More Applications of the Polynomial Method to Algorithm Design

Amir Abboud* Ryan Williams† Huacheng Yu‡

Abstract

In low-depth circuit complexity, the *polynomial method* is a way to prove lower bounds by translating weak circuits into low-degree polynomials, then analyzing properties of these polynomials. Recently, this method found an application to algorithm design: Williams (STOC 2014) used it to compute all-pairs shortest paths in $n^3/2^{\Omega(\sqrt{\log n})}$ time on dense n-node graphs. In this paper, we extend this methodology to solve a number of problems in combinatorial pattern matching and Boolean algebra, considerably faster than previously known methods.

First, we give an algorithm for BOOLEAN ORTHOGONAL DETECTION, which is to *detect* among two sets $A, B \subseteq \{0,1\}^d$ of size n if there is an $x \in A$ and $y \in B$ such that $\langle x, y\rangle = 0$. For vectors of dimension $d = c(n)\log n$, we solve BOOLEAN ORTHOGONAL DETECTION in $n^{2-1/O(\log c(n))}$ time by a Monte Carlo randomized algorithm. We apply this as a subroutine in several other new algorithms:

- In BATCH PARTIAL MATCH, we are given n query strings from from $\{0,1,\star\}^{c(n)\log n}$ ($\star$ is a "don't care"), n strings from $\{0,1\}^{c(n)\log n}$, and wish to determine for each query whether or not there is a string matching the query. We solve this problem in $n^{2-1/O(\log c(n))}$ time by a Monte Carlo randomized algorithm.
- Let $t \le v$ be integers. Given a DNF F on $c\log t$ variables with t terms, and v arbitrary assignments on the variables, F can be evaluated on all v assignments in $v \cdot t^{1-1/O(\log c)}$ time, with high probability.
- There is a randomized algorithm that solves the Longest Common Substring with don't cares problem on two strings of length n in $n^2/2^{\Omega(\sqrt{\log n})}$ time.
- Given two strings S, T of length n, there is a randomized algorithm that computes the length of the longest substring of S that has Edit-Distance less than k to a substring of T in $k^{1.5}n^2/2^{\Omega(\sqrt{\frac{\log n}{k}})}$ time.
- Symmetric Boolean Constraint Satisfaction Problems (CSPs) with n variables and m constraints are solvable in poly$(m) \cdot 2^{n(1-1/O(\log mn))}$ time.

*Computer Science Department, Stanford University. Supported by the Stanford SOE Hoover Fellowship of Virginia Vassilevska Williams, NSF Grant CCF-1417238 and BSF Grant BSF:2012338.

†Computer Science Department, Stanford University, `rrw@cs.stanford.edu`. Supported in part by a David Morgenthaler II Faculty Fellowship, and NSF CCF-1212372. 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.

‡Computer Science Department, Stanford University. Supported in part by NSF CCF-1212372.

1 Introduction

The PARTIAL MATCH problem states: given a database D of n points in $\{0,1\}^d$, process D to support queries of the form $q \in \{0,1,\star\}^d$ which either report a point $x \in D$ that matches all the non-$\star$ characters in q or reports that no x exists. (The $\star$'s are "don't cares" or wildcards.) This problem naturally captures several near-neighbor search scenarios; indeed, PARTIAL MATCH is believed by some to be more difficult than nearest neighbor in $\mathbb{R}^d$ [PTW08]. PARTIAL MATCH is easily seen to be equivalent to another important problem called SUBSET QUERY [CIP02]. In SUBSET QUERY, we wish to preprocess a database $\mathcal{D}$ of *sets* from $[d]$ such that, for all query subsets $T \subseteq [d]$, we can determine if there is an $S \in \mathcal{D}$ such that $T \subseteq S$.[1]

PARTIAL MATCH has been thoroughly studied for decades (e.g. Rivest's PhD thesis [Riv74]). However, there has been only minor algorithmic progress beyond the two obvious solutions of storing $2^{\Omega(d)}$ space for all possible queries, or taking $\Omega(n)$ time to try all points in the database. It is generally believed that PARTIAL MATCH is intractable for sufficiently large dimension d—this is one version of the "curse of dimensionality" hypothesis. In models such as the cell-probe model, strong lower bounds are known [MNSW98, BOR99, JKKR04, PT06, PTW08]. For example, Patrascu [Pat11] showed that any data structure for partial match that probes only t cells must use space $2^{\Omega(d/t)}$ when the word size is $O(n^{1-\varepsilon}/t)$. The best known data structures for answering partial match queries are due to Charikar, Indyk, and Panigrahy [CIP02] for the general case (discussed in more detail below), and Cole, Gottlieb, and Lewenstein [CGL04] for queries with a bounded number of $\star$'s. Large gaps remain between the data structures and the known lower bounds.

In this paper, we study the natural off-line variant of answering multiple partial match queries at once. In contrast to data structures, lower bounds in this setting will be much more difficult to attain (if they exist at all). In the BATCH PARTIAL MATCH problem, we have n queries $x_1, \ldots, x_n \in \{0,1,\star\}^d$ and a database $D \subseteq \{0,1\}^d$ of size n, and wish to answer all queries.[2] We also define BATCH SUBSET QUERY

[1] See the Preliminaries for an overview of the reduction.

[2] Although we set the number of queries and the database size to be the same, that is only for simplicity: our algorithms easily extend to cases with different sizes of query sets and databases.

in the natural way. The obvious algorithm for these problems runs in $O(n^2 d)$ time, and by preprocessing all possible queries to the database, one can also solve them in $O(n2^d)$ time. However, just as in the usual partial match setting, Batch Partial Match is truly theoretically interesting for $d \geq \Omega(\log n)$; in that case, we wish to solve the problem in time that is *sub-quadratic* in n.

Given a data structure for Partial Match with $O(p)$ processing time and $O(q)$ query time, we can clearly solve Batch Partial Match in $O(p + nq)$ time. Applying the data structures of Charikar, Indyk, and Panigrahy, one obtains the following algorithms for Batch Partial Match:

- $\tilde{O}(n \cdot 2^{O(\sqrt{c} \cdot d \log^2 d / \log^{1/2} n)} + n^2/2^c)$ time, for any parameter c. Setting $c = \log n / \log^{2+\varepsilon} d$, the running time is $\tilde{O}(n \cdot 2^{O(d/\log^{\varepsilon} d)} + n^{2-1/\log^{2+\varepsilon} d})$. Hence for $d = O(\log n)$, one can obtain $n^{2-1/(\log\log n)^{2+\varepsilon}}$ running time.
- $\tilde{O}(n \cdot d^c + n^2 d/c)$ time, for any $c \leq n$. For $d = \Omega(\log n)$, this does not yield a running time improvement better than $O(n^2/\text{poly}(\log n))$.

The following improved algorithms for Batch Partial Match were also known.

- For sufficiently large d, one can apply fast matrix multiplication to solve the problem faster than $O(n^2 d)$. Indeed, it is easy to observe that Batch Subset Query with n sets over the universe $[d]$ can be solved with an $n \times d \times n$ matrix product. However, this approach cannot yield an $o(n^2)$ time algorithm.
- The Vector Domination problem is defined as: given two sets A, B, each of n vectors in $c \log n$ dimensions, determine if there is $x \in A$ and $y \in B$ such that $x_i \leq y_i$ for all i. Impagliazzo, Lovett, Paturi, and Schneider [ILPS14] (building on prior work of Impagliazzo, Paturi, and Schneider [IPS13] and Chan [Cha05]) give an $n^{2-1/\text{poly}(c)}$ time algorithm for the Vector Domination problem: given two sets A, B, each of n vectors in $c \log n$ dimensions, determine if there is $x \in A$ and $y \in B$ such that $x_i \leq y_i$ for all i. When the vectors are over $\{0, 1\}$, it is easy to see that this problem is equivalent to Batch Subset Query, hence their algorithm can be used to solve Batch Partial Match in $n^{2-1/\text{poly}(c)}$ time when the dimensionality is $c \log n$. This is a substantial improvement over applying Charikar, Indyk, and Panigrahy's data structure.

While some lower bounds for Partial Match (the data structure problem) are known, no lower bounds are known for Batch Partial Match at all, being an "offline" problem. However, there is some evidence that Batch Partial Match may not be solvable in sub-quadratic time. For example, it is known that if Batch Boolean Orthogonal Detection (hence also Batch Partial Match) is solvable in $n^{2-\varepsilon} \cdot 2^{o(d)}$ time for some $\varepsilon > 0$, then CNF-SAT on formulas with n variables and m clauses is solvable in $2^{n(1-\varepsilon/2)} \cdot 2^{o(m)}$ time [Wil05]. The Strong Exponential Time Hypothesis ([IP01, CIP09]) effectively asserts that such SAT algorithms do not exist. (In fact, several conditional lower bounds based on assuming SETH can actually be based on assuming that Batch Partial Match needs $n^{2-o(1)}$ time instead, such as the graph diameter inapproximability results of Roddity and Vassilevska [RV13].)

