Orthogonal Vectors and Related Problems

MA 589: M.Sc. Project Stage I

by

Deepanshu Kush (140110013)

under the guidance of

Prof. Srikanth Srinivasan

Department of Mathematics
Indian Institute of Technology, Bombay
Mumbai 400 076

1 Introduction

1.1 Overview of the thesis

In the remainder of section 1, we define the *Orthogonal Vectors* problem and discuss its trivial algorithms. We also state the *Orthogonal Vectors Conjecture* and *Strong Exponential Time Hypothesis* and explore their connections. In section 2, we define *Partial Match* and *Subset Query* and show their equivalences to Orthogonal Vectors. Further, we discuss a *novel* generalization of such kinds of problems and prove a dichotomy result based on a graph theoretic characterization: in some cases this generalized problem is at least as hard as Orthogonal Vectors and in the rest, it can be solved in almost linear time. In section 3, we discuss a faster-than-trivial algorithm for Orthogonal Vectors due to Abboud, Williams, and Yu [AWY15], motivating every step along the way. In section 4, we show that listing (almost) all orthogonal pairs is sub-quadratically equivalent to detecting even a single orthogonal pair. Finally in section 5, we discuss a few natural extensions that can be explored.

1.2 Preliminaries

Given two vectors $u, v \in \{0,1\}^d$, we say that they are *orthogonal* if $\sum_{i=1}^{d} u_i v_i = 0$ where the sum is considered over the field of real numbers $\mathbb{R}$ (and crucially, not over $\mathbb{F}_2$). Equivalently, they are orthogonal if $\bigvee_{i\in[d]} u_i \wedge v_i = 0$; spelled out in words, this means that there is no $i \in [d]$ such that $u_i = v_i = 1$. The Orthogonal Vectors problem is defined as follows.

Input: Lists A, B of N d-dimensional 0-1 vectors each.

Output: A pair $u \in A$ and $v \in B$ such that u and v are orthogonal, if such a pair exists.

We shall call the corresponding decision version of this problem *Orthogonal Detection*. One immediately notices that a brute-force algorithm for this problem runs in $O(N^2 \cdot d)$ time: for every pair of vectors $(u, v) \in A \times B$, simply check if they are orthogonal in $O(d)$ time. Another simple idea is to do the following:

- For each vector $v \in \{0,1\}^d$, check if v belongs to A. If it does, add it to the list A'. Similarly, create a new list B'.

- Do a brute-force search for orthogonal vectors on these new lists A' and B'.

Observe that the first step takes $O(2^d N \cdot d)$ time. The purpose of creating these new lists is to remove all redundant entries in the original lists. Once this is done, the second step, in the worst case, simply takes $O(2^{2d} \cdot d)$ time for a total running time of $O(2^d(N + 2^d) \cdot d)$. The running times of these algorithm clearly suggest considering the two cases $d > \log N$ and $d < \log N$. In the latter case, the second algorithm performs better than the first. More precisely, observe that if $d = (1-\epsilon)\log N$, then it has a running time of $O(N^{2-\epsilon} \cdot d)$. Whereas, in the former case, using the first algorithm of running time $O(N^2 \cdot d)$ would be preferable. So it is now natural to ask: can we also obtain a running time of $O(N^{2-\epsilon} \cdot d)$ when d grows faster than $\log N$? The *Orthogonal Vectors Conjecture* states that this is impossible:

Conjecture 1 (OVC, [Wil05, AWW14])**.** *If $d = \omega(\log N)$, then there is no algorithm for Orthogonal Vectors problem that runs in time $\tilde{O}(N^{2-\epsilon})$ for any $\epsilon > 0$.*

Here the notation $\tilde{O}(\cdot)$ is the same as $O(\cdot)$ except that it hides any logarithmic factors in N. It is convenient to use here as we have already convinced ourselves that an interesting regime for d is when it grows as $c \log N$ for some constant $c > 0$. We shall only work in this regime throughout this document.

While the orthogonal vectors problem is of interest in multiple areas, especially in computational geometry to prove lower bounds assuming OVC (see for example[Wil18], here we shall see its connection to one of the most important problems in computer science, the boolean satisfiability problem (CNF-SAT), which is defined as follows:

> **Input**: Boolean variables $x_1, \ldots, x_n$ and a formula in the conjunctive normal form i.e. of the form $C_1 \wedge \ldots \wedge C_m$ where each C_i is the logical OR of these variables or their negations
>
> **Output**: 1 if there exists an assignment to these variables on which this formula evaluates to 1 and 0 otherwise

The same problem but with the added constraint that each clause of the input formula contains at most k literals (a variable or its negation is called a literal) is called k-CNF-SAT.

1.3 Connection between OVC and SETH

We will now see a reduction from k-CNF-SAT to Orthogonal Vectors for any k.

