


# The Orthogonal Vectors Conjecture for Branching Programs and Formulas

Daniel M. Kane<sup>1</sup>

CSE and Mathematics, UC San Diego, La Jolla CA, USA  
dakane@ucsd.edu

Richard Ryan Williams<sup>2</sup>

EECS and CSAIL, MIT, 32 Vassar St., Cambridge MA, USA  
rrw@mit.edu

 <https://orcid.org/0000-0003-2326-2233>

---

## Abstract

---

In the **ORTHOGONAL VECTORS (OV)** problem, we wish to determine if there is an orthogonal pair of vectors among  $n$  Boolean vectors in  $d$  dimensions. The *OV Conjecture (OVC)* posits that **OV** requires  $n^{2-o(1)}$  time to solve, for all  $d = \omega(\log n)$ . Assuming the OVC, optimal time lower bounds have been proved for many prominent problems in **P**, such as Edit Distance, Frechet Distance, Longest Common Subsequence, and approximating the diameter of a graph.

We prove that OVC is true in several computational models of interest:

- For all sufficiently large  $n$  and  $d$ , **OV** for  $n$  vectors in  $\{0, 1\}^d$  has branching program complexity  $\tilde{\Theta}(n \cdot \min(n, 2^d))$ . In particular, the lower and upper bounds match up to polylog factors.
- **OV** has Boolean formula complexity  $\tilde{\Theta}(n \cdot \min(n, 2^d))$ , over all complete bases of  $O(1)$  fan-in.
- **OV** requires  $\tilde{\Theta}(n \cdot \min(n, 2^d))$  wires, in formulas comprised of gates computing arbitrary symmetric functions of unbounded fan-in.

Our lower bounds basically match the best known (quadratic) lower bounds for *any* explicit function in those models. Analogous lower bounds hold for many related problems shown to be hard under OVC, such as Batch Partial Match, Batch Subset Queries, and Batch Hamming Nearest Neighbors, all of which have very succinct reductions to **OV**.

The proofs use a certain kind of input restriction that is different from typical random restrictions where variables are assigned independently. We give a sense in which independent random restrictions cannot be used to show hardness, in that OVC is false in the “average case” even for  $AC^0$  formulas:

For all  $p \in (0, 1)$  there is a  $\delta_p > 0$  such that for every  $n$  and  $d$ , **OV** instances with input bits independently set to 1 with probability  $p$  (and 0 otherwise) can be solved with  $AC^0$  formulas of  $O(n^{2-\delta_p})$  size, on all but a  $o_n(1)$  fraction of instances. Moreover,  $\lim_{p \rightarrow 1} \delta_p = 1$ .

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Circuit complexity

**Keywords and phrases** fine-grained complexity, orthogonal vectors, branching programs, symmetric functions, Boolean formulas

**Digital Object Identifier** 10.4230/LIPIcs.ITCS.2019.48

**Acknowledgements** We are very grateful to Ramamohan Paturi for raising the question of whether the OV conjecture is true for  $AC^0$  circuits. We also thank Abhishek Bhrushundi, Ludmila Glinskikh, and the anonymous reviewers for helpful comments.

---

<sup>1</sup> Supported by NSF Award CCF-1553288 (CAREER) and a Sloan Research Fellowship.

<sup>2</sup> Supported by NSF CCF-1741615 (CAREER: Common Links in Algorithms and Complexity). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.



## 1 Introduction

We investigate the following basic problem:

**ORTHOGONAL VECTORS (OV)**  
 Given:  $n$  vectors  $v_1, \dots, v_n \in \{0, 1\}^d$   
 Decide: Are there  $i, j$  such that  $\langle v_i, v_j \rangle = 0$ ?

An instructive way of viewing the **OV** problem is that we have a collection of  $n$  sets over  $[d]$ , and wish to find two disjoint sets among them. The obvious algorithm runs in time  $O(n^2 \cdot d)$ , and  $\log(n)$  factors can be shaved [33]. For  $d < \log_2(n)$ , stronger improvements are possible: there are folklore  $O(n \cdot 2^d \cdot d)$ -time and  $\tilde{O}(n + 2^d)$ -time algorithms (for a reference, see [1]). Truly subquadratic-time algorithms have recently been developed for even larger dimensionalities: the best known result in this direction is that for all constants  $c \geq 1$ , **OV** with  $d = c \log n$  dimensions can be solved in  $n^{2-1/O(\log c)}$  time [7, 19]. However, it seems inherent that, as the vector dimension  $d$  increases significantly beyond  $\log n$ , the time complexity of **OV** approaches the trivial  $n^2$  bound.

Over the last several years, a significant body of work has been devoted to understanding the following plausible lower bound conjecture:

► **Conjecture 1** (Orthogonal Vectors Conjecture (OVC) [37, 6, 9, 3]). *For every  $\varepsilon > 0$ , there is a  $c \geq 1$  such that **OV** cannot be solved in  $n^{2-\varepsilon}$  time on instances with  $d = c \log n$ .*

In other words, OVC states that **OV** requires  $n^{2-o(1)}$  time on instances of dimension  $\omega(\log n)$ . The popular Strong Exponential Time Hypothesis [27, 18] (on the time complexity of CNF-SAT) implies OVC [37]. For this reason, and the fact that the **OV** problem is very simple to work with, the OVC has been the engine under the hood of many recent conditional lower bounds on classic problems solvable within P. For example, the OVC implies nearly-quadratic time lower bounds for Edit Distance [9], approximating the diameter of a graph [34], Frechet Distance [13, 15], Longest Common Substring and Local Alignment [6], Regular Expression Matching [10], Longest Common Subsequence, Dynamic Time Warping, and other string similarity measures [3, 14], Subtree Isomorphism and Largest Common Subtree [2], Curve Simplification [16], intersection emptiness of two finite automata [35], first-order properties on sparse finite structures [24] as well as average-case hardness for quadratic-time [11]. Other works surrounding the OVC (or assuming it) include [38, 36, 5, 20, 8, 23, 26, 17, 30, 22].

Therefore it is of strong interest to prove the OVC in reasonable computational models. Note that **OV** can be naturally expressed as a depth-three formula with unbounded fan-in: an OR of  $n^2$  NORs of  $d$  ANDs on two input variables: an  $AC^0$  formula of size  $O(n^2 \cdot d)$ . Are there smaller formulas for **OV**?

### 1.1 OVC is True in Restricted Models

In this paper, we study how well **OV** can be solved in the Boolean formula and branching program models. Among the aforementioned **OV** algorithms, only the first two seem to be efficiently implementable by formulas and branching programs: for example, there are DeMorgan formulas for **OV** of size only  $O(n^2 d)$  and size  $O(nd2^d)$ , respectively (see Proposition 4).