1.1 Our Results

In this paper, we present a faster algorithm for Batch Partial Match for all dimensions $d \geq \Omega(\log n)$, and exploit the versatility of partial matches to derive faster algorithms for other basic problems. The key idea is to leverage a strategy outlined in recent prior work ([Wil14]) on all-pairs shortest paths. After a series of reductions, we show how to rephrase the problem as a type of *Boolean circuit evaluation* problem, then combine tools from circuit complexity and from the algorithms literature (such as fast matrix multiply) to solve the evaluation problem efficiently.

Our initial algorithm solves the Boolean Orthogonal Detection problem, which is to detect among two sets $A, B \subseteq \{0, 1\}^d$ of size n if there is an $x \in A$ and $y \in B$ such that $\langle x, y \rangle = 0$. In what follows, let $c : \mathbb{N} \to \mathbb{N}$ satisfy $c(n) \leq n^{\varepsilon}$ for all $\varepsilon > 0$.

Theorem 1.1. *For vectors of dimension $d = c(n) \log n$, Boolean Orthogonal Detection can be solved in $n^{2-1/O(\log c(n))}$ time by a randomized algorithm that is correct with high probability.*

This algorithm is a significant improvement over the $n^{2-1/\text{poly}(c)}$ running time of [ILPS14] mentioned above (for the Boolean case), and can be used to recover the fastest known running time for CNF-SAT (via a known reduction from CNF-SAT to Boolean Orthogonal Detection [Wil05]).

As alluded to in the above, the theorem proceeds by reducing this detection problem to a circuit evaluation problem, then provide a way to efficiently evaluate this circuit. After that, we give a sub-quadratic time reduction from the batch version of Boolean Orthogonal Detection to Boolean Orthogonal Detection itself, which roughly preserves the running time. Invoking reductions between orthogonal detection and partial match, this yields a new sub-quadratic time algorithm for answering partial match queries in batch:

Corollary 1.1. *The Batch Partial Match problem with n queries and n strings from $\{0, 1\}^{c(n) \log n}$ can be solved in $n^{2-1/O(\log c(n))}$ time by a randomized algorithm that is correct with high probability.*

An analogous statement holds for the batch version of subset query.

Evaluating DNFs. Next, we prove an *equivalence* between (a) evaluating DNF formulas on many assignments and (b) answering many partial match queries on a database, resulting in a new algorithm for evaluating a DNF on many assignments:

COROLLARY 1.2. *Given a DNF F on $c \log t$ variables with t terms, and v arbitrary assignments on the variables with $t \leq v$, F can be evaluated on all v assignments in $v \cdot t^{1-1/O(\log c)}$ time, with high probability.*

That is, for exponential size DNF, we can obtain a genuine polynomial improvement over the obvious $\tilde{O}(v \cdot t)$ time bound for evaluating a t-term DNF on v assignments. It is instructive to think of Corollary 1.2 as a multivariate Boolean version of fast univariate polynomial evaluation. Ideally, one would like to see that a DNF with t terms can be evaluated on t arbitrary assignments in $\tilde{O}(t)$ time (but note that this would be impossible, if orthogonal detection is hard!).

Longest Common Substring with Don't Cares. Fast algorithms for computing certain similarity measures between sequences (strings) are a classical area of study in computer science. Famous examples include the Longest Common Subsequence and Edit-Distance problems. We apply our orthogonal detection algorithm to solve natural extensions of the *longest common substring* problem.

Given two strings $S, T \in \Sigma^n$ of length n, the LONGEST COMMON SUBSTRING problem asks for the length of the longest string that appears in both S and T as a contiguous substring. The problem can be solved in optimal $\Theta(n)$ time using Suffix-Trees [Gus97] (See [KSV14] for a recent space-efficient algorithm). We consider the variant in which the strings S and T are over $(\Sigma \cup \{\star\})^n$, where $\star$'s correspond to "don't care" characters. In the LONGEST COMMON SUBSTRING WITH DON'T CARES problem (abbreviated as LCS*), we ask for the longest string over Σ that is a partial match with both a contiguous substring of S and a contiguous substring of T. LCS* is natural for modeling real-life text processing and bioinformatics, where the $\star$ characters may correspond to noisy or erroneous data.

The classic Smith-Waterman $O(n^2)$ time dynamic programming algorithm for Local Alignment [SW81] and the $O(n^2/\log^2 n)$ time improvements for Edit Distance [CLZU03, MP80, BFC08] can be adapted to solve LCS*, but no faster algorithms were known. There are many clever algorithms for related problems. For instance, the classical pattern matching with don't cares problem is the special case of LCS* where we ask whether the whole string (or pattern) S appears in T as a substring. Kalai's algorithm [Kal02] improves previous algorithms of Indyk [Ind98], Muthukrishnan and Palem [MP94], and Fisher and Paterson [FP73], and solves the problem in $O(|T| \log |S|)$ time.

Recently, as an attempt to explain the lack of faster algorithms for LCS*, Abboud, Vassilevska Williams, and Weimann [AVW14] proved that SETH implies an $n^{2-o(1)}$ lower bound for LCS* over binary alphabets. Thus, under this plausible hypothesis we are left with a sub polynomial gap in our understanding of the complexity of LCS* and the question becomes whether we can decrease this gap, e.g. by "shaving more log factors". Our reductions imply a new algorithm for LCS* running faster than $O(n^2/\log^c n)$ for any constant $c > 0$, thus obtaining a "clean shave" of polylog factors for this problem.

THEOREM 1.2. *There is a randomized algorithm that solves the Longest Common Substring with don't cares problem on two strings of length n in $n^2/2^{\Omega(\sqrt{\log n})}$ time.*

Longest Substrings with Small Edit Distance. An important related task is: given two long sequences, find two substrings are as long as possible and still not far from each other in Edit-Distance. That is, the two substrings are not required to be equal as in LCS*, but we want them to be highly similar. This situation is common in a context where small changes to the data are inherent, e.g. in biology.

DEFINITION 1.1. (THE $\texttt{LMS}_\texttt{k}$ PROBLEM) *Given S and $T \in (\Sigma \cup \{\star\})^n$, return the length of the longest substring s of S such that there is a substring t of T with Edit-Distance less than k to s.*

This problem can capture many of the applications in sequence alignment that the Local Alignment problem solves. In Local Alignment, one specifies a scoring function that defines the similarity between two substrings. Typically, this scoring function tries to reward long substrings while penalizing large weighted edit distance. The implicit scoring function in $\texttt{LMS}_\texttt{k}$ is of the same flavor: longer substrings score higher, as long as the edit distance is less than k.

The significance of finding good algorithms for these tasks is witnessed by the 50,000+ citations to the paper introducing BLAST [AGM+90], a heuristic algorithm for Local Alignment, making it one of the most cited scientific papers of the 1990s. Since the sizes of genomes can reach billions of "letters", even log-factor improvements over the quadratic upper bound of Smith-Waterman can have a noticeable impact on our ability to analyze biological data.

One variant of $\texttt{LMS}_\texttt{k}$ has received a lot of attention: given S and T, determine whether there is a substring of T that has Edit-Distance less than k to the *entire* string S. Landau and Vishkin [LV86] solve this problem without don't care symbols in $O((k + \log |S|)|T|)$ time. Akutsu [Aku95] showed how to use their framework to obtain an $O(\sqrt{k|S|} \cdot |T|)$ algorithm which allows for don't cares. If we replace Edit-Distance with Hamming-Distance, we obtain the k-Mismatches problem which is solved by Clifford *et al.* [CEPR10] in $O(|T|(k + \log|S|\log k)\log|T|)$ time, and by Amir, Lewenstein, and Porat [ALP04] in the case without don't cares in $O(|T|\sqrt{k \log k})$ time. However, these algorithms do not extend to find the optimal substrings of S and T in subquadratic time, which is perhaps a more relevant task in computational biology [Gus97].

By reduction to Boolean orthogonal vectors, we present an algorithm for $\texttt{LMS}_\texttt{k}$ that is faster than $O(n^2/\log^c n)$ time for any constant $c > 0$, when the size of the alphabet and the error threshold k are constant.

THEOREM 1.3. *There is a randomized algorithm that solves the Longest k-Matching Substring problem on two strings of length n in $k^{1.5}n^2/2^{\Omega(\sqrt{\frac{\log n}{k}})}$ time.*

Our algorithm can also be adapted to solve the simpler "longest common substring with k-mismatches with don't cares" problem, in a similar runtime. Recently, practical [FGKU14] and subquadratic [Gra14] algorithms were proposed for the version of the problem without don't cares.

