k-SUM Hardness Implies Treewidth-SETH

Michael Lampis
Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France

Abstract

We show that if k-SUM is hard, in the sense that the standard algorithm is essentially optimal, then a variant of the SETH called the Primal Treewidth SETH is true. Formally: if there is an $\varepsilon > 0$ and an algorithm which solves SAT in time $(2-\varepsilon)^{\mathrm{tw}}|\phi|^{O(1)}$, where tw is the width of a given tree decomposition of the primal graph of the input, then there exists a randomized algorithm which solves k-SUM in time $n^{(1-\delta)\frac{k}{2}}$ for some $\delta > 0$ and all sufficiently large k. We also establish an analogous result for the k-XOR problem, where integer addition is replaced by component-wise addition modulo 2.

An interesting aspect of our proof is that we rely on two key ideas from different topics. First, inspired by the classical perfect hashing scheme of Fredman, Komlós, and Szemerédi, we show that k-SUM admits an interactive proof protocol using integers of absolute value only $O(n^{k/2})$. Second, using the intuition that SAT formulas of treewidth tw can encode the workings of alternating Turing machines using tw bits of space, we are able to encode this protocol into a formula of treewidth roughly $\frac{k}{2}\log n$ and obtain the main result.

As an application of our reduction we are able to revisit tight lower bounds on the complexity of several fundamental problems parameterized by treewidth (Independent Set, Max Cut, k-Coloring). Our results imply that these bounds, which were initially shown under the SETH, also hold if one assumes the k-SUM or k-XOR Hypotheses, arguably increasing our confidence in their validity.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases SETH, Treewidth, Parameterized Complexity, Fine-grained Complexity

Funding This work was supported by ANR project ANR-21-CE48-0022 (S-EX-AP-PE-AL).

Contents

1 Introduction

1.1 Executive Summary

The main result of this paper is to establish the following implication: if there exists a faster-than-expected algorithm solving SAT parameterized by treewidth, then there exists a faster-than-expected (randomized) algorithm for k-SUM and k-XOR. Equivalently, the k-SUM Hypothesis implies the tw-SETH.

The immediate application (and initial motivation) of this work is in parameterized complexity, where our results establish that several DP algorithms for standard problems (Independent Set, Max Cut, k-Coloring) parameterized by treewidth are optimal *under the* k-SUM Hypothesis; this is new evidence for these important lower bounds, which were previously known under the SETH and its variants. Beyond this immediate application, our result is, to the best of our knowledge, one of the first non-trivial implications reducing a Sum-based hypothesis to a SAT-based hypothesis. This is valuable in the wider line of research attempting to draw connections among the numerous hypotheses in fine-grained complexity and offers some new information about which hypotheses should be considered more solid.

An interesting aspect of our result is that we obtain it by combining intuitions from complexity theory (space-bounded non-deterministic or alternating machines) and perfect hashing schemes. Our starting point is a natural question about k-SUM: what relation should one expect to have between the number of given integers (n) and their values? It is known that one can assume without loss of generality that the maximum absolute value is at most (roughly) $O(n^k)$ and this can be achieved via hashing. Could one go further?

We expect the answer to this question to be negative, barring some very clever idea. However, we work around this difficulty and hash an arbitrary k-SUM instance into one where all integers have absolute values $O(n^{k/2})$ by *cheating* in the sense that we change the rules of the question. In particular, we consider an interactive proof setting where a verifier challenges a prover to produce a solution in a hashed instance with small absolute values, and then the verifier is allowed to pose a follow-up challenge to eliminate false positives. The key idea is then that, because SAT parameterized by treewidth captures the workings of non-deterministic machines with bounded space and a small number of alternations, this protocol can be encoded in a SAT formula of treewidth (roughly) $\frac{k}{2} \log n$. Hence, our proof is an example of how intuitions from complexity theory can help us make non-trivial connections between problems which initially look quite different.

1.2 Background

The investigation of fine-grained complexity hypotheses and the algorithmic lower bounds they imply has been a topic of intense study in the last two decades. The three most prominent such hypotheses are probably the Strong Exponential Time Hypothesis (SETH), the 3-Sum Hypothesis, and the All-Pairs Shortest Paths (APSP) Hypothesis. Numerous tight lower bounds are known based on these hypotheses (we refer the reader to the survey of Vassilevska Williams [58]) and this is an area of very active research.

One weakness of this line of work is that at the moment these three main hypotheses are believed to be orthogonal, in the sense that neither one is known to imply any of the others. Indeed, determining the relations between these hypotheses is sometimes stated as a major open problem in the field [59], even though there is some tentative evidence that some potential reductions may be impossible [21]. Compounding this problematic situation

further, not only is it hard to classify the main hypotheses in order of relative strength, but there has also been a proliferation of other plausible hypotheses in the literature, which often have no clear relation to the three mentioned ones (we review some examples below).

Our goal in this paper is to attempt to ameliorate this situation in a direction that is of special interest for a specific well-studied sub-field of fine-grained complexity, namely the fine-grained complexity of *structurally parameterized* problems. In this area, numerous results are known proving that dynamic programming algorithms used in conjunction with standard parameters, such as treewidth, are optimal under the SETH. In particular, the pioneering work of Lokshtanov, Marx, and Saurabh [53] started a line of work which succeeded in attacking problems for which an algorithm running in time $c^{\mathrm{tw}}n^{O(1)}$ was known, for c a constant, and showing that an algorithm with running time $(c-\varepsilon)^{\mathrm{tw}}n^{O(1)}$ would falsify the SETH[1]. Results of this type have been obtained for virtually all the major problems that admit single-exponential algorithms parameterized by treewidth, as well as for other parameters, such as pathwidth, clique-width, and cutwidth (we review some results below).

Although this line of research has been quite successful, it was somewhat undesirable that the only evidence given for these numerous lower bounds was the SETH, a hypothesis whose veracity is not universally accepted. This motivated a recent work of Lampis [49] where it was shown that many (perhaps most) of these lower bounds can be obtained from a weaker, more plausible assumption, called the Primal Pathwidth SETH (pw-SETH). One of the main advantages of this approach is that the pw-SETH is more credible, as it was shown to be implied by several assumptions other than the SETH. Nevertheless, this line of work still made no connection between SAT-based hypotheses and hypotheses from the two other main families (Sum-based and APSP-based) and hence all evidence for the optimality of DP algorithms parameterized by treewidth still mostly rests on variants of the SETH.

1.3 Our results

In this paper we make what is, to the best of our knowledge, one of the first such connections between a Sum-based and a SAT-based fine-grained hypothesis and as our main application obtain new evidence for several (known) fine-grained lower bounds for standard problems parameterized by treewidth. In a nutshell, our main result is that a (slightly) stronger variant of the 3-Sum Hypothesis, implies a (weaker, but still quite useful) variant of the SETH.

In order to be more precise, let us define exactly the relevant hypotheses, starting with two Sum-related hypotheses. The k-SUM problem is the following: we are given k arrays (for some fixed k), each containing n integers and the goal is to decide if it is possible to select one integer from each array so that the sum of the selected integers is 0. It is a well-known fact that a meet in the middle approach can solve this problem in time $\widetilde{O}(n^{\lceil \frac{k}{2} \rceil})$ and the research for faster algorithms has been extremely active [10, 22, 24, 38, 39]. The k-SUM Hypothesis states that the meet in the middle algorithm is almost optimal, meaning that for all fixed $k \geq 3$ and $\varepsilon > 0$ it is impossible to solve the problem in time $O(n^{\lceil \frac{k}{2} \rceil - \varepsilon})$ (even for randomized algorithms). Observe that since the k-SUM Hypothesis is stated for all $k \geq 3$ it implies the 3-Sum Hypothesis, which is the special case for $k = 3$. It is currently unknown whether the two hypotheses are in fact equivalent. A close variant of these problems is the k-XOR problem, where rather than summing integers we are summing boolean vectors (component-wise modulo 2). The meet in the middle algorithm is similarly conjectured to be

[1] Here tw denotes the width of a given tree decomposition of the input graph. Throughout the paper we assume the reader is familiar with the basics of parameterized complexity, as given in standard textbooks [27]

optimal and this is referred to as the k-XOR Hypothesis. It is currently unknown whether the k-XOR Hypothesis is stronger or weaker than the k-SUM Hypothesis [29].

Let us also recall some relevant SAT-related hypotheses. The aforementioned pw-SETH states that the "obvious" dynamic programming algorithm for solving SAT parameterized by pathwidth is optimal. More precisely, it states that for all $\varepsilon > 0$, it is impossible to decide if a CNF formula ϕ given together with a path decomposition of its primal graph of width pw is satisfiable in time $(2-\varepsilon)^{\mathrm{pw}}|\phi|^{O(1)}$. Observe that the SETH trivially implies the pw-SETH. Replacing path by tree decompositions in the statement of the pw-SETH gives us the tw-SETH, which was first explicitly posited by Iwata and Yoshida [42]. Since every path decomposition is a tree decomposition, the pw-SETH trivially implies the tw-SETH.

Having set the groundwork, we are now ready to informally state the main result of this paper.

▶ **Theorem 1** (Informal). *The following statements hold:*

1. *The k-SUM hypothesis implies the tw-SETH.*
2. *The k-XOR hypothesis implies the tw-SETH.*

More strongly, we have the following: if the tw-SETH is false, then there exists an $\varepsilon > 0$ such that for sufficiently large k, both k-SUM and k-XOR admit randomized algorithms running in time $n^{(1-\varepsilon)\frac{k}{2}}$.

A summary of the relations between several hypotheses, where our result is placed into context, is given in Figure 1.

On a high level, the heart of our main result is a randomized reduction which transforms a given k-SUM or k-XOR instance into a CNF formula of treewidth $\frac{k}{2}\log n$. The technical part of this paper is devoted to describing this transformation and proving that it works correctly with high probability.

It is worth noting here that the k-SUM Hypothesis is usually stated in a quite careful way, where the exponent is $\lceil \frac{k}{2} \rceil - \varepsilon$, that is, it makes a difference whether k is odd or even. In contrast to this, we establish that if the tw-SETH were false, then we would strongly falsify the k-SUM Hypothesis, in the sense that we would be able to improve upon the standard algorithm not only by an additive but by a multiplicative constant (albeit, for sufficiently large k). As a result, in the remainder of the paper it will not be crucial whether k is odd or even (and for practical purposes, we will usually assume it is even).

Applications In light of the known barriers to proving that the SETH implies the 3-Sum Hypothesis, the question of finding plausible variants of the two conjectures such that one implies the other becomes inherently interesting. Nevertheless, our motivation is a little more precise than just looking for a connection. In a sense, we are searching for new evidence for a hypothesis which is already known to imply lower bounds of major importance to parameterized complexity.

To put it more clearly, the immediate application of our results is that we are able to revisit several lower bounds on the complexity of standard problems parameterized by treewidth. These lower bounds were initially known under the SETH [53] and were more recently shown under the pw-SETH [49]. Our main reduction allows us to also obtain them as consequences of the k-SUM and k-XOR Hypotheses, hence arguably increasing our confidence in their validity.

To obtain these results we rely in part on the work of Iwata and Yoshida [42] who showed that falsifying the tw-SETH is equivalent to improving the state-of-the-art DP algorithm

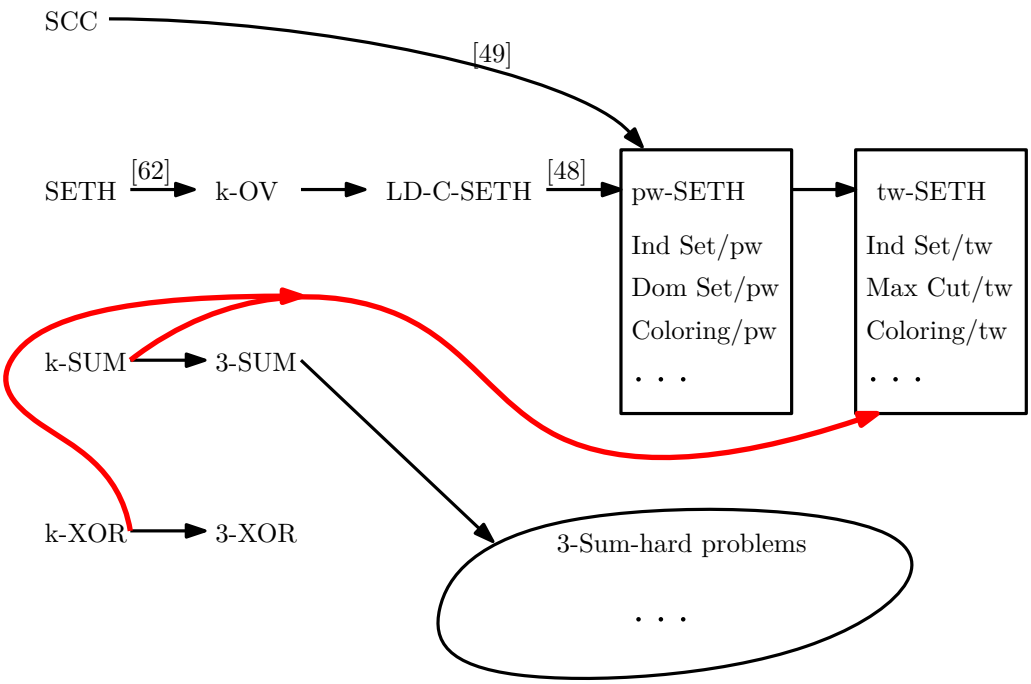

Figure 1 Summary of the relations between some notable fine-grained hypotheses with the new implications of this paper marked in red. In particular, the k-SUM and k-XOR Hypotheses now imply tight lower bounds for several standard problems parameterized by treewidth. We give definitions of all hypotheses and references to all other relations in Section 2, but mention briefly that SCC refers to the Set Cover Conjecture and LD-C-SETH refers to the SETH for n-input boolean circuits of size s and depth εn.

for several other problems parameterized by treewidth. By using their reductions, and also adding some new results building on the framework of [49] we obtain the following:

▶ **Theorem 2.** *If the k-SUM hypothesis is true, or the k-XOR hypothesis is true, then all the following statements are true, for all $\varepsilon > 0$:*

1. *There is no algorithm solving Independent Set in time $(2-\varepsilon)^{tw}n^{O(1)}$.*
2. *There is no algorithm solving Independent Set in time $(2-\varepsilon)^{cw}n^{O(1)}$, where cw is the width of a given clique-width expression.*
3. *There is no algorithm solving Max Cut in time $(2-\varepsilon)^{tw}n^{O(1)}$.*
4. *For all $k \geq 3$, there is no algorithm solving k-Coloring in time $(k-\varepsilon)^{tw}n^{O(1)}$.*

The first two items of Theorem 2 follow immediately from our main result and the results of [42], where it was shown that achieving the stated running time for Independent Set for parameters treewidth and clique-width is equivalent to falsifying the tw-SETH. The third item also follows from our result and a reduction from the result of [42] for Max-2-SAT. The last item is a reduction we give here, which, however, relies on standard techniques and straightforwardly generalizes the reduction from pw-SETH to k-Coloring given in [49].

These results should be interpreted as a *small representative sample* of what is possible. Indeed, one of the main thrusts of the work of Lampis [49] was to show how SETH-based lower bounds can, with a minimal amount of effort, be transformed to pathwidth-preserving reductions from SAT. This allows us to lift SETH-based lower bounds to pw-SETH-based ones and [49] verified that this can be done for a wide variety of problems (see [40, 51] for

follow-up works). It is not hard to see that in the vast majority of cases, pathwidth-preserving reductions are also treewidth-preserving, so we expect that, as a rule, it should not be a problem to lift lower bounds for problems parameterized by treewidth to tw-SETH-based bounds, and hence obtain k-SUM-hardness for many other problems. However, to keep the presentation manageable, we only focus on the problems mentioned above.

1.4 Comparison with previous results

Let us give some context about SAT-based and Sum-based hypotheses and their relations, explaining why we have selected to focus on these particular hypotheses.

As mentioned, the question of whether the 3-Sum Hypothesis implies or is implied by the SETH is a major open problem in fine-grained complexity theory. In one direction, it was established in [21] that if one could show that the SETH implies the 3-Sum hypothesis (or the APSP hypothesis), then the *non-deterministic* version of the SETH would be false. This poses a natural barrier to proving an implication in one direction. The possibility of a reduction in the converse direction is less studied, probably because reducing 3-Sum to SAT in a way that would show that the 3-Sum Hypothesis implies the SETH would likely require us to produce a CNF formula of bounded arity with $2 \log n$ variables (so that a purported algorithm would give a sub-quadratic algorithm for 3-Sum). This seems, however, impossible, because then the total size of the formula would be poly-logarithmic in n, meaning that we would be efficiently compressing the given instance. It is therefore natural that we attempt to reduce to a weaker variant of the SETH. That being said, to the best of our knowledge, there is no known barrier against showing that the 3-Sum Hypothesis implies a hypothesis that is weaker than the SETH but stronger than the tw-SETH, such as for example, the SETH for circuits of linear depth, mentioned in Figure 1, or even the SETH for CNF formulas of unbounded arity.