The other algorithms do not seem to be implementable in small space, in particular with small-size branching programs. Our first theorem shows that the simple constructions solving **OV** with  $O(n^2 \cdot d)$  and  $O(n \cdot 2^d \cdot d)$  work are essentially optimal for all choices of  $d$  and  $n$ :

► **Theorem 2** (OVC For Formulas of Bounded Fan-in). *For every constant  $c \geq 1$ , **OV** on  $n$  vectors in  $d$  dimensions does not have  $c$ -fan-in formulas of size  $O(\min\{n^2/(\log d), n \cdot 2^d/(d^{1/2} \log d)\})$ , for all sufficiently large  $n, d$ .*

► **Theorem 3** (OVC For Branching Programs). ***OV** on  $n$  vectors in  $d$  dimensions does not have branching programs of size  $O(\min\{n^2, n \cdot 2^d/(d^{1/2})\}/(\log(nd) \log(d)))$ , for all sufficiently large  $n, d$ .*

As far as we know, size- $s$  formulas of constant fan-in may be more powerful than size- $s$  branching programs (but note that DeMorgan formulas can be efficiently simulated with branching programs). Thus the two lower bounds are incomparable. These lower bounds are tight up to the (negligible) factor of  $\min\{\sqrt{\log n}, d^{1/2}\} \log(d) \log(nd)$ , as the following simple construction shows:

► **Proposition 4.** ***OV** has  $AC^0$  formulas (and branching programs) of size  $O(dn \cdot \min(n, 2^d))$ .*

**Proof.** The  $O(dn^2)$  bound is obvious: take an OR over all  $\binom{n}{2}$  pairs of vectors, and use an  $AND \circ OR$  of  $O(d)$  size to determine orthogonality of the pair. For the  $O(dn2^d)$  bound, our strategy is to try all  $2^d$  vectors  $v$ , and look for a  $v$  that is equal to one input vector and is orthogonal to another input vector. To this end, take an OR over all  $2^d$  possible vectors  $w$  over  $[d]$ , and take the AND of two conditions:

1. There is a vector  $v$  in the input such that  $v = w$ . This can be computed with an OR over all  $n$  vectors of an  $O(d)$ -size formula, in  $O(nd)$  size.
2. There is a vector  $u$  in the input such that  $\langle u, w \rangle = 0$ . This can be computed with a *parallel* OR over all  $n$  vectors of an  $O(d)$ -size formula, in  $O(nd)$  size.

Note that the above formulas have constant-depth, with unbounded fan-in AND and OR gates. Since DeMorgan formulas of size  $s$  can be simulated by branching programs of size  $O(s)$ , the proof is complete.<sup>3</sup> ◀

### Formulas with symmetric gates

As mentioned above, **OV** can be naturally expressed as a depth-three formula of unbounded fan-in: an  $AC_3^0$  formula of  $O(n^2d)$  wires. We show that this wire bound is also nearly optimal, even when we allow arbitrary symmetric Boolean functions as gates. Note this circuit model subsumes both  $AC$  (made up of AND, OR, and NOT gates) and  $TC$  (made up of MAJORITY and NOT gates).

► **Theorem 5** (OVC For TC Formulas). *Every formula computing **OV** composed of arbitrary symmetric functions with unbounded fan-in needs at least  $\Omega(\min\{n^2/(\log d), n \cdot 2^d/(d^{1/2} \log d)\})$  wires, for all  $n$  and  $d$ .*

## 1.2 Lower Bounds for Batch Partial Match, Batch Subset Query, Batch Hamming Nearest Neighbors, etc.

A primary reason for studying **OV** is its ubiquity as a “bottleneck” special case of many other basic search problems. In particular, many problems have very succinct reductions from **OV** to them, and our lower bounds extend to these problems.

<sup>3</sup> This should be folklore, but we couldn’t find a reference; see the Appendix B. An anonymous referee thought we should attribute Borodin [12], but that reference only shows that circuits of depth  $s$  can be simulated in space  $O(s)$ . We need something much stronger: branching programs of size  $O(s)$  correspond to  $\log_2(s) + O(1)$  space. Lynch [31] showed that formulas of size  $s$  can be simulated in  $O(\log s)$  space, but in our case we need  $\log_2(s) + O(1)$  space.

We say that a *linear projection reduction* from a problem  $A$  to problem  $B$  is a circuit family  $\{C_n\}$  where each  $C_n$  has  $n$  input and  $O(n)$  outputs, each output of  $C_n$  depends on at most one input, and  $x \in A$  if and only if  $C_{|x|}(x) \in B$ , for all possible inputs  $x$ . Under this constrained reduction notion, it is easy to see that if **OV** has a linear projection reduction to  $B$ , then size lower bounds for **OV** (even in our restricted settings) imply analogous lower bounds for  $B$  as well. Via simple linear projection reductions which preserve both  $n$  and  $d$  (up to constant multiplicative factors), analogous lower bounds hold for many other problems which have been commonly studied, such as:

BATCH PARTIAL MATCH

Given:  $n$  “database” vectors  $v_1, \dots, v_n \in \{0, 1\}^d$  and  $n$  queries  $q_1, \dots, q_n \in \{0, 1, \star\}^d$   
Decide: Are there  $i, j$  such that  $v_i$  is a partial match of  $q_j$ , i.e. for all  $k$ ,  $q_j[k] \in \{v_i[k], \star\}$ ?

BATCH SUBSET QUERY

Given:  $n$  sets  $S_1, \dots, S_n \subseteq [d]$  and  $n$  queries  $T_1, \dots, T_n \subseteq [d]$   
Decide: Are there  $i, j$  such that  $S_i \subseteq T_j$ ?

BATCH HAMMING NEAREST NEIGHBORS

Given:  $n$  points  $p_1, \dots, p_n \in \{0, 1\}^d$  and  $n$  queries  $q_1, \dots, q_n \in \{0, 1\}^d$ , integer  $k$   
Decide: Are there  $i, j$  such that  $p_i$  and  $q_j$  differ in at most  $k$  positions?

### 1.3 “Average-Case” OVC is False, Even for AC<sup>0</sup>

The method of proof in the above lower bounds is an input restriction method that does *not* assign variables independently (to 0, 1, or  $\star$ ) at random. (Our restriction method could be viewed as a random process, just not one that assigns variables independently.) Does **OV** become easier under natural product distributions of instances, e.g., with each bit of each vector being an independent random variable? Somewhat surprisingly, we show that a reasonable parameterization of average-case OVC is false, even for AC<sup>0</sup> formulas.

For  $p \in (0, 1)$ , and for a given  $n$  and  $d$ , we call  $\mathbf{OV}(p)_{n,d}$  the distribution of **OV** instances where all bits of the  $n$  vectors are chosen independently, set to 1 with probability  $p$  and 0 otherwise. We would like to understand when  $\mathbf{OV}(p)_{n,d}$  can be efficiently solved on almost all instances (i.e., with probability  $1 - o(1)$ ). We give formulas of truly sub-quadratic size for every  $p > 0$ :

► **Theorem 6.** *For every  $p \in (0, 1)$ , and every  $n$  and  $d$ , there is an AC<sup>0</sup> formula of size  $n^{2-\varepsilon_p}$  that correctly answers all but a  $o_n(1)$  fraction of  $\mathbf{OV}(p)_{n,d}$  instances on  $n$  vectors and  $d$  dimensions, for an  $\varepsilon_p > 0$  such that  $\varepsilon_p \rightarrow 1$  as  $p \rightarrow 1$ .*

Interestingly, our AC<sup>0</sup> formulas have one-sided error, even in the worst case: if there is no orthogonal pair in the instance, our formulas *always* output 0. However, they may falsely report that there is no orthogonal pair, but this only occurs with probability  $o(1)$  on a random  $\mathbf{OV}(p)_{n,d}$  instance, for any  $n$  and  $d$ .