Symmetric Boolean CSPs. Finally, we solve a vastly generalized version of the CNF satisfiability problem, in a running time that is competitive with the fastest known CNF SAT algorithms. In particular, we show how BOOLEAN ORTHOGONAL DETECTION can be used to solve constraint satisfaction problems where each constraint is an *arbitrary* symmetric function on a subset of variables.

A *symmetric Boolean CSP* [CD04] is a conjunction of Boolean constraints over Boolean variables, such that each constraint models some symmetric function—that is, the truth of each constraint only depends on the number of true literals in the constraint. In our setting, each constraint in an instance may be modeled by a different symmetric function: there is no restriction on which symmetric functions are used. (Some constraint could be an XOR, another could be an AND, another could be a MAJORITY, etc.) In the language of Boolean circuit complexity, such a CSP is an *AND of SYM gates with negations at the bottom*; the CNF SAT problem is the special case where the SYM gates are OR gates.

THEOREM 1.4. *Symmetric Boolean CSPs with n variables and m constraints are solvable in* $\text{poly}(m) \cdot 2^{n(1-1/O(\log mn))}$ *time.*

The fastest known CNF-SAT algorithms run in $O(2^{n(1-1/O(\log m/n))})$ time [CIP06, DH09] on instances with n variables and m clauses; this compares favorably with Theorem 1.4 when $m = \text{poly}(n)$. It is also useful to compare Theorem 1.4 with the 0-1 integer programming algorithm of Impagliazzo-Lovett-Paturi-Scheider [ILPS14] running in $2^{n(1-1/\text{poly}(\log m/n))}$ time: theirs can be viewed as an algorithm for satisfiability of ANDs of linear threshold functions, whereas our algorithm works for arbitrary symmetric functions on each constraint and runs faster in the case of $m \geq n^{1+\varepsilon}$.

2 Preliminaries

In this section, we review a few prior known results that are applied in this work.

Equivalence of partial match, subset query, and orthogonal vectors. Here we briefly sketch the equivalences between the three aforementioned problems. First, using the correspondence

$$S \subseteq T \iff S \cap \overline{T} = \varnothing,$$

it is easy to see how to reduce between subset query and orthogonal vectors: turn sets over $[d]$ into d-bit vectors (or vice-versa) and flip the bits of the database set. We can easily simulate a subset query with a partial match query, by putting $\star$'s in components corresponding to elements not in the query set, and 1s in components corresponding to elements in the query set. Finally, partial match queries $q \in \{0, 1, \star\}^d$ can be simulated by orthogonal vector queries v_q of length $2d$. Namely, for all $i = 0, \ldots, d-1$,

- if $q_{i+1} = 0$, set $v_q[2i+1] := 1$ and $v_q[2i+2] := 0$
- if $q_{i+1} = 1$, set $v_q[2i+1] := 0$ and $v_q[2i+2] := 1$,
- if $q_{i+1} = \star$, set $v_q[2i+1] = 0$ and $v_q[2i+2] := 0$,

modifying the database accordingly.

Efficient matrix multiplication and polynomial evaluation. One component from prior work requires a fast rectangular matrix multiplication algorithm. The precise statement we need is:

LEMMA 2.1. (COPPERSMITH [COP82]) *For all sufficiently large N, multiplication of an $N \times N^{.172}$ matrix with an $N^{.172} \times N$ matrix can be done in $O(N^2 \log^2 N)$ arithmetic operations.*

A full proof of this statement can be found in the appendix of Williams' paper on all-pairs shortest paths [Wil14]. We will also need the following basic lemma on efficiently evaluating polynomials on a combinatorial rectangle. We include a sketch of it here, for completeness.

LEMMA 2.2. ([WIL14]) *Given $P(x_1, \ldots, x_d, y_1, \ldots, y_d)$, a polynomial over $\mathbb{F}_2$ with at most $n^{0.1}$ monomials, and two sets of n inputs $A = \{a_1, \ldots, a_n\} \subseteq \{0,1\}^d$, $B = \{b_1, \ldots, b_n\} \subseteq \{0,1\}^d$, we can evaluate P on all pairs $(a_i, b_j) \in A \times B$ in $\tilde{O}(n^2)$ time.*

Proof. (Sketch) The idea is to reduce the problem of evaluating P to fast rectangular matrix multiplication. Let $m \leq n^{0.1}$, and put an arbitrary ordering on the monomials of P. In particular, we create an $n \times m$ matrix M with rows indexed by strings in A, and columns indexed by monomials of P, and an $m \times n$ matrix N with rows indexed by monomials of P and columns indexed by strings in B. In particular, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$, define $M[i, j]$ to be the value of the jth monomial of P restricted to the x-variables evaluated on a_i. (That is, we set all y-variables in the jth monomial to 1, and plug in the assignment defined by a_i for the variables $x_1, \ldots, x_d$.) Similarly for $j = 1, \ldots, m$ and $k = 1, \ldots, n$, define $N[j, k]$ to be the value of the jth monomial of P restricted to the y-variables evaluated on b_k. Then, $M[i, j] \cdot N[j, k]$ equals the value of the jth monomial of P on the assignment (a_i, b_k), and

$$(M \cdot N)[i, k] = \sum_{j=1}^{m} M[i, j] \cdot N[j, k] = P(a_i, b_k);$$

that is, the i, k entry of the matrix product equals the value of P on the assignment (a_i, b_k). Therefore, we have reduced the evaluation problem to multiplication of an $n \times n^{0.1}$ and $n^{0.1} \times n$ matrix over $\mathbb{F}_2$, which can be done in $\tilde{O}(n^2)$ time by Lemma 2.1. □

A tool from low-depth circuit complexity. We will also need a well-known construction from circuit complexity. Suppose we wish to compute the AND over d variables $z_1, \ldots, z_d$ with a low-degree polynomial over $\mathbb{F}_2$. In general this is impossible, but we can choose a *distribution* of polynomials such that for any particular input, a random polynomial does the job. Let $t \geq 1$ be an integer, choose independently and uniformly at random $t \cdot d$ bits $r_{1,1}, \ldots, r_{1,d}, r_{2,1}, \ldots, r_{2,d}, \quad \ldots \quad, r_{t,1}, \ldots, r_{t,d} \in \{0, 1\}$, and consider the expression

$$A_t(y_1, ..., y_d) = \prod_{i=1}^{t} \left(1 \oplus \bigoplus_{j=1}^{d} r_{i,j} \cdot (y_j \oplus 1) \right),$$

where $\oplus$ is addition modulo 2. For fixed $r_{i,j} \in \{0, 1\}$, A_t is a product of t sums of at most $d + 1$ variables y_j, along with possibly the constant 1, over the field $\mathbb{F}_2$. Razborov [Raz87] and Smolensky [Smo87] showed that for large t, the low-degree arithmetic expression A_t simulates the AND of the original d variables, with high probability:

LEMMA 2.3. *For every fixed* $(y_1, ..., y_d) \in \{0, 1\}^d$,

$$\Pr_{r_{i,j}} [A_t(y_1, ..., y_d) = AND(y_1, \ldots, y_d)] \geq 1 - 1/2^t.$$

Alignments and Edit Distance. An *alignment* $\mathcal{A}$ of a string of length n_1 to a string of length n_2 is a pair of sets $\mathcal{A}_{Match} \subseteq [n_1] \times [n_2]$ and $\mathcal{A}_{Mis} \subseteq ([n_1] \cup \{-\}) \times ([n_2] \cup \{-\})$ such that:

- every index $i \in [n_1]$ appears exactly once in a pair (i, x) in $\mathcal{A}_{Match} \cup \mathcal{A}_{Mis}$ and every index $i \in [n_2]$ appears exactly once in a pair (y, i) in $\mathcal{A}_{Match} \cup \mathcal{A}_{Mis}$, for some $x, y \in [n] \cup \{-\}$,
- the pair $(-, -)$ does not belong to $\mathcal{A}_{Mis}$, and
- for every pair of pairs $(i_1, j_1), (i_2, j_2) \in \mathcal{A}_{Match} \cup \mathcal{A}_{Mis}$, if $i_1 < i_2$ then $j_1 < j_2$.

In words, $\mathcal{A}_{Match}$ contains the pairs of indices that are "matched" in the alignment $\mathcal{A}$ while $\mathcal{A}_{Mis}$ contains the mismatched pairs and the indels (insertions and deletions). We denote by A_{n_1,n_2} the set of all alignments of strings of lengths n_1, n_2. The cost of an alignment $\mathcal{A} \in A_{n_1,n_2}$ is the size of $\mathcal{A}_{Mis}$. An alignment $\mathcal{A} \in A_{n_1,n_2}$ is *valid* for a pair of strings $X \in (\Sigma \cup \{\star\})^{n_1}, Y \in (\Sigma \cup \{\star\})^{n_2}$ iff for every pair $(i, j) \in \mathcal{A}_{Match}$, $X[i] \equiv Y[j]$ (i.e. X[i]=Y[j] or one of them is the $\star$ symbol).