The above discussion partly explains why we attempt to prove an implication towards a weaker variant of the SETH and we have already explained why this weaker variant is still useful for our application (lower bounds for treewidth-based DP). Let us also explain why we start from the hypothesis that k-SUM, rather than 3-Sum, is hard. For this, it is instructive to revisit a recent result of [49], which established that the k-OV Hypothesis implies the pw-SETH. Recall that this hypothesis states that k-Orthogonal Vectors cannot be solved faster than roughly n^k time[2]. The result of [49] is then obtained by constructing a formula with pathwidth $k \log n$. Then, a purported algorithm solving SAT in $(2-\varepsilon)^{\text{pw}}$ would run in time $n^{(1-\delta)k}$ for the instances constructed by the reduction. There is, however, one complication with this strategy, which is that the purported SAT algorithm is allowed to have an arbitrary polynomial dependence on the formula size (which logically should be at least linear in the original input size n). In order to render this extra dependence irrelevant, we then need to allow k to become sufficiently large, and this is the reason why the results of [49] only establish that the k-OV Hypothesis implies the pw-SETH, rather than that the 2-OV Hypothesis does (which would have been stronger). In a similar fashion, we are forced to rely on the k-SUM Hypothesis.

k-SUM is not a brute-force problem We have therefore explained why it is natural to start from k-SUM and k-XOR, and why we need to reduce to a weaker variant of the SETH. Let us now also explain how the problem we are solving is different (and significantly

[2] See the next section for a more careful definition.

harder) from previous work, and in particular, from the proof that the k-OV Hypothesis implies the pw-SETH we mentioned above. As we explain in more detail in Section 3 the crucial difference is that whereas for the k-OV problem the (conjectured) best algorithm is essentially a brute-force algorithm that tries all solutions, this is *not* the case for k-SUM or k-XOR. Informally, it is easy to encode the workings of a brute-force algorithm with a CNF formula and then show that if we could decide the satisfiability of the formula faster than expected, we would be able to predict the answer given by the algorithm. This becomes much more complicated when the algorithm does something more clever than non-deterministically picking a solution and then verifying it.

The fact that the "obvious" brute force algorithm for 3-Sum is not optimal is a known obstacle to reductions from this problem. In fact, it was already observed by Pătraşcu [55] who was attempting to reduce 3-Sum to various other problems. Pătraşcu's solution was to reduce 3-Sum to a 3-Sum-Convolution problem, where the obvious algorithm is (hypothesized to be) optimal and indeed this problem has proven extremely useful in reductions establishing 3-Sum hardness. Unfortunately, this solution is not suitable for us because the key property of the Convolution problem is that the index of the last integer to be selected is fully specified from the indices of the others. For $k = 3$ this makes brute-force optimal, but for larger k this only decreases the obvious search space from n^k to n^{k-1}, whereas we need to bring this down to $n^{k/2}$. For this reason we need to rely on completely different ideas to formulate our reduction from k-SUM and k-XOR to SAT.

To summarize, we have explained why we are reducing from k-SUM (rather than 3-Sum) to SAT. We then explained that the main challenge in making the reduction efficient enough is that the (conjectured) optimal algorithm for k-SUM is much more clever than a brute-force algorithm that guesses a solution and then verifies it. We will therefore be forced to exploit the full power afforded to us by an algorithm solving SAT given a graph decomposition and in particular we will crucially rely on the extra power of tree over path decompositions.

1.5 Techniques and Proof Overview

We will formulate a reduction from k-SUM or k-XOR to SAT instances of treewidth essentially $\frac{k}{2}\log n$. As a result, if there is an algorithm solving SAT in time $(2-\varepsilon)^{\mathrm{tw}}|\phi|^{O(1)}$, we will obtain an algorithm for k-SUM and k-XOR with complexity $n^{(1-\delta)\frac{k}{2}}$. Let us describe step-by-step our approach, the obstacles we face, and our solutions. Along the way, we will clarify the subtle but intriguing point of why the consequence we obtain is the tw-SETH, rather than the pw-SETH.

The first step is to recall that SAT formulas of pathwidth pw have a tight correspondence with non-deterministic algorithms using space $\mathrm{pw} + O(\log n)$, as shown by Iwata and Yoshida [42]. This seems like an encouraging start, as k-SUM is clearly in NL. However, the obvious non-deterministic logarithmic space algorithm for k-SUM is to guess the k indices of the solution and then verify it. This algorithm uses space $k \log n$, therefore corresponds to a formula of pathwidth $k \log n$, rather than $\frac{k}{2}\log n$. This is exactly the point where the obstacle we discussed above (that the best algorithm for k-SUM is not brute-force search) manifests itself.

We are therefore motivated to come up with an alternative non-deterministic algorithm for k-SUM, using space only roughly $\frac{k}{2}\log n$. This space is clearly not enough to store the indices of all selected elements. However, an alternative approach to solve k-SUM (and Subset Sum) non-deterministically is to consider elements one by one and maintain a counter storing the sum (rather than the indices) of selected elements. If the maximum absolute value in the input is W, this algorithm uses space $\log W + O(\log n)$, so it would

suffice if $W = O(n^{k/2})$. Unfortunately, k-SUM under this restriction is not known to be equivalent to the general case. Instead, what is known is that by applying appropriate hash functions we can edit an arbitrary k-SUM instance to ensure that $W = O(n^k)$ (see Lemma 2.2 or [29]). This seems like a dead-end because it gives the same space complexity as the trivial algorithm. Nevertheless, we will push this idea by attempting to further decrease W.

Of course, it seems (to us) hopeless to attempt to show that k-SUM remains hard on instances with $W = O(n^{k/2})$. To see why, recall that the way that W can be brought down to roughly n^k is to apply to the input a hash function that is (almost-)linear, meaning it (almost) satisfies $h(x+y) = h(x)+h(y)$ for all inputs x, y. Applying this function means that all k-tuples which sum to 0 become hashed k-tuples that sum to 0; while any other k-tuple will (hopefully) map to 0 with probability $O(\frac{1}{W})$. Since there are n^k possible k-tuples, if we set W a bit larger that n^k we can claim by union bound that we have no false positives. However, if W is smaller than n^k the probability of false positives rises quite quickly.

In order to deal with this obstacle we return to the root of our problems, which is the meet in the middle algorithm. Observe that k-SUM and k-XOR can be seen as instances of List Disjointness: if L is the set of the (at most $n^{k/2}$) sums constructible from the first $k/2$ arrays and R is the set of sums constructible from the rest, the problem is essentially to check if $L \cap R \neq \emptyset$. The meet in the middle algorithm relies on the fact that List Disjointness for $|L| = |R| = N$ takes $\widetilde{O}(N)$ rather than $O(N^2)$ time to solve. This formulation nicely meshes with the idea of using hash functions, because one way to test if $L \cap R \neq \emptyset$ is to apply a hash function to the two sets and then check disjointness in the hashed values. This seems to improve our situation because we are mapping a collection of $|L \cup R| = O(n^{k/2})$ elements (compared to n^k k-tuples). However, in order to produce a collision-free hash function, we need the range we are mapping to to be *quadratic* in the number of elements (due to the birthday paradox). So this still gives W in the order of n^k.

We now arrive at the key idea of our construction, which will make clear why we need to use treewidth rather than pathwidth. We are in a situation where we wish to map $N = n^{k/2}$ elements (the set $L \cup R$) into a range of size roughly N, while avoiding collisions, so we draw some ideas from the classical 2-level hashing scheme of Fredman, Komlós, and Szemerédi [36]. The main idea of this scheme is to use an initial *main* hash function to map the N elements into N buckets; then, if the hash function is good enough, even though there will be some collisions, all buckets will have small load; we can therefore use a secondary hash function inside each bucket to eliminate the remaining collisions. In particular, the secondary hash function can afford to be quadratic in the size of each bucket and we know that in this case the secondary hash function is collision-free with probability at least $\frac{1}{2}$.

In order to illustrate how this idea will be used in our setting consider the following *Interactive Proof* protocol for k-SUM (formulated as List Disjointness). A verifier selects a random *main* hash function h^* which maps the $N = O(n^{k/2})$ elements of $L \cup R$ to a range of size N. The prover is then challenged to produce a value v_1 such that there exist $x \in L$ and $y \in R$ which satisfy $h^*(x) = v_1 = h^*(y)$. The prover can clearly do this if a solution exists, but he will also be able to do it if h^* has some collisions and this will happen with high probability. Suppose, however, that h^* is *balanced*, in the sense that with high probability, the maximum load M of any bucket is sub-polynomial in N ($M = N^{o(1)}$). The verifier then selects a random secondary hash function h_2 with range M^2, so with probability at least $\frac{1}{2}$ this function is collision-free for the bucket v_1. The prover is then challenged to produce $x \in L, y \in R$ such that $(h^*(x), h_2(x)) = (v_1, v_2) = (h^*(y), h_2(y))$, that is, the prover is challenged to produce a collision in h_2 *while respecting the commitment made* for the value of h^*. If no true solution exists, this will be possible with probability at most $\frac{1}{2}$,

and repeating this second step many times can make this probability very small.

Observe that the interactive protocol we sketched above has some positive and some negative aspects. On the positive side, the *memory* the verifier needs to use is as small as desired: we need to keep track of the prover's initial response ($\frac{k}{2}\log n$ bits), while the responses given in each round consist of $o(k \log n)$ additional bits (because $M^2 = N^{o(1)}$), which can be discarded after the answer is verified. On the negative side, this protocol seems to differ significantly from a non-deterministic algorithm, because we force the prover to commit to a value $v_1 = h^*(x) = h^*(y)$ before we make additional queries. This seems to involve a quantifier alternation: whereas one round of interaction can be encoded by a non-deterministic machine that guesses the prover's response, for several rounds a machine would have to guess the prover's initial response, so that *for all* subsequent requests with distinct secondary hash functions *there exists* a prover response that convinces the verifier.

We have finally arrived at the justification for the use of treewidth. As mentioned, there is a tight connection between the pw-SETH and non-deterministic log-space machines, so barring an improvement to the above protocol that would eliminate the quantifier alternation *without increasing* the verifier's memory, it seems very hard to obtain a formula of the desired pathwidth. Nevertheless, recent works on the classes XALP and XNLP give us the intuition that whereas pathwidth captures non-deterministic log-space algorithms, treewidth captures *alternating* log-space algorithms[3], informally because Introduce nodes are non-deterministic (we have to guess the value of a new variable), but Join nodes are co-non-deterministic (we have to ensure the current assignment can be extended to both children). The key intuition is then that we can encode the above protocol in a formula of *treewidth* $(1 + o(1))\frac{k}{2}\log n$. This is achieved by constructing a separate formula of *pathwidth* $(1 + o(1))\frac{k}{2}\log n$ for each round of the interaction and then adding constraints that ensure the prover has to remain consistent with respect to the main hash function h^*. These extra constraints transform the union of path decompositions into a tree decomposition.

Putting it all together Let us now sketch how the ideas described above lead to the reduction. We use the same general strategy for k-XOR and k-SUM, though for reasons we describe below the reduction is simpler for k-XOR. Initially, we pick a main hash function h^* that is (almost) linear and has range of the order $n^{k/2}$. Suppose that we know that h^* is balanced in the sense that, for all inputs, with high probability the maximum load of any bucket is $M = n^{o(k)}$. We pick many (say $10k \log n$) secondary hash functions $h_1, \ldots, h_{10k \log n}$, with range of the order M^2.

Now, for every pair (h^*, h_ℓ), for $\ell \in [10k \log n]$, we construct a SAT instance ϕ_ℓ of pathwidth $\frac{k}{2}\log n + O(\log M) = (1 + o(1))\frac{k}{2}\log n$. The formula ϕ_ℓ encodes the execution of the non-deterministic Subset Sum algorithm which uses a counter to keep track of the current sum, applied to the input instance after we apply to each element the pair of hash functions (h^*, h_ℓ). The bags of the decomposition of ϕ_ℓ are now supposed to contain the contents of the memory of the algorithm after every step. We focus on the "mid-point" bag that represents the sum calculated right after the first $k/2$ arrays have been processed. For each $\ell \in [10k \log n - 1]$ we add constraints ensuring the formulas ϕ_ℓ and $\phi_{\ell+1}$ agree on the value of h^* at the mid-point of the execution of the algorithm. Hence, we obtain a formula of treewidth $(1 + o(1))\frac{k}{2}\log n$ which simulates the interactive protocol we discussed above. Namely, in order to satisfy the formula one would first have to commit to a value of the main hash function at the mid-point (that is, to a value $v_1 = h^*(x) = h^*(y)$) and then produce a

[3] With a logarithmic number of alternations, but this is fine for our purposes.

collision for all secondary hash functions while respecting the commitment. This intuition drives the proof of correctness.

Finally, let us explain how the reductions for k-XOR and k-SUM differ. When dealing with k-XOR we use linear transformations (we multiply each input vector with a random boolean matrix). This makes it easy to concatenate two hash functions (h^*, h_ℓ). Crucially, linear hash functions in this context have been shown to achieve maximum load comparable to a fully random function, that is, *logarithmic* in the number of elements [8, 43]. Since for our purposes any sub-polynomial bound on the maximum load works, we can put these ingredients together and obtain a reduction along the lines sketched above.

In contrast to this, for k-SUM we have to deal with several additional problems. First, concatenating hash functions means we are dealing with a Vector Subset Sum problem, but this is easy to deal with. Second, the hash functions which are available in the context of integers (we use a function due to Dietzfelbinger [30]) are generally almost-linear, in the sense that due to rounding errors, $h(x+y)$ is only guaranteed to be close to $h(x)+h(y)$, rather than equal. This is a common difficulty in this context and is also not too hard to deal with. The main problem is that for this class of almost-linear hash functions over the integers it is currently an open problem if we can guarantee a maximum load better than $M = N^{1/3}$ when mapping N elements into a range of size N [46]. This forces us to use a function that is made up of a concatenation of many smaller independent hash functions. Then, using the fact that the base function is pair-wise independent [30, 31] we are able to show that the maximum load can be upper-bounded sufficiently well for our purposes.

Because of these added difficulties for k-SUM, we begin our presentation in Section 3 with the reduction for k-XOR. Then, in Section 4 we mainly focus on overcoming the additional obstacles for k-SUM and assume the reader is already familiar with the general strategy of the construction. We therefore suggest to the interested reader to begin with Section 3. Applications of our main result are given in Section 5.

1.6 Open problems

In general, we believe it would be fruitful to continue investigating the connections between different flavors of fine-grained complexity hypotheses, as this can both increase our confidence in the validity of our lower bounds and provide valuable insight. Beyond this general direction, let us mention some concrete questions directly relevant to this work:

Pathwidth The most natural problem we leave open is the following: does the k-SUM (or k-XOR) Hypothesis imply the pw-SETH? We have already explained the reasons why our current proof needs to rely on treewidth, rather than pathwidth. Furthermore, by the characterization of Iwata and Yoshida [42], if it were possible to obtain a reduction from k-SUM to SAT with pathwidth $\frac{k}{2}\log n$ (which seems necessary), then we would have a non-deterministic algorithm solving k-SUM in the same amount of space. The "easy" algorithmic ideas one would try clearly fall short of this, so it merits further investigation whether this can be done, or whether some circumstantial evidence against such an algorithm can be found (which would also be circumstantial evidence that the pw-SETH and tw-SETH are not equivalent hypotheses).

Derandomization Another natural problem we have not addressed is the question of whether our reduction can be made deterministic. Even though reductions from 3-Sum and k-SUM are often randomized, it has recently become a topic of interest to investigate whether such reductions can be derandomized [23, 33]. Can these ideas be applied to our reduction?

Clique Moving away from k-SUM, it would be very interesting to make a connection between SAT-based hypothesis, where it is conjectured that we have to try all possibilities in some sense, and other problems where more clever algorithms exist. A very notable case is detecting k-cliques in a graph, which can be done in time $n^{\frac{\omega k}{3}}$, where ω is the matrix multiplication constant. Our concrete question is then the following: if the tw-SETH is false, does this imply an algorithm which can detect if a graph has a k-clique in time $n^{\frac{\omega k}{3}-\varepsilon}$ or even $n^{\frac{2k}{3}-\varepsilon}$? In the same way that to obtain the results of this paper we had to encode the main idea of the meet in the middle algorithm for k-SUM, it appears likely that to obtain this implication we would need a reduction that needs some elements of fast matrix multiplication algorithms, or at least, efficient algorithms which can verify the correctness of matrix multiplication.

We stress here that in the question formulated above we are discussing k-clique detection, because this is a problem for which a non-trivial algorithm is known and is conjectured to be optimal. For the (harder) problems of detecting a k-clique in a hypergraph or a k-clique of minimum (edge-)weight, it is known that if these problems are as hard as conjectured (n^k), then the k-OV Hypothesis is true [1], therefore the pw-SETH is true. Even without using the results of [1] it is straightforward to show that these problems can be encoded by a CNF formula of pathwidth $k \log n$, giving the implication to the pw-SETH. What makes the k-clique detection question interesting, then, is that the conjectured best bound does not correspond to brute-force search.