Theorem 1. *There is an algorithm which on an instance F of k-CNF-SAT with n variables and m clauses produces an instance T of Orthogonal Detection such that F is yes instance if and only if T is a yes instance. Further, this reduction runs in time $O(2^{n/2}m)$.*

Proof. The technique we shall use to give the reduction is called *split and list.* Consider a k-CNF formula F on variable set $x_1, \ldots, x_n$ and with claues $C_1, \ldots, C_m$. We will construct two sets of d-dimensional boolean vectors A and B with $N = |A| = |B| = 2^{n/2}$ and $d = m$.

Split the variables into two sets and let α be an assigment to the first set of variables. Define the vector A_α as

$$(A\alpha)_j = \begin{cases} 1, & \text{if } \alpha \text{ does not satisfy } C_i \\ 0, & \text{otherwise} \end{cases}$$

Symmetrically, for β an assignment to the second set of variables define B_β as

$$(B_\beta)_j = \begin{cases} 1, & \text{if } \beta \text{ does not satisfy } C_i \\ 0, & \text{otherwise} \end{cases}$$

We define A as the set of all A_α obtained that way and B as the set of all B_β. We have that α, β satisfies the formula if and only if for all j either α or β satisfies C_j. Hence either $(A_\alpha)_j = 0$ or $(B_\beta)_j = 0$, which is the case exactly if A_α and B_β are orthogonal. The running time is $O(2^{n/2}m)$ as for each assignment α (respectively β) to the first set of variables, it takes $O(m)$ time to construct the boolean vector A_α (respectively B_β). $\square$

Analogous to OVC, a conjecture about the lower bound for Orthogonal Vectors, is a conjecture about the lower bound for SAT, which is called the *Strong Exponential Time Hypothesis*:

Conjecture 2 (SETH, [CIP06]). *For every $\epsilon > 0$, there exists $k \in \mathbb{N}$ such that k-CNF-SAT requires $\Omega(2^{n-\epsilon n})$ time.*

Next, we observe the following relationship between the two conjectures using theorem 1.

Corollary 2. *SETH implies OVC.*

Proof. Let us prove the contrapositive. Assume that we have an algorithm for orthogonal vectors that runs in time $\tilde{O}(N^{2-\epsilon})$ for some $\epsilon > 0$ if $d = \omega(\log N)$. Then we have such an algorithm for $d = C \log N$ for all constants C. Hence, using theorem 1, for any k we get a total running time to solve k-CNF-SAT (on n variables and m clauses) of

$$O(2^{n/2} \cdot m + (2^{n/2})^{2-\epsilon}) = O(2^{n-\frac{n\epsilon}{2}})$$

which contradicts SETH. □

Remark 3. *The reason we can ignore m when multiplied with 2^n in the above expression is because we can assume it be $O(n)$ due to the following result which we shall only state informally.*

Lemma 4 (Sparsification Lemma, Informal)**.** *There is a sub-exponential time algorithm which reduces the number of clauses of a k-CNF-SAT formula to $O(n)$, for any constant k.*

2 A generalization to larger alphabets

2.1 Equivalence with Partial Match and Subset Query

Before we attempt to generalize this problem by considering alphabets larger than $\{0, 1\}$, let us motivate it by first seeing a couple of examples of problems that are known to be equivalent to orthogonal vectors and an example of a problem which can be solved much faster. The *Partial Match* problem can be defined as follows:

> **Input**: Queries $x_1, \ldots, x_n$ from $\{0, 1, \star\}^d$ and a database $D \subseteq \{0, 1\}^d$ of size n
>
> **Output**: Determine if for some query x_i, there is a string $y \in D$ such that x_i matches y at all its non-$\star$ positions

In a similar spirit, we define the *Subset Query* problem:

> **Input**: Query subsets $S_1, \ldots, S_n$ of $[d]$ and a database $\mathcal{D}$ of size n containing subsets of $[d]$
>
> **Output**: Determine if for some query S_i, there is a set $T \in \mathcal{D}$ such that $S_i \subseteq T$

Let us see the equivalence between the three problems. Note that we have a natural correspondence between $S \subseteq [d]$ and a vector $v_S \in \{0,1\}^d$ where the i^{th} coordinate is 1 if and only if $i \in S$. Further, observe that $S \subseteq T \Leftrightarrow S \cap \overline{T} = \emptyset$. To reduce between subset query and orthogonal vectors: we simply turn query sets $S \subseteq [d]$ into their corresponding d-bit vectors v_S (or vice-versa) and flip the bits of the database sets $T \in \mathcal{D}$ i.e. take the ones' complement of v_T, denoted by $\overline{v_T}$. Finally, observe that $S \subseteq T$ if and only if v_S and $\overline{v_T}$ are orthogonal.