We say that the edit distance between two strings $X \in (\Sigma \cup \{\star\})^{n_1}, Y \in (\Sigma \cup \{\star\})^{n_2}$ is k, and denote $ED(X, Y) = k$, iff the alignment $\mathcal{A} \in A_{n_1,n_2}$ of minimal cost that is valid for X, Y has cost $|\mathcal{A}_{Mis}| = k$.

3 Detecting an orthogonal pair of Boolean vectors

We begin by presenting an algorithm for BOOLEAN ORTHOGONAL DETECTION: given two sets of vectors $A, B \subseteq \{0, 1\}^d$, each of cardinality n, determine if there is an $x \in A$ and $y \in B$ such that $\langle x, y \rangle = 0$.

REMINDER OF THEOREM 1.1 *For vectors of dimension* $d = c(n) \log n$, BOOLEAN ORTHOGONAL DETECTION *can be solved in* $n^{2-1/O(\log c(n))}$ *time by a randomized algorithm that is correct with high probability.*

In the following sections, we give a series of reductions showing how the algorithm of Theorem 1.1 can be used to derive the other algorithms mentioned in the introduction of the paper.

Proof of Theorem 1.1. Before giving details, let's sketch the idea of the algorithm. Suppose we are given sets $A, B \subseteq \{0, 1\}^d$. We divide the n vectors of A and n vectors of B into $\lceil n/s \rceil$ groups of size at most s. Next, we design a "small" low-depth Boolean circuit C which takes a group A' of A and a group B' of B, and outputs 1 if and only if there is an orthogonal pair of vectors in (A', B'). Then we use Lemma 2.3 to construct an efficiently samplable distribution of polynomials $\mathcal{D}$ such that for each input x, $C(x) = P(x)$ for a randomly chosen $P \in \mathcal{D}$, with probability at least $2/3$. Since the circuit C is "small", the polynomial P will have a "somewhat small" number of monomials. By Lemma 2.2, we can evaluate P on all pairs of groups efficiently. Finally we sample $O(\log n)$ polynomials, and take a majority over all pairs of groups to increase the probability of correctness.

We now turn to the circuit construction. Let s be a parameter such that $s \leq 2^{(d+1)/6}$; later we shall choose s

optimally. Given two groups $\{x_i\}, \{y_j\}$ of s vectors from $\{0,1\}^d$, we can design a low-depth circuit C detecting if there is an orthogonal pair in a straightforward way: For all pairs of vectors (x_i, y_j), compute the expression

$$E(x_i, x_j) = \wedge_{k=1}^{d}(\neg x_i[k] \vee \neg y_j[k]).$$

Note $E(x_i, y_j) = 1$ if and only if $\langle x_i, y_j \rangle = 0$. Then we define the circuit C to be the OR over all s^2 pairs (x_i, y_j) of the expression $E(x_i, y_j)$. The circuit C is therefore an OR of s^2 ANDs of d ORs of two negations of input bits, of size $O(s^2 d)$.

We can 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 construction of Razborov [Raz87] and Smolensky [Smo87] from Lemma 2.3, each AND gate in C of fan-in d can be replaced by an arithmetic expression $A_{3\log s}$ which is a product (i.e., an AND gate) of $3 \log s$ XORs of $d+1$ fan-in. Given an input y, each $A_{3\log s}$ will be incorrect on this input with probability at most $1/s^3$. Finally, for the top OR gate of C, we replace it with a NOT-AND-NOT using DeMorgan's law, and apply Razborov-Smolensky to the AND gate of fan-in s^2, replacing it with an expression A_2, which is an AND of two XORs of fan-in s^2+1. This replacement will be incorrect on an input with probability at most $1/4$. Taking a union bound over all gates, for each input, with probability at least $1 - 1/4 - s/s^3 > 2/3$, these randomized replacements do not change the output. Let C' be this new circuit over XOR and AND, which is an arithmetic circuit over $\mathbb{F}_2$.

The circuit C' is an AND of two XORs of s^2+1 ANDs of $3 \log s$ XORs of $d+1$ ANDs of at most two variables. Applying the distributive law to the ANDs of $3 \log s$ XORs of $d+1$ terms, we can expand each AND of XORs into an XOR of ANDs. Naively, the size of the resulting expression after this expansion would be $\Omega((d+1)^{3\log s})$. However, using the facts:

(a) $x^2 = x$ over $\mathbb{F}_2$,

(b) $s \leq 2^{(d+1)/6}$ (hence, $3 \log s < (d+1)/2$), and

(c) each XOR is over a subset of the same d products of two variables,

we obtain an XOR of at most $O(\binom{d+1}{3\log s})$ ANDs of variables. Once this expansion has occurred for every AND gate in the middle layer, we have a circuit C'' which is an AND of two XORs of $O(s^2\binom{d+1}{3\log s})$ ANDs. Expanding the top AND gate, we obtain an XOR of $O(s^4\binom{d+1}{3\log s}^2)$ ANDs of variables; that is, we have a polynomial P over $\mathbb{F}_2$ with $O(s^4\binom{d+1}{3\log s}^2)$ monomials. Let

$$m = O\left(s^4\binom{d+1}{3\log s}^2\right)$$

be the resulting upper bound on monomials.

Let us check that expanding this circuit into a sum of monomials can be done efficiently. Note that m is an upper bound on the total number of monomials we could ever generate, while expanding the polynomial via distributivity. Each addition or multiplication of two polynomials takes $\tilde{O}(m)$ time, and we need to do $\tilde{O}(s^2 d)$ such polynomial operations. In the below, we will set $s < n^{1/400}$ and $m \leq n^{1/10}$. Hence it takes $\tilde{O}(s^2 dm) \leq O(n)$ time to generate the expanded polynomial from the circuit C'.

The above randomized procedure generates a polynomial P such that for every two groups of s d-bit vectors, evaluating P on these two groups determines whether there is an orthogonal pair in the two groups, with probability at least $2/3$. By Lemma 2.2, we can evaluate P on all $O(n^2/s^2)$ pairs of groups in A and B in time $\tilde{O}((n/s)^2)$, provided that $m \leq n^{0.1}$.

Now we need to set the parameter s so that $m \leq n^{0.1}$ is satisfied and s is maximized. Let $s := 2^{\varepsilon \log n/\log(d/\log n)}$ for sufficiently small $\varepsilon > 0$. Note that $\varepsilon \log n/\log(d/\log n) \leq (d+1)/6$ when $d \geq \Omega(\log n)$ and ε is small enough; therefore, the constraint $s \leq 2^{(d+1)/6}$ holds. By standard estimates,

$$\begin{aligned} m &\leq O\left(s^4\binom{d+1}{3\log s}^2\right) \\ &\leq O\left(s^4\left(\frac{e(d+1)}{3\log s}\right)^{6\log s}\right) \\ &\leq O\left(s^{10}\left(\frac{(d+1)}{3\log s}\right)^{6\log s}\right) \\ &\leq O\left(s^{10}\left(\frac{(d+1)}{3\varepsilon\log n/\log(d/\log n)}\right)^{6\varepsilon\log n/\log c}\right) \\ &\leq O\left(s^{10}\left(\frac{(c\log n+1)\log c}{3\varepsilon\log n}\right)^{6\varepsilon\log n/\log c}\right). \end{aligned}$$

Taking logarithms, we have

$$\begin{aligned} \log m &\leq 10 \cdot \log s + 6\varepsilon \cdot \frac{\log n}{\log c} \log\left(\frac{c\log n + 1}{3\varepsilon\log n} \cdot \log c\right) \\ &\leq 10\varepsilon \cdot \frac{\log n}{\log c} + 6\varepsilon \cdot \frac{\log n}{\log c} \log\left(\left(\frac{c\log n+1}{3\varepsilon\log n}\right)^2\right) \\ &\leq 10\varepsilon \cdot \frac{\log n}{\log(d/\log n)} + 18\varepsilon \cdot \log n \end{aligned}$$

for sufficiently large n. Setting $\varepsilon < 1/400$, we have $\log m \leq 0.1 \log n$. Then, the above procedure will run in time

$$\begin{aligned} \tilde{O}(n^2/s^2 + m^{2.5}) &\leq \tilde{O}(n^2/2^{\varepsilon\log n/\log(d/\log n)} + n^{0.25}) \\ &\leq n^{2-1/O(\log(d/\log n))}. \end{aligned}$$

The above only generates a polynomial P which is correct on a given point with probability $2/3$. To amplify the probability of success, generate $k = 10 \log n$ independent polynomials $P_1, \dots, P_k$, evaluate all P_i on all pairs of groups from A and B, and for each pair, compute the majority value reported by $P_1, \dots, P_k$. By the Chernoff bound, with probability at least $1 - n^{-3}$, the majority value on a pair of groups is the correct value of the circuit C on that pair. Hence by the union bound, with probability at least $1 - n^{-1}$, we will determine the correct output of C on all pairs of groups from A and B. This only adds an $O(\log n)$ additional factor to the running time of the algorithm.