1.7 Other related work

As mentioned, fine-grained complexity theory is a very vibrant area and numerous complexity hypotheses can be found in the literature. In this paper we focus on variants of the Strong Exponential Time Hypothesis (SETH), formulated by Impagliazzo and Paturi [41], which states that for all ε there exists k such that k-SAT on inputs with n variables cannot be solved in time $(2-\varepsilon)^n$. The 3-Sum Hypothesis goes back to [37] and has been used as the basis for numerous fine-grained lower bounds. It would be impractical to review here the numerous other fine-grained hypotheses that can be found in the literature. We do mention, however, two other hypotheses which are close to our topic. The Set Cover Conjecture [26] states that Set Cover on instances with m sets and n elements cannot be solved in time $(2-\varepsilon)^n m^{O(1)}$; and the Max-3-SAT Hypothesis, which states that Max-3-SAT cannot be solved in time $(2-\varepsilon)^n$ [32]. It is an open problem whether either one of these hypothesis implies or is implied by the SETH.

Establishing connections between the different branches of hypotheses of fine-grained complexity theory is a topic that has already attracted attention. A previous work that is very close in spirit to what we do here is that of Abboud, Bringmann, Dell, and Nederlof [1], who showed that if it is hard to find a minimum weight k-clique in time $n^{(1-\varepsilon)k}$ or if it is hard to detect a k-clique in a hypergraph in the same amount of time, then the k-OV problem also requires time $n^{(1-\varepsilon)k}$. Because the min-weight 3-clique problem is equivalent to the APSP Hypothesis, the thrust of the result of [1] is similar to ours in the sense that they start from a hypothesis that is stronger than APSP (min-weight k-clique, rather than 3-clique) and show that it implies a hypothesis that is weaker than the SETH (the k-OV Hypothesis). This can also be seen by comparing Figure 1 with Figure 1 of [1]. In other words, whereas [1] draws a connection between the APSP and SETH branches of fine-grained complexity hypotheses, we draw a connection between the 3-Sum and SETH branches.

More broadly, motivated by the somewhat speculative nature of many assumptions, much work has been devoted to reproving lower bounds known under some hypothesis, while

using a weaker or orthogonal hypothesis. For instance, Abboud, Bringmann, Hermelin, and Shabtay [2] show that the optimality of the textbook DP algorithm for Subset Sum, which was previously known under the SCC, is also implied by the SETH. More broadly, Abboud, Vassilevska Williams, and Yu [5] started an effort to define a problem whose hardness is implied by all three major hypotheses, with the hope that this problem would serve as a more solid starting point for reductions. In a similar vein, Abboud, Hansen, Vassilevska Williams, and Williams [3] advocated for the use of the NC-SETH (the version of the SETH for circuits of poly-logarithmic depth) as a more solid starting point for reductions than the SETH.

We also note that the validity of hypotheses used in fine-grained complexity should not in general be taken for granted. Notably, recent work by Björklund and Kaski [14] followed-up by Pratt [56] established that the SCC and the Tensor Rank Conjecture (a conjecture from the realm of matrix multiplication algorithms) cannot both be true. Also, Williams has disproved a plausible-sounding strengthening of the non-deterministic SETH [61].

Several recent works have focused on proving that the SETH *does not* imply various lower bounds, often using the non-deterministic SETH as a hypothesis, in a spirit similar to the work of Carmosino et al. we mentioned above [21]. In particular, Aggarwal and Kumar [6] show that hardness for some cases of the Closest Vector Problem cannot be based on the SETH, Trabelsi shows a similar result for All-Pairs Max Flow [57], and Li shows a similar result for some cases of approximating the diameter [52]. Furthermore, Belova et al. [13], building on previous work [12] show that proving SETH-based hardness for various problems in P will be challenging. In particular, a result of [13] that is very relevant to us is that proving a SETH-based lower bound of $n^{1+\varepsilon}$ for k-SUM would require disproving a stronger version of the SETH, which would in turn imply some circuit lower bounds. Note, however, that these results discuss the difficulty of reducing SAT to k-SUM, while our main result is a reduction in the converse direction. A recent work that does go in the same direction as we do in this paper is that of Belova, Chukhin, Kulikov, and Mihajlin [11], who reduce Subset Sum to the Orthogonal Vectors problem. However, their motivation is to obtain a Subset Sum algorithm using less space, rather than to show hardness for Orthogonal Vectors.

The intuition that SAT parameterized by pathwidth encodes non-deterministic logarithmic space algorithms comes from [42], while the intuition that the difference between treewidth and pathwidth in this setting is that treewidth simulates alternations comes from works on the classes XNLP and XALP [15, 16, 17]. This intuition goes back to the work of Allender, Chen, Lou, Papakonstantinou, and Tang [7] who showed that whereas SAT on instances of pathwidth $O(\log n)$ is complete for NL, SAT on instances of treewidth $O(\log n)$ is complete for SAC^1, the class of problems decidable by logarithmic depth boolean circuits of semi-unbounded fan-in. This is a somewhat weaker class than AC^1 (circuits of unbounded fan-in), which in turn corresponds exactly to logarithmic space alternating non-deterministic machines which perform $O(\log n)$ alternations.

2 Preliminaries

As mentioned, we expect the reader to be familiar with the basics of parameterized complexity, such as DP algorithms parameterized by treewidth, and refer to standard textbooks for the relevant definitions [27]. For a CNF formula ϕ we define the primal graph of ϕ to be the graph obtained if we construct a vertex for each variable of ϕ and make two vertices adjacent if the corresponding variables appear in a clause together. When we refer to the treewidth

or pathwidth of a formula ϕ we will mean the corresponding width of the primal graph.

Regarding the hypotheses of Figure 1 we have the following definitions:

1. SETH: For all $\varepsilon > 0$ there exists k such that k-SAT cannot be solved in time $(2-\varepsilon)^n$ on instances with n variables.
2. pw-SETH: For all $\varepsilon > 0$ there exists no algorithm solving SAT in time $(2-\varepsilon)^{\text{pw}}|\phi|^{O(1)}$, where pw is the width of a given path decomposition of the input formula ϕ.
3. tw-SETH: The same as the previous statement, but given a tree decomposition instead of a path decomposition.
4. LD-C-SETH: For all $\varepsilon > 0$, there exists no algorithm which takes as input a bounded fan-in boolean circuit of size s with n inputs and depth at most εn and decides if the circuit is satisfiable in time $(2-\varepsilon)^n$.
5. k-OV Hypothesis: For all $k \geq 2, \varepsilon > 0$, there exists no algorithm which takes as input k arrays, each containing n boolean vectors of length d and decides if there is a way to select one vector from each array so that the selected vectors have product 0 in time $n^{k-\varepsilon}d^{O(1)}$.
6. SCC: For all $\varepsilon > 0$ there exists k such that no algorithm can solve Set Cover on inputs with n elements and m sets of size at most k in time $(2-\varepsilon)^n m^{O(1)}$.

We have already defined k-SUM and k-XOR in the introduction. Regarding the relations between hypotheses, it was shown by Williams that the SETH implies the k-OV Hypothesis [62], while it is easy to reduce the k-OV problem to the satisfiability of a boolean circuit with $k \log n$ inputs and small depth. The LD-C-SETH is a hypothesis posited in [48], where it was shown that it implies (a stronger form of) the pw-SETH. Notice that the NC-SETH, mentioned in [3], trivially implies the LD-C-SETH. The fact that the SCC implies the pw-SETH was shown in [49].

In general we try to use standard graph-theoretic notation. When we write $[i,j]$ for i,j positive integers, we mean the set of all integers between i and j inclusive, that is, $[i,j] = \{i, i+1, \ldots, j\}$. When we write $[i]$ we mean $[1,i]$. A function $f(x)$ is affine if $f(x) = ax + b$, for a, b constants, where we call b the offset. A function is linear if it is affine and has offset 0.

We do not pay much attention to the specific machine model, since we ignore poly-logarithmic factors, but the reader may assume we are using the RAM model. We also in general allow randomized algorithms and our main result will be a reduction that succeeds with high probability and one-sided error. For k-SUM and k-XOR one can assume that the given integers/vectors have length bounded by $O(\log n)$ [29], but one point of focus will be to determine the constant hidden in the Oh-notation and how it depends on k.

3 k-XOR implies tw-SETH

Our goal in this section is to show the following theorem:

▶ **Theorem 3.** *(k-XOR ⇒ tw-SETH) Suppose that there exists an $\varepsilon > 0$ and an algorithm which, given a CNF formula ϕ and a tree decomposition of its primal graph of width tw decides if ϕ is satisfiable in time $(2-\varepsilon)^{tw}|\phi|^{O(1)}$. Then, there exist $\delta > 0, k_0 > 0$ and a randomized algorithm which for all $k > k_0$ solves k-XOR on instances with k arrays each containing n vectors in time $n^{(1-\delta)\frac{k}{2}}$ with one-sided error and success probability $1 - o(1)$. In other words, if the tw-SETH is false, then the randomized k-XOR hypothesis is false.*

On a high level, we will establish Theorem 3 by giving a (randomized) reduction which transforms a k-XOR instance into a CNF formula of treewidth roughly $\frac{k}{2}\log n$. There are, however, several technical obstacles that we need to overcome to achieve this.

To understand the main challenge, it is instructive to compare Theorem 3 with the results of [49], where it was shown that the hypothesis that the k-Orthogonal Vectors (k-OV) problem is hard implies the pw-SETH. In both k-OV and k-XOR the solution can be described as a string of $k\log n$ bits, giving the indices of the vectors we are looking for. However, a crucial difference is that, whereas for k-OV the (conjectured) best algorithm needs to try out all n^k possibilities, for k-XOR there is an algorithm running in time (essentially) $n^{k/2}$. As a result, while the reduction from k-OV to SAT is straightforward (we construct a variable for each bit of the certificate and just need to check that the certificate is valid), the reduction from k-XOR to SAT needs to be much more efficient, because encoding the indices of the selected vectors is likely to produce a formula with treewidth $k\log n$, rather than $\frac{k}{2}\log n$. Such a formula would not help us solve k-XOR, even if the tw-SETH were false, as this would only guarantee a running time of $n^{(1-\varepsilon)k}$, which is much slower than the best algorithm.

As discussed, the fact that brute force is not optimal for k-SUM and this hinders reductions from this problem was already observed by Pătraşcu [55], who worked around this obstacle by reducing 3-Sum to 3-Sum-Convolution, a problem for which brute force is optimal. However, the natural generalization of this approach to larger values of k leads to a problem for which the brute force algorithm has complexity n^{k-1} and is sub-optimal (see e.g. [4]). We therefore need a different approach.

Our solution will rely on three main ingredients. The first is a version of a reduction given in [49] which established that if the pw-SETH is false, then there is an algorithm for Subset Sum on n integers with target value t running in time $t^{1-\varepsilon}n^{O(1)}$. This reduction can easily be adapted (indeed simplified) to handle sums modulo 2, so we can apply it to k-XOR. It is also known that, given an arbitrary instance of k-XOR, it is possible to ensure that all vectors have $k\log n + O(1)$ bits at most, without loss of generality (see Lemma 2.2 of [29]). We can therefore view the input instance as a Subset Sum instance, with the target t being at most $O(n^k)$. Unfortunately, this is, once again, not efficient enough, even though the algorithm we are encoding is completely different from the brute force algorithm. We would need to arrive at a Subset Sum instance where $t = O(n^{k/2})$ to use the reduction of [49].

The second ingredient we therefore need is some tool that will allow us to preserve the anwer to the original k-XOR instance, while compressing the vectors from $k\log n$ to (roughly) $\frac{k}{2}\log n$ bits. A natural approach to follow here is to use a linear hash function[4], that is, produce a random boolean array R of dimensions $\frac{k}{2}\log n \times k\log n$ and replace each vector v of the input with Rv. Unfortunately, this simple approach will not work, as it will likely produce a large number of false positives (and indeed, if it did work, this would imply that we can simplify any k-XOR instance to ensure that vectors have $\frac{k}{2}\log n$ bits, rather than $k\log n$, which would be surprising).

The third (and final) main ingredient we need is therefore some idea that will help us root out false positive solutions. Here we use an idea inspired from the classical work of Fredman, Komlós, and Szemerédi [36] on 2-level hashing. Recall that in this scheme we have N elements which we want to map (hash) into a range of size M. For our application (k-XOR) we can think of $N = n^{k/2}$, because an equivalent formulation of the problem is the following: is there a vector which is simultaneously among the (at most $n^{k/2}$) vectors constructible as

[4] Indeed, this is how Lemma 2.2 of [29] reduces k-XOR to the case where vectors have length $k\log n + O(1)$.

sums using the first $k/2$ arrays, and the (at most $n^{k/2}$) vectors constructible as sums using the last $k/2$ arrays. If we had a collision-free hash function, this List Disjointness question could be solved by examining hashed values. Unfortunately, in order to hope that a random hash function will be collision-free (with reasonable probability), we need M to be at least N^2, which in our case would again force us to use vectors of $k \log n$ bits. For our general scheme to work, we need to avoid collisions while still keeping $M \approx N$.

The key idea of [36] is then to observe that if we have a (sufficiently good) hash function mapping N elements into a range of size not much larger than N, even though we cannot fully avoid collisions, we can perhaps expect to have few collisions in each target value (bucket). The idea of [36] is then to run a second hash function, whose range is quadratic in the size of each bucket. Even though this second step is less efficient, this inefficiency is acceptable because the buckets have size significantly smaller than N, and the combination of the two hash functions eliminates all collisions.

High-level technique summary Putting it all together we perform a reduction based on the following steps:

1. We select a random linear function the cuts down the length of all vectors of the initial instance to $\frac{k}{2} \log n$. Call this the main hash function h^*.
2. Using the aforementioned reduction from Subset Sum to SAT from [49] we can reduce the hashed instance to a SAT instance with pathwidth $\frac{k}{2} \log n$. From the satisfying assignments to this instance we can extract a value y such that $y = h^*(v_1) = h^*(v_2)$, where v_1, v_2 are vectors constructible from the first and last $k/2$ arrays respectively. Unfortunately, h^* could have many collisions, so this step could produce many false positives (that is, pairs v_1, v_2 with $v_1 \neq v_2$ but $h^*(v_1) = h^*(v_2)$).
3. We observe that the main hash function h^* we used has a good balancing property, as shown by the work of Alon et al. on random linear functions [8], namely, the maximum number of collisions in each target value is not far from what we would expect from a completely random function.
4. As a result, we can apply a secondary hash function, whose range is quadratic in the expected bucket size (which is poly-logarithmic, rather than polynomial, in n) making the inefficiency of this second function insignificant. The secondary hash function is collision-free with constant probability.

Finally, there is one last step missing to complete this construction. Observe that, for any *fixed* value $y = h^*(v_1) = h^*(v_2)$, as described in step 2 above, extending h^* with a secondary hash function (of much smaller range) will eliminate collisions with constant probability and hence make the reduction correct. The problem is that to get the reduction to work as described above we would need to essentially exchange the order of quantifiers in the above statement[5], which does not seem possible. Indeed, this is the main obstacle to proving that the k-XOR hypothesis implies the pw-SETH (rather than the tw-SETH).

To deal with this problem we use the implicit quantifier alternation that becomes available to us when we parameterize SAT by treewidth rather than pathwidth. We produce many (say $\Omega(k \log n)$) copies of the construction sketched above, all using the same main hash function, but each using a different secondary hash function. We then connect these copies in a way

[5] To see what we mean, read the statement of the previous paragraph as "for all buckets y, most secondary hash functions make y collision-free" and compare with the (stronger) statement "most secondary hash functions make all buckets y collision-free".

that forces a satisfying assignment to commit to a value $y = h^*(v_1) = h^*(v_2)$ throughout the instance. For any fixed y, the instance is then falsely satisfiable if bucket y has a collision for all $\Omega(k \log n)$ secondary hash functions, which happens with probability $n^{-\Omega(k)}$; hence by union bound the probability that there exists a value y which produces a false positive is very low. The key fact here is that connecting the instances in this way may increase the pathwidth, but we can still bound the treewidth of the new instance by the desired bound.

In the rest of this section we first review the basic tools we will need in Section 3.1, namely a version of the aforementioned Subset Sum reduction and known results on linear hash functions; we then describe our reduction from k-XOR to SAT in Section 3.2; and finally we prove the correctness of our reduction (and hence Theorem 3) in Section 3.3.

3.1 Tools

As mentioned we will make use of and adapt two main ingredients from previous work. The first is a reduction from Subset Sum to SAT which produces an instance whose pathwidth is $\log t$, where t is the target value. We describe a version of such a reduction, adapted from [49], below. Our second ingredient is the fact that random linear hash functions behave "well", in the sense that the maximum load is comparable to that of a truly random hash function, as proved in [8]. We review these facts in Section 3.1.2

3.1.1 Subset (XOR)-Sum to Pathwidth SAT

We start with a lemma informally stating the following: suppose we are given a k-XOR instance, where each vector has u bits. It is possible to encode this instance using a CNF formula of pathwidth (almost) u and produce a path decomposition of the resulting formula that has the following special property: the last bag of the decomposition encodes exactly the sums which are constructible from the vectors of the instance. More precisely, the last bag of the decomposition has u (ordered) variables and an assignment to these variables can be extended to a satisfying assignment of the whole formula if and only if the vector corresponding to this assignment is constructible in the k-XOR instance by selecting one vector from each array and computing their sum.