Next, given an instance of orthogonal vectors with lists A, B of n d-dimensional vectors each, we can reduce it to an instance of partial match as follows: for each $u \in A$, define a query $u^* \in \{0,1,\star\}^d$ by replacing a 0-position in u to $\star$ in u^* and keeping the 1-positions as they are. Let the database D be the collection of all $\overline{v}$ (ones' complement of v), where $v \in B$. Recall that $u \in A$ and $v \in B$ are orthogonal if and only if there is no index $i \in [d]$ such that $u_i = v_i = 1$, which if true implies $v_i = 0$ whenever $u_i = 1$. In other words, $\overline{v}$ matches u^* at all its non-$\star$ positions if and only if $u \in A$ and $v \in B$ are orthogonal. Conversely, given partial match queries $q \in \{0,1,\star\}^d$ and database vectors $v \in \{0,1\}^d$, we create an instance of orthogonal vectors as follows: construct list A, which contains $u_q \in \{0,1\}^{2d}$ for every query q, defined as: for all indices $i = 1, 2, \ldots, d$,

- if $q_i = 0$, define $(u_q)_{2i-1} = 1$ and $(u_q)_{2i} = 0$.
- if $q_i = 1$, define $(u_q)_{2i-1} = 0$ and $(u_q)_{2i} = 1$.
- if $q_i = \star$, define $(u_q)_{2i-1} = 0$ and $(u_q)_{2i} = 0$.

Similarly construct list B which contains $v' \in \{0,1\}^{2d}$ for all database vectors v, defined as follows: for all indices $i = 1, 2, \ldots, d$,

- if $v_i = 0$, define $v'_{2i-1} = 0$ and $v'_{2i} = 1$.
- if $v_i = 1$, define $v'_{2i-1} = 1$ and $v'_{2i} = 0$.

Then it is straightforward to observe that a query q matches with a database vector v at all non-$\star$ positions if and only if u_q and v' are orthogonal.

It is important to note that all these reductions run in $O(nd)$ time and therefore, in the regime when $d = C \log n$ which is what we are interested in, the existence of an $\tilde{O}(n^{2-\epsilon})$ algorithm for one problem implies so for the other two as well.

2.2 Almost linear time algorithm for a similar problem

Next, we study a problem very similar to partial match, which we shall call *Exact Match*:

> **Input**: Queries $x_1, \dots, x_n$ from $\{0,1\}^d$ and a database $D \subseteq \{0,1\}^d$ of size n
>
> **Output**: Determine if for some query x_i, there is a string $y \in D$ such that $x_i = y$

Despite the obvious similarity to partial match, we observe that *Exact Match* can be solved rather quickly.

Theorem 5. *There is an algorithm that solves Exact Match with running time $\tilde{O}(n)$.*

Proof. On input queries $x_1, \dots, x_n$ from $\{0,1\}^d$ and database $D = \{y_1, \dots, y_n\} \subseteq \{0,1\}^d$, do the following:

1. Treat each x_i as the binary representation of some integer and sort $x_1, \dots, x_n$ using any optimal sorting algorithm (for example *Merge Sort*).
2. For each $j \in [n]$, use binary search to determine if there is an $i \in [n]$ such that $y_j = x_i$.

Using the fact that sorting of d-bit numbers takes $O(dn \log n)$ time and binary search takes $O(d \log n)$ time, it is evident that this algorithm runs in $\tilde{O}(n)$ time. □

2.3 The generalization and dichotomy result

In the remainder of this section, we aim to study the reason behind this stark contrast between the runtime of Exact Match and the runtime of Partial Match (assuming OVC is true). Let us begin with a few definitions.

Definition 6 (Problem $\Pi(f, \Sigma_1, \Sigma_2)$)**.** *Suppose we are given two lists A, B of d-bit strings where the strings come from Σ_1^d and Σ_2^d respectively for some constant-sized alphabets Σ_1, Σ_2. Further, assume that $f : \Sigma_1 \times \Sigma_2 \to \{0,1\}$ is any boolean function (also called predicate). Then define $\Pi(f, \Sigma_1, \Sigma_2)$ to be the problem of determining if there are strings $s \in A, t \in B$ such that $\bigwedge_{i \in [d]} f(s_i, t_i)$.*

We readily observe that all problems that we have discussed so far immediately fall into this framework. For example, consider the example of *Partial Match.* What f, Σ_1, Σ_2 does it correspond to? Clearly, we must have $\Sigma_1 = \{0, 1, \star\}$ and $\Sigma_2 = \{0,1\}$. f must be given by the predicate PM which takes the following values:

- $PM(\star, 0) = PM(\star, 1) = 1$.
- $PM(0, 0) = PM(1, 1) = 1$.
- $PM(0, 1) = PM(1, 0) = 0$.

Definition 7 (Graph $G_{f,\Sigma_1 \times \Sigma_2}$ of $\Pi(f, \Sigma_1, \Sigma_2)$)**.** *Given f, Σ_1, Σ_2 as above, we create a bipartite graph $G_{f,\Sigma_1 \times \Sigma_2}$ with parts Σ_1, Σ_2 such that there is an edge between $u \in \Sigma_1$ and $v \in \Sigma_2$ if and only if $f(u, v) = 1$.*

The purpose of considering the graphs corresponding to the predicate f and the alphabets Σ_1, Σ_2 is to use them to precisely characterize the hardness of $\Pi(f, \Sigma_1, \Sigma_2)$: when is it as hard as *Orthogonal Vectors* and when is it as easy as *Exact Match*? Before moving on this characterization, let us first look at the graphs corresponding to Orthogonal Vectors and Exact Match. Let OV be the Orthogonal Vectors predicate on $\Omega = \{0, 1\}$ i.e. $OV(a, b) = 0 \Leftrightarrow (a, b) = (1, 1)$. Similarly, let EM be the Exact Match predicate on Ω i.e. $EM(a, b) = 1 \Leftrightarrow a = b$.