For a given instance A, B, the above algorithm determines subsets $A' \subset A$ and $B' \subset B$ such that $|A'| = |B'| \leq s$ and A', B' contain an orthogonal pair (if such a pair exists). Because $s \leq n^{0.1}$, we can find an explicit pair with only $O(n^{0.2} \cdot d)$ extra running time, by exhaustive search over all pairs of vectors in $A' \times B'$. □

4 Applications

In this section, we give some consequences of the algorithm from Theorem 1.1 in the previous section. First, we can quickly re-derive the fastest known CNF-SAT algorithm (up to constants in the savings) [CIP06, DH09] using the orthogonal detection algorithm.

THEOREM 4.1. *CNF-SAT on n variables and $m \leq 2^{o(n)}$ clauses is solvable in $2^{n(1-1/O(\log(m/n)))}$ time.*

Proof. (Sketch) Williams [Wil05] showed how to reduce CNF-SAT with m clauses and n variables to BOOLEAN ORTHOGONAL DETECTION with $N = 2^{n/2}$ size sets of vectors in m dimensions. (The proof is very simple; the reader is encouraged to find it.) Theorem 1.1 gives an $N^{2-1/O(\log(m/n))} = 2^{n(1-1/O(\log(m/n)))}$ time algorithm. □

The above reduction shows that it may be difficult to improve the running time dependence on the dimension of our algorithm in Theorem 1.1. For example, an $n^{2-1/O(\log\log c)}$ time algorithm would yield a faster CNF-SAT algorithm, strong enough to prove new circuit lower bounds (see [JMV13]).

4.1 Processing Queries in Batch Next, we show how to use solutions to the decision/finding problem solved in Theorem 1.1 to solve problems with multiple outputs.

In BATCH BOOLEAN ORTHOGONAL DETECTION, we are given $A, B \subseteq \{0,1\}^d$ with $|A| = |B| = n$, and must decide for every $x \in A$ if there is a $y \in B$ such that $\langle x, y \rangle = 0$ (over the integers). That is, we decide for *every* vector in one set if it has an orthogonal vector in the other set. BATCH BOOLEAN ORTHOGONAL DETECTION looks strictly harder than BOOLEAN ORTHOGONAL DETECTION, simply because much more information needs to be computed. However, the two are actually *equivalent* in a precise sense: a subquadratic-time algorithm for BOOLEAN ORTHOGONAL DETECTION implies a subquadratic-time algorithm for the batch version.

LEMMA 4.1. *If* BOOLEAN ORTHOGONAL DETECTION *is in time $T(n, d)$, then* BATCH BOOLEAN ORTHOGONAL DETECTION *is in time $O(n \cdot T(\sqrt{n}, d))$.*

Proof. We first give an algorithm for BATCH BOOLEAN ORTHOGONAL DETECTION, assuming an oracle for BOOLEAN ORTHOGONAL DETECTION. Initialize an n-bit table T to be all-zeros. Divide the n vectors of A and B into $\lceil n/s \rceil$ groups of size at most s. For each pair (A', B') of groups, call an oracle for BOOLEAN ORTHOGONAL DETECTION on the two sets of size $O(s)$, respectively. Repeat the following until no orthogonal pair $(x_i, y_j) \in A' \times B'$ is found: set $T[i] := 1$ and remove x_i from A (and hence A' as well). (We can also *find* a vector y_j such that $\langle x_i, y_j \rangle = 0$ as well and store it, if desired.)

This concludes the algorithm; now we analyze it. For each oracle call that returns an orthogonal vector pair, we remove one vector from A; hence there can be at most n such calls. For each pair of groups, there is exactly one oracle call that reports there is no orthogonal pair in the pair. Therefore the total running time is $O((n+n^2/s^2) \cdot T(O(s), d))$. Setting $s = \Theta(\sqrt{n})$ to balance the terms, the running time becomes $O(n \cdot T(\sqrt{n}, d))$. □

It follows that, in order to solve the batch orthogonal vectors problem (and hence, BATCH PARTIAL MATCH and BATCH SUBSET QUERY) in subquadratic time, one only has to solve BOOLEAN ORTHOGONAL DETECTION in subquadratic time:

COROLLARY 4.1. *For every $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$,* BOOLEAN ORTHOGONAL DETECTION *is in $O(n^2/g(n, d) \cdot f(d))$ time if and only if* BATCH BOOLEAN ORTHOGONAL DETECTION *is in $O(n^2/g(\sqrt{n}, d) \cdot f(d))$ time.*

Proof. One direction is obvious; the other directly follows from Lemma 4.1. □

By the known equivalences between BOOLEAN ORTHOGONAL DETECTION, PARTIAL MATCH, and SUBSET QUERY, it follows from Theorem 1.1 and Corollary 4.1 that:

REMINDER OF COROLLARY 1.1 *The* BATCH PARTIAL MATCH *problem with n queries and n strings from $\{0,1\}^{c(n) \log n}$ can be solved in $n^{2-1/O(\log c(n))}$ time by a randomized algorithm that is correct with high probability.*

COROLLARY 4.2. BATCH SUBSET QUERY *with n query sets and n database sets from $[c(n) \log n]$ can be solved in $n^{2-1/O(\log c(n))}$ time by a randomized algorithm that is correct with high probability.*

4.2 Evaluating DNFs The algorithm for Batch Partial Match can be used to evaluate DNF (and consequently, CNF) formulae with t terms on v variable assignments faster than the naive method that takes $\Omega(t \cdot v)$ time. We provide a simple black-box reduction, and state the resulting running time as a corollary.

Theorem 4.2. *Given an algorithm for the* Batch Boolean Orthogonal Detection *problem with m d-bit vectors in one set and n d-bit vectors in the other set running in $T(m, n, d)$ time, there is an algorithm for evaluating DNF formulae with d variables and m terms on n variable assignments in $T(m, n, 2d)$ time.*

Proof. Let F be a DNF formula over the variables $x_1, \ldots, x_d$, and let $S \subseteq \{0,1\}^d$ be a set of assignments on which to evaluate F. First, we reduce the problem to an "inverse" version of Batch Partial Match, where we wish to determine for all database strings if they are a yes-answer to some query. Set the string database $\mathcal{D}$ to be S. For every conjunct c of F, define $q_c \in \{0, 1, \star\}^d$ as follows:

- $q_c[i] := 0$ if the literal $\neg x_i$ appears in c,
- $q_c[i] := 1$ if the literal x_i appears in c, and
- $q_c[i] := \star$ otherwise.

The d-bit strings matching q_c correspond directly to the set of satisfying assignments to the conjunct c. Therefore, the problem of determining which elements of $\mathcal{D}$ match some query q_c is exactly the problem of evaluating F on the set of assignments S.

This inverse batch partial match problem can be solved by reducing partial match queries on strings of length d to orthogonal vector queries in $2d$ dimensions (as described in the Preliminaries). Observe that, in contrast to the partial match problem, the orthogonal vector problem is "symmetric" in that the database list and query list are both of the same type. Hence for all database vectors we can compute if there is a query vector which is orthogonal, using an algorithm for Batch Boolean Orthogonal Detection. □

Reminder of Corollary 1.2 *Given a DNF F on $c \log t$ variables with t terms, and v arbitrary assignments on the variables with $t \le v$, F can be evaluated on the v assignments in $v \cdot t^{1-1/O(\log c)}$ time, with high probability.*

Proof. First, we reduce the problem with "many" assignments and "few" terms to a collection of instances with the same number of assignments as there are terms. Assuming $v \ge t$, we divide the set of assignments into $\lceil v/t \rceil$ groups of size at most t each, and evaluate the t-term DNF on each of the t assignments in $t^{2-\Omega(1/\log n)}$ time, by Theorem 1.1 and Theorem 4.2. The batch Boolean Orthogonal Detection algorithm is called for $O(v/t)$ times, yielding the claimed running time. □

Note that if $v \le t$ instead, we can obtain $t \cdot v^{1-1/O(\log c)}$ time by an analogous argument.

4.3 Symmetric Boolean CSP Here, we show how to orthogonal detection can be applied to solve a large class of Boolean constraint satisfaction problems faster than exhaustive search, in running time that is competitive with the fastest known CNF-SAT algorithms.