## 1.4 Intuition

Our lower bounds give some insight into why **OV** is hard. There are two main ideas:

1. **OV** instances with  $n$   $d$ -dimensional vectors can encode difficult Boolean functions on  $d$  inputs, requiring circuits of size  $\tilde{\Omega}(\min(2^d, n))$ . This can be accomplished by encoding those strings with “middle” Hamming weight from the truth table of a hard function with the vectors in an **OV** instance, in such a way that finding an orthogonal pair is equivalent to evaluating the hard Boolean function at a given  $d$ -bit input. This is an inherent property of **OV** that is independent of the computational model.
2. Because we are working with simple computational models, we can generally make the following kind of claim: given an algorithm for solving **OV** and given a partial assignment to all input vectors except for one appropriately chosen vector, we can propagate this partial assignment through the algorithm, and “shrink” the size of the algorithm by a factor of  $\Omega(n)$ . This sort of argument was first used by Nechiporuk [32] in the context of branching program lower bounds, and can be also applied to formulas.

Combining the two ideas, if we can “shrink” our algorithm by a factor of  $n$  by restricting the inputs appropriately, and argue that the remaining subfunction requires circuits of size  $\tilde{\Omega}(\min(2^d, n))$ , we can conclude that the original algorithm for **OV** must have had size  $\tilde{\Omega}(\min(n2^d, n^2))$ . (Of course, there are many details to verify, but this is the basic idea.)

The small  $\text{AC}^0$  formulas for **OV**( $p$ ) (the average-case setting) involve several ideas. First, given the probability  $p \in (0, 1)$  of 1 and the number of vectors  $n$ , we observe a simple phase transition phenomenon: there is only a particular range of dimensionality  $d$  in which the problem is non-trivial, and outside of this range, almost all instances are either “yes” instances or “no” instances. Second, within this “hard” range of  $d$ , the orthogonal vector pairs are expected to have a special property: with high probability, at least one orthogonal pair in a “yes” instance has noticeably fewer ones than a typical vector in the distribution. To obtain a sub-quadratic size  $\text{AC}^0$  formula from these observations, we partition the instance into small groups such that the orthogonal pair (if it exists) is the only “sparse” vector in its group, whp. Over all pairs of groups  $i, j$  in parallel, we take the component-wise OR of all sparse vectors in group  $i$  (call the resulting vector  $u_i$ ), and similarly for group  $j$  (call the result  $v_j$ ). Then we test if  $u_i$  and  $v_j$  are orthogonal. By doing so, if our formula ever reports 1, then there is some orthogonal pair in the instance (even in the worst case).

## 2 Lower Bounds

### Functions hard for the middle layer of the hypercube

In our lower bound proofs, we will use functions on  $d$ -inputs for which every small circuit fails to agree with the function on inputs of Hamming weight about  $d/2$ . Let  $\binom{[d]}{k}$  denote the set of all  $d$ -bit vectors of Hamming weight  $k$ .

► **Lemma 7.** *Let  $d$  be even, let  $\mathcal{C}$  be a set of Boolean functions, let  $N(d, s)$  be the number of functions in  $\mathcal{C}$  on  $d$  inputs of size at most  $s$ , and let  $s^* \in \mathbb{N}$  satisfy  $\log_2(N(d, s^*)) < \binom{[d]}{d/2}$ .*

*Then there is a sequence  $S$  of  $\binom{[d]}{d/2}$  pairs  $(x_i, y_i) \in \binom{[d]}{d/2} \times \{0, 1\}$ , such that each  $x_i \in \binom{[d]}{d/2}$  appears exactly once among the pairs in  $S$ , and every function  $f : \{0, 1\}^d \rightarrow \{0, 1\}$  satisfying  $f(x_i) = y_i$  (for all  $i = 1, \dots, \binom{[d]}{d/2}$ ) requires  $\mathcal{C}$ -size at least  $s^*$ .*

**Proof.** By definition, there are  $N(d, s)$  functions of size  $s$  on  $d$  inputs from  $\mathcal{C}$ , and there are  $2^{\binom{[d]}{d/2}}$  possible input/output sequences  $(x_i, y_i) \in \binom{[d]}{d/2} \times \{0, 1\}$  defined over all  $d$ -bit vectors of Hamming weight  $d/2$ . When  $2^{\binom{[d]}{d/2}} > N(d, s)$ , there is at least one input/output sequence that is not satisfied by any function in  $\mathcal{C}$  of size  $s$ . ◀

Note that it does not matter *what* is meant by “size” in the above lemma: the size measure could be gates, wires, etc., and the lemma still holds (as it is just counting). The above simple lemma applies to formulas, as follows:

► **Corollary 8.** *Let  $c \geq 2$  be a constant. There are  $\binom{d}{d/2}$  pairs  $(x_i, y_i) \in \binom{[d]}{d/2} \times \{0, 1\}$ , such that every function  $f : \{0, 1\}^d \rightarrow \{0, 1\}$  satisfying  $f(x_i) = y_i$  (for all  $i = 1, \dots, \binom{d}{d/2}$ ) needs  $c$ -fan-in formulas of size at least  $\Omega(2^d / (d^{1/2} \log d))$ .*

**Proof.** There are  $N(d, s) \leq d^{k_c \cdot s}$  formulas of size  $s$  on  $d$  inputs, where the constant  $k_c$  depends only on  $c$ . When  $2^{\binom{d}{d/2}} > d^{k_c \cdot s}$ , Lemma 7 says that there is an input/output sequence of length  $\binom{d}{d/2}$  that *no* formula of size  $s$  can satisfy. Thus to satisfy that sequence, we need a formula of size  $s$  at least large enough that  $2^{\binom{d}{d/2}} \leq d^{k_c \cdot s}$ , i.e.,  $s \geq \Omega\left(\binom{d}{d/2} / \log(d)\right) \geq \Omega(2^d / (d^{1/2} \log d))$ . ◀

## 2.1 Lower Bound for Constant Fan-in Formulas

We are now ready to prove the lower bound for Boolean formulas of constant fan-in:

► **Reminder of Theorem 2.** *For every constant  $c \geq 1$ , **OV** on  $n$  vectors in  $d$  dimensions does not have  $c$ -fan-in formulas of size  $O(\min\{n^2 / (\log d), n \cdot 2^d / (d^{1/2} \log d)\})$ , for all sufficiently large  $n, d$ .*

All of the lower bound proofs have a similar structure. We will give considerably more detail in the proof of Theorem 2 to aid the exposition of the later lower bounds.

**Proof.** To simplify the calculations, assume  $d$  is even in the following. Let  $F_{n,d+1}(v_1, \dots, v_n)$  be a  $c$ -fan-in formula of minimal size  $s$  computing **OV** on  $n$  vectors of dimension  $d + 1$ , where each  $v_i$  denotes a sequence of  $d + 1$  Boolean variables  $(v_{i,1}, \dots, v_{i,d+1})$ .

Let  $\ell$  be the number of leaves of  $F_{n,d+1}$ . Since  $F_{n,d+1}$  is minimal, each gate has fan-in at least two (gates of fan-in 1 can be “merged” into adjacent gates). Therefore (by an easy induction on  $s$ ) we have

$$s \geq \ell \geq s/2. \tag{1}$$

Observe there must be a vector  $v_{i^*}$  (for some  $i^* \in [n]$ ) whose  $d + 1$  Boolean variables appear on at most  $\ell/n$  leaves of the formula  $F_{n,d}$ . Our plan now is to leave the  $d + 1$  variables corresponding to  $v_{i^*}$  free, and set all other variables in a particular way, so that the remaining subfunction requires many gates. We consider two cases (based on how  $d$  compares with  $n$ ), which determine how we set those variables.