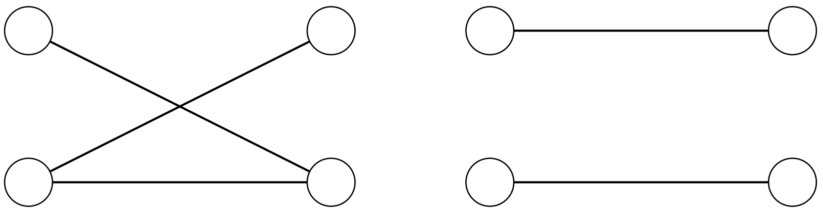
(a) A picture of G_{OV,Ω^2} (b) A picture of G_{EM,Ω^2}

Figure 1: Difference between the graphs of Orthogonal Vectors and Exact Match

We observe that G_{OV,Ω^2} is precisely a path of length 3. Also, G_{EM,Ω^2} is simply a perfect matching of size 2. Further, note that one can also define Exact Match on a larger alphabet Σ of constant size c, where the bits of both the query strings and database strings come from Σ. It is easy to see that theorem 5 will continue to hold in this case, as instead of treating strings as binary representations of integers, we can think of them as representations in base c. Moreover, G_{EM,Σ^2} will still be a perfect matching, albeit of size c.

Consider the graphs of the two problems we've shown to be equivalent to Orthogonal Vectors: Subset Query and Partial Match. What can we say about their graphs? Observe that they both contain G_{OV,Ω^2} as induced subgraphs! This leads us to the following

Theorem 8. *There is a dichotomy in the hardness of $\Pi(f, \Sigma_1, \Sigma_2)$:*

- *If $G_{f,\Sigma_1\times\Sigma_2}$ contains a path of length 3 as an induced subgraph, then $\Pi(f,\Sigma_1,\Sigma_2)$ is at least as hard as Orthogonal Vectors.*
- *Otherwise, there is an algorithm that solves $\Pi(f,\Sigma_1,\Sigma_2)$ in time $\tilde{O}(n)$.*

Proof.

- If $G_{f,\Sigma_1\times\Sigma_2}$ contains G_{OV,Ω^2} as induced subgraph, then upon relabelling if necessary, it is easy to see that any instance of Orthogonal Vectors can be simulated using $\Pi(f,\Sigma_1,\Sigma_2)$. In other words, a sub-quadratic algorithm (i.e. an algorithm with running time $\tilde{O}(n^{2-\epsilon})$ for some constant $\epsilon > 0$) for $\Pi(f,\Sigma_1,\Sigma_2)$ would imply one for Orthogonal Vectors.
- Suppose $G_{f,\Sigma_1\times\Sigma_2}$ does not contain any path of length 3 as an induced subgraph. Without loss of generality, we may assume that it does not contain any isolated vertices. Using the terminology of definition 6, this is because if $a \in \Sigma_1$ is isolated and s is a string in list A in which a appears, then we can straight away rule out the possibility of there being a $t \in B$ for which $\bigwedge_{i\in[d]} f(s_i,t_i)$. We now appeal to the following

  Lemma 9. *Suppose G is bipartite graph with parts X and Y having no isolated vertices such that it does not contain a path of length 3 as an induced subgraph. Then it must be the disjoint union of complete bipartite graphs.*

  Proof. Let us prove the contrapositive of this statement. Suppose G has a connected component G_j with parts $X_j \subseteq X$ and $Y_j \subseteq Y$ such that G_j is not complete. This means that there exist vertices $x \in X_j$ and $y \in Y_j$ that are *not* adjacent. But by assumption, neither of them are isolated; thus, there must be vertices $u \in X_j$ and $v \in Y_j$ such that x is adjacent to v and y is adjacent to u. But then the graph induced by the vertices x, y, u, v is precisely a path of length 3. □

  Applying this lemma to $G_{f,\Sigma_1\times\Sigma_2}$, we obtain that it must be the disjoint union of $G_1,\dots,G_c$ where each G_j is a complete bipartite graph with parts $X_j \subseteq \Sigma_1$ and $Y_j \subseteq \Sigma_2$. Define an alphabet $\Sigma = \{1,2,\dots,c\}$. Then given an instance of $\Pi(f,\Sigma_1,\Sigma_2)$, we reduce it to an instance of $\Pi(EM,\Sigma,\Sigma)$: for a string $s \in A \subseteq \Sigma_1^d$, define a new string $s' \in \Sigma^d$ as follows. We know that each bit s_i lies in a unique set X_j where $1 \le j \le c$. Define s'_i to be this precise $j \in \Sigma$. Let the collection of all such s' be called A'. Similarly define t' for all strings $t \in B$ using the partition $\{Y_j\}_{j=1}^c$ of Σ_2 and call this collection B'. Note that this reduction takes $O(nd)$ time. It is now clear that $\bigwedge_{i\in[d]} f(s_i,t_i) \Leftrightarrow s'_i = t'_i$. Or in other words, this is true if and only if $\bigwedge_{i\in[d]} EM(s'_i,t'_i)$. But we have already seen that Exact Match can be solved in $\widetilde{O}(n)$ time. This concludes the proof.