▶ **Lemma 4.** *Suppose we are given ℓ arrays $A_1, \ldots, A_\ell$, each containing a collection of boolean vectors of u bits. There exists a polynomial-time algorithm which produces a formula ψ of pathwidth $u + O(1)$ and a path decomposition of ψ such that the last bag of the decomposition B_0 contains u variables $x_1, \ldots, x_u$ and such that we have the following: for each truth assignment σ to the variables of B_0, σ can be extended to a satisfying assignment of ψ if and only if there exist $i_1, i_2, \ldots, i_\ell$ such that $\sum_{j \in [\ell]} A_j[i_j] = (\sigma(x_1), \sigma(x_2), \ldots, \sigma(x_u))$.*

Proof. We first describe the construction for $\ell = 1$ and then reuse this inductively to arrive at the general case.

Suppose $\ell = 1$, so we are given a single array A_1 containing n vectors of u bits. We construct a CNF formula containing the following variables:

1. The variables $x_{i,j}$, for $i \in [0, n], j \in [u]$. The intuitive meaning of these is that, for a fixed i, the u variables $x_{i,j}$ encode the sum we have constructed after having considered the first i elements of A_1.
2. The variables s_i, for $i \in [n]$. The intuitive meaning is that s_i is set to 1 if the i-th element of A_1 is selected.
3. The variables y_i, for $i \in [n]$. The intuitive meaning is that y_i is set to 1 if at least one of $s_1, s_2, \ldots, s_i$ is set to 1.

The idea is that the formula we are constructing will encode the workings of a non-deterministic algorithm, which considers each element of A_1 in turn, decides whether to select it (this is encoded by the s_i variables) and keeps track of the sum of the currently selected elements. The y_i variables will allow us to check that exactly one element per array is selected.

We add the following clauses:

1. For $i = 1$, add clauses ensuring that $y_1 = s_1$. For $i \in [2, n]$, add clauses ensuring that $y_i = (y_{i-1} \vee s_i)$.
2. For $i \in [2, n]$ add the clause $(\neg s_i \vee \neg y_{i-1})$, ensuring that we cannot select the i-th vector if we have already selected one among the first $i - 1$ vectors. Also, add the unit clause y_n, ensuring that at least one vector is selected in total.
3. For $i \in [n]$, for $j \in [u]$, add clauses ensuring that if s_i is set to 0, then $x_{i-1,j} = x_{i,j}$. That is, if the i-th vector is not selected, then the value encoded by $x_{i,j}$ is the same as the value encoded by $x_{i-1,j}$.
4. For $i \in [n]$, for $j \in [u]$ such that the bit in position j of $A_1[i]$ is set to 1, add clauses ensuring that if s_i is set to 1 then $x_{i,j} \neq x_{i-1,j}$.
5. For $i \in [n]$, for $j \in [u]$ such that the bit in position j of $A_1[i]$ is set to 0, add clauses ensuring that if s_i is set to 1 then $x_{i,j} = x_{i-1,j}$. The clauses of this and the previous step ensure that if the i-th vector is taken, $x_{i,j}$ is calculated by adding $A_1[i]$ to $x_{i-1,j}$.

Correctness: The formula we have constructed is satisfiable. Indeed, it is not hard to see that all satisfying assignments set exactly one s_i variable to 1 and once we have fixed which s_i is set to 1 there is a unique way to extend this to a satisfying assignment of the whole formula (modulo the assignment to $x_{0,j}$ for $j \in [u]$): set all other $s_{i'}$, for $i' \neq i$ to 0; set $y_{i'}$ to 1 if and only if $i' \geq i$; set $x_{i',j} = x_{0,j}$ for $i' < i$; set $x_{i',j} = x_{0,j} + A_1[i]_j$ for $i' \geq i$, where $A_1[i]_j$ is the j-th bit of $A[i]$. In other words, we can satisfy the formula if and only if we select a single $i \in [n]$ and then ensure that the vectors $x_{0,j}$ and $x_{n,j}$ differ exactly by $A_1[i]$. For the base case it now suffices to add constraints to the formula so that $x_{0,j} = 0$ for all j.

Pathwidth: We can construct a path decomposition by starting with $n+1$ bags $B_i, i \in [0, n]$, with each bag B_i containing the u variables $x_{i,j}$. For each $i \in [n]$ we add to both B_i and B_{i-1} the variables s_i, y_i. This has covered the clauses of the first two steps. To cover the remaining clauses, for each $i \in [n]$ we insert a sequence of $2u$ new bags between B_{i-1} and B_i. Start with B_{i-1} and for each $j \in [u]$ insert after the current bag a copy of the same bag with $x_{i,j}$ added, and then a copy of this second bag with $x_{i-1,j}$ removed, and make this new bag the current bag. Repeat in this way, at each step exchanging $x_{i-1,j}$ with $x_{i,j}$, until we arrive at B_i. This process covers all remaining clauses. In the end, after the last bag we add a bag containing $x_{n,j}$ for $j \in [u]$.

General construction: Given the construction for $\ell = 1$, we can handle the general case of larger values of ℓ as follows: first, construct a formula for the first $A_{\ell-1}$ arrays and a path decomposition where the last bag contains u variables which encode all constructible vectors. Use the base case construction to encode the array A_ℓ with a CNF formula, without adding the clauses that force $x_{0,j}$ to 0; instead, identify the $x_{0,j}$ variables with the variables of the last bag of the decomposition of the formula representing $A_{\ell-1}$. This completes the construction, preserves pathwidth, and correctness can easily be established by induction. ◀

Because in the previous lemma the "special" bag whose variables encode the constructible sums is the last bag of the decomposition, we can glue together two copies of this construction to encode a k-XOR instance.

▶ **Corollary 5.** *Suppose we are given k arrays (for k even) $A_1, \ldots, A_k$, each containing a collection of boolean vectors of u bits. There exists a polynomial-time algorithm which produces a formula ψ of pathwidth $u + O(1)$ and a path decomposition of ψ such that some "special" bag of the decomposition B^* contains u variables $x_1, \ldots, x_u$ and such that we have the following: for each truth assignment σ to the variables of B^*, σ can be extended to a satisfying assignment of ψ if and only if there exist $i_1, i_2, \ldots, i_k$ such that $\sum_{j\in[k/2]} A_j[i_j] = (\sigma(x_1), \sigma(x_2), \ldots, \sigma(x_u)) = \sum_{j\in[k/2+1,k]} A_j[i_j]$. Therefore, ψ is satisfiable if and only if the given k-XOR instance has a solution.*

Proof. We invoke Lemma 4 on the k-XOR instance $A_1, \ldots, A_{k/2}$ and the k-XOR instance $A_{k/2+1}, \ldots, A_k$. This produces two formulas ϕ_1, ϕ_2 and two path decompositions, where the last bags of the decompositions have the properties described in the lemma. We obtain a formula for the whole instance by taking the union of ϕ_1, ϕ_2 and identifying the u variables of the last bags (respecting the ordering). We construct a path decomposition by merging the last bags of the two decompositions, and call the bag resulting from this merging B^*. ◀

3.1.2 Linear Hash Functions

We now recall some known properties of linear hash functions. In this section (and following sections) when we say that we construct a random linear function $h : \{0,1\}^u \to \{0,1\}^r$ we mean that we construct an $r \times u$ boolean matrix R whose elements are set independently to 0 or 1 with probability 1/2. The function is then defined as $h(v) = Rv$. Furthermore, when we write $h^{-1}(y)$, we mean the set of pre-images of y, that is, $h^{-1}(y) = \{\ x \mid h(x) = y\ \}$.

▶ **Theorem 6.** *[Theorem 4 of [8]] Let u, r be integers, with $r < u$ and $\mathcal{H}$ be the set of all linear functions from $\{0,1\}^u$ to $\{0,1\}^r$. For all sets $S \subseteq \{0,1\}^u$ such that $|S| \leq 2^r$ we have the following: $E_{h\in\mathcal{H}}[\max_{y\in\{0,1\}^r} |h^{-1}(y) \cap S|] = O(r \log r)$.*

We remark that is was shown in [8] that in fact the same upper bound on the maximum load can be shown even if $|S|$ is slightly larger than 2^r (namely, the theorem also holds under the assumption $|S| < 2^r r$). Conversely, Babka [9] gives an improvement to Theorem 6 which upper bounds the expected maximum load by $O(r)$ (rather than $O(r \log r)$) and this was much more recently further improved by Jaber, Kumar, and Zuckerman to match the performance of a truly random function, that is, $O(r/\log r)$ [43]. Nevertheless, neither of these improvements is necessary for our purposes and indeed Theorem 6 is already more than good enough (it would have been sufficient for us to have an upper bound of $2^{o(r)}$ on the maximum load). We therefore stick with the bound of Theorem 6.

Recall that a class of functions $\mathcal{H}$ from $\{0,1\}^u$ to $\{0,1\}^r$ is called 1-universal if for all distinct $x_1, x_2 \in \{0,1\}^u$ we have that $Pr_{h\in\mathcal{H}}[h(x_1) = h(x_2)] \leq \frac{1}{2^r}$. The following is a standard fact about 1-universal functions.

▶ **Theorem 7.** *Let u, r be integers, with $r < u$ and $\mathcal{H}$ be a 1-universal set of functions from $\{0,1\}^u$ to $\{0,1\}^r$. Then, for all sets $S \subseteq \{0,1\}^u$ such that $|S| \leq 2^{r/2}$ we have the following: $Pr_{h\in\mathcal{H}}[\max_{y\in\{0,1\}^r} |h^{-1}(y) \cap S| > 1] \leq \frac{1}{2}$.*

Proof. There are at most $\binom{|S|}{2} \leq \frac{2^r}{2}$ possible colliding pairs and each pair has probability at most $\frac{1}{2^r}$ of actually colliding. Therefore, the expected number of collisions is at most $\frac{1}{2}$. By

Markov's inequality, the probability that there exists a pair of distinct values that collide is at most $\frac{1}{2}$. ◀

It is also known (and not too hard to observe) that the linear hash functions we are using are indeed 1-universal.

▶ **Observation 8.** *Let u, r be integers, with $r < u$ and $\mathcal{H}$ be the set of all linear functions from $\{0,1\}^u$ to $\{0,1\}^r$. Then, for all distinct $x_1, x_2 \in \{0,1\}^u$ we have $Pr_{h\in\mathcal{H}}[h(x_1) = h(x_2)] \leq \frac{1}{2^r}$.*

Proof. It is sufficient to prove the observation for $r = 1$, as in our context a hash function with r bits of output is just a concatenation of r independent hash functions with one bit of output, therefore if the probability of collision is at most $1/2$ in this case, it is at most $\frac{1}{2^r}$ in general.

Let R then be a random boolean vector of dimension u and we want to show that $Pr[Rx_1 = Rx_2] \leq \frac{1}{2}$ when $x_1 \neq x_2$. Let j be the last bit in which x_1, x_2 differ. Let x_2' be the vector obtained from x_2 by flipping the j-th bit. Then $Pr[Rx_1 = Rx_2] = \frac{1}{2}Pr[Rx_1 = Rx_2'] + \frac{1}{2}Pr[Rx_1 \neq Rx_2']$. This follows because when $Rx_1 = Rx_2'$, then $Rx_1 = Rx_2$ if and only if the j-th bit of R is 0 (which happens with probability $1/2$); and similarly, when $Rx_1 \neq Rx_2'$, then $Rx_1 = Rx_2$ if and only if the j-th bit of R is 1. However, $Pr[Rx_1 = Rx_2'] + Pr[Rx_1 \neq Rx_2'] = 1$. ◀

3.2 The Reduction

In this section we present a construction which reduces k-XOR to SAT in a way that will allow us to establish Theorem 3. We are given a k-XOR instance: k arrays (we assume without loss of generality that k is even) $A_1, \ldots, A_k$, each containing n boolean vectors of u bits each. The question is whether there exist $i_1, i_2, \ldots, i_k$ such that $\sum_{j\in[k]} A_j[i_j] = 0$. Equivalently, the question is whether there exists a boolean vector v with u bits and $i_1, i_2, \ldots, i_k$ such that $v = \sum_{j\in[k/2]} A_j[i_j] = \sum_{j\in[k/2+1,k]} A_j[i_j]$. We assume without loss of generality that n is a power of 2.

Let $r_0 = \frac{k}{2}\log n + 1$ and pick a hash function h^* uniformly at random from the space of all linear functions from $\{0,1\}^u$ to $\{0,1\}^{r_0}$. In the remainder we call h^* our *main* hash function.

Let $r_1 = \lceil 3\log(r_0)\rceil$. For each $\ell \in [10k\log n]$ pick a hash function h_ℓ uniformly at random from the space of all linear functions from $\{0,1\}^u$ to $\{0,1\}^{r_1}$. We will call these our *secondary* hash functions.

Our construction now proceeds as follows: for each $\ell \in [10k\log n]$ we construct a new instance of k-XOR made up of k arrays of size n, where each vector has $r_0 + r_1$ bits. Let $B_1^\ell, B_2^\ell, \ldots, B_k^\ell$ be the arrays of the new instance. We set for all $i \in [k], j \in [n]$ that $B_i^\ell[j] = (h^*(A_i[j]), h_\ell(A_i[j]))$. In other words, to obtain instance ℓ we apply to each element of the original instance the main hash function as well as the ℓ-th secondary hash function and replace the element with the concatenation of the two returned values.

For each of the $10k\log n$ instances of k-XOR we have produced we invoke Corollary 5. This produces $10k\log n$ formulas $\phi_1, \ldots, \phi_{10k\log n}$ and their corresponding path decompositions of width $r_0 + r_1 + O(1)$. Recall that by Corollary 5 each decomposition contains a special bag B^* with $r_0 + r_1$ variables, such that an assignment to this bag is extendible to a satisfying assignment to the formula if and only if the vector of the assignment is constructible as a sum from the first $k/2$ arrays and also as a sum from the last $k/2$ arrays.

So far we have constructed $10k\log n$ independent CNF formulas (and their path decompositions). The last step of our construction adds some extra clauses to ensure that all satisfying

assignments must consistently select the same value for the main hash functions. More precisely, for each $\ell \in [10k \log n - 1]$ consider the formulas ϕ_ℓ and $\phi_{\ell+1}$ and the corresponding special bags of their decompositions, which contain $r_0 + r_1$ variables each. We add clauses ensuring that the first r_0 variables of the special bag of ϕ_ℓ (encoding the value of the main hash function) must receive the same assignments as the r_0 corresponding variables of the special bag of $\phi_{\ell+1}$.

3.3 Proof of Correctness

▶ **Lemma 9.** *The algorithm of Section 3.2 produces a formula of treewidth* $(1 + o(1))\frac{k}{2} \log n$. *Furthermore, a tree decomposition of this width can be constructed in polynomial time.*

Proof. We start with the union of the path decompositions of the formulas $\phi_1, \ldots, \phi_{10k \log n}$ guaranteed by Corollary 5. These cover the whole construction except the clauses we added in the last step to ensure consistency for the main hash function between different CNF formulas. Let us then explain how to obtain a tree decomposition that also covers these new clauses, while at the same time connecting the forest of path decompositions into a single tree decomposition.

For each $\ell \in [10k \log n - 1]$ we do the following: consider the special bags $B^*_\ell, B^*_{\ell+1}$ of the path decompositions of $\phi_\ell, \phi_{\ell+1}$. The primal graph induced by these two bags is a matching with r_0 edges, say from variables $x^\ell_1, x^\ell_2, \ldots, x^\ell_{r_0}$ to variables $x^{\ell+1}_1, x^{\ell+1}_2, \ldots, x^{\ell+1}_{r_0}$, because we have added clauses (of arity two) ensuring that $x^\ell_j = x^{\ell+1}_j$. We connect B^*_ℓ and $B^*_{\ell+1}$ in the decomposition by a sequence of $2r_0$ bags as follows: start with B^*_ℓ as the current bag and then at each step $j \in [r_0]$, add a bag that is a copy of the current bag with $x^{\ell+1}_j$ added, and then add a copy of the bag with x^ℓ_j removed and make that the current bag. Continue in this way until the last bag contains all of $x^{\ell+1}_j$, for $j \in [r_0]$, and make that bag a neighbor of $B^*_{\ell+1}$. It is not hard to see that this constructs a valid tree decomposition of the whole instance.