**Case 1.** Suppose  $\binom{d}{d/2} \leq n - 1$ . Let  $\{(x_i, y_i)\} \subseteq \binom{[d]}{d/2} \times \{0, 1\}$  be a list of hard pairs from Corollary 8, and let  $f : \{0, 1\}^d \rightarrow \{0, 1\}$  be any function that satisfies  $f(x_i) = y_i$ , for all  $i$ . By Corollary 8, such an  $f$  needs  $c$ -fan-in formulas of size at least  $\Omega(2^d / (d^{1/2} \log d))$ . Let  $\{x'_1, \dots, x'_t\} \subseteq \binom{[d]}{d/2}$  be those  $d$ -bit strings of Hamming weight  $d/2$  such that  $f(x'_i) = 1$ , for some  $t \leq \binom{d}{d/2} \leq n - 1$ .

**Case 2.** Suppose  $\binom{d}{d/2} \geq n - 1$ . Then we claim that there is a list of input/output pairs  $(x_1, y_1), \dots, (x_{n-1}, y_{n-1}) \in \binom{[d]}{d/2} \times \{0, 1\}$  such that for every  $f : \{0, 1\}^d \rightarrow \{0, 1\}$  satisfying  $f(x_i) = y_i$ , for all  $i$ ,  $f$  needs formulas of size at least  $\Omega(n / \log d)$ . To see this, note that if we take  $n - 1$  distinct strings  $x_1, \dots, x_{n-1}$  from  $\binom{[d]}{d/2}$ , there are  $2^{n-1}$  possible choices for the list of pairs. So when  $2^{n-1} > d^{k_c \cdot s}$ , there is a list of hard pairs  $(x_1, y_1), \dots,$

$(x_{n-1}, y_{n-1})$  that no formula of size  $s$  satisfies. For any function  $f : \{0, 1\}^d \rightarrow \{0, 1\}$  such that  $f(x_i) = y_i$  for all  $i = 1, \dots, n-1$ , its formula size  $s$  must be at least  $\Omega(n/\log d)$  in this case.

Let  $\{x'_1, \dots, x'_t\} \subseteq \binom{[d]}{d/2}$  be those  $d$ -bit strings of Hamming weight  $d/2$  such that  $(x'_i, 1)$  is on the list of hard pairs, for some  $t \leq n-1$ .

In either of the two cases, we will use the list of  $t \leq \min\{n-1, \binom{d}{d/2}\}$  strings  $\{x'_1, \dots, x'_t\}$  to assign all variables  $v_i$  of our **OV** formula, for all  $i \neq i^*$ . In particular, for the first  $t$  integers  $i \in [n]$  such that  $i \neq i^*$ , we substitute each  $(d+1)$ -bit input vector  $v_i$  with a distinct  $(d+1)$ -bit string  $\overline{1x'_{i'}}$  (the bit 1 concatenated with the complement of  $x_{i'}$ , obtained by flipping all the bits of  $x'_{i'}$ ). If  $t < n-1$  (which can happen in Case 1), we also substitute all other input vectors  $v_j$  where  $j \neq i^*$  with the  $(d+1)$ -bit vector  $\vec{1}$ . Note that all of the pairs of vectors substituted so far are not orthogonal to each other: for all  $i \neq i'$ , we have  $\langle \overline{1x'_i}, \overline{1x'_{i'}} \rangle \neq 0$ , and for all  $i$  we have  $\langle \overline{1x'_i}, \vec{1} \rangle \neq 0$ . Finally, we substitute a 0 in the first input bit of  $\vec{v}_{i^*}$ .

After these substitutions, the remaining formula  $F'_n$  has only  $d$  inputs, namely the last  $d$  bits of the vector  $v_{i^*}$ . Moreover,  $F'_n$  is a formula with at most  $\ell/n$  leaves labeled by literals: the rest of the leaves are labeled with 0/1 constants. After simplifying the formula (replacing all gates with some 0/1 inputs by equivalent functions of smaller fan-in, and replacing gates of fan-in 1 by wires), the total number of leaves of  $F'_n$  is now at most  $\ell/n$ . Therefore by (1) we infer that

$$\text{size}(F'_n) \leq 2\ell/n. \quad (2)$$

Since  $F_{n,d+1}$  computes **OV**, it follows that for every input vector  $z \in \{0, 1\}^d$  of Hamming weight  $d/2$ ,  $F'_n$  on input  $z$  outputs 1 if and only if there is some  $i$  such that  $\langle \overline{1x'_i}, 0z \rangle = 0$ . Note that since both  $\overline{1x'_i}$  and  $z$  have Hamming weight exactly  $d/2$ , we have  $\langle \overline{1x'_i}, 0z \rangle = 0$  if and only if  $z = x_i$ .

Let  $f : \{0, 1\}^d \rightarrow \{0, 1\}$  be any function that is consistent with the list of hard pairs  $\{(x_i, y_i)\}$  (from Corollary 8 in case 1, and our claim in case 2). By our choice of  $x_i$ 's, it follows that for all  $z \in \{0, 1\}^d$  of Hamming weight  $d/2$ ,  $F'_n(z) = 1$  if and only if  $f(z) = 1$ . By our choice of  $f$ , we must have

$$\text{size}(F'_n) \geq \min\{\Omega(2^d/(d^{1/2} \log d)), \Omega(n/\log d)\}, \quad (3)$$

depending on whether  $\binom{d}{d/2} \leq n-1$  or not (case 1 or case 2). Combining (2) and (3), we infer that

$$\ell \geq \Omega(n \cdot \min\{2^d/(d^{1/2} \log d), n/\log d\}). \quad (4)$$

Therefore the overall lower bound on formula size is  $s \geq \Omega\left(\min\left\{\frac{n^2}{\log d}, \frac{n \cdot 2^d}{d^{1/2} \log d}\right\}\right)$ .  $\blacktriangleleft$

### Remark on a Red-Blue Variant of **OV**

In the literature, **OV** is sometimes posed in a different form, where half of the vectors are colored red, half are colored blue, and we wish to find a red-blue pair which is orthogonal. Calling this form **OV'**, we note that **OV'** also exhibits the same lower bound up to constant factors. Given an algorithm/formula/circuit  $A$  for computing **OV'** on  $2n$  vectors ( $n$  of which are red, and  $n$  of which are blue), it is easy to verify that an algorithm/formula/circuit for **OV** on  $n$  vectors results by simply putting two copies of the set of vectors in the red and blue parts. Thus our lower bounds hold for the red-blue variant as well.

## 2.2 Lower Bound for Branching Programs

Recall that a branching program of size  $S$  on  $n$  variables is a directed acyclic graph  $G$  on  $S$  nodes, with a distinguished start node  $s$  and exactly two sink nodes, labeled 0 and 1 respectively. All non-sink nodes are labeled with a variable  $x_i$  from  $\{x_1, \dots, x_n\}$ , and have one outgoing edge labeled  $x_i = 1$  and another outgoing edge labeled  $x_i = 0$ . The branching program  $G$  evaluated at an input  $(a_1, \dots, a_n) \in \{0, 1\}^n$  is the subgraph obtained by only including edges of the form  $x_i = a_i$ , for all  $i = 1, \dots, n$ . Note that after such an evaluation, the remaining subgraph has a unique path from the start node  $s$  to a sink; the sink reached on this unique path (be it 0 or 1) is defined to be the output of  $G$  on  $(a_1, \dots, a_n)$ .

► **Reminder of Theorem 3.** **OV** on  $n$  vectors in  $d$  dimensions does not have branching programs of size  $O(\min\{n^2, n \cdot 2^d / (d^{1/2})\} / (\log(nd) \log(d)))$ , for all sufficiently large  $n, d$ .