□

Remark 10. *Notice that we write '$G_{f,\Sigma_1\times\Sigma_2}$ contains a path of length 3 as an induced subgraph' and not simply '...contains a path of length 3'. This is important because $K_{2,2}$ is a graph containing the latter but not the former. And indeed, a problem Π whose graph is $K_{2,2}$ is reducible to Exact Match and thus, can be solved in almost linear time.*

3 A fast algorithm for Orthogonal Vectors

In this section, we shall describe a faster randomized algorithm for Orthogonal Vectors due to Abboud, Williams and Yu [AWY15]. As we can expect, the running time of this algorithm is not sub-quadratic, but nevertheless it improves significantly on the trivial $O(n^2d)$ algorithm.

Theorem 11. *For vectors of dimension $d = c(n)\log n$, Orthogonal Detection can be solved in $n^{2-1/O(\log c(n))}$ time by a randomized algorithm that is correct with high probability.*

Proof. Before moving on to the formal description and analysis of this algorithm, it is imperative that we describe the key high-level ideas involved in the proof. Suppose we are given lists A, B of n d-dimensional 0-1 vectors each, as input. There are four important ideas:

1. Reduce the problem to many subproblems of very small size i.e. divide both lists A and B into $q = \lceil \frac{n}{s} \rceil$ many sub-lists ($A_1,\ldots,A_q$ and $B_1,\ldots,B_q$) of size s each. A subproblem now is to determine whether there is a pair of orthogonal vectors in $A_i \times B_j$.

2. Construct small boolean circuits for solving these subproblems. We've already seen that x and y are orthogonal if and only if $\bigvee_{i\in[d]} x_i \wedge y_i = 0$ or in other words, if $\bigwedge_{i\in[d]} \neg x_i \vee \neg y_i$ which when expressed as a circuit is an *AND* of fan-in d over *OR*s of fan-in 2. Build on this to come up with a circuit for a subproblem (A_i, B_j) which outputs 1 if and only if there is a pair of orthogonal vectors in $A_i \times B_j$.

3. Evaluate circuits using *probabilistic* polynomials (defined as a distribution over certain polynomials) of low degree i.e. construct polynomials (over $\mathbb{F}_2$) that on an input (A_i, B_j), determine if there is a pair of orthogonal vectors in $A_i \times B_j$, *with high probability.* But how do we transform these small circuits into low-degree polynomials? This requires a tool (due to Razborov and Smolensky).

4. Finally, evaluate this polynomial over all possible q^2 pairs (A_i, B_j) in $\tilde{O}(n^2/s^2)$ time, using fast rectangular matrix multiplication (due to Coppersmith).

Now that we have this big picture in mind, let us first formally describe the two major tools that we are going to use to come up with this algorithm. The first of them is a probabilistic construction of polynomials over $\mathbb{F}_2$ that approximates small boolean circuits, due to Razborov[Raz87] and Smolensky[Smo87]. Suppose we wish to compute the OR of the bits $y_1, \dots, y_d$ by evaluating a polynomial over $\mathbb{F}_2$. It is not hard to figure out that $1 + \Pi_{j=1}^{d}(1 + y_j)$ does the job. But the question that we want to ask ourselves is that can a smaller degree polynomial do the job? If not always, then perhaps with a high probability?

Lemma 12. *Let $t, d \in \mathbb{N}$ and suppose r_{ij} are chosen independently and uniformly at random from $\mathbb{F}_2$ for $1 \leq i \leq t$ and $1 \leq j \leq d$. Define the expression*

$$A_t(y_1, \dots, y_d) = \prod_{i=1}^{t}(1 + \sum_{j=1}^{d} r_{ij}(1 + y_j)).$$

Then $\Pr_{r_{ij}}[A_t(y_1, \dots, y_d) = AND(y_1, \dots, y_d)] \geq 1 - 1/2^t$ for all $(y_1, \dots, y_d) \in \{0, 1\}^d$.

Proof. Suppose $b_1, \dots, b_d$ are chosen independently and uniformly at random from $\mathbb{F}_2$. Consider the expression $\sum_{j=1}^{d} b_j y_j$. If all $y_j = 0$, then this expression computes $OR(y_1, \dots, y_d)$ correctly with probability 1. Otherwise there is a j for which $y_j = 1$ and by the Schwartz-Zippel lemma, this expression computes $OR(y_1, \dots, y_d)$ correctly with probability at least 1/2. To bring down the error probability to $1/2^t$, we simply run this process t times, and take the AND over all outcomes. In other words, we conclude that the expression

$$B_t(y_1, \dots, y_d) = \prod_{i=1}^{t}\left(\sum_{j=1}^{d} r_{ij} y_j\right).$$

equals $OR(y_1, \dots, y_d)$ with probability at least $1 - 1/2^t$ for all choices of $y_1, \dots, y_d$. Finally, to see the conclusion about A_t, just observe that $AND(y_1, \dots, y_d) = \overline{OR(\overline{y_1}, \dots, \overline{y_d})}$, where the overline denotes the complement. □

Next, we state the main result due to Coppersmith[Cop82] that allows us to perform fast rectangular matrix multiplication.