The width of this decomposition is dominated by the width of the path decompositions given by Corollary 5, which is $r_0 + r_1 + O(1)$. Since $r_0 = \frac{k}{2} \log n + O(1)$ and $r_1 = O(\log r_0) = o(k \log n)$ we have that the width is at most $(1 + o(1))\frac{k}{2} \log n$. ◀

▶ **Lemma 10.** *If the k-XOR instance on which the algorithm of Section 3.2 is executed has a solution, then the resulting CNF formula is satisfiable.*

Proof. If the given k-XOR instance has a solution given by indices $i_1, i_2, \ldots, i_k$, then there exists a boolean vector v such that $v = \sum_{j \in [k/2]} A_j[i_j] = \sum_{j \in [k/2+1,k]} A_j[i_j]$. We obtain a satisfying assignment of the formula as follows.

For each $\ell \in [10k \log n]$ we assign to the $r_0 + r_1$ variables of the special bag B^*_ℓ the values $(h^*(v), h_\ell(v))$. Observe that since we are using the same value for the first r_0 variables of the special bags, this satisfies the consistency clauses added in the final step of the construction. Furthermore, by Corollary 5 for each $\ell \in [10k \log n]$ the assignment we have set is extendible to a satisfying assignment of ϕ_ℓ. This is because $\sum_{j \in [k/2]} B^\ell_j[i_j] = \sum_{j \in [k/2+1,k]} B^\ell_j[i_j] = (h^*(v), h_\ell(v))$, as follows from the linearity of the functions h^*, h_ℓ. ◀

▶ **Lemma 11.** *If the k-XOR instance on which the algorithm of Section 3.2 is executed has no solution, then the resulting CNF formula is satisfiable with probability at most* $o(1)$.

Proof. Suppose that the given k-XOR instance has no solution. Let S_1 be the set of vectors constructible from $A_1, \ldots, A_{k/2}$, that is, $v \in S$ if and only if there exist $i_1, \ldots, i_{k/2}$ such that

$v = \sum_{j\in[k/2]} A_j[i_j]$. Similarly, let S_2 be the set of vectors constructible from $A_{k/2+1}, \ldots, A_k$ and we have by assumption that $S_1 \cap S_2 = \emptyset$. Furthermore, $|S_1|, |S_2| \leq n^{k/2}$.

Suppose now that we apply the main hash function h^* to $S_1 \cup S_2$. Since $|S_1 \cup S_2| \leq 2n^{k/2} = 2^{r_0}$, by Theorem 6 we have that $E[\max_{y\in\{0,1\}^{r_0}} |(h^*)^{-1}(y) \cap (S_1 \cup S_2)|] = O(r_0 \log r_0)$. By using Markov inequality we conclude that the probability that there exists $y \in \{0,1\}^{r_0}$ such that $|(h^*)^{-1}(y) \cap (S_1 \cup S_2)| > r_0 \log^2 r_0$ is $o(1)$, that is, tends to 0 as n tends to infinity. In the remainder we will therefore condition the analysis under the assumption that for all $y \in \{0,1\}^{r_0}$ we have $|(h^*)^{-1}(y) \cap (S_1 \cup S_2)| < r_0 \log^2 r_0$.

By the constraints added in the last step of our construction of the CNF formula we have that if a satisfying assignment exists, it must pick the same values for the first r_0 variables of the special bags B^* of each of the $10k \log n$ instances. Fix now a value $y \in \{0,1\}^{r_0}$. We want to show that the probability that a satisfying assignment exists which uses y as the truth assignment for the first r_0 variables of the special bags is $o(\frac{1}{n^{k/2}})$. Since there are $2^{r_0} = O(n^{k/2})$ such assignments y, if we show this, then by union bound we will have that the probability that a satisfying assignment exists is $o(1)$.

Consider now the set $S_y = (h^*)^{-1}(y) \cap (S_1 \cup S_2)$. As mentioned, we are assuming that $|S_y| < r_0 \log^2 r_0$. Consider now a value $\ell \in [10k \log n]$ and suppose that the secondary hash function h_ℓ is collision-free for S_y, that is, for all $z \in \{0,1\}^{r_1}$ we have $|h_\ell^{-1}(z) \cap S_y| \leq 1$. We claim that in this case the CNF formula indeed has no satisfying assignment which picks y as the assignment to the r_0 special variables. Indeed, otherwise we would have a satisfying assignment to the sub-formula ϕ_ℓ, which would by Corollary 5 correspond to a collection $i_1, \ldots, i_k$ such that $\sum_{j\in[k/2]} B_i^\ell[i_j] = (y, z) = \sum_{j\in[k/2+1,k]} B_i^\ell[i_j]$. In other words, we would have to find two distinct elements of S_y which have a collision for the secondary hash function h_ℓ, which is impossible as we assumed that this function is collision-free on S_y.

We now observe that for each $\ell \in [10k \log n]$, the probability that h_ℓ is collision-free on S_y is at least $1/2$. Indeed, by Theorem 7, if $|S_y| \leq r_0 \log^2 r_0$, since the domain of the secondary hash function has size $2^{r_1} \geq r_0^3 > |S_y|^2$ for sufficiently large n, we have that h_ℓ is not collision-free on S_y with probability at most $1/2$. However, for each $\ell \in [10k \log n]$, we pick each h_ℓ independently and uniformly at random, so the probability that all the secondary hash functions have collisions on S_y is at most $2^{-10k \log n} = n^{-10k}$. In particular, for a fixed assignment y to the r_0 special variables the constructed formula is satisfiable only if all secondary hash functions have collisions, and this probability is at most $n^{-10k} = o(n^{-k/2})$, therefore, by union bound the probability of any satisfying assignment existing is $o(1)$. ◀

Proof of Theorem 3. Suppose there exists an algorithm solving SAT parameterized by treewidth in time $2^{(1-\varepsilon)\mathrm{tw}}|\phi|^C$ for some constant C. We are given a k-XOR instance on k arrays of n vectors each and we execute the construction of Section 3.2. By Lemma 10 and Lemma 11 deciding if the constructed formula is satisfiable is equivalent (up to a very small probability of one-sided error) to deciding if the original instance has a solution.

Let us then examine the running time of this procedure. The treewidth of the new formula is, according to Lemma 9, at most $(1 + \frac{\varepsilon}{2})\frac{k}{2} \log n$ (for sufficiently large n). Furthermore, since the construction runs in randomized polynomial time, we have $|\phi| = (kn)^{O(1)}$. Therefore, the running time of the whole procedure is at most $2^{(1-\varepsilon)(1+\frac{\varepsilon}{2})\frac{k}{2} \log n} n^{C'}$, for some constant C'. We have $(1-\varepsilon)(1+\frac{\varepsilon}{2}) < 1 - \frac{\varepsilon}{2}$. Suppose then that $k > k_0 = \lceil \frac{8C'}{\varepsilon} \rceil$. Then $\frac{\varepsilon k}{4} \geq \frac{\varepsilon k}{8} + C'$. We therefore have that the algorithm runs in time at most $n^{(1-\frac{\varepsilon}{8})\frac{k}{2}}$, disproving the hypothesis that k-XOR is hard. ◀

4 k-SUM implies tw-SETH

Our goal in this section is to establish the following theorem:

▶ **Theorem 12.** *(k-SUM ⇒ tw-SETH) Suppose there exists an $\varepsilon > 0$ and an algorithm which, given a CNF formula ϕ and a tree decomposition of its primal graph of width tw decides if ϕ is satisfiable in time $(2-\varepsilon)^{tw}|\phi|^{O(1)}$. Then, there exist $\delta > 0, k_0 > 0$ and a randomized algorithm which for all $k > k_0$ solves k-SUM on instances with k arrays each containing n integers in time $n^{(1-\delta)\frac{k}{2}}$ with one-sided error and success probability $1 - o(1)$. In other words, if the tw-SETH is false, then the randomized k-SUM hypothesis is false.*

The high-level strategy we will follow is the same as for Theorem 3, but there are some additional technical obstacles we need to overcome. In particular, the first ingredient will once again be a reduction from (a variant of) Subset Sum with a target value that can be encoded with t bits to SAT on instances with pathwidth t. It is therefore crucial to make t small (that is, at most roughly $\frac{k}{2}\log n$), which in turn seems to imply that we need to edit the given k-SUM instance so that the largest absolute value is also of the order of $n^{k/2}$.

In general it is known that an arbitrary k-SUM instance can be reduced (via a randomized algorithm) to an equivalent one where integers have absolute values $O(n^k)$ (Lemma 2.2 of [29]). This is achieved by applying an almost-linear hash function to the input integers. The almost-linearity of the function guarantees that if k integers sum to zero the sum of their hashed values will also be almost zero, while it can be shown that the probability of false positives (k integers whose sum is non-zero, but whose hashed values sum to zero) is small. We will use this approach, based on an almost-linear (in fact, almost-affine) hash function due to Dietzfelbinger [30, 31].

The problem we are faced with now is that we cannot afford to allow the integers of the hashed instance to have a range of size n^k, but we rather need the range to be in the order of $n^{k/2}$. However, setting the range of the hash function to be so small will almost surely produce many false positives.

Recall the strategy we used to deal with this problem for Theorem 3: we reformulated the problem as a List Disjointness question (if L, R are the sets of at most $n^{k/2}$ integers constructible from the first and last $k/2$ arrays respectively, is $L \cap R \neq \emptyset$?) which can be solved by examining hashed values; then, a purported solution is encoded by picking a specific hash value y such that $x_1 \in L, x_2 \in R$ and $h(x_1) = h(x_2) = y$; a key fact now was that even though false positives exist in the hashed instance, once we fix a specific y the number of false positive pairs for this particular y can be guaranteed to be small ($\log^{O(1)} n$), using a result of [8] (Theorem 6). We then used a second-level hash function to eliminate these (few) collisions.

The obstacle is now that a result analogous to Theorem 6 is not known for the almost-affine hash functions we wish to use. Indeed, in our setting we essentially want to hash a set of at most $N = n^{k/2}$ integers (the sums constructible from the first $k/2$ arrays) into a range of size roughly $N = n^{k/2}$. Whereas for a linear hash function of the type of Theorem 6 this will with high probability produce a maximum load poly-logarithmic in N, for almost-linear hash functions dealing with integers, such as the ones we use, the best known upper bound on the maximum load in $O(N^{1/3})$ given by Knudsen [46] and it is a known open problem whether this can be improved further (Westover's work nicely summarizes what is known [60]). Unfortunately, this bound on the maximum load is definitely not good enough for our purposes, because our plan is to use a second-level hash function whose range will be quadratic in the maximum load, but we cannot afford to increase the number of output bits (and therefore the treewidth) by another $\log(N^{2/3}) = \frac{2k}{3}\log n$ bits.

Thinking a little more carefully we observe that what we need for the strategy of Theorem 3 to work is a bound on the maximum load that is stronger than what is known for our hash function, but could be weaker than the bound of Theorem 6. In particular, it would be good enough to show that the maximum load produced by hashing N elements in a range of size N is at most sub-polynomial (that is, $N^{o(1)}$). Our main effort in this section is then devoted to constructing an almost-affine integer hash function which achieves this maximum load.

Let us then outline how our construction of an almost-affine hash function achieving small maximum load works. The basic idea is simple: rather than using Dietzfelbinger's hash function to map $N = n^{k/2}$ integers to integers with $\frac{k}{2}\log n$ bits, we select independently $\frac{\log n}{2}$ hash functions each of which produces as output an integer on k bits. In other words, we hash integers to integer *vectors* of dimension $\frac{\log n}{2}$, where each coordinate has k bits. To give some intuition why we do this, recall that Dietzfelbinger's hash function maps elements in a pairwise independent way. This fact is of little help if we map N items into N buckets, as the maximum load of a pairwise independent hash function could be $\Theta(\sqrt{N})$; however, if we map N elements into a small number (say 2^k) buckets, the maximum load is $\frac{N}{2^k} + O(\sqrt{N}) \leq (1+o(1))\frac{N}{2^k}$, that is, with high probability every bit we are using is decreasing the load by a factor of almost 2. We can then analyze the performance of the concatenation of the hash functions by using the fact that each function is selected independently, allowing us to show that with high probability most hash functions will work as expected. The precise details of this approach are given in Lemma 21.

Given this approach we still need to solve a few secondary problems to apply the strategy we used for k-XOR. Namely, we need to adapt the reduction from Subset Sum to SAT so that it works for vector sums (this is straightforward); and we need to refine our analysis to take into account that the functions we use are not linear but almost-affine. We review the basic tools we will need in Section 4.1; we then describe our reduction from k-SUM to SAT in Section 4.2; and finally we prove the correctness of our reduction, including the claims about the maximum load of our main hash function, in Section 4.3.

4.1 Tools

In this section we first give another variation of the reduction from Subset Sum to SAT (similar to Lemma 4) which this time is able to handle integer vectors. We then review some facts we will need about the hash functions we will be using, including how Chebyshev's inequality and pairwise independence can be used to provide concentration bounds on the load of any bucket (Lemma 16).

4.1.1 Subset Sum to Pathwidth SAT

Similarly to Section 3.1.1 we present a variant of the reduction from Subset Sum to SAT of [49] which is adapted to our construction. In particular, we will focus on a version of Subset Sum where the elements are vectors with integer coordinates, that is, elements of $[2^u]^d$, where each integer coordinate is made up of u bits and we have d coordinates for each vector. The question is whether it is possible to select one vector from each set, so that their component-wise sum (modulo 2^u) is equal to a given target $t \in [2^u]^d$. The lemma below states that it is possible to encode a Subset Sum instance into a CNF formula with a path decomposition such that the last bag contains du variables with the following property: if we read these variables as encoding an element $t \in [2^u]^d$, then the satisfying assignments of the formula correspond exactly to the sums t which can be realized in the instance.

For technical reasons, we formulate this lemma in a more general way, allowing for the

ranges of the integers in each component of the given vectors to be distinct. We will therefore assume that we are dealing with d-dimensional vectors, where for each $j \in [d]$ the integers of the j-th coordinate are in the range $[0, 2^{u_j} - 1]$.

▶ **Lemma 13.** *Let d be a positive integer, $u_1, u_2, \ldots, u_d$ be d positive integers and suppose we are given ℓ arrays $A_1, \ldots, A_\ell$, each containing a collection of elements of $[0, 2^{u_1} - 1] \times [0, 2^{u_2} - 1] \times \ldots \times [0, 2^{u_d} - 1]$. There exists a polynomial-time algorithm which produces a formula ψ of pathwidth $(\sum_{j \in [d]} u_j) + O(1)$ and a path decomposition of ψ such that the last bag of the decomposition B_0 contains $t = \sum_{j \in [d]} u_j$ variables $x_1, \ldots, x_t$ and such that we have the following: for each truth assignment σ to the variables of B_0, σ can be extended to a satisfying assignment of ψ if and only if there exist $i_1, i_2, \ldots, i_\ell$ such that $\sum_{j \in [\ell]} A_j[i_j] = ((\sigma(x_1)\sigma(x_2)\ldots\sigma(x_{u_1})), (\sigma(x_{u_1+1})\ldots\sigma(x_{u_1+u_2})), \ldots, (\sigma(x_{t-u_d+1})\ldots\sigma(x_t)))$, where additions are component-wise and modulo 2^{u_j} for the j-th component, $j \in [d]$.*

Proof. The proof is essentially the same as for Lemma 4, so we sketch the details and only focus on the differences. As in Lemma 4, we will describe the construction for $\ell = 1$; the general construction can be obtained inductively by gluing together a formula for the first $\ell - 1$ arrays with the formula constructed for A_ℓ. Suppose then for the remainder that we have a single array A containing n vectors.

Similarly to Lemma 4 we construct the variables $x_{i,(j_1,j_2)}$ for $i \in [0, n], j_1 \in [d], j_2 \in [u_{j_1}]$; s_i for $i \in [n]$; and y_i for $i \in [n]$. The intuitive meanings of these variables are the same, namely, for a given $i \in [0, n]$, the $t = \sum_{j_1 \in [d]} u_{j_1}$ variables $x_{i,(j_1,j_2)}$ encode the value of the sum we have constructed after considering the first i elements of the array; s_i is set to 1 if we select the i-th element of the array; and y_i is 1 if at least one element has been selected among the first i elements of the array.

The clauses involving the y_i and s_i variables are identical to Lemma 4. Furthermore, again as in Lemma 4 we add clauses ensuring that if s_i is 0, then $x_{i-1,(j_1,j_2)} = x_{i,(j_1,j_2)}$, for all $j_1 \in [d], j_2 \in [u_{j_1}]$.

The main difference with the proof of Lemma 4 is the clauses we add to cover the case when s_i is set to 1. In this case we have to ensure that the value encoded by $x_{i,(j_1,j_2)}$ is equal to the value encoded by $x_{i-1,(j_1,j_2)}$ plus $A[i]$ and this is slightly more complicated than in Lemma 4 because rather than bitwise XOR we have to compute integer additions, which means we have to take into account the calculation of carry bits.