**Proof Sketch.** The proof is similar to Theorem 2; here we focus on the steps of the proof that are different. Let  $G$  be a branching program with  $S$  nodes computing **OV** on  $n$  vectors in  $d + 1$  dimensions. Each node of  $G$  reads a single input bit from one of the input vectors; thus there is an input vector  $v_{i^*}$  that is read only  $O(S/n)$  times in the entire branching program  $G$ .

We will assign all variables other than the  $d + 1$  variables that are part of  $v_{i^*}$ . Using the same encoding as Theorem 2, by assigning  $n - 1$  other vectors, we can implement a function  $f : \{0, 1\}^d \rightarrow \{0, 1\}$  that is hard for branching programs to compute on the  $d$ -bit inputs in  $\binom{[d]}{[d/2]}$ . In particular, we substitute  $(d + 1)$ -bit vectors which represent inputs from  $f^{-1}(1) \cap \binom{[d]}{[d/2]}$  for all  $n - 1$  input vectors different from  $v_{i^*}$ . For each of these assignments, we can reduce the size of the branching program accordingly: for each input bit  $x_j$  that is substituted with the bit  $a_j$ , we remove all edges with the label  $x_j = \neg a_j$ , so that every node labeled  $x_j$  now has outdegree 1. After the substitution, two properties hold:

1. There is a hard function  $f$  such that the minimum size  $T$  of a branching program computing  $f$  on the  $n - 1$  inputs satisfies  $T \log_2(T) \geq \Omega(\min\{\binom{[d]}{[d/2]}, n\} / \log(d))$ . To see that such an  $f$  exists, we again use a counting argument. First note there are  $d^T \cdot 2^{\Theta(T \log(T))}$  branching programs of size  $T$  on  $d$  inputs (there are  $d^T$  choices for the node labels, and  $2^{\Theta(T \log(T))}$  choices for the remaining graph on  $T$  nodes). In contrast, there are at least  $2^{\min\{\binom{[d]}{[d/2]}, n-1\}}$  choices for the hard function  $f$ 's values on  $d$ -bit inputs of Hamming weight  $d/2$ . Therefore there is a function  $f$  such that  $d^T \cdot 2^{\Theta(T \log(T))} \geq 2^{\min\{\binom{[d]}{[d/2]}, n-1\}}$ , or

$$T + \Theta(T \log(T)) \geq \min \left\{ \binom{[d]}{[d/2]}, n - 1 \right\} / \log_2(d).$$

2. The minimum size of a branching program computing a function  $f : \{0, 1\}^d \rightarrow \{0, 1\}$  on the remaining  $d$  bits of input is at most  $O(S/n)$ . This follows because every node  $v$  with outdegree 1 can be removed from the branching program without changing its functionality: for every arc  $(u, v)$  in the graph, we can replace it with the arc  $(u, v')$ , where  $(v, v')$  is the single edge out of  $v$ , removing the node  $v$ .

Combining these two points, we have  $(S/n) \cdot \log(S/n) \geq \Omega\left(\min\left\{\binom{[d]}{[d/2]}, n\right\} / \log(d)\right)$ , or

$$S \geq \Omega\left(\frac{\min\{n \binom{[d]}{[d/2]}, n^2\}}{\log(S/n) \cdot \log(d)}\right).$$

Since  $S \leq n^2 d$ , we have

$$S \geq \Omega\left(\frac{\min\{n \binom{[d]}{[d/2]}, n^2\}}{\log(nd) \cdot \log(d)}\right) \geq \Omega\left(\frac{\min\{n \cdot 2^d / d^{1/2}, n^2\}}{\log(nd) \cdot \log(d)}\right).$$

This concludes the proof. ◀



## 2.3 Formulas With Symmetric Gates

We will utilize a lower bound on the number of functions computable by symmetric-gate formulas with a small number of wires:

► **Lemma 9.** *There are  $n^{O(w)}$  symmetric-gate formulas with  $w$  wires and  $n$  inputs.*

**Proof.** There is an injective mapping from the set of trees of unbounded fan-in and  $w$  wires into the set of binary trees with at most  $2w$  nodes: simply replace each node of fan-in  $k$  with a binary tree of at most  $2k$  nodes. The number of such binary trees is  $O(4^{2w})$  (by upper bounds on Catalan numbers). This counts the number of “shapes” for the symmetric formula; we also need to count the possible gate assignments. There are  $2^{k+1}$  symmetric functions on  $k$  inputs. So for a symmetric-gate formula with  $g$  gates, where the  $i$ th gate has fan-in  $w_i$  for  $i = 1, \dots, g$ , the number of possible assignments of symmetric functions to its gates is  $\prod_{i=1}^g 2^{w_i+1} = 2^{g+\sum_i w_i} = 2^{g+w}$ . There are at most  $w$  leaves, and there are  $n^w$  ways to choose the variables read at each leaf. Since  $g \leq w$ , we conclude that there are at most  $4^{2w} \cdot 2^{2w} \cdot n^w \leq n^{O(w)}$  symmetric-gate formulas with  $w$  wires. ◀

► **Reminder of Theorem 5.** *Every formula computing  $\mathbf{OV}$  composed of arbitrary symmetric functions with unbounded fan-in needs at least  $\Omega(\min\{n^2/(\log d), n \cdot 2^d/(d^{1/2} \log d)\})$  wires, for all  $n$  and  $d$ .*

**Proof.** (Sketch) With Lemma 9 in hand, the proof is quite similar to the previous lower bounds, so we just sketch the ideas. Let  $F$  be a symmetric-gate formula for computing  $\mathbf{OV}$  with unbounded fan-in and  $w$  wires. Let  $w_i$  be the number of wires touching inputs and  $w_g$  be the number of wires that do not touch inputs. Since  $F$  is a formula, we have (by a simple induction argument) that  $w_i \geq w_g$ , thus

$$w \leq 2w_i. \tag{5}$$

As before, each leaf of the formula is labeled by an input from one of the input  $n$  vectors; in this way, every leaf is “owned” by one of the  $n$  input vectors. We will substitute a 0/1 variable assignment to all vectors, except the vector  $\vec{z}^*$  which owns the fewest leaves. This gives a 0/1 assignment to all but  $O(w_i/n)$  of the  $w_i$  wires that touch inputs.

After any such variable assignment, we can simplify  $F$  as follows. For every symmetric-function gate  $g$  which has  $w_g$  input wires with  $k$  wires assigned 0/1, we can replace  $g$  with a symmetric function  $g'$  that has only  $w_g - k$  inputs, and no input wires assigned 0/1 (a partial 0/1 assignment to a symmetric function just yields another symmetric function on a smaller set of inputs). If  $g'$  is equivalent to a constant function itself, then we remove it from the formula and substitute its output wire with that constant, repeating the process on the gates that use the output of  $g$  as input. When this process completes, our new formula  $F'$  has  $d$  inputs and no wires that are assigned constants. So  $F'$  has  $O(w_i/n)$  wires touching inputs, and therefore by (5) the total number of wires in  $F'$  is  $O(w/n)$ .