Lemma 13. *For all sufficiently large N, multiplication of an $N \times N^{0.172}$ matrix with an $N^{0.172} \times N$ matrix can be done in $O(N^2 \log^2 N)$ arithmetic operations.*

As mentioned earlier, we also want to understand how we can use matrix multiplication in evaluating polynomials over many inputs simultaneously.

Lemma 14. *For a polynomial $P(x_1, \ldots, x_\ell, y_1, \ldots, y_\ell)$ over $\mathbb{F}_2$ with at most $N^{0.1}$ monomials, and two lists of N inputs $A = \{u_1, \ldots, u_N\} \subseteq \{0,1\}^\ell, B = \{v_1, \ldots, v_N\} \subseteq \{0,1\}^\ell$, we can evaluate P on all pairs $(u_i, v_j) \in A \times B$ in $\tilde{O}(N^2)$ time.*

Proof. We shall reduce this problem of evaluating P to fast rectangular matrix multiplication and then use lemma 13. Let $m \leq N^{0.1}$ be the number of monomials of P. First, we order the monomials of P arbitrarily. Next, we construct an $N \times m$ matrix K with its rows indexed by strings in A, and columns indexed by monomials of P in the fixed order that we have assigned to them. Similarly, we also construct an $m \times N$ matrix L whose rows are indexed by monomials of P in this assigned order and columns are indexed by strings in B. More precisely, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$, we define K_{ij} to be the value of the j^{th} monomial of P restricted to the x-variables evaluated on u_i (i.e. we set all y-variables in the j^{th} monomial to 1, and plug in the assignment defined by u_i for the variables $x_1, \ldots, x_\ell$.). Similarly for $j = 1, \ldots, m$ and $k = 1, \ldots, n$, define L_{jk} to be the value of the j^{th} monomial of P restricted to the y-variables evaluated on v_k. Then, $K_{ij} \cdot L_{jk}$ equals the value of the j^{th} monomial of P on the assignment (u_i, v_k), and therefore,

$$(L \cdot K)_{ik} = \sum_{j=1}^{m} K_{ij} \cdot L_{jk}.$$

Or in other words, the (i,k)-th entry of the product matrix $L \cdot K$ equals the value of P on the assignment (u_i, v_k). But by lemma 13, we know that this product matrix can be computed in $\tilde{O}(N^2)$ time. $\square$

Now let us expand on the second idea of writing the existence of an orthogonal vector pair in a subproblem (A', B') (where $A' \subseteq A, B' \subseteq B$ are of size s each, and where s will eventually be chosen as something small in terms of n) as a boolean circuit. Given $x_i \in A'$ and $y_j \in B'$, we know that

$$F(x_i, y_j) = \bigwedge_{k \in [d]} \neg x_i[k] \vee \neg y_j[k]$$

is true if and only if x_i and y_j are orthogonal. Therefore, the circuit

$$C(A', B') = \bigvee_{i,j=1}^{s} F(x_i, y_j)$$

evaluates to 1 if and only if there is an orthogonal pair in $A' \times B'$. Note that the circuit C is therefore an OR of s^2 ANDs of d ORs of two negations of input bits, of size $O(s^2 d)$, as can be seen from figure 2.

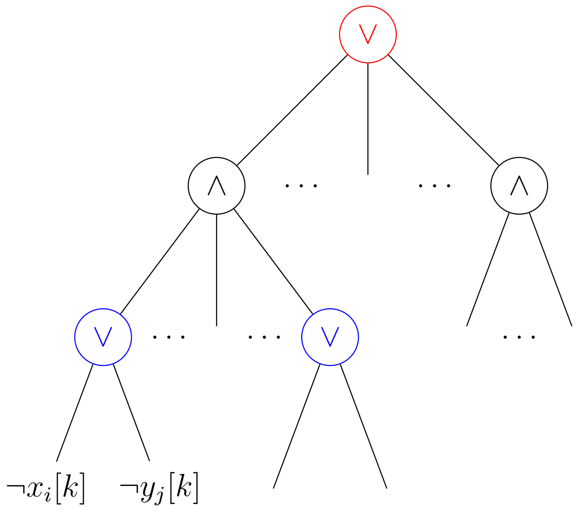

Figure 2: A picture of $C(A', B')$: the top OR gate (in red) has fan-in s^2; the bottom OR gates (in blue) have fan-in 2; the AND gates have fan-in d.