Theorem 4.3. *If the* Boolean Orthogonal Detection *problem with n vectors in d dimensions is solvable in $T(n, d)$ time, then symmetric Boolean CSPs with n variables and m constraints is solvable in $\tilde{O}(T(2^{n/2}, O(m \cdot n^2)))$ time.*

The reduction shows how to reduce a symmetric Boolean CSP instance to the problem of satisfying a small CNF problem restricted to a special combinatorial rectangle $A \times B$ of variable assignments. This is surprising, as a small CNF cannot possibly implement general symmetric functions, such as the MAJORITY function! Our trick is to re-encode the CSP problem in both the CNF *and the choice of the rectangle* $A \times B$, so that the CNF evaluation problem actually takes place over a new set of variables, clauses, and assignments, compared to the original CSP.

Proof. Let C be a symmetric Boolean CSP with n variables and m constraints; for simplicity, assume n is even. Divide the variables of C into two halves X and Y such that $|X| = |Y| = n/2$. We will first build a related CNF F based on this variable partition.

For each constraint c in C, create $t = 2\log(n/2)$ new variables $x_{c,1}, \ldots, x_{c,t/2}$ and $y_{c,1}, \ldots, y_{c,t/2}$ in the new CNF F. Add $O(n^2)$ clauses to F over these t variables, where each clause corresponds to a pair of numbers $(p, q) \in [n/2]^2$ such that $p + q$ true literals (p from variables of X, q from variables of Y) makes the symmetric constraint c evaluate to *false*. More precisely, let p and q be $\log(n/2)$-bit binary strings corresponding to non-negative numbers such that $p + q$ true literals yield a false output for the symmetric function for c. Then the clause $s_{p,q}$ for the pair of numbers (p, q) is as follows: for variable x_i, if $p_i = 1$ then put $\neg x_i$ in the clause $s_{p,q}$; if $p_i = 0$ then put x_i in $s_{p,q}$. We do analogously for the variables y_i and bits q_i. The resulting clause $s_{p,q}$ of literals is false precisely on the variable assignment (p, q).

The conjunction of these $O(n^2)$ clauses $s_{p,q}$ form a CNF which is true on a variable assignment

$$(a_{c,1}, \ldots, a_{c,t/2}, b_{c,1}, \ldots, b_{c,t/2}) \in \{0,1\}^{2\log(n/2)}$$

if and only if the non-negative integer a represented by the bit string $a_{c,1} \cdots a_{c,t/2}$ and the non-negative integer b represented by $b_{c,1} \cdots b_{c,t/2}$ sums to a number of *true* inputs which satisfies the symmetric constraint c. In total, the CNF F has $O(m \cdot n^2)$ clauses.

For each of the $2^{n/2}$ assignments P to the $n/2$ variables of X in C, make an assignment to all variables $x_{c,1}, \dots, x_{c,t/2}$ of D corresponding to the number of true literals in c set by P, for all constraints c in C. Let the collection of such assignments be $\mathcal{X}$. An analogous procedure is done for all $2^{n/2}$ assignments to the $n/2$ variables of Y, call this assignment collection $\mathcal{Y}$. We have two collections $\mathcal{X}$ and $\mathcal{Y}$ of variable assignments such that there is a pair of assignments (one from $\mathcal{X}$ and one from $\mathcal{Y}$) satisfying the CNF F if and only if the original CSP is satisfiable.

Now we reduce this restricted satisfiability problem on F to BOOLEAN ORTHOGONAL DETECTION, similar to Theorem 4.1. For every assignment $P' \in \mathcal{X}$, make a Boolean vector $v_{P'}$ with dimension equal to the number of clauses of F. The vector $v_{P'}$ and has a 1 in the ith component if and only if the ith clause of F is not satisfied by the partial assignment P'. Similarly, for every assignment $Q' \in \mathcal{Y}$, make a Boolean vector $w_{Q'}$ with dimension equal to the number of clauses of F, and put a 1 in the ith component if and only if the ith clause is not satisfied by Q'. Observe that $\langle v_{P'}, w_{Q'} \rangle = 0$ if and only if the variable assignment (P', Q') satisfies F. Therefore, a call to BOOLEAN ORTHOGONAL DETECTION on these $N = O(2^{n/2})$ vectors of dimension $d = O(m \cdot n^2)$ will solve the original CSP. □

REMINDER OF THEOREM 1.4 *Symmetric Boolean CSPs with n variables and m constraints are solvable in $O(2^{n(1-1/O(\log mn))})$ time.*

Proof. Apply Theorem 1.1 to the previous theorem. □

4.4 Longest Matching Substrings Our algorithms for $\texttt{LCS}^*$ and $\texttt{LMS}_\texttt{k}$ follow the same approach. We first show how to use our new algorithm for BOOLEAN ORTHOGONAL DETECTION to solve the problems faster when the optimal substring is short, then we devise an algorithm (in Section 5) that is fast when the optimal substring is long.

LEMMA 4.2. *If* BOOLEAN ORTHOGONAL DETECTION *with n vectors in d dimensions is solvable in $T(n, d)$ time, then, given two strings S, T of length n over $\Sigma \cup \{\star\}$, we can find length of the longest common substring of S and T, or report that it is of length at least Δ, in $T(n, O(\Delta \log |\Sigma|)) \cdot O(\log \Delta)$ time.*

Proof. Binary search for the maximum length $K \in \{0, \dots, \Delta\}$ for which there are common substrings of length K. For a given guess K we generate all $N = n - k$ substrings of length exactly K of S, $Sub_K(S) = \{s_1, \dots, s_N\}$, and of T, $Sub_K(T) = \{t_1, \dots, t_N\}$.

For each σ in the alphabet Σ, associate it with a unique bit vector v_σ of length $4 \log |\Sigma|$ that has exactly $2 \log |\Sigma|$ ones. Since there are $\binom{4 \log |\Sigma|}{2 \log |\Sigma|} > |\Sigma|$ such vectors, this is possible. For every s_i, we replace each alphabet symbol σ in s_i with the bit vector v_σ, and we replace every $\star$ with the all-zeroes vector of length $4 \log |\Sigma|$. For every t_j, we replace each σ in t_j with the *complement* of v_σ (the unique vector with 1's flipped to 0's and 0's flipped to 1's), and every $\star$ is replaced with the all-zeroes vector.

This results in bit vectors $s'_1, ..., s'_N, t'_1, ..., t'_N$ of length $4K \log |\Sigma|$ such that $\langle s'_i, t'_j \rangle = 0$ if and only if s_i and t_j match. Therefore, given the algorithm for BOOLEAN ORTHOGONAL DETECTION we can check if there are matching substrings of length K in $T(n, 4K \log |\Sigma|)$ and the overall runtime is as claimed. □

REMINDER OF THEOREM 1.2 *There is a randomized algorithm that solves the Longest Common Substring with don't cares problem on two strings of length n in $n^2/2^{\Omega(\sqrt{\log n})}$ time.*

Proof. Lemma 5.2 in Section 5 gives an algorithm that finds the optimal substring of length at least Δ in $\tilde{O}(n^2/\sqrt{\Delta})$ time. Applying Theorem 1.1 to Lemma 4.2, we solve the case in which the optimal substring is of length at most Δ in $n^{2-1/O\left(\log \frac{\Delta \log |\Sigma|}{\log n}\right)} = n^{2-1/O(\log \Delta)}$ time. Running both algorithms and setting $\Delta = 2^{\Omega(\sqrt{\log n})}$, we find the optimal substring in $n^2/2^{\Omega(\sqrt{\log n})}$ time. □

We now turn to the $\texttt{LMS}_\texttt{k}$ problem.

LEMMA 4.3. *If* BOOLEAN ORTHOGONAL DETECTION *with n vectors in d dimensions is solvable in $T(n, d)$ time, then, given two strings S, T of length n over $\Sigma \cup \{\star\}$, we can find length of the longest substring of S with edit distance less than k to a substring of T, or report that it is of length at least Δ, in $T(n, O(\Delta \log |\Sigma|)) \cdot O(\Delta^k)$ time.*

Proof. We will look for the largest $K_1 \leq \Delta$ such that there exists $K_2 \leq \Delta$ and two substrings s of S and t of T where $|s| = K_1, |t| = K_2$ for which $ED(s, t) \leq k$.

By definition of the edit distance, for two strings s, t of lengths K_1, K_2, $ED(s, t) \leq k$ if and only if there is an alignment $\mathcal{A} \in A_{K_1,K_2}$ of cost up to k that is valid for s and t. Observe that the number of alignments of two strings of length $\leq \Delta$ that have cost up to k can be upper bounded by $3^k \cdot \binom{\Delta+k}{k}$. For a pair of lengths K_1, K_2, let us define the set of all such alignments $A^{\leq k}_{K_1,K_2} = \{\mathcal{A} \in A_{K_1,K_2} \mid |\mathcal{A}_{Mis}| \leq k\}$. Note that this set depends on k and K_1, K_2 but not on the strings s, t.