For each $i \in [n]$ and for each $j_1 \in [d]$ we construct u_{j_1} "carry" variables $c_{i,(j_1,j_2)}$, $j_2 \in [u_{j_1}]$. We denote by $A[i]_{j_1,j_2}$ the j_2-th least significant bit of the j_1-th coordinate of $A[i]$. We add clauses ensuring that if s_i is set to 1, then for all $j_1 \in [d]$ we have (i) $x_{i,(j_1,1)} = x_{i-1,(j_1,1)} + A[i]_{j_1,1}$ (ii) $c_{i,(j_1,1)} = x_{i-1,(j_1,1)} \wedge A[i]_{j_1,1}$ (ii) for all $j_2 \in [2, u_{j_1}]$ we ensure that $x_{i,(j_1,j_2)} = x_{i-1,(j_1,j_2)} + A[i]_{j_1,j_2} + c_{i,(j_1,j_2-1)}$ and $c_{i,(j_1,j_2)} = 1$ if and only if at least two of $x_{i-1,(j_1,j_2)}, A[i]_{j_1,j_2}, c_{i,(j_1,j_2-1)}$ are set to 1.

We now observe that the clauses added above implement the desired property, namely, that when s_i is set to 1 we must give values to $x_{i-1,(j_1,j_2)}$ and $x_{i,(j_1,j_2)}$ such that the difference of their encoded values is $A[i]$. Furthermore, it is not hard to see that we can build a path decomposition of the new instance in a manner similar to that of Lemma 4 with the only difference that in the bags that contain $x_{i-1,(j_1,j_2)}$ and $x_{i,(j_1,j_2)}$ we also add the carry variables $c_{i,(j_1,j_2-1)}$ and $c_{(i,j_1,j_2)}$. ◀

We now obtain a corollary similar to Corollary 5 by gluing together the formulas representing the first half and second half of the arrays of a k-SUM instance.

▶ **Corollary 14.** *Let d be a positive integer, $u_1, \ldots, u_d$ be d positive integers and suppose we are given k arrays (for k even) $A_1, \ldots, A_k$, each containing a collection of integer vectors from*

$[0, 2^{u_1} - 1] \times [0, 2^{u_2} - 1] \times \ldots \times [0, 2^{u_d} - 1]$. *There exists a polynomial-time algorithm which produces a formula ψ of pathwidth $(\sum_{j\in[d]} u_j) + O(1)$ and a path decomposition of ψ such that some "special" bag of the decomposition B^* contains $t = \sum_{j\in[d]} u_j$ variables $x_1, \ldots, x_t$ and such that we have the following: for each truth assignment σ to the variables of B^*, σ can be extended to a satisfying assignment of ψ if and only if there exist $i_1, i_2, \ldots, i_k$ such that $\sum_{j\in[k/2]} A_j[i_j] = ((\sigma(x_1)\sigma(x_2)\ldots\sigma(x_{u_1})), (\sigma(x_{u_1+1})\ldots\sigma(x_{u_1+u_2})), \ldots, (\sigma(x_{t-u_d+1})\ldots\sigma(x_t))) = \sum_{j\in[k/2+1,k]} A_j[i_j]$. Therefore, ψ is satisfiable if and only if the given k-SUM instance has a solution.*

Proof. Identically to the proof of Corollary 5, we produce formulas ϕ_1, ϕ_2 from $A_1, \ldots, A_{k/2}$ and $A_{k/2+1}, \ldots, A_k$ respectively, and then identify the variables of the last bags, obtaining the special bag B^*. ◀

4.1.2 Linear and Affine Hash Functions

The main hash function we will rely on is the following function due to Dietzfelbinger [30, 31]. Let $U = \{0, \ldots, u-1\}$ be a universe, $M = \{0, \ldots, m-1\}$ a target range, and $r = um$, where u, m are positive integers with $m < u$. Then we define the hash function $h_{a,b}(x) = \left\lfloor \frac{(ax+b) \bmod r}{u} \right\rfloor$ and the class of hash functions $\mathcal{H}_{u,m} = \{h_{a,b} \mid 0 \le a, b < r\}$. The following theorem was shown (in a more general form) in [30, 31]:

▶ **Theorem 15.** *If u, m are powers of 2, then $\mathcal{H}_{u,m}$ is a pairwise independent family of hash functions, that is, for all distinct $x_1, x_2 \in U$ and for all $y_1, y_2 \in M$ we have $Pr_{h\in\mathcal{H}_{u,m}}[h(x_1) = y_1 \wedge h(x_2) = y_2] = \frac{1}{|M|^2}$.*

In the remainder, whenever we use this family of hash functions we will always have that u, m are powers of 2. Observe that Theorem 15 immediately implies that the family $\mathcal{H}_{u,m}$ is 1-universal, so Theorem 7 applies. However, using Chebyshev's inequality, we can also establish the following:

▶ **Lemma 16.** *Let u, m be powers of 2. Then, for all $\delta > 0$, $S \subseteq U$ and $y \in M$ we have $Pr_{h\in\mathcal{H}_{u,m}}[|h^{-1}(y) \cap S| \ge (1+\delta)\frac{|S|}{|M|}] \le \frac{|M|}{\delta^2|S|}$*

Proof. Let $Y = |h^{-1}(y) \cap S|$ be the random variable we are interested in. We can see Y as the sum of $|S|$ Bernoulli random variables where for each $x \in S$ the corresponding variable takes value 1 if $h(x) = y$, that is, with probability $\frac{1}{|M|}$. We have $E[Y] = \frac{|S|}{|M|}$ and $Var[Y] = \frac{|S|}{|M|}(1 - \frac{1}{|M|}) \le E[Y]$ where we used the fact that the variance of the sum of pairwise independent variables is equal to the sum of their variances. By Chebyshev's inequality we have that $Pr[Y \ge E[Y] + \delta E[Y]] \le \frac{Var[Y]}{\delta^2(E[Y])^2} \le \frac{|M|}{\delta^2|S|}$. ◀

Furthermore, 1-universality will not be sufficient for our k-SUM construction, because the functions we are using are not exactly linear (or affine). We will therefore need the following variation of Theorem 7 which states informally that if the range we are mapping is k^2 times larger than what is needed for Theorem 7, then not only do we have no collisions but in fact distinct keys are mapped into buckets at distance at least k from each other.

▶ **Lemma 17.** *Let u, m be powers of 2 and k a positive integer. Then, for all sets $S \subseteq [0, u-1]$ with $|S| \le \frac{\sqrt{m}}{k}$ we have the following: $Pr_{h\in\mathcal{H}_{u,m}}[\max_{y\in[0,m-1]} |S \cap (\bigcup_{j\in[0,k-1]} h^{-1}((y + j) \bmod m))| > 1] \le \frac{1}{2}$.*

Proof. Fix a $y \in [0, m-1]$ and consider the values $(y + j) \bmod m$, for $j \in [0, k-1]$. Consider two distinct elements $x_1, x_2 \in S$. The probability that $h(x_1) = y + j_1 \bmod m$ and

$h(x_2) = y + j_2 \bmod m$, for some given $j_1, j_2 \in [0, k-1]$ is at most $\frac{1}{m^2}$, by Theorem 15. By union bound, the probability that there exist $j_1, j_2 \in [0, k-1]$ such that $h(x_1) = y + j_1 \bmod m$ and $h(x_2) = y + j_2 \bmod m$ is at most $\frac{k^2}{m^2}$. Again by union bound, the probability that there exist distinct x_1, x_2 such that there exist y_1, y_2 as stated is at most $\binom{|S|}{2}\frac{k^2}{m^2} \leq \frac{|S|^2}{2}\frac{k^2}{m^2} \leq \frac{1}{2m}$. Taking a union bound over the m possible values of y completes the proof. ◀

It will at some point be more convenient to work with the linear functions of $\mathcal{H}_{u,m}$, that is, the function $h_{a,b}(x)$ with $b = 0$. Given a function $h(x) = \lfloor \frac{(ax+b) \bmod r}{u} \rfloor$ we define $\hat{h}(x) = \lfloor \frac{(ax) \bmod r}{u} \rfloor$, that is, $\hat{h}$ is the modification of $h(x)$ where we have removed the offset b. The functions $\hat{h}$ are almost-linear in the following sense:

▶ **Observation 18.** *For all $h \in \mathcal{H}_{u,m}$ and $x_1, x_2 \in U$ we have that $\hat{h}(x_1 + x_2) \equiv \hat{h}(x_1) + \hat{h}(x_2) + \kappa' \bmod m$, where $k' \in \{0, 1\}$.*

Proof. Let $\hat{h}(x) = \lfloor \frac{(ax) \bmod r}{u} \rfloor$. Then, $\hat{h}(x_1 + x_2) = \lfloor \frac{(ax_1 + ax_2) \bmod r}{u} \rfloor$, while $\hat{h}(x_1) + \hat{h}(x_2) = \lfloor \frac{(ax_1) \bmod r}{u} \rfloor + \lfloor \frac{(ax_2) \bmod r}{u} \rfloor = \lfloor \frac{(ax_1) \bmod r + (ax_2) \bmod r}{u} \rfloor + \kappa$, where $\kappa \in \{0, 1\}$. We now observe that the difference between $(ax_1) \bmod r + (ax_2) \bmod r$ and $(ax_1 + ax_2) \bmod r$ is an integer multiple of r; therefore, dividing this difference by u gives an integer multiple of m (since $r = mu$); therefore the two values are equal if calculations are performed modulo m. ◀

4.2 The Reduction

We are given a k-SUM instance: k arrays (we assume without loss of generality that k is even) $A_1, \ldots, A_k$, each containing n integers. The question we want to answer is whether there exist $i_1, \ldots, i_k$ such that $\sum_{j \in [k]} A_j[i_j] = 0$. Equivalently, if we multiply all entries of $A_{k/2+1}, \ldots, A_k$ by -1 the question becomes whether there exists an integer v and $i_1, \ldots, i_k$ such that $v = \sum_{j \in [k/2]} A_j[i_j] = \sum_{j \in [k/2+1, k]} A_j[i_j]$. We will focus on the latter formulation of the problem and assume without loss of generality that n is a power of 4. Furthermore, by adding $\max_{j \in [k], i \in [n]} |A_j[i]|$ to all entries we can now assume that all integers are non-negative without affecting the answer. Let $W = \max_{j \in [k], i \in [n]} A_j[i]$ be the largest integer of the resulting instance.

Similarly to the proof of Theorem 3 we will now construct a *main* hash function. However, our main hash function will now be a composition of many hash functions with smaller range. In particular, let $m = 2^k$ and let u be the smallest power of 2 such that $kW < u$. For $\beta \in [\frac{\log n}{2}]$ pick independently and uniformly at random a hash function $h^*_\beta \in \mathcal{H}_{u,m}$. The main hash function is the concatenation $h^* = (h^*_1, h^*_2, \ldots, h^*_{\frac{\log n}{2}})$. Observe that if we apply h to an integer in the input we obtain a vector of $\frac{\log n}{2}$ integers, each of k bits, so $\frac{k}{2} \log n$ bits of output.

Let $\delta > 0$ be a small fixed constant (we will precisely define δ in the proof of correctness in a way that will depend on the running time of the supposed algorithm falsifying the tw-SETH). Let m' be the smallest power of 2 such that $m' > 4k^2 n^{4\delta k}$. For each $\ell \in [10k \log n]$ pick a (secondary) hash function h_ℓ independently and uniformly at random from $\mathcal{H}_{u,m'}$.

For each $\ell \in [10k \log n]$ we now construct a distinct (hashed) instance of k-SUM, using the main hash function $h^* = (h^*_1, h^*_2, \ldots, h^*_{\frac{\log n}{2}})$ and the secondary hash function h_ℓ. In order to do so, we will need to take into account that the functions we are using are in fact not linear but almost-affine transformations, that is, each function $h_{a,b}(x)$ adds to the output a constant offset b and then taking the floor of the result may introduce a small error.

Recall that for a hash function $h_{a,b}(x) = \lfloor \frac{(ax+b) \bmod r}{u} \rfloor$ we defined $\hat{h}_{a,b}(x) = h_{a,0}(x) = \lfloor \frac{ax \bmod r}{u} \rfloor$, that is, the function we obtain if we remove the offset b. For the main hash function

h^* we set $\hat{h}^* = (\hat{h}^*_1, \ldots, \hat{h}^*_{\frac{\log n}{2}})$. We construct the ℓ-th k-SUM instance by constructing k arrays $B^\ell_1, \ldots, B^\ell_k$ and setting for all $j \in [k], i \in [n]$ that $B^\ell_j[i] = (\hat{h}^*(A_j[i]), \hat{h}_\ell(A_j[i]))$. Furthermore, we add to the instance two more arrays B^ℓ_0, B^ℓ_{k+1} as follows. Let $b^*_1, b^*_2, \ldots, b^*_{\frac{\log n}{2}}$ be the offset values of $h^*_1, h^*_2, \ldots, h^*_{\frac{\log n}{2}}$ respectively, and b_ℓ be the offset value of h_ℓ. Then, we add to B^ℓ_0 and to B^ℓ_{k+1} all vectors $(\lfloor \frac{b^*_1}{u} \rfloor, \lfloor \frac{b^*_2}{u} \rfloor, \ldots, \lfloor \frac{b^*_{\log n/2}}{u} \rfloor, \lfloor \frac{b_\ell}{u} \rfloor) + v$, where $v \in [0,k]^{\frac{\log n}{2}+1}$.

We now invoke Corollary 14 on each of the $[10k \log n]$ instances we constructed to produce $10k \log n$ CNF formulas and corresponding decompositions of pathwidth $\frac{k}{2} \log n + 4\delta k \log n + 2 \log k + O(1)$, where the bound on the pathwidth follows because the output of the main hash function consists of $\frac{\log n}{2}$ integers on k bits and the output of each secondary hash function is an integer on $\log m' \leq \log(8k^2 n^{4\delta k})$ bits. Finally, in the same way as for k-XOR, for each $\ell \in [10k \log n - 1]$ we add constraints between the first $\frac{k}{2} \log n$ variables of the special bag B^*_ℓ and the special bag $B^*_{\ell+1}$ to ensure that a consistent value needs to be selected for the main hash function across all instances. This completes the construction.

4.3 Proof of Correctness

We now present the proof of correctness of the construction of the previous section, following the same outline as for Theorem 3. The main difference is that before establishing the "hard" part of the reduction (if the formula is satisfiable, then with high probability there is a k-SUM solution, given in Lemma 22) we need to prove that the expected maximum load of our main hash function is sub-polynomial, as promised. This is done in Lemma 21.

▶ **Lemma 19.** *The algorithm of Section 4.2 produces a CNF formula with primal treewidth at most* $\frac{k}{2} \log n + 4\delta k \log n + O(\log k)$*, as well as a decomposition of this width.*

Proof. As mentioned, when we invoke the algorithm of Corollary 14 we obtain path decompositions of width $\frac{k}{2} \log n + 4\delta k \log n + O(\log k)$. The extra constraints we added to ensure consistency between instances can be handled in a tree decomposition in the same way as in the proof of Lemma 9. ◀

▶ **Lemma 20.** *If the original k-SUM instance on which the algorithm of Section 4.2 is applied has a solution, then the resulting formula is satisfiable.*

Proof. This lemma is slightly more complicated to establish than Lemma 10 because the hash functions we use are not exactly linear, but the floor operation may introduce a small error. This is the reason why we added to our construction the extra arrays B^ℓ_0, B^ℓ_{k+1}, which allow us to add to the input a desired error correction value.

Let us be more precise. Suppose there is a solution $i_1, i_2, \ldots, i_k$ to the given k-SUM instance such that $\sum_{j \in [k/2]} A_j[i_j] = \sum_{j \in [k/2+1,k]} A_j[i_j] = v$. We will attempt to construct a satisfying assignment by giving a value encoding $h^*(v)$ to the first $\frac{k}{2} \log n$ variables of the special bags of the $10k \log n$ formulas. This satisfies the consistency constraints added in the end, so we need to show that for all $\ell \in [10k \log n]$ this can be extended to a satisfying assignment of the formula representing the ℓ-th k-SUM instance.

We complete the assignment to the special bag B^*_ℓ by giving value $h_\ell(v)$ to the remaining variables. We now claim there exist i_0 and i_{k+1} such that $\sum_{j \in [0,k/2]} B^\ell_j[i_j] = \sum_{j \in [k/2,k+1]} B^\ell_j[i_j] = (h^*(v), h_\ell(v))$. In that case, there exists a satisfying assignment by Corollary 14.

A potential difficulty now is that because the functions $\hat{h}$ are not exactly linear, the sum $\sum_{j \in [k/2]} B^\ell_j[i_j]$ may not exactly correspond to the target value because (i) we have used the

functions $\hat{h}$ to produce the entries of B^ℓ_j, so the offset of our hash functions is missing (ii) the functions $\hat{h}$ are not exactly linear.

However, by invoking Observation 18 we can see that the functions $\hat{h}$ have the property that for any k integers $x_1, x_2, \ldots, x_k$ we have $\hat{h}(\sum_{i\in[k]} x_i) \equiv \sum_{i\in[k]} \hat{h}(x_i) + \kappa' \mod m$, where $\kappa' \in [0, k-1]$. Because B^ℓ_0 contains entries which are equal to the missing offset vectors plus any vector from $[0,k]^{\frac{\log n}{2}+1}$, it is always possible to select a vector $B^\ell_0[i_0]$ (and similarly from B^ℓ_{k+1}) so that we can construct the target sum and hence satisfy the formula. ◀

Before we proceed with Lemma 22 which establishes the more challenging direction of the reduction we need to establish that the main hash function we are using has the desired property that with high probability no bucket has very high load. This is shown in the following lemma which states that if we select $\frac{\log n}{2}$ hash functions with k bits of output each (as we do in our construction) the probability that there exists a set of size $n^{\delta k}$ which collide on all hash functions is very small. In other words, the expected maximum load for our main hash function is $n^{o(1)}$ with high probability.