As described earlier, the  $n - 1$  vectors we assign can implement  $2^{\min\{n-1, \binom{d}{d/2}\}}$  different functions on  $d$ -bit inputs, but there are at most  $d^{O(w/n)}$  functions computable by the symmetric formula remaining, by Lemma 9. Thus the number of wires  $w$  must satisfy  $d^{O(w/n)} \geq 2^{\min\{n-1, \binom{d}{d/2}\}}$ , or

$$w \geq \Omega(\min\{n^2, n \cdot 2^d/(d^{1/2})\}/(\log d)).$$

This completes the proof. ◀

### 3 Small Formulas for OV in the Average Case

Recall that for  $p \in (0, 1)$  and for a fixed  $n$  and  $d$ , we say that  $\mathbf{OV}(p)_{n,d}$  is the distribution of  $\mathbf{OV}$  instances where all bits of the  $n$  vectors from  $\{0, 1\}^d$  are chosen independently, set to 1 with probability  $p$  and 0 otherwise. We will often say that a vector is “sampled from  $\mathbf{OV}(p)$ ” if each of its bits are chosen independently in this way. We would like to understand how efficiently  $\mathbf{OV}(p)_{n,d}$  can be solved on almost all instances (i.e., with probability  $1 - o(1)$ ), for every  $n$  and  $d$ .

► **Reminder of Theorem 6.** *For every  $p \in (0, 1)$ , and every  $n$  and  $d$ , there is an  $\text{AC}^0$  formula of size  $n^{2-\varepsilon_p}$  that correctly answers all but a  $o_n(1)$  fraction of  $\mathbf{OV}(p)_{n,d}$  instances on  $n$  vectors and  $d$  dimensions, for an  $\varepsilon_p > 0$  such that  $\varepsilon_p \rightarrow 1$  as  $p \rightarrow 1$ .*

**Proof.** Let  $\varepsilon > 0$  be sufficiently small in the following. First, we observe that  $\mathbf{OV}(p)_{n,d}$  is very easy, unless  $d$  is close to  $(2/\log_2(1/(1-p^2)))\log_2(n)$ . In particular, for dimensionality  $d$  that is significantly smaller (or larger, respectively) than this quantity, all but a  $o(1)$  fraction of the  $\mathbf{OV}(p)_{n,d}$  instances are “yes” (or “no”, respectively). To see this, note that two randomly chosen  $d$ -dimensional vectors under the  $\mathbf{OV}(p)_{n,d}$  distribution are orthogonal with probability  $(1-p^2)^d$ . For  $d = (2/\log_2(1/(1-p^2)))\log_2(n)$ , a random pair is orthogonal with probability

$$(1-p^2)^{(2/\log_2(1/(1-p^2)))\log_2(n)} = 1/n^2.$$

Thus an  $\mathbf{OV}(p)_{n,d}$  instance with  $n$  vectors has nontrivial probability of being a yes instance for  $d$  approximately  $(2/\log_2(1/(1-p^2)))\log_2(n)$ .

Therefore if  $d > (2/\log_2(1/(1-p^2)) + \varepsilon)\log_2(n)$ , or  $d < (2/\log_2(1/(1-p^2)) - \varepsilon)\log_2(n)$ , then the random instance is either almost surely a “yes” instance, or almost surely a “no” instance, respectively. These comparisons could be done with the quantities  $(2/\log_2(1/(1-p^2)) - \varepsilon)\log_2(n)$  and  $(2/\log_2(1/(1-p^2)) + \varepsilon)\log_2(n)$  (which can be hard-coded in the input) with a  $\text{poly}(d, \log n)$ -size branching program, which can output 0 and 1 respectively if this is the case.<sup>4</sup>

From here on, assume that

$$d \in [(2/\log_2(1/(1-p^2)) - \varepsilon)\log_2(n), (2/\log_2(1/(1-p^2)) + \varepsilon)\log_2(n)].$$

Note that for  $p$  sufficiently close to 1, the dimensionality  $d$  is  $\delta \log n$  for a small constant  $\delta > 0$  that is approaching 0. Thus in the case of large  $p$ , the  $\text{AC}^0$  formula given in Proposition 4 has sub-quadratic size. In particular, the size is

$$O(n \cdot 2^d \cdot d) \leq n^{1+2/\log_2(1/(1-p^2))+o(1)}. \quad (6)$$

For  $p \geq 0.867 > \sqrt{3/4}$ , this bound is sub-quadratic. For smaller  $p$ , we will need a more complex argument.

Suppose  $u, v \in \{0, 1\}^d$  are randomly chosen according to the distribution of  $\mathbf{OV}(p)$  (we will drop the  $n, d$  subscript, as we have fixed  $n$  and  $d$  at this point).

We now claim that, conditioned on the event that  $u, v$  is an orthogonal pair, both  $u$  and  $v$  are expected to have between  $(p/(1+p) - \varepsilon)d$  and  $(p/(1+p) + \varepsilon)d$  ones, with  $1 - o(1)$

<sup>4</sup> As usual,  $\text{poly}(m)$  refers to an unspecified polynomial of  $m$  of fixed degree.

probability. The event that both  $u[i] = v[i] = 1$  holds with probability  $p^2$ ; conditioned on this event never occurring, we have

$$\begin{aligned}\Pr[u[i] = 0, v[i] = 0 \mid \neg(u[i] = v[i] = 1)] &= (1-p)^2/(1-p^2), \\ \Pr[u[i] = 1, v[i] = 0 \mid \neg(u[i] = v[i] = 1)] &= p(1-p)/(1-p^2), \\ \Pr[u[i] = 0, v[i] = 1 \mid \neg(u[i] = v[i] = 1)] &= p(1-p)/(1-p^2).\end{aligned}$$

Hence the expected number of ones in  $u$  (and in  $v$ ) is only  $p(1-p)d/(1-p^2) = pd/(1+p)$ , and the number of ones is within  $(-\varepsilon d, \varepsilon d)$  of this quantity with probability  $1 - o(1)$ . (For example, in the case of  $p = 1/2$ , the expected number of ones is  $d/3$ , while a typical vector has  $d/2$  ones.)

Say that a vector  $u$  is *light* if it has at most  $(p/(1+p) + \varepsilon)d$  ones. It follows from the above discussion that, conditioned on an  $\mathbf{OV}(p)$  instance being a “yes” instance, there is an orthogonal pair with two light vectors, with probability  $1 - o(1)$ . Since the expected number of ones is  $pd$ , the probability that a randomly chosen  $u$  is light is

$$\begin{aligned}\Pr\left[u \text{ has } \leq \left(\frac{p}{1+p} + \varepsilon\right)d = pd\left(1 - \frac{p}{p+1} + \frac{\varepsilon}{p}\right) \text{ ones}\right] &\leq e^{-(p/(p+1) - \varepsilon/p)^2 pd/2} \\ &= e^{-p^3 d / (2(p+1)^2) + \Theta_p(\varepsilon)d},\end{aligned}$$

by a standard Chernoff tail bound (see Theorem 11 in Appendix A). So with high probability, there are at most  $n \cdot e^{-p^3 d / (2(p+1)^2) + \Theta_p(\varepsilon)d} = n^{1-\alpha}$  light vectors in an  $\mathbf{OV}(p)$  instance, where

$$\alpha = \frac{\log_2(e) \cdot p^3}{(p+1)^2 \log_2(1/(1-p^2))} + \Theta_p(\varepsilon) / \log_2(1/(1-p^2)).$$

Divide the  $n$  vectors of the input arbitrarily into  $n^{1-\alpha(1-\varepsilon)}$  groups  $G_1, \dots, G_{n^{1-\alpha(1-\varepsilon)}}$ , of  $O(n^{\alpha(1-\varepsilon)})$  vectors each. WLOG, suppose an orthogonal pair  $u, v$  lies in different groups  $u \in G_i$  and  $v \in G_j$ , with  $i \neq j$  (note that, conditioned on there being an orthogonal pair, this event also occurs with  $1 - o(1)$  probability). Since every vector is independently chosen, and given that  $\Pr_v[v \text{ is light}] \leq 1/n^\alpha$ , note that

$$\Pr_{v_1, \dots, v_{n^{\alpha(1-\varepsilon)}}}[\text{all } v_i \text{ in group } G_a \text{ are not light}] \geq (1 - 1/n^\alpha)^{n^{\alpha(1-\varepsilon)}} \geq 1 - 1/n^{\varepsilon\alpha},$$

for every group  $G_a$ . Thus the groups  $G_i$  and  $G_j$  have at most one light vector with probability  $1 - o(1)$ .