Now, we want to formalize the third idea. How do we construct probabilistic polynomials that evaluate to C with good probability? We will randomly convert C into a low-degree polynomial, as follows. First, the bottom ORs of the form $(\neg a \vee \neg b)$ can be directly converted into polynomials over $\mathbb{F}_2$, by replacing each of them with the expression $1 + a \cdot b$. Next, applying the Razborov-Smolensky construction from lemma 12 to the AND gates, we replace each of them by the expression $A_{3 \log s}$. This replacement of one AND gate will yield an error of at most $1/s^3$. Thus, the probability that any of them errs is bounded by $\frac{s^2}{s^3} = \frac{1}{s}$. Finally, we replace the top OR gate by the expression B_2 (recall its definition from the proof of lemma 12). The probability that B_2 will evaluate OR incorrectly on its inputs is at most $1/4$. Let P be the final polynomial obtained (on $2 \cdot sd$ variables i.e. each bit of each string of A' and B') after making all these replacements. Therefore, by the union bound, the probability that P does not compute $C(A', B')$ correctly is bounded by $\frac{1}{s} + \frac{1}{4} \leq \frac{1}{3}$ (we shall see that $1/s$ will turn out to be small enough for instances with sufficiently large n).

Let us now discuss the fourth and final idea. We have constructed a polynomial P that computes whether (A', B') has an orthogonal vector pair or not with probability at least $2/3$. Now, we wish to compute P efficiently on all subproblems (A_i, B_j) where $1 \leq i, j \leq q$ simultaneously (recall that $q = \lceil n/s \rceil$). To this end, we make use of lemma 14. But to do this, we first need to estimate the number of monomials in P and ensure that it is bounded by $n^{0.1}$. This will then give an idea of what s needs to be.

We note that when any of the middle ANDs is replaced with $A_{3\log s}$ (after replacing the bottom OR with the expression $1+x_i[k]\cdot y_i[k]$), the expression becomes

$$\prod_{t=1}^{3\log s}\left(1+\sum_{k=1}^{d} r_{tk}\cdot x_i[k]\cdot y_j[k]\right).$$

each of whose brackets contains $d+1$ terms and therefore, the expansion contains $O((d+1)^{3\log s})$ monomials. Now the replacement for the top OR is B_2 which has two brackets each of which is the sum of s^2 terms, each of which corresponds to one middle AND. Therefore, the total number of monomials in P can be bounded by $(s^2(d+1)^{3\log s})^2 = s^4(d+1)^{6\log s}$, which we force to be upper bounded by $n^{0.1}$. Let $s = 2^{\epsilon\frac{\log n}{\log d}}$ (the reason behind this peculiar choice will become apparent in a moment) for $\epsilon = 1/160$. Then, the number of monomials m of P is at most

$$s^4(d+1)^{6\log s} = s^4(d+1)^{6\epsilon\frac{\log n}{\log d}}$$

and therefore,

$$\begin{aligned}\log m &\leq 4\epsilon\frac{\log n}{\log d} + 6\epsilon\log n\frac{\log(d+1)}{\log d}\\ &\leq 4\epsilon\log n + 12\epsilon\log n\\ &\leq 16\epsilon\log n\\ &= 0.1\log n\end{aligned}$$

and thus, lemma 14 is applicable.

Running Time. First, we need to figure out the runtime of the process of expanding out P into a sum of monomials. Note that as written above, each of the $t_1 = 3\log s$ brackets of $A_{3\log s}$ contains $d+1$ terms. Degree of each term after expansion is t_1 and thus, writing down one term takes $O(t_1)$ time. Therefore, writing down all the terms takes $O((d+1)^{t_1}t_1)$ time. Doing so for each AND gate and adding them up takes $O(s^2(d+1)^{t_1}t_1)$ time. Finally, note that we are multiplying two such brackets to compute B_2 and therefore, in all, it takes time

$$O((s^2(d+1)^{t_1}t_1)^2) = O(s^4(d+1)^{6\log s}\log^2 s) = \tilde{O}(n^{0.1})$$

which is insignificant compared to the time taken to run the algorithm from lemma 14 i.e. $\tilde{O}(n^2/s^2)$, which as $s = n^{\epsilon/\log d}$, is $n^{2-\frac{1}{O(\log d)}}$.

Remark 15. *The reader might notice that this is not quite the runtime that is promised in the theorem statement; indeed, what we have proved is a slightly weaker result. To get the runtime mentioned in the theorem, we must use a stronger bound on the number of monomials of $A_{3\log s}$: $\binom{d+1}{3\log s}$ instead of $O((d+1)^{3\log s})$, coupled with a slightly different expression for s. The rest of the proof however, is identical and doesn't involve any new ideas.*

Error Reduction. We have seen that the polynomial P that we construct only computes the correct answer with probability 2/3. To bring down the error to say, $1/n$, we simply repeat this algorithm $10\log n$ times (i.e. construct $P_1, \ldots, P_{10\log n}$) and then, simply output the majority value of the outputs of these $10\log n$ trials. Chernoff bound guarantees that the majority value will be correct with probability $1 - 1/n$. Further, this only blows up the runtime of our algorithm by a $\log n$ factor. $\square$

4 A list version of Orthogonal Vectors

The Orthogonal Detection problem simply asks *if* there is an orthogonal pair in $A \times B$. What if we were to to ask for the list of all such pairs? How hard is this problem? Certainly on the face of it, it seems like a harder problem than Orthogonal Detection. Perhaps that's why, the following result might seem surprising.

