, i_k]} \sum_{t=1}^{n/\ell} x_{t+jn/\ell}.$$

We ideally want our hyper clique to be formed by all $\binom{\ell}{k}$ choices of k elements from $\{v_1, v_2, \ldots, v_\ell\}$ where $v_i \in V_i$ is an assignment of the variables in X_i. If we choose two $v_i, v_i' \in V_i$ to be part of our clique there will be no edge that contains both by construction, thus there will be no hyperclique.

Note that this means that every k-uniform ℓ-hyper-clique effectively gives an assignment to every variable. The clique has the weight of every monomial in $P = \sum_{i=1}^{m} p_i$ represented in W_1 exactly once. And, the clique has the weight of each variable represented in W_2 exactly once.

An edge could be responsible for at most mn^k possible monomials each of which have a coefficient at most 2^k. So the range of possible W_1 weights is $[-mn^k 2^k, mn^k 2^k]$.

Each edge can be responsible for at most kn/ℓ variables for a range of possible W_2 weights of $[0, kn/\ell]$. Both k and ℓ are constants and $k < \ell$ so for simplicity we will simply bound W_2 by n.

It takes $\ell 2^{n/\ell}$ time to produce all the nodes. Every edge takes $m\binom{n}{k}$ time to produce the weights for at most. There are

$O(\ell \binom{\ell}{k} 2^{kn/\ell})$ edges. This takes a total of $O(\ell m \binom{n}{k} \binom{\ell}{k} 2^{kn/\ell})$ time. Removing the constant factors we get $O(mn^k 2^{kn/\ell})$ time.

We will now use this lemma to show that maximum ℓ-hyperclique finding and exact weight ℓ-hyperclique finding solve maximum degree-k-CSP and weighted degree-k-CSP respectively.

LEMMA C.3. *If the maximum ℓ-hyperclique problem on a k-uniform hypergraph with weights in the range $[-mn^k 2^k, mn^k 2^k]$ can be solved in time $T(n)$ then the maximum degree-k-CSP problem can be solved in $O(T(\ell 2^{n/\ell}) + mn^k \ell 2^{kn/\ell})$ time. When ℓ and k are constants.*

Proof. We use Lemma C.2 to transform the maximum degree-k-CSP problem into a k-uniform hypergraph, H, on $\ell 2^{kn/\ell}$ nodes in $O(mn^k \ell 2^{kn/\ell})$ time. Next we create H' by taking every edge, e, from H and assigning it weight $W_1(e)$. Then we run maximum k-hyperclique on H' the maximum hyperclique found will correspond to the assignment $\vec{a}$ that maximizes $\sum_{i=1}^{m} p_i(\vec{a})$.

LEMMA C.4. *If the exact weight ℓ-hyperclique problem on a d-uniform hypergraph with weights in the range $[-mn^{k+1} 2^k \binom{\ell}{k}/\ell, mn^{k+1} 2^k \binom{\ell}{k}/\ell]$ can be solved in time $T(n)$ then the weighted degree-k-CSP problem can be solved in $O(T(\ell 2^{n/\ell}) + mn^k \ell 2^{kn/\ell})$ time. When ℓ and k are constants.*

Proof. We use Lemma C.2 to transform the maximum degree-k-CSP problem into a k-uniform hypergraph, H, on $\ell 2^{n/\ell}$ nodes in $O(mn^k \ell 2^{kn/\ell})$ time. Next we create H' by taking every edge, e, from H and assigning it weight $W_1(e) 2 \binom{\ell}{k} n/\ell + W_2(e)$ in H'. Note that the weight of W_2 is upper-bounded by n and there are $\binom{\ell}{k}$ edges in a hyper-clique so the low order bits correspond to $\sum_{i=1}^{n} a_i$ and the high order bits correspond to $\sum_{i=1}^{m} p_i(\vec{a})$.

Then we run exact weight ℓ-hyperclique on H' looking for weight $K_g = K_p 2 \binom{\ell}{k} n/\ell + K_v$ if we find a solution it will correspond to the assignment $\vec{a}$ that achieves $\sum_{i=1}^{m} p_i(\vec{a}) = K_p$ and $\sum_{i=1}^{n} a_i = K_v$.

C.2 Degree-k-CSP to Unweighted Hyperclique We will now extend the reduction to the unweighted version of (ℓ, k) hyperclique. This reduction will introduce a large polynomial overhead. However, for many choices of degree-k-CSPs the best known algorithms for both the maximization and weighted variants require exponential time. Notably, for max-3-SAT no $2^{(1-\varepsilon)n}$ algorithm is known.