▶ **Lemma 21.** *For all $S \subseteq [u]$ with $|S| \leq n^{k/2}$ and for all $\delta > 0$ we have the following: suppose we select independently and uniformly at random $\frac{\log n}{2}$ hash functions $h_1, \ldots, h_{\frac{\log n}{2}} \in \mathcal{H}_{u,m}$ with $m = 2^k$. Then, the probability there exists $(y_1, y_2, \ldots, y_{\frac{\log n}{2}}) \in [0, 2^k-1]^{\frac{\log n}{2}}$ such that $|\bigcap_{j\in\frac{\log n}{2}} (h_j^{-1}(y_j) \cap S)| \geq n^{\delta k}$ tends to 0 as n tends to infinity.*

Proof. Fix a $\delta > 0$ and a specific vector $(y_1, y_2, \ldots, y_{\frac{\log n}{2}}) \in [0, 2^k-1]^{\frac{\log n}{2}}$. We will show that for any such vector, the probability that $|\bigcap_{j\in\frac{\log n}{2}} (h_j^{-1}(y_j) \cap S)| \geq n^{\delta k}$ is $o(\frac{1}{n^{k/2}})$. Since there are $n^{k/2}$ such vectors, by union bound we will have that the probability that a vector satisfying the stated property exists tends to 0.

Let $S_0 = S$ and for $b \in [\frac{\log n}{2}]$ we define a random variable S_b as $S_b = |\bigcap_{j\in[b]} (h_j^{-1}(y_j) \cap S)|$. In other words, S_b is the set of elements of S which are hashed to a vector that is consistent with the b first components of our selected vector. We want to argue that the probability that $|S_{\frac{\log n}{2}}| \geq n^{\delta k}$ is small.

Define for each $b \in [\frac{\log n}{2}]$ a random variable X_b which is equal to 1 if $|S_b| \leq (1 + \frac{\delta}{4})\frac{|S_{b-1}|}{2^k}$. We have that $Pr[X_b = 0] \leq \frac{2^{k+4}}{\delta^2 |S_{b-1}|}$ by Lemma 16. We now distinguish two cases:

1. Suppose $\sum_{b\in[\frac{\log n}{2}]} X_b > \frac{\log n}{2} - \frac{10}{\delta}$. Observe that for all $b \in [\frac{\log n}{2}]$ we have $|S_b| \leq |S_{b-1}|$ and furthermore, if $X_b = 1$ then $|S_b| \leq (1 + \frac{\delta}{4})\frac{|S_{b-1}|}{2^k}$. We therefore have $|S_{\frac{\log n}{2}}| \leq |S| \cdot \left(\frac{1+\frac{\delta}{4}}{2^k}\right)^{\frac{\log n}{2} - \frac{10}{\delta}} \leq \left(1 + \frac{\delta}{4}\right)^{\frac{\log n}{2}} \cdot 2^{\frac{10k}{\delta}} \leq e^{\frac{\delta \log n}{8}} 2^{\frac{10k}{\delta}} \leq n^{\frac{\delta}{2}}$ where we used that $|S| \leq n^{k/2} = (2^k)^{\frac{\log n}{2}}$, that $1 + \frac{\delta}{4} \leq e^{\frac{\delta}{4}}$, that $e^{\log n} < n^2$, and that $2^{\frac{10k}{\delta}} < n^{\frac{\delta}{4}}$ for sufficiently large n. So in this case $S_{\frac{\log n}{2}}$ is as small as desired.
2. The remaining case is $\sum_{b\in[\frac{\log n}{2}]} X_b \leq \frac{\log n}{2} - \frac{10}{\delta}$, therefore there exist at least $\frac{10}{\delta}$ distinct variables X_b which have value 0. We claim that the probability of this happening is very small. Indeed, because the hash functions $h_1, \ldots, h_{\frac{\log n}{2}}$ are chosen independently, the variables $X_b, b \in [\frac{\log n}{2}]$ are also independent. Furthermore, if $|S_{b-1}| \geq n^{\delta k}$, then $Pr[X_b = 0] \leq \frac{2^{k+4}}{\delta^2 n^{\delta k}}$. If $|S_{\frac{\log n}{2}}| \geq n^{\delta k}$, then $|S_b| \geq n^{\delta k}$ for all $b \in [\frac{\log n}{2}]$. Hence, if $|S_{\frac{\log n}{2}}| \geq n^{\delta k}$, then there are f variables X_b set to 0 with probability at most $\binom{\frac{\log n}{2}}{f}\left(\frac{2^{k+4}}{\delta^2 n^{\delta k}}\right)^f \leq \left(\frac{2^{k+4}\log n}{2\delta^2 n^{\delta k}}\right)^f$. Since this is a decreasing function of f (for n sufficiently large) and we are

in the case $f \geq \frac{10}{\delta}$, we have that the probability that $\sum_{b \in [\frac{\log n}{2}]} X_b < \frac{\log n}{2} - \frac{10}{\delta}$ is at most $\frac{\log n}{2} \left(\frac{2^{k+4} \log n}{2\delta^2 n^{\delta k}} \right)^{\frac{10}{\delta}} = o(\frac{1}{n^k})$, as desired.

In other words, the above analysis establishes that either almost all hash functions are successful (in the sense that they place at most an $(1 + \frac{\delta}{4})$ fraction of the remaining items on the bucket we care about), in which case in the end our bucket contains few items; or many hash functions fail. However, by Lemma 16 this can happen with probability at most (roughly) $n^{-\delta k}$ if the current collection of items is large (larger than $n^{\delta k}$), so it is very unlikely that more than $\frac{10}{\delta}$ hash functions fail in this way. ◀

▶ **Lemma 22.** *Suppose that the algorithm of Section 4.2 is applied to an instance of k-SUM that has no solution. If* $\log k < \delta k$*, then the probability that a satisfiable formula is output is* $o(1)$*.*

Proof. We follow a strategy similar to the proof of Lemma 11 with the main difference being that we need to account for the fact that the use of the floor function in our hash introduces some minor errors, as seen in Observation 18. Suppose that the given k-SUM instance has no solution and let S_1, S_2 respectively be the sums constructible from $A_1, \ldots, A_{k/2}$ and $A_{k/2+1}, \ldots, A_k$ respectively. By assumption $S_1 \cap S_2 = \emptyset$. Also, $|S_1|, |S_2| \leq n^{k/2}$.

Our main hash function is made up of $\frac{\log n}{2}$ independently chosen hash functions, as required by Lemma 21. We will therefore condition our analysis on the event that the statement of Lemma 21 is satisfied, that is, for all $y = (y_1, y_2, \ldots, y_{\frac{\log n}{2}})$, there are at most $n^{\delta k}$ elements of $|S_1 \cup S_2|$ mapped to y by h^*.

Suppose that we want to find a satisfying assignment to the produced CNF formula and observe that because of the consistency constraints added in the last step we must select a consistent value to the $\frac{k}{2} \log n$ variables that represent the output of the main hash function h^*. Fix then such an assignment $y = (y_1, \ldots, y_{\frac{\log n}{2}})$ and we will show that the probability that this assignment can be extended to an assignment that satisfies the formula is $o(\frac{1}{n^{k/2}})$ and the lemma will follow by union bound.

Consider all vectors $\tilde{y}$ which are "close" to y, meaning that each coordinate differs from the corresponding coordinate of y by at most $2k$. The set of all such vectors is at most $(4k)^{\frac{\log n}{2}} < n^{\log k}$. Since we are conditioning on the event that no vector has more than $n^{\delta k}$ elements of $S_1 \cup S_2$ mapped to it, the total number of elements of $S_1 \cup S_2$ which are mapped to a vector $\tilde{y}$ that is close to y is at most $n^{\delta k + \log k} < n^{2\delta k}$.

We now observe that if we fix y as the assignment encoding the output of h^*, by Corollary 14 for all $\ell \in [10k \log n]$ in order to produce a satisfying assignment to the formula representing the ℓ-th k-SUM instance, we must select elements of S_1, S_2 which are mapped by h^* close to y. This can be seen because by an analysis similar to that of Lemma 20 the sums of the hashed values differ from the hashed value of a sum by at most $k/2$ (using Observation 18) and the extra elements of B_0^ℓ allow us to modify a sum by at most k. Therefore, there are at most $n^{2\delta k}$ elements of $S_1 \cup S_2$ which correspond to possible satisfying assignments of the ℓ k-SUM instances (extending the assignment encoding y).

Now we recall that we have set up the range of the secondary hash function to be at least $4k^2 n^{4\delta k}$, meaning that, according to Lemma 17 each secondary hash function is collision-free with probability at least $\frac{1}{2}$ on the set of elements of $S_1 \cup S_2$ which may lead to a satisfying assignment extending the current one. More strongly, Lemma 17 establishes not only collision-freeness but the property that any two distinct elements are mapped to values at distance more than $2k$. If there exists a secondary hash function that achieves this strong collision-freeness property, then the corresponding k-SUM instance cannot be satisfied while

extending the current assignment (because using Observation 18 we can bound the error due to non-linearity by at most $k/2$). We now observe that the probability that no secondary hash function achieves the strong collision-freeness property is at most $2^{-10k\log n} = o(\frac{1}{n^k})$. Hence, the probability that the assignment encoding y can be extended to a satisfying assignment to the whole instance is small, as desired. ◀

Proof of Theorem 12. Suppose that we have an algorithm solving SAT in time $2^{(1-\varepsilon)\mathrm{tw}}|\phi|^{O(1)}$. We set $\delta = \frac{\varepsilon}{40}$ and k_0 sufficiently large (to be defined later). We now execute the algorithm of Section 4.2 to obtain an instance which preserves the answer with high probability according to Lemmas 20 and 22.

Suppose the original instance has k arrays of n integers. We constructed $O(k\log n)$ hashed instances, with total size $O(k^{O(\log n)}) = n^{O(\log k)}$ (this is dominated by the arrays B_0^ℓ, B_{k+1}^ℓ which contain a vector for each $[0,k]^{\log n/2}$). The algorithm of Corollary 14 runs in polynomial time, so the final formula has size $n^{O(\log k)}$, hence the $|\phi|^{O(1)}$ factor of the running time contributes at most $n^{C\log k}$ for some constant C. Let k_0 be sufficiently large that $C\log k_0 < \delta k_0$. Then, the total running time is at most $2^{(1-\varepsilon)(\frac{k}{2}\log n + 4\delta k\log n)}n^{\delta k}$ which is at most $n^{\frac{k}{2}(1-\varepsilon+10\delta)}$ and since δ is sufficiently small we obtain the theorem. ◀

5 Applications

As mentioned, the main application of the results of this paper is in the fine-grained parameterized complexity of problems parameterized by treewidth. In this area, numerous results are known proving that known dynamic programming algorithms are optimal, under the SETH, or more recently the pw-SETH. In particular, Lokshtanov, Marx, and Saurabh [53] started a line of work which succeeded in attacking problems for which an algorithm running in $c^{\mathrm{tw}}n^{O(1)}$ was known, for c a constant, and showing that an algorithm with running time $(c-\varepsilon)^{\mathrm{tw}}n^{O(1)}$ would falsify the SETH. The list of problems for which such tight results are known is long and includes Independent Set, Dominating Set, k-Coloring, Max Cut [53], Steiner Tree, Feedback Vertex Set [28], #-Perfect Matching [25], and numerous other problems [20, 34, 35, 45, 50, 51, 54]. Analogous results are also known for other structural parameters, such as clique-width and cut-width [18, 19, 44, 47].

We will rely heavily on the work of Iwata and Yoshida who first put forward the tw-SETH as a hypothesis (though not explicitly under this name) [42]. Their main result was the following:

▶ **Theorem 23.** *The following statements are equivalent:*

1. *(The tw-SETH is false): There exists $\varepsilon > 0$ and an algorithm solving SAT in time $(2-\varepsilon)^{tw}|\phi|^{O(1)}$, where tw is the width of a given tree decomposition of the primal graph of ϕ.*
2. *The same as the previous statement, but for 3-SAT.*
3. *The same as the previous statement, but for Max-2-SAT.*
4. *There exists an $\varepsilon > 0$ and an algorithm solving Independent Set in time $(2-\varepsilon)^{tw}n^{O(1)}$.*
5. *There exists an $\varepsilon > 0$ and an algorithm solving Independent Set in time $(2-\varepsilon)^{cw}n^{O(1)}$, where cw is the width of a given clique-width expression.*

The main result of this paper is that the k-SUM and k-XOR Hypotheses imply the tw-SETH. As a result we immediately obtain that the two hypotheses imply all the statements of Theorem 23. Going a bit further, we prove the statements of Theorem 2 (stated in Section 1).

Proof of Theorem 2. The first two statements follow immediately from Theorems 3, 12, and the last two statements of Theorem 23.

The third statement follows from the same theorems and the third statement of Theorem 23 via the following simple reduction from Max-2-SAT to Max-Cut.

▷ Claim 24. There is a treewidth-preserving reduction from Max-2-SAT to Max-Cut.

Proof. We describe a reduction that produces an edge-weighted instance of Max-Cut where parallel edges are allowed. This can be replaced by an unweighted instance by replacing edges of weight w with w parallel edges of weight 1; and this can in turn be reduced to a simple graph by replacing all edges with paths of length 3 through new vertices and adding $2m$ to the target cut value, where m is the number of edges. These transformations do not affect the treewidth of the instance.

Suppose then that we are given a 2-CNF formula ϕ with n variables and m clauses. We construct a graph that has one vertex for each variable, as well as a special vertex s_0. For each clause $c = (\ell_1 \vee \ell_2)$ we construct two vertices c_{ℓ_1} and c_{ℓ_2} and make them adjacent to each other and to s_0 via edges of weight 1. Furthermore, if ℓ_1 is a positive appearance of variable x we connect c_{ℓ_1} to the vertex representing x via a path of length 2 through a new vertex and assign weight $2m$ to its edges; while if ℓ_1 is a negative appearance of x we connect c_{ℓ_1} to x via an edge of weight $4m$ (and we do the same for ℓ_2). If the original instance had target value t we set the target for the new instance to $8m^2 + 2t$.

This completes the construction and we claim that the new graph has the same treewidth as the primal graph of ϕ up to a small additive constant. Indeed, we can take the tree decomposition of the primal graph and add s_0 everywhere. Then, for each clause $c = (\ell_1 \vee \ell_2)$ involving variables x_1, x_2 we observe that there must be a bag B containing x_1, x_2; we make a new bag which is adjacent to B and contains $x_1, x_2, s_0, c_{\ell_1}, c_{\ell_2}$ and any other (degree 2) vertices of the gadget of clause c. Proceeding in this way we obtain a valid tree decomposition.

Now, if there is an assignment to ϕ satisfying t clauses, we place in the new graph all false variable and literal vertices on the same side as s_0 and all other vertices on the other side. Observe that this cuts all edges of weight $2m$ and $4m$, which contributes $8m^2$ to the cut. Furthermore, if a clause is satisfied, at least one of the literal vertices is on the other side of s_0, so the corresponding gadget contributes 2 more and we achieve the target cut size.

For the converse direction, observe that in a cut that achieves the target value we must cut all edges of weight $2m$ or $4m$ because if one of them is not cut the cut has weight at most $8m^2 - 2m + 2t \leq 8m^2$ which is below the target. However, if all such edges are cut we can obtain an assignment to ϕ by setting to 0 all variables whose vertices are on the same side as s_0 and this assignment is consistent with the assignment we obtain if we do the same for literal vertices. Observe now that since for each clause we added a triangle, each clause gadget contributes at most 2 to the cut. Furthermore, no clause gadget can contribute 1 from its unweighted edges. Therefore, there exist at least t gadgets that contribute exactly 2. But, this can happen only if one of the literal variables is on a different side from s_0, that is, if the corresponding clause is satisfied. ◁

Finally, to obtain the last statement on k-Coloring we make the following claims, which reuse existing tools from the literature (in fact, we simply verify that the details of previous reductions go through in our setting):

▷ Claim 25. If the tw-SETH holds, then for all $\varepsilon > 0$ and $B \geq 3$, 2-CSP instances ψ over alphabets of size B cannot be solved in time $(B - \varepsilon)^{\mathrm{tw}}|\psi|^{O(1)}$.

Proof. We recall that in [49] the same statement was shown for pathwidth rather than treewidth (Lemma 13 of [49]). The proof of Lemma 13 of [49] in turn relies on Lemma 12, which states that one can reduce the arity of a given CSP instance from an arbitrary constant to 2 without increasing the pathwidth.