The idea is to go over all relevant $O(\Delta^2)$ pairs $K_1, K_2 \in [\Delta]$ and for each such pair we enumerate over all $3^k \cdot \binom{\Delta+k}{k}$ alignments $A \in A^{\leq k}_{K_1,K_2}$ and try to find two substrings s, t of S, T for which A is a valid alignment using a single call to our BOOLEAN ORTHOGONAL DETECTION algorithm. The number of calls becomes $O(\Delta^2 \cdot 3^k \cdot \binom{\Delta+k}{k}) = O(\Delta^k)$.

Observe that checking whether $A \in A^{\le k}_{K_1,K_2}$ is a valid alignment for a pair s, t is equivalent to checking that for every pair $(i, j) \in A_{Match}$ either $s[i] = t[j]$ or one of $s[i], t[j]$ is a $\star$. To do this, we can assign a distinct index $I(i, j)$ to each of the pairs $(i, j) \in A_{Match}$ and then reduce s, t to vectors s', t' as follows. First, define s'', t'' by deleting and reordering the characters in s, t so that for each pair $(i, j) \in A_{Match}$ we have $s[i]$ at the $I(i, j)$ position in s'' and $t[j]$ at the $I(i, j)$ position in t''. Note that s'', t'' must be of the same length $K'' \le K_1, K_2$ and that s'', t'' match if and only if A is a valid assignment for s, t. Then, convert s'', t'' to binary vectors s', t' of length $4K'' \log |\Sigma|$ as in the proof of Lemma 4.2, so that s'', t'' match if and only if $\langle s', t' \rangle = 0$. To check whether A is valid for any substrings $|s| = K_1, |t| = K_2$ of S, T we generate all $O(n)$ substrings of S, T of the given lengths and convert each substring to a binary vector of length $O(\Delta \log |\Sigma|)$ as described, and check for an orthogonal pair of vectors. The total running time becomes $T(n, O(\Delta \log |\Sigma|)) \cdot O(\Delta^k)$. □

Reminder of Theorem 1.3 *There is a randomized algorithm that solves the Longest k-Matching Substring problem on two strings of length n in $k^{1.5}n^2/2^{\Omega(\sqrt{\frac{\log n}{k}})}$ time.*

Proof. Lemma 5.1 in Section 5 gives an algorithm that finds the optimal substring of length at least Δ in $\tilde{O}(k^{1.5}n^2/\sqrt{\Delta})$ time. Applying Theorem 1.1 to Lemma 4.2 we solve the case in which the optimal substring is of length no more than Δ in $\Delta^k n^{2-1/O(\log \frac{\Delta \log |\Sigma|}{\log n})} = \Delta^k n^{2-1/O(\log \Delta)}$ time. Running both algorithms and setting $\Delta = 2^{\Omega(\sqrt{\frac{\log n}{k}})}$ we get the optimal substring in time $k^{1.5}n^2/2^{\Omega(\sqrt{\frac{\log n}{k}})}$. □

5 Finding long matching substrings

In this section we describe an algorithm that gets two strings $S, T \in (\Sigma \cup \{\star\})^n$ and a parameter Δ and returns the longest matching substring of S and T, if there is such a substring of length at least Δ. We will first focus on $\texttt{LMS}_\texttt{k}$ and then remark that the algorithm for $\texttt{LCS}^*$ is similar but simpler.

For a string $S \in (\Sigma \cup \{\star\})^n$ and integer $K \in [n]$ let B^S_K be a partition of S into $N = \lceil n/K \rceil$ segments (or blocks) $b_1, \ldots, b_N$ of length up to K as follows: $\forall i \in [N-1] : b_i = S[(i-1)K+1, \cdots, iK]$ and $b_N = S[(N-1)K+1, \cdots, n]$.

Akutsu [Aku95] presents an algorithm for the problem of finding all substrings of a text T of length n that have Edit Distance less than k to a given pattern P of length m, where both strings might contain don't care symbols, that runs in $\tilde{O}(\sqrt{k}mn)$ time. This algorithm combines the Landau-Vishkin [LV86] dynamic programming procedure with an algorithm for pattern-matching with don't cares [FP73, Ind98, Kal02].

For any block $b_i \in B^S_K$, we can use Akutsu's algorithm to check whether for some $t \subseteq T : ED(b_i, t) \le k$ in $\tilde{O}(\sqrt{kK}n)$ time. Doing this for each block, we can check whether any block in B^S_K matches a substring of T in $\tilde{O}(n/K \cdot n\sqrt{kK}) = \tilde{O}(\sqrt{k}n^2/\sqrt{K})$ time.

We perform this check for $K = n, n/2, n/4, \ldots, \Delta$ and find K^*; the largest such K for which $\exists b_i \in B^S_K, t \subseteq T$ such that $ED(b_i, t) \le k$. If no such $K \ge \Delta$ exists, we return that there is no matching substring of length more than 2Δ. The following simple claim shows that we will be correct.

Claim 1. *If z is the longest substring of S such that $\exists t \subseteq T : ED(z, t) \le k$, and $K \le |z|/2$, then there exists a block $b_i \in B^S_K$ and $t' \subseteq T$ such that $ED(b_i, t') \le k$.*

Proof. Our optimal substring z must contain at least one block $b_i \in B^S_K$ since otherwise $|z| \le 2K - 2$. Letting t' be the substring of t that is aligned with b_i when aligning z and t we get that $ED(b_i, t') \le ED(z, t) \le k$. □

This claim also shows that in case we found $K^* \ge \Delta$ in the above checks, we are certain that the length of the optimal substring $z \subseteq S$ is between K^* and $2K^*$. An additional observation is that in the optimal alignment of z to some $t \subseteq T$ we must have a substring $z' \subseteq z \subseteq S$ of length at least $|z'| \ge K/k$ and a substring $t' \subseteq t \subseteq T$ such that z' and t' are aligned without any mismatches/indels and, in particular, $z' \equiv t$. Note that otherwise $ED(z, t) > k$. Therefore, if we set $K' = K/2k$, we are certain that there exists a block $b_i \in B^S_{K'}$ and some $t'' \subseteq T$ such that in the optimal alignment of z and t, b_i is aligned to t'' without any mismatches/indels. Consequently, if we go over all pairs $b_i \in B^S_{K'}, y \subseteq T$ for which $b_i \equiv y$ and then try to expand these substrings left and right as much as possible without incurring more than k mismatches/indels, we will end up with the optimal substrings z and t.

This is exactly what we will do. Kalai's algorithm for pattern matching with don't cares [Kal02] can find all occurrences of a pattern P of length m in a text T of length n, when don't care symbols are allowed, in $O(n \log m)$ time. Thus, we can find all pairs $b_i \in B^S_{K'}, y \subseteq T$ for which $b_i \equiv y$ in $\tilde{O}(n/K' \cdot n) = \tilde{O}(kn^2/K^*)$. Our goal is now to expand each of these pairs as much as possible, which gives rise to the following definition:

Definition 5.1. (Longest Matching Prefix) *For two strings $S, T \in (\Sigma \cup \{\star\})^n$ and a parameter $k \in [n]$, $\texttt{LMP}(S, T, k)$ is the largest $i \in [n]$ such that $\exists i' \in [n] : ED(S[1 \cdots i], T[1 \cdots i']) \le k$.*

Fix a pair $b_i \in B^S_{K'}, y \subseteq T$ that we are trying to expand and let $b^+_i \subseteq S$ be the substring of S of length $2K^*$ that is immediately to the right of b_i. Similarly, let $y^+ \subseteq T$ be the substring of T of length $2K^*$ that is immediately to the right of y. Analogously we define b^-_i and y^- to be the substrings on the left of our pair. Let $\alpha \in [k]$ and consider

$a = \texttt{LMP}(b_i^+, y^+, \alpha)$, the length of the longest prefix of b_i^+ that has edit distance less than α to a prefix of y^+. Observe that if we expand b_i to $b_i \circ (b_i^+[1 \cdots a])$ we get an expansion to the right of our pair with α mismatches/indels. Similarly, let $\beta \in [k]$ and consider $b = \texttt{LMP}((b_i^-)^T, (y^-)^T, \beta)$, to get an expansion to the left with β mismatches/indels. Moreover, we can combine these two expansions to get two superstrings of our pair, of additional length $a+b$, that have Edit Distance no more than $\alpha+\beta$. A subtle point is that although expanding in this way might sometimes result in strings whose Edit Distance is less than $\alpha+\beta$, and this might be suboptimal, we are guaranteed to find the optimal alignment of our optimal pair z and t this way. This is because we start the expansion from a pair of substrings that is aligned to each other in the optimal alignment and we solve the left and right parts optimally.