Theorem 16. *Suppose Orthogonal Vectors has an algorithm $\mathcal{A}$ that on input lists $A, B \subseteq \{0,1\}^d$ of size n each, runs in $\tilde{O}(n^{2-\epsilon})$ time. Then there is an algorithm to list all orthogonal pairs in $A \times B$ that runs in $O(n^{2-\frac{\epsilon\delta}{2}})$ time, assuming $A \times B$ has at most $\Delta = O(n^{2-\delta})$ many orthogonal pairs.*

Proof. Similar to the proof of theorem 11, the idea again is to use a divide and conquer strategy. We start by partitioning both lists into blocks such that each block is of size m (to be chosen later, in terms of n and Δ) so that there are n^2/m^2 pairs of blocks. On every pair $A' \subseteq A, B' \subseteq B$, run $\mathcal{A}$ and collect those pairs which *do* have an orthogonal pair. For each such pair (A', B'), do the following: split A' and B' into equal halves $A'_1, A'_2 \subset A$ and $B'_1, B'_2 \subset B$ and for each of the 4 possible pairs of halves (A'_i, B'_j) for $i, j \in \{1, 2\}$, determine if it contains an orthogonal pair by running $\mathcal{A}$. If such a pair (A'_i, B'_j) contains an orthogonal pair, recurse with the following provision: at each level i of the recursion (i goes from 0 to $\lceil \log m \rceil$), maintain a global counter c_i of the number of recursion calls that have been executed at that level. Once $c_i > \Delta$, do not recurse on any more pairs at recursion level i anymore. Once a pair consists of just singletons, we check if the vectors are orthogonal or not.

Running Time. Note that at level i, there are $\frac{m}{2^i}$ vectors in each list. Further, at every level, only at most Δ pairs containing an orthogonal pair are examined, owing to the global counter constraint. Thus, the complexity of the algorithm for level $i \geq 1$ is $O(\Delta \left(\frac{m}{2^i}\right)^{2-\epsilon})$ and at level 0 (i.e. the very first step) is $O(\frac{n^2}{m^2} \cdot m^{2-\epsilon}) = O(n^2/m^\epsilon)$ so that the total complexity is

$$O\left(\frac{n^2}{m^\epsilon} + \Delta m^{2-\epsilon} \sum \left(\frac{1}{2^{2-\epsilon}}\right)^i\right) = O\left(\frac{n^2}{m^\epsilon} + \Delta m^{2-\epsilon}\right).$$

Therefore, we decide to fix $m = n/\sqrt{\Delta}$ to equate the two terms. We obtain the claimed running time upon substituting $\Delta = O(n^{2-\delta})$. □

Remark 17. *This theorem actually establishes the sub-quadratic equivalence between Orthogonal Detection and listing sub-quadratically many orthogonal pairs, the formal (Turing Machine) definition of which we omit here.*

The proof of theorem 16 is inspired from that of theorem 3.3 of [WW18], which shows the sub-cubic equivalence between *finding a negative weight triangle* and *listing subcubically many negative weight triangles* in a graph.

5 Further Work

Given that the list version of Orthogonal Vectors is sub-quadratically equivalent to Orthogonal Vectors, one can ask the same question about $\Pi(f, \Sigma_1, \Sigma_2)$, particularly in the case when $G_{f,\Sigma_1 \times \Sigma_2}$ does not contain an induced subgraph of length 3. Also, note that we have only shown that $\Pi(f, \Sigma_1, \Sigma_2)$ is *at least* as hard as Orthogonal Vectors in such a case. Therefore, in principle, it might be possible to come up with a predicate and a pair of alphabets for which it is relatively easy to show that a sub-quadratic algorithm is impossible. These observations warrant making further efforts into such investigations.

6 Acknowledgements

The author would like to thank Prof. Srinivasan for continued discussions on all parts of this thesis and more, and in helping gain a better intuition of all involved concepts.

Bibliography

[AWW14] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 39–51, 2014.

[AWY15] Amir Abboud, Richard Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 218–230, 2015.

[CIP06] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for SAT. In *21st Annual IEEE Conference on Computational Complexity (CCC 2006), 16-20 July 2006, Prague, Czech Republic*, pages 252–260, 2006.

[Cop82] Don Coppersmith. Rapid multiplication of rectangular matrices. *SIAM J. Comput.*, 11(3):467–471, 1982.

[Raz87] Alexander A. Razborov. Lower bounds for the size of circuits of bounded depth over a complete basis with logical addition. *Mathematical notes of the Academy of Sciences of the USSR*, 41(4):333–338, 1987.

[Smo87] Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 77–82, 1987.

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

[Wil18] Ryan Williams. On the difference between closest, furthest, and orthogonal pairs: Nearly-linear vs barely-subquadratic complexity. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1207–1215, 2018.

[WW18] Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018.