Let  $Light(v)$  be the function which outputs 1 if and only if the  $d$ -bit input vector  $v$  is light. Since every symmetric function has  $\text{poly}(d)$ -size formulas [29],  $Light(v)$  also has  $\text{poly}(d)$ -size formulas. We can now describe our formula for  $\mathbf{OV}(p)$ , in words:

Take the OR over all  $n^{2-2\alpha(1-\varepsilon)}$  pairs  $(i, j) \in [n^{1-\alpha(1-\varepsilon)}]^2$  with  $i < j$ :  
 Take the  $\neg$ OR over all  $k = 1, \dots, d$ , of the AND of two items:  
 1. The OR over all  $O(n^{\alpha(1-\varepsilon)})$  vectors  $u$  in group  $G_i$  of  $(Light(u) \wedge u[k])$ .  
 2. The OR over all  $O(n^{\alpha(1-\varepsilon)})$  vectors  $v$  in group  $G_j$  of  $(Light(v) \wedge v[k])$ .

To see that this works, we observe:

- If there is an orthogonal pair  $u, v$  in the instance, then recall that with probability  $1 - o(1)$ , (a)  $u$  and  $v$  are light, (b)  $u$  and  $v$  appear in different groups  $G_i$  and  $G_j$ , and (c) there are no other light vectors in  $G_i$  and no other light vectors in  $G_j$ . Thus the inner ORs

- over the group  $G_i$  (and respectively  $G_j$ ) will only output the bits of the vector  $u$  (and respectively  $v$ ). Thus the above formula, by guessing the pair  $(i, j)$ , and checking over all  $k = 1, \dots, d$  that  $(u[k] \wedge v[k])$  is *not* true, will find that  $u, v$  are orthogonal, and output 1.
- If there is no orthogonal pair, then we claim that the formula *always* outputs 0. Suppose the formula outputs 1. Then there is some  $(i, j) \in [n^{1-\alpha(1-\varepsilon)}]^2$  such that the inner product of two vectors  $V_i$  and  $W_j$  is 0, where  $V_i$  is the OR of *all* light vectors in group  $G_i$  and  $W_j$  is the OR of all light vectors in group  $G_j$ . But for these two vectors to have zero inner product, it must be that *all* pairs of light vectors (one from  $G_i$  and one from  $G_j$ ) are orthogonal to each other. Thus there is an orthogonal pair in the instance.

Using the  $\text{poly}(d)$ -size formulas for *Light*, the DeMorgan formula has size

$$O(n^{2-2\alpha(1-\varepsilon)} \cdot d \cdot n^{\alpha(1-\varepsilon)} \cdot \text{poly}(d)) \leq O(n^{2-\alpha(1-\varepsilon)} \cdot \text{poly}(d)). \tag{7}$$

Substituting in the value for  $\alpha$ , the exponent becomes

$$2 - \frac{p^3(1-\varepsilon)}{(p+1)^2 \ln(1/(1-p^2))} + \Theta_p(\varepsilon)/\ln(1/(1-p^2)).$$

Recalling that we are setting  $\varepsilon$  to be arbitrarily small (its value only affects the  $o(1)$  probability of error), the formula size is

$$n^{2 - \frac{p^3}{2(p+1)^2 \ln(1/(1-p^2))} + o(1)}.$$

Observe that our formula can in fact be made into an  $\text{AC}^0$  formula of similar size; this is easy to see except for the  $\text{poly}(d)$ -size formula for *Light*. But for  $d = O(\log n)$ , any formula of  $\text{poly}(\log n)$ -size on  $O(\log n)$  bits can be converted into an  $\text{AC}^0$  circuit of depth  $c/\varepsilon$  and size  $2^{(\log n)^\varepsilon}$ , for some constant  $c \geq 1$  and any desired  $\varepsilon > 0$ .

The final formula is the minimum of the formulas of (6) and (7). For every fixed  $p \in (0, 1]$ , we obtain a bound of  $n^{2-\varepsilon_p}$  for an  $\varepsilon_p > 0$ . ◀

## 4 Conclusion

Let us highlight two interesting directions for future work.

### Lower Bounds for AC0 Circuits?

While we give fairly strong lower bounds for Boolean formulas, our results seem to leave open the following compelling conjecture about the difficulty of **OV** with low-depth circuits:

► **Conjecture 10.** *There is a  $\delta > 0$  such that **OV** is not computable by depth-3  $\text{AC}^0$  circuits with  $n^{2-\varepsilon}$  wires.*

The obvious translation of a depth-3 circuit into a formula would blow up the size too much, so our formula lower bounds do not (easily) extend to resolve the above conjecture. Typical random restriction methods (e.g., [25]) appear to be too coarse to handle such conjectures, but perhaps deterministic restriction methods (e.g., [21]) can.

### Generalizations of OV?

It is important to note that the *largest* known lower bound for branching programs computing any explicit function is due to Neciporuk [32] from 1966, and is only  $\Omega(N^2/\log^2 N)$  for inputs of length  $N$ . A similar statement holds for Boolean formulas over the full binary basis (see for

example [28]). Our lower bounds for **OV** match these bounds up to polylogarithmic factors. Thus it would be a significant breakthrough to generalize our results to other problems believed to require *cubic* time, such as:

3-ORTHOGONAL VECTORS (3-**OV**)  
 Given:  $n$  vectors  $v_1, \dots, v_n \in \{0, 1\}^d$   
 Decide: Are there  $i, j, k$  such that  $\sum_{\ell=1}^d v_i[\ell] \cdot v_j[\ell] \cdot v_k[\ell] = 0$ ?

It is known that the Strong Exponential Time Hypothesis also implies that 3-**OV** requires  $n^{3-o(1)}$  for dimensionality  $d = \omega(\log n)$  [37, 4].