We observe that the proof of Lemma 12 easily goes through in our case, as it relies on replacing a constraint of high arity with a cycle on new variables and then attaching this cycle somewhere in the decomposition where all the variables of the original constraint appear. This is still possible in a tree decomposition (in fact, it is easier than in a path decomposition, as we can attach a new branch to the corresponding bag).

The proof of Lemma 13 also goes through in our case. In particular, given a decomposition of a CNF formula, we partition the boolean variables of each bag in groups of δ variables so that each will be represented by γ variables of the new CSP instance. We will have that $B^\gamma \sim 2^\delta$. The remaining construction is identical except we have consistency constraints between any two neighboring bags of the decomposition, but it can be shown that this gives the desired treewidth in the same way that these constraints are shown to have the desired pathwidth in [49]. ◁

▷ Claim 26. If there exist $\varepsilon > 0, k \geq 3$ such that k-Coloring can be solved in time $(k-\varepsilon)^{\mathrm{tw}} n^{O(1)}$, then there is an algorithm that can solve 2-CSP instances ψ over alphabets of size k in time $(k-\varepsilon)^{\mathrm{tw}} |\psi|^{O(1)}$.

Proof. This follows from the construction of Lemma 22 of [49], where for each constraint of the 2-CSP we consider each non-satisfying assignment and represent by a simple gadget which is essentially a short path between the vertices representing the variables of the constraint. This construction does not increase the pathwidth by more than an additive constant, and it also clearly does not significantly increase the treewidth either. ◁

We remark that, as in [49], it is not too hard to also obtain a reduction in the converse direction (from k-Coloring parameterized by treewidth to tw-SETH), but this is not necessary to establish Theorem 2. ◀

References

1 Amir Abboud, Karl Bringmann, Holger Dell, and Jesper Nederlof. More consequences of falsifying SETH and the orthogonal vectors conjecture. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 253–266. ACM, 2018. doi:10.1145/3188745.3188938.

2 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-based lower bounds for subset sum and bicriteria path. *ACM Trans. Algorithms*, 18(1):6:1–6:22, 2022. doi:10.1145/3450524.

3 Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 375–388. ACM, 2016. doi:10.1145/2897518.2897653.

4 Amir Abboud and Kevin Lewi. Exact weight subgraphs and the k-sum conjecture. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, volume 7965 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2013. doi:10.1007/978-3-642-39206-1_1.

5 Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. *SIAM J. Comput.*, 47(3):1098–1122, 2018. doi:10.1137/15M1050987.

6 Divesh Aggarwal and Rajendra Kumar. Why we couldn't prove SETH hardness of the closest vector problem for even norms! In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 2213–2230. IEEE, 2023. doi:10.1109/FOCS57990.2023.00138.

7 Eric Allender, Shiteng Chen, Tiancheng Lou, Periklis A. Papakonstantinou, and Bangsheng Tang. Width-parametrized SAT: time–space tradeoffs. *Theory Comput.*, 10:297–339, 2014. URL: https://doi.org/10.4086/toc.2014.v010a012, doi:10.4086/TOC.2014.V010A012.

8 Noga Alon, Martin Dietzfelbinger, Peter Bro Miltersen, Erez Petrank, and Gábor Tardos. Linear hash functions. *J. ACM*, 46(5):667–683, 1999. doi:10.1145/324133.324179.

9 Martin Babka. A note on the size of largest bins using placement with linear transformations. *CoRR*, abs/1810.04161, 2018. URL: http://arxiv.org/abs/1810.04161, arXiv:1810.04161.

10 Ilya Baran, Erik D. Demaine, and Mihai Pătraşcu. Subquadratic algorithms for 3sum. *Algorithmica*, 50(4):584–596, 2008. URL: https://doi.org/10.1007/s00453-007-9036-3, doi:10.1007/S00453-007-9036-3.

11 Tatiana Belova, Nikolai Chukhin, Alexander S. Kulikov, and Ivan Mihajlin. Improved space bounds for subset sum. In Timothy M. Chan, Johannes Fischer, John Iacono, and Grzegorz Herman, editors, *32nd Annual European Symposium on Algorithms, ESA 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, volume 308 of *LIPIcs*, pages 21:1–21:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. URL: https://doi.org/10.4230/LIPIcs.ESA.2024.21, doi:10.4230/LIPICS.ESA.2024.21.

12 Tatiana Belova, Alexander Golovnev, Alexander S. Kulikov, Ivan Mihajlin, and Denil Sharipov. Polynomial formulations as a barrier for reduction-based hardness proofs. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 3245–3281. SIAM, 2023. URL: https://doi.org/10.1137/1.9781611977554.ch124, doi:10.1137/1.9781611977554.CH124.

13 Tatiana Belova, Alexander S. Kulikov, Ivan Mihajlin, Olga Ratseeva, Grigory Reznikov, and Denil Sharipov. Computations with polynomial evaluation oracle: ruling out superlinear SETH-based lower bounds. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM*

Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024, pages 1834–1853. SIAM, 2024. doi:10.1137/1.9781611977912.73.

14 Andreas Björklund and Petteri Kaski. The asymptotic rank conjecture and the set cover conjecture are not both true. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 859–870. ACM, 2024. doi:10.1145/3618260.3649656.

15 Hans L. Bodlaender, Carla Groenland, Hugo Jacob, Lars Jaffke, and Paloma T. Lima. XNLP-completeness for parameterized problems on graphs with a linear structure. In Holger Dell and Jesper Nederlof, editors, *17th International Symposium on Parameterized and Exact Computation, IPEC 2022, September 7-9, 2022, Potsdam, Germany*, volume 249 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. URL: https://doi.org/10.4230/LIPIcs.IPEC.2022.8, doi:10.4230/LIPICS.IPEC.2022.8.

16 Hans L. Bodlaender, Carla Groenland, Hugo Jacob, Marcin Pilipczuk, and Michal Pilipczuk. On the complexity of problems on tree-structured graphs. In Holger Dell and Jesper Nederlof, editors, *17th International Symposium on Parameterized and Exact Computation, IPEC 2022, September 7-9, 2022, Potsdam, Germany*, volume 249 of *LIPIcs*, pages 6:1–6:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. URL: https://doi.org/10.4230/LIPIcs.IPEC.2022.6, doi:10.4230/LIPICS.IPEC.2022.6.

17 Hans L. Bodlaender, Carla Groenland, Jesper Nederlof, and Céline M. F. Swennenhuis. Parameterized problems complete for nondeterministic FPT time and logarithmic space. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 193–204. IEEE, 2021. doi:10.1109/FOCS52979.2021.00027.

18 Narek Bojikian, Vera Chekan, Falko Hegerfeld, and Stefan Kratsch. Tight bounds for connectivity problems parameterized by cutwidth. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPIcs*, pages 14:1–14:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL: https://doi.org/10.4230/LIPIcs.STACS.2023.14, doi:10.4230/LIPICS.STACS.2023.14.

19 Narek Bojikian and Stefan Kratsch. A tight monte-carlo algorithm for steiner tree parameterized by clique-width. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 29:1–29:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. URL: https://doi.org/10.4230/LIPIcs.ICALP.2024.29, doi:10.4230/LIPICS.ICALP.2024.29.

20 Glencora Borradaile and Hung Le. Optimal dynamic program for r-domination problems over tree decompositions. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, volume 63 of *LIPIcs*, pages 8:1–8:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.8.

21 Marco L. Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In Madhu Sudan, editor, *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 261–270. ACM, 2016. doi:10.1145/2840728.2840746.

22 Timothy M. Chan. More logarithmic-factor speedups for 3sum, (median, +)-convolution, and some geometric 3sum-hard problems. *ACM Trans. Algorithms*, 16(1):7:1–7:23, 2020. doi:10.1145/3363541.

23 Timothy M. Chan and Qizheng He. Reducing 3sum to convolution-3sum. In Martin Farach-Colton and Inge Li Gørtz, editors, *3rd Symposium on Simplicity in Algorithms, SOSA 2020, Salt Lake City, UT, USA, January 6-7, 2020*, pages 1–7. SIAM, 2020. doi:10.1137/1.9781611976014.1.

24 Timothy M. Chan and Moshe Lewenstein. Clustered integer 3sum via additive combinatorics. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 31–40. ACM, 2015. doi:10.1145/2746539.2746568.

25 Radu Curticapean and Dániel Marx. Tight conditional lower bounds for counting perfect matchings on graphs of bounded treewidth, cliquewidth, and genus. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1650–1669. SIAM, 2016. URL: https://doi.org/10.1137/1.9781611974331.ch113, doi:10.1137/1.9781611974331.CH113.

26 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. *ACM Trans. Algorithms*, 12(3):41:1–41:24, 2016. doi:10.1145/2925416.

27 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

28 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *ACM Trans. Algorithms*, 18(2):17:1–17:31, 2022. doi:10.1145/3506707.

29 Mina Dalirrooyfard, Andrea Lincoln, Barna Saha, and Virginia Vassilevska Williams. Average-case hardness of parity problems: Orthogonal vectors, k-sum and more. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 4613–4643. SIAM, 2025. doi:10.1137/1.9781611978322.158.

30 Martin Dietzfelbinger. Universal hashing and k-wise independent random variables via integer arithmetic without primes. In Claude Puech and Rüdiger Reischuk, editors, *STACS 96, 13th Annual Symposium on Theoretical Aspects of Computer Science, Grenoble, France, February 22-24, 1996, Proceedings*, volume 1046 of *Lecture Notes in Computer Science*, pages 569–580. Springer, 1996. doi:10.1007/3-540-60922-9_46.

31 Martin Dietzfelbinger. Universal hashing via integer arithmetic without primes, revisited. In Hans-Joachim Böckenhauer, Dennis Komm, and Walter Unger, editors, *Adventures Between Lower Bounds and Higher Altitudes - Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, volume 11011 of *Lecture Notes in Computer Science*, pages 257–279. Springer, 2018. doi:10.1007/978-3-319-98355-4_15.

32 Baris Can Esmer, Jacob Focke, Dániel Marx, and Pawel Rzazewski. Fundamental problems on bounded-treewidth graphs: The real source of hardness. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 34:1–34:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. URL: https://doi.org/10.4230/LIPIcs.ICALP.2024.34, doi:10.4230/LIPICS.ICALP.2024.34.

33 Nick Fischer, Piotr Kaliciak, and Adam Polak. Deterministic 3sum-hardness. In Venkatesan Guruswami, editor, *15th Innovations in Theoretical Computer Science Conference, ITCS 2024, January 30 to February 2, 2024, Berkeley, CA, USA*, volume 287 of *LIPIcs*, pages 49:1–49:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. URL: https://doi.org/10.4230/LIPIcs.ITCS.2024.49, doi:10.4230/LIPICS.ITCS.2024.49.

34 Jacob Focke, Dániel Marx, Fionn Mc Inerney, Daniel Neuen, Govind S. Sankar, Philipp Schepper, and Philip Wellnitz. Tight complexity bounds for counting generalized dominating sets in bounded-treewidth graphs. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 3664–3683. SIAM, 2023. URL: https://doi.org/10.1137/1.9781611977554.ch140, doi:10.1137/1.9781611977554.CH140.

35 Jacob Focke, Dániel Marx, and Pawel Rzazewski. Counting list homomorphisms from graphs of bounded treewidth: tight complexity bounds. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, pages 431–458. SIAM, 2022. doi:10.1137/1.9781611977073.22.

36 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *J. ACM*, 31(3):538–544, 1984. doi:10.1145/828.1884.

37 Anka Gajentaan and Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995. doi:10.1016/0925-7721(95)00022-2.

38 Omer Gold and Micha Sharir. Improved bounds for 3sum, k-sum, and linear degeneracy. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*, pages 42:1–42:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. URL: https://doi.org/10.4230/LIPIcs.ESA.2017.42, doi:10.4230/LIPICS.ESA.2017.42.

39 Allan Grønlund and Seth Pettie. Threesomes, degenerates, and love triangles. *J. ACM*, 65(4):22:1–22:25, 2018. doi:10.1145/3185378.

40 Tim A. Hartmann and Dániel Marx. Independence and domination on bounded-treewidth graphs: Integer, rational, and irrational distances. In Olaf Beyersdorff, Michal Pilipczuk, Elaine Pimentel, and Kim Thang Nguyen, editors, *42nd International Symposium on Theoretical Aspects of Computer Science, STACS 2025, March 4-7, 2025, Jena, Germany*, volume 327 of *LIPIcs*, pages 44:1–44:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025. URL: https://doi.org/10.4230/LIPIcs.STACS.2025.44, doi:10.4230/LIPICS.STACS.2025.44.

41 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. URL: https://doi.org/10.1006/jcss.2000.1727, doi:10.1006/JCSS.2000.1727.

42 Yoichi Iwata and Yuichi Yoshida. On the equivalence among problems of bounded width. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 754–765. Springer, 2015. doi:10.1007/978-3-662-48350-3_63.

43 Michael Jaber, Vinayak M. Kumar, and David Zuckerman. Linear hashing is optimal, 2025. URL: https://arxiv.org/abs/2505.14061, arXiv:2505.14061.

44 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structural parameters, tight bounds, and approximation for (k, r)-center. *Discret. Appl. Math.*, 264:90–117, 2019. URL: https://doi.org/10.1016/j.dam.2018.11.002, doi:10.1016/J.DAM.2018.11.002.

45 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structurally parameterized d-scattered set. *Discret. Appl. Math.*, 308:168–186, 2022. URL: https://doi.org/10.1016/j.dam.2020.03.052, doi:10.1016/J.DAM.2020.03.052.

46 Mathias Bæk Tejs Knudsen. Linear hashing is awesome. *SIAM J. Comput.*, 48(2):736–741, 2019. doi:10.1137/17M1126801.

47 Michael Lampis. Finer tight bounds for coloring on clique-width. *SIAM J. Discret. Math.*, 34(3):1538–1558, 2020. doi:10.1137/19M1280326.

48 Michael Lampis. Circuits and backdoors: Five shades of the SETH. *CoRR*, abs/2407.09683, 2024. URL: https://doi.org/10.48550/arXiv.2407.09683, arXiv:2407.09683, doi:10.48550/ARXIV.2407.09683.

49 Michael Lampis. The primal pathwidth SETH. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 1494–1564. SIAM, 2025. doi:10.1137/1.9781611978322.47.

50 Michael Lampis and Manolis Vasilakis. Structural parameterizations for two bounded degree problems revisited. *ACM Trans. Comput. Theory*, 16(3):17:1–17:51, 2024. doi:10.1145/3665156.

51 Michael Lampis and Manolis Vasilakis. Structural parameterizations for induced and acyclic matching. *CoRR*, abs/2502.14161, 2025. URL: https://doi.org/10.48550/arXiv.2502.14161, arXiv:2502.14161, doi:10.48550/ARXIV.2502.14161.

52 Ray Li. Settling SETH vs. approximate sparse directed unweighted diameter (up to (NU)NSETH). In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1684–1696. ACM, 2021. doi:10.1145/3406325.3451045.

53 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018. doi:10.1145/3170442.

54 Dániel Marx, Govind S. Sankar, and Philipp Schepper. Degrees and gaps: Tight complexity results of general factor problems parameterized by treewidth and cutwidth. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 95:1–95:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. URL: https://doi.org/10.4230/LIPIcs.ICALP.2021.95, doi:10.4230/LIPICS.ICALP.2021.95.

55 Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 603–610. ACM, 2010. doi:10.1145/1806689.1806772.

56 Kevin Pratt. A stronger connection between the asymptotic rank conjecture and the set cover conjecture. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 871–874. ACM, 2024. doi:10.1145/3618260.3649620.

57 Ohad Trabelsi. (Almost) ruling out SETH lower bounds for all-pairs max-flow. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 2132–2156. SIAM, 2025. doi:10.1137/1.9781611978322.69.

58 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the international congress of mathematicians: Rio de janeiro 2018*, pages 3447–3487. World Scientific, 2018.

59 Virginia Vassilevska Williams. 3sum and related problems in fine-grained complexity (invited talk). In Kevin Buchin and Éric Colin de Verdière, editors, *37th International Symposium on Computational Geometry, SoCG 2021, June 7-11, 2021, Buffalo, NY, USA (Virtual Conference)*, volume 189 of *LIPIcs*, pages 2:1–2:2. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. URL: https://doi.org/10.4230/LIPIcs.SoCG.2021.2, doi:10.4230/LIPICS.SOCG.2021.2.

60 Alek Westover. Linear hashing: No shift, non-prime modulus, for real! *CoRR*, abs/2307.13016, 2023. URL: https://doi.org/10.48550/arXiv.2307.13016, arXiv:2307.13016, doi:10.48550/ARXIV.2307.13016.

61 Richard Ryan Williams. Strong ETH breaks with merlin and arthur: Short non-interactive proofs of batch evaluation. In Ran Raz, editor, *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, volume 50 of *LIPIcs*, pages 2:1–2:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. URL: https://doi.org/10.4230/LIPIcs.CCC.2016.2, doi:10.4230/LIPICS.CCC.2016.2.

62 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. URL: https://doi.org/10.1016/j.tcs.2005.09.023, doi:10.1016/J.TCS.2005.09.023.