THEOREM C.1. *If the unweighted ℓ-hyperclique problem on a k-uniform hypergraph can be solved in time $T(n)$ then the maximum degree-k-CSP problem can be solved in $O((mn^k)^{\binom{\ell}{k}} T(2^{n/\ell}) + mn^k 2^{kn/\ell})$ time. When ℓ and k are constants.*

Proof. We apply Lemma C.1 and generate a instance of the ℓ-hyperclique problem on a k-uniform hypergraph with weights in the range $[-mn^k 2^k, mn^k 2^k]$ in time $O(mn^k 2^{kn/\ell})$ time.

We want an unweighted version. We can do this by guessing the weights of the edges in our hyperclique and deleting edges that don't have these weights. In generating the hyper graph we will split the variables $x_1, \ldots, x_n$ into k groups $X_i = [x_{1+in/\ell}, \ldots, x_{(i+1)n/\ell}]$. We create a node for all $2^{n/\ell}$ assignments for each X_i call this set V_i. We can consider the edges $E[i_1, \ldots, i_k]$ which have exactly one vertex from each of $V_{i_1}, \ldots, V_{i_k}$.

We will make a guess about the weight of the $E[i_1, \ldots, i_k]$ edges, call the weight we guess $g[i_1, \ldots, i_k]$. There are total of $\binom{\ell}{k}$ edges in a hyperclique, we will make guesses for each edge. The guesses, g of the weights can be limited to the range $[-mn^k 2^k, mn^k 2^k]$. This gives a total of $O((mn^k)^{\binom{\ell}{k}})$ values we have to guess. Any hyperclique we find in a graph where we delete all $E[i_1, \ldots, i_k]$ edges have weight $g[i_1, \ldots, i_k]$ will have a total weight equal to the sum of all the guessed $g[i_1, \ldots, i_k]$. Formally, if $\binom{\{1,\ldots,\ell\}}{k}$ is the set of all k element subsets of $\{1, \ldots, \ell\}$ then the total weight of any hyperclique is $\sum_{\{i_1,\ldots,i_k\} \in \binom{\{1,\ldots,\ell\}}{d}} g[i_1, \ldots, i_k]$.

To solve the maximum k-degree CSP we exhaustively search through assignments of $g_{\{1,\ldots,k\}}, \ldots, g_{\{i_0,\ldots,i_{k-1}\}}, \ldots, g_{\{\ell-k,\ldots,\ell\}}$ in order from the maximum sum of guesses to the minimal sum. We delete all edges where the weight of the edge disagrees with the guess and then remove the weight from all the remaining edges. The first instance of ℓ-hyperclique that finds a ℓ-hyperclique has found the maximum weight ℓ-hyperclique.

Thus, we can solve $(mn^k)^{\binom{\ell}{k}}$ instances of ℓ-hyperclique in a k regular hypergraph with $N = \ell 2^{n/\ell}$ nodes and $O(2^{kn/\ell})$ hyperedges to solve maximum degree-k-CSP. This takes time $\tilde{O}((mn^k)^{\binom{\ell}{k}} T(2^{n/\ell}) + 2^{nk/\ell})$.

THEOREM C.2. *If the unweighted ℓ-hyperclique problem on a k-uniform hypergraph can be solved in time $T(n)$ then the weighted degree-k-CSP problem can be solved in $O((mn^{k+1})^{\binom{\ell}{k}} T(2^{n/\ell}) + mn^k 2^{kn/\ell})$ time.*

Proof. We apply Lemma C.2 and generate a instance of the ℓ-hyperclique problem on a k-uniform hypergraph with weights in the range $[-mn^{k+1} 2^k, mn^{k+1} 2^k]$ in time $O(mn^k 2^{kn/\ell})$ time and a target sum K_g.

We want an unweighted version. We can do this by guessing the weights of the edges in our hyperclique and deleting edges that don't have these weights. In generating the hyper graph we will split the variables $x_1, \ldots, x_n$ into k groups $X_i = [x_{1+in/\ell}, \ldots, x_{(i+1)n/\ell}]$. We create a node for all $2^{n/\ell}$ assignments for each X_i call this set V_i. We can consider

the edges $E[i_1, \ldots, i_k]$ which have exactly one vertex from each of $V_{i_1}, \ldots, V_{i_k}$.

We will make a guess about the weight of the $E[i_1, \ldots, i_k]$ edges, call the weight we guess $g[i_1, \ldots, i_k]$. There are total of $\binom{\ell}{k}$ edges in a hyperclique, we will make guesses for each edge. The guesses, g of the weights can be limited to the range $[-mn^k 2^k, mn^k 2^k]$. This gives a total of $O((mn^k)^{\binom{\ell}{k}})$ values we have to guess. Any hyperclique we find in a graph where we delete all $E[i_1, \ldots, i_k]$ edges have weight $g[i_1, \ldots, i_k]$ will have a total weight equal to the sum of all the guessed $g[i_1, \ldots, i_k]$. Formally, if $\binom{\{1,\ldots,\ell\}}{k}$ is the set of all k element subsets of $\{1, \ldots, \ell\}$ then the total weight of any hyperclique is $\sum_{\{i_1,\ldots,i_k\} \in \binom{\{1,\ldots,\ell\}}{k}} g[i_1, \ldots, i_k]$.