---

### References

- 1 Pairwise comparison of bit vectors, January 20, 2017. URL: <https://cstheory.stackexchange.com/questions/37361/pairwise-comparison-of-bit-vectors>.
- 2 Amir Abboud, Arturs Backurs, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Or Zamir. Subtree Isomorphism Revisited. In *SODA*, pages 1256–1271, 2016.
- 3 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight Hardness Results for LCS and Other Sequence Similarity Measures. In *FOCS*, pages 59–78, 2015.
- 4 Amir Abboud and Virginia Vassilevska Williams. Popular Conjectures Imply Strong Lower Bounds for Dynamic Problems. In *FOCS*, pages 434–443, 2014.
- 5 Amir Abboud, Virginia Vassilevska Williams, and Joshua Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *SODA*, pages 377–391. Society for Industrial and Applied Mathematics, 2016.
- 6 Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of Faster Alignment of Sequences. In *ICALP*, pages 39–51, 2014.
- 7 Amir Abboud, Richard Ryan Williams, and Huacheng Yu. More Applications of the Polynomial Method to Algorithm Design. In *SODA*, pages 218–230, 2015.
- 8 Thomas Dybdahl Ahle, Rasmus Pagh, Ilya P. Razenshteyn, and Francesco Silvestri. On the Complexity of Inner Product Similarity Join. In *PODS*, pages 151–164, 2016.
- 9 Arturs Backurs and Piotr Indyk. Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false). In *STOC*, pages 51–58, 2015.
- 10 Arturs Backurs and Piotr Indyk. Which Regular Expression Patterns Are Hard to Match? In *FOCS*, pages 457–466, 2016.
- 11 Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Average-Case Fine-Grained Hardness. *IACR Cryptology ePrint Archive*, 2017:202, 2017. URL: <http://eprint.iacr.org/2017/202>.
- 12 Allan Borodin. On Relating Time and Space to Size and Depth. *SIAM J. Comput.*, 6(4):733–744, 1977. doi:10.1137/0206054.
- 13 Karl Bringmann. Why Walking the Dog Takes Time: Fréchet Distance Has No Strongly Subquadratic Algorithms Unless SETH Fails. In *FOCS*, pages 661–670, 2014.
- 14 Karl Bringmann and Marvin Künnemann. Quadratic Conditional Lower Bounds for String Problems and Dynamic Time Warping. In *FOCS*, pages 79–97, 2015.
- 15 Karl Bringmann and Wolfgang Mulzer. Approximability of the discrete Fréchet distance. *JoCG*, 7(2):46–76, 2016.
- 16 Kevin Buchin, Maike Buchin, Maximilian Konzack, Wolfgang Mulzer, and André Schulz. Fine-grained analysis of problems on curves. In *EuroCG, Lugano, Switzerland*, 2016.
- 17 Massimo Cairo, Roberto Grossi, and Romeo Rizzi. New Bounds for Approximating Extremal Distances in Undirected Graphs. In *SODA*, pages 363–376, 2016.

- 18 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The Complexity of Satisfiability of Small Depth Circuits. In *Parameterized and Exact Complexity (IWPEC)*, pages 75–85, 2009.
- 19 Timothy M. Chan and Ryan Williams. Deterministic APSP, Orthogonal Vectors, and More: Quickly Derandomizing Razborov-Smolensky. In *SODA*, pages 1246–1255, 2016. doi:10.1137/1.9781611974331.ch87.
- 20 Krishnendu Chatterjee, Wolfgang Dvořák, Monika Henzinger, and Veronika Loitzenbauer. Model and Objective Separation with Conditional Lower Bounds: Disjunction is Harder than Conjunction. In *LICS*, pages 197–206, 2016.
- 21 Shiva Chaudhuri and Jaikumar Radhakrishnan. Deterministic restrictions in circuit complexity. In *STOC*, pages 30–36. ACM, 1996.
- 22 Lijie Chen and Ryan Williams. An Equivalence Class for Orthogonal Vectors. In *SODA*, page to appear, 2019.
- 23 Jacob Evald and Søren Dahlgaard. Tight Hardness Results for Distance and Centrality Problems in Constant Degree Graphs. *CoRR*, abs/1609.08403, 2016. arXiv:1609.08403.
- 24 Jiawei Gao, Russell Impagliazzo, Antonina Kolokolova, and R. Ryan Williams. Completeness for First-Order Properties on Sparse Structures with Algorithmic Applications. In *SODA*, pages 2162–2181, 2017.
- 25 Johan Håstad. Almost Optimal Lower Bounds for Small Depth Circuits. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, pages 6–20, 1986.
- 26 Costas S. Iliopoulos and Jakub Radoszewski. Truly Subquadratic-Time Extension Queries and Periodicity Detection in Strings with Uncertainties. In *27th Annual Symposium on Combinatorial Pattern Matching, CPM 2016, June 27-29, 2016, Tel Aviv, Israel*, pages 8:1–8:12, 2016.
- 27 Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.
- 28 Stasys Jukna. *Boolean Function Complexity: Advances and Frontiers*. Springer-Verlag, 2012.
- 29 V. M. Khrapchenko. The complexity of the realization of symmetrical functions by formulae. *Mathematical notes of the Academy of Sciences of the USSR*, 11(1):70–76, 1972.
- 30 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the Fine-grained Complexity of One-Dimensional Dynamic Programming. *CoRR*, abs/1703.00941, 2017. arXiv:1703.00941.
- 31 Nancy A. Lynch. Log Space Recognition and Translation of Parenthesis Languages. *J. ACM*, 24(4):583–590, 1977. doi:10.1145/322033.322037.
- 32 E. I. Nechiporuk. On a Boolean function. *Doklady of the Academy of Sciences of the USSR*, 169(4):765–766, 1966. English translation in *Soviet Mathematics Doklady* 7:4, pages 999–1000.
- 33 Paul Pritchard. A Fast Bit-Parallel Algorithm for Computing the Subset Partial Order. *Algorithmica*, 24(1):76–86, 1999.
- 34 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *STOC*, pages 515–524, 2013.
- 35 Michael Wehar. Intersection Non-Emptiness for Tree-Shaped Finite Automata. Available at <http://michaelwehar.com/documents/TreeShaped.pdf>, February 2016.
- 36 Richard Ryan Williams. Strong ETH Breaks With Merlin and Arthur: Short Non-Interactive Proofs of Batch Evaluation. In *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, pages 2:1–2:17, 2016. doi:10.4230/LIPIcs.CCC.2016.2.
- 37 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. See also ICALP’04.
- 38 Ryan Williams and Huacheng Yu. Finding orthogonal vectors in discrete structures. In *SODA*, pages 1867–1877, 2014. doi:10.1137/1.9781611973402.135.

## A Chernoff Bound

We use the following standard tail bound:

► **Theorem 11.** *Let  $p \in (0, 1)$  and let  $X_1, \dots, X_d \in \{0, 1\}$  be independent random variables, such that for all  $i$  we have  $\Pr[X_i = 1] = p$ . Then for all  $\delta \in (0, 1)$ ,*

$$\Pr \left[ \sum_i X_i < (1 - \delta)pd \right] \leq e^{-\delta^2 pd/2}.$$

## B DeMorgan Formulas into Branching Programs

Here we describe at a high level how to convert a DeMorgan formula (over AND, OR, NOT) of size  $s$  into a branching program of size  $O(s)$ . Our construction is similar to the log-space algorithm for formula evaluation of Lynch [31], except we have to be extremely careful with the time and space efficiency to get a branching program with only  $O(s)$  total nodes (corresponding to  $\log_2(s) + O(1)$  space).

Our branching program will perform an in-order traversal of the DeMorgan formula, maintaining a counter (from 1 to  $s$ ) of the current node being visited in the formula. The branching program begins at the root (output) of the formula. If the current node is a leaf, its value  $b$  is returned to the parent node. If the current node is not a leaf, the branching program recursively evaluates its left child (storing no memory about the current node).

The left child returns a value  $b$ . If the current node is an AND and  $b = 0$ , or the current node is an OR and  $b = 1$ , the branching program propagates the bit  $b$  up the tree (moving up to the parent). If the current node is a NOT, then the branching program moves to the parent with the value  $\neg b$ .

If none of the above cases hold, then the branching program erases the value  $b$ , and recursively evaluates the right child, which returns a value  $b$ . This value is simply propagated up the tree (note the fact that we visited the right child means that we know what the left child's value was).

Observe that we only hold the current node of the formula in memory, as well as  $O(1)$  extra bits.