To solve the weighted k-degree CSP we exhaustively search through assignments of $g_{\{1,\ldots,k\}}, \ldots, g_{\{i_0,\ldots,i_{k-1}\}}, \ldots, g_{\{\ell-k,\ldots,\ell\}}$ where the sum of the guesses equals our target sum K_g. We delete all edges where the weight of the edge disagrees with the guess and then remove the weight from all the remaining edges. The first instance of ℓ-hyperclique that finds a ℓ-hyperclique has found a exact weight ℓ-hyperclique.

Thus, we can solve $(mn^{k+1})^{\binom{\ell}{k}}$ instances of ℓ-hyperclique in a k regular hypergraph with $N = \ell 2^{n/\ell}$ nodes and $O(2^{kn/\ell})$ hyperedges to weighted degree-k-CSP. This takes time $\tilde{O}((mn^{k+1})^{\binom{\ell}{k}} T(2^{n/\ell}) + 2^{nk/\ell})$.

Note that if $\ell >> k$ then the time to solve the hyperclique problem overwhelms the time it takes to produce the graph. Further note that Max ℓ-SAT is a special case of the maximum degree-ℓ-CSP problem.

COROLLARY C.1. *An algorithm which breaks the (ℓ, k)-hyperclique hypothesis runs in time $n^{\ell-\varepsilon}$ for some $\varepsilon > 0$.*

When ℓ and k constant such an algorithm can be used to solve the maximum degree-k-CSP problem and weighted degree-k-CSP problems in time $2^{(1-\varepsilon/\ell-o(1))n}$.

D Bounds for Graph Densities for NonInteger L

We get tight *mn* hardness for graphs with densities of the form $m = n^{1+1/L}$ for integer $L \geq 1$. For graphs with densities between $m = \tilde{o}(n^{1+1/L})$ and $m = \tilde{\omega}(1 + 1/(L+1))$ we get the lower bound of $n^{2+1/(L+1)}$. So our lower bound looks like a staircase (see Figure 7). The best known upper bound for all of these problems is *mn*. Ideally there would be a tight lower bound between the points $m = n^{1+1/L}$ and $m = n^{1+1/(L+1)}$. This does not exist right now, but we can present an improved lower bound for some densities in between $m = n^{1+1/L}$ and $m = n^{1+1/(L+1)}$. As a result we can rule out the "stair case" lower bound, making improvements between these densities even less likely.

At a high level we will use random color coding on the vertices with c colors. This will cause the node sets to be of
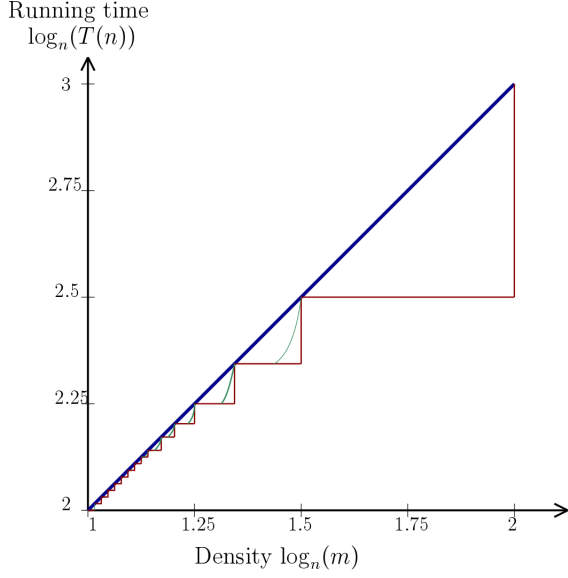
Figure 7: The heavy blue line represents the upper bound of $\tilde{O}(nm)$ and the jagged lighter red line represents the lower bound at all densities $m = \tilde{\Theta}(n^{1+1/L})$. The curved green lines show the lower bounds from appendix D.

size approximately n/c. The edge sets will effectively be colored by pairs of these colors (the color of their endpoints) for a total of c^2 colors. So the number of edges between nodes of two specifically chosen colors is in expectation m/c^2. If the graph is sparse to begin with then this coloring will cause the graph to become increasingly sparse. However, this reduction from denser to less dense graphs is inefficient; we need to consider c^k possible choices of colors of vertices. For densities close to $m = n^{1+1/L}$ we will get a better upper bound than before. Sadly this approach does not work for $L = 1$ because color coding a dense graph generates many dense graphs.

We will re-phrase and use Lemma 5 from Pagh and Silvestri's paper.

THEOREM D.1. *We randomly color a graph's nodes with c colors. Let the set of edges between color c_i and c_j be called $E_{i,j}$. Then $P(|E_{i,j}| > Ex/c^2) \leq 1/x^2$. [PS14]*

We apply this to get a self-reduction from k-cycle to k-cycle at a different density.

LEMMA D.1. *If we can solve the k-cycle problem in $T(n,m)$ time then for all $g \in [1,n]$ and all $c \in [1,n]$:*

$$T(n,m) = O(nm\lg(n)/g + nm\lg(n)/c + c^k T(n\lg(n)/c, m\sqrt{g}/c^{1.5}))$$

Proof. First we take the graph G and turn it into a k-circle-layered graph using Lemma 2.2. Then we check for all high

degree nodes with degree greater than gm/n to find what the minimum k-cycle through them is. There are n/g such nodes at most and we can use Dijstra to find the minimum k-cycle in time nm/g. Remove all the high degree nodes from the graph and call this new graph G'.

We randomly color the nodes of the altered graph G' where $|V'| = n$ and $|E'| = m$ with c colors. Let V_i be the set of vertices colored color i. Let $E_{i,j}$ be the number of edges between vertices colored i and j.

There are $O(n\lg(n)/c)$ nodes in sets $V_i > n\lg(n)/c$ in expectation. In $O(nm\lg(n)/c)$ time we can remove all nodes in overloaded sets.

By Theorem D.1 we have that the number of sets with $E_{i,j} > m\sqrt{g}/c^{1.5}$ in expectation is $1/(cg)$ removing every node in a set attached to these edges takes at most $nm\lg(n)/g$ time.

We can find all k-cycles in the remaining graph by trying all n^k choices of colorings of the k-cycle. Each subproblem has $kn\lg(n)/c$ nodes and $km\sqrt{g}/c^{1.5}$ edges.

The minimum k-cycle found in any of these steps can be returned.

Now we can state what our improved bound. It drops quickly away from the ideal mn bound, but we beat the "staircase" density depicted in red in Figure 7. This lower bound rules out the possibility that the true answer would be the staircase difficulty.

LEMMA D.2. *Assuming the k-clique where $k > 5$ and odd conjecture we have that for $c = n^\delta$ for all $\delta > 0$:*

$$T(n^{1+o(1)}/c, mn^{o(1)}/c^{1.5}) = \Omega(mn^{1-o(1)}c^{-k}).$$

This bound beats the bound from $k+1$-clique when $\delta < 4/((k-1)(k^2-k+4))$.

Proof. If the k-clique conjecture is true then there exists some $f(n) = n^{o(1)}$ such that k-clique takes $\omega(nm/f(n))$. Set the g from Lemma D.1 equal to $f(n)\lg(n)$.

Then k-clique takes $\omega(nm\lg(n)/g + nm\lg(n)/c)$ because c is a polynomially large factor. Then re-stating the formula from Lemma D.1 we get:

$$T(n^{1+o(1)}/c, mn^{o(1)}/c^{1.5}) = \Omega(mn^{1-o(1)}c^{-k}).$$

Let $N_c = n^{1+o(1)}/c$ and $M_c = mn^{o(1)}/c^{1.5}$. When k odd we show k-cycle is mn hard for density $m = n^{(k+1)/(k-1)} = n^{1-2/(k-1)}$ and $k+2$-cycle is $m'n$ hard for density $m = n^{(k+3)/(k+1)}$.

Note that if $c = 1$ we get that $M = N^{(k+1)/(k-1)}$. Further note that if $k > 5$ and $c = n^{8/((k-1)(k-3))}$ then $M = N^{(k+3)/(k+1)+o(1)}$. So with $c \in [1, n^{8/((k-1)(k-3))}]$ spans the distance between the two densities we care about for $k > 5$.

Next recall that the lower bound we have on graphs between density $m = n^{(k+1)/(k-1)}$ and $m = n^{(k+3)/(k+1)}$ is $\Omega(n^{2+2/(k+1)-o(1)})$. For what values of c do we beat this lower bound? Well the question can be reformulated as: for what values of c is $N^{2+2/(k+1)-o(1)} = o(mn^{1-o(1)}c^{-k})$. After some algebraic manipulation we find that $c = \tilde{o}\left(n^{4/((k-1)(k^2-k+4))}\right)$. So when $c = n^\delta$ and $k > 5$ we have an improved lower bound when $\delta < 4/((k-1)(k^2-k+4))$.