Akutsu's implementation [Aku95] of the Landau-Vishkin [LV86] dynamic programming algorithm for approximate pattern matching allows us to find all substrings of a text T that are of edit-distance less than e to a maximal prefix of a pattern P, for every $e \in [k]$, in $\tilde{O}(\sqrt{k|P|} \cdot |T|)$ time. To support don't cares, Akutsu replaces the Suffix Trees of Landau and Vishkin with a less efficient lookup table that he computes using a pattern matching with don't cares algorithm. Although finding all such occurrences is not a direct goal of the Landau-Vishkin algorithm, the dynamic programming table $L[d,e]$ (see [LV86, Aku95]) records the information about maximal prefixes which is very useful for us. We will run Akutsu's algorithm with T as the text and b_i^+ as the pattern, for every substring $b_i \in B_{K'}^S$ and obtain all the required $\texttt{LMP}(b_i^+, y^+, \alpha)$ values. The computations for b_i^- can be done similarly. The running time of this stage is $\tilde{O}(\frac{n}{K'} \cdot \sqrt{kK^*} \cdot n) = \tilde{O}(k^{1.5} n^2/\sqrt{K})$.

Thus, for each pair $b_i \in B_{K'}^S, y \subseteq T$, we will go over all pairs of integers $\alpha, \beta \in [k]$ such that $\alpha + \beta = k$ and consider the corresponding indices $a, b \in [2K']$. Each such pair α, β gives us an expansion by $a + b$, and we can take the pair that maximizes $a+b$ as the best expansion of b_i and y. Expanding each pair and returning the longest substring found is guaranteed to give us the optimal value, and we are done. This final computation takes $O(k \cdot n^2/K') = O(k^2 n^2/K)$.

The overall running time is $\tilde{O}(k^{1.5} n^2/\sqrt{\Delta})$ since $K^* \geq \Delta$ and $K > k$.

LEMMA 5.1. *Given two strings S, T of length n over $\Sigma \cup \{\star\}$, we can find length of the longest substring of S with edit distance less than k to a substring of T, or report that it is of length less than Δ, in $\tilde{O}(k^{1.5} n^2/\sqrt{\Delta})$ time.*

To solve $\texttt{LCS}^*$ we follow the same approach with some simplifications. To find K^* and substrings of S of length K^* such that expanding them to length up $2K^*$ would result in one of the optimal substrings, we use Kalai's algorithm instead of Akutsu's and the running time of this stage decreases to $\tilde{O}(n^2/\Delta)$. To expand each of the $O(n/K^*)$ blocks of S with its $O(n)$ matching substrings of T we first compute a matrix recording for each of the $O(n^2/\sqrt{\Delta})$ pairs of a block of length $\sqrt{\Delta}$ of S and substring of length $\sqrt{\Delta}$ of T whether they match. This matrix is computed using Kalai's algorithm in $\tilde{O}(n^2/\sqrt{\Delta})$ time. Then, using this table, we can expand each block from length $K*$ to length up to K^* using $K^*/\sqrt{\Delta} + \sqrt{\Delta}$ steps. The total running time becomes $\tilde{O}(n^2/K^* \cdot (K^*/\sqrt{\Delta} + \sqrt{\Delta})) = \tilde{O}(n^2/\sqrt{\Delta})$ since $K^* \geq \Delta$.

LEMMA 5.2. *Given two strings S, T of length n over $\Sigma \cup \{\star\}$, we can find length of the longest common substring of S and T, or report that it is of length less than Δ, in $\tilde{O}(n^2/\sqrt{\Delta})$ time.*

6 Conclusion

We have given a new algorithm for detecting orthogonal vectors in a Boolean domain, with several applications to processing partial match queries in batch, evaluating circuits, and improved optimization algorithms. Our reductions demonstrate the considerable power of Boolean orthogonal detection. While the reductions we introduce in this paper are applied in a positive way, one can also think of them as more evidence that BATCH PARTIAL MATCH and BOOLEAN ORTHOGONAL DETECTION will be hard to solve in $n^{2-\varepsilon}$ time for some universal $\varepsilon > 0$: such an algorithm would not only imply faster algorithms for solving CNF-SAT, but it would also yield faster algorithms for *every* symmetric Boolean CSP (for example).

We believe the most tantalizing open problem is to extend our results to *Hamming nearest neighbors*: is there an algorithm with similar running time for finding a closest pair of Boolean vectors under the Hamming metric? This is a natural next step for developing good exact neighbor search algorithms, and to apply this technique to problems like Local Alignment and Edit Distance. However, it seems to require the evaluation of a circuit that we do not yet know how to handle. If one could efficiently evaluate an OR of MAJORITY of XORs on a combinatorial rectangle of inputs, that would yield improved exact algorithms for closest pair in the Hamming metric.

Acknowledgements. We thank the SODA reviewers for their helpful comments.

References

[AGM$^+$90] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[Aku95] Tatsuya Akutsu. Approximate string matching with don't care characters. *Inf. Process. Lett.*, 55(5):235–239, 1995.

[ALP04] Amihood Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with k mismatches. *J. Algorithms*, 50(2):257–275, 2004.

[AVW14] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *ICALP (1)*, pages 39–51, 2014.

[BFC08] P. Bille and M. Farach-Colton. Fast and compact regular expression matching. *Theoretical Computer Science,*, 409(3):486–496, 2008.

[BOR99] Allan Borodin, Rafail Ostrovsky, and Yuval Rabani. Lower bounds for high dimensional nearest neighbor search and related problems. In *STOC*, pages 312–321, 1999.

[CD04] Nadia Creignou and Hervé Daudé. Combinatorial sharpness criterion and phase transition classification for random csps. *Inf. Comput.*, 190(2):220–238, 2004.

[CEPR10] Raphal Clifford, Klim Efremenko, Ely Porat, and Amir Rothschild. Pattern matching with don't cares and few errors. *Journal of Computer and System Sciences*, 76(2):115 – 124, 2010.

[CGL04] Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and dont cares. In *STOC*, pages 91–100, 2004.

[Cha05] Timothy M. Chan. All-pairs shortest paths with real weights in $O(n^3/\log n)$ time. *Algorithmica*, 50(2):236–243, 2008. See also WADS'05.

[CIP02] Moses Charikar, Piotr Indyk, and Rina Panigrahy. New algorithms for subset query, partial match, orthogonal range searching, and related problems. In *ICALP*, pages 451–462, 2002.

[CIP06] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for sat. In *IEEE Conf. Computational Complexity*, pages 252–260, 2006.

[CIP09] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *Parameterized and Exact Computation*, pages 75–85. Springer, 2009.

[CLZU03] M. Crochemore, G.M. Landau, and M. Ziv-Ukelson. A subquadratic sequence alignment algorithm for unrestricted scoring matrices. *SIAM Journal on Computing*, 32:1654–1673, 2003.

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

[DH09] Evgeny Dantsin and Edward A. Hirsch. Worst-case upper bounds. 185:403–424, 2009.

[FGKU14] Tomás Flouri, Emanuele Giaquinta, Kassian Kobert, and Esko Ukkonen. Longest common substrings with k mismatches. *CoRR*, abs/1409.1694, 2014.

[FP73] M.J. Fischer and M.S. Paterson. String matching and other products. *SIAM-AMS Proc.*, 7:113–125, 1973.

[Gra14] Szymon Grabowski. A note on the longest common substring with k-mismatches problem. *CoRR*, abs/1409.7217, 2014.

[Gus97] Dan Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge University Press, 1997.

[ILPS14] Russell Impagliazzo, Shachar Lovett, Ramamohan Paturi, and Stefan Schneider. 0-1 integer linear programming with a linear number of constraints. *CoRR*, abs/1401.5512, 2014.

[Ind98] P. Indyk. Faster algorithms for string matching problems: Matching the convolution bound. In *Proc. FOCS*, pages 166–, 1998.

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

[IPS13] Russell Impagliazzo, Ramamohan Paturi, and Stefan Schneider. A satisfiability algorithm for sparse depth two threshold circuits. In *FOCS*, pages 479–488, 2013.

[JKKR04] T. S. Jayram, Subhash Khot, Ravi Kumar, and Yuval Rabani. Cell-probe lower bounds for the partial match problem. *J. Comput. Syst. Sci.*, 69(3):435–447, 2004.

[JMV13] Hamid Jahanjou, Eric Miles, and Emanuele Viola. Local reductions. *arXiv preprint arXiv:1311.3171*, 2013.

[Kal02] A. Kalai. Efficient pattern-matching with don't cares. In *SODA*, pages 655–656, 2002.

[KSV14] Tomasz Kociumaka, Tatiana Starikovskaya, and Hjalte Wedel Vildhøj. Sublinear space algorithms for the longest common substring problem. *CoRR*, arXiv:1407.0522, 2014.

[LV86] Gad M. Landau and Uzi Vishkin. Efficient string matching with k mismatches. *Theoretical Computer Science*, 43(0):239 – 249, 1986.

[MNSW98] Peter Bro Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson. On data structures and asymmetric communication complexity. *J. Comput. Syst. Sci.*, 57(1):37–49, 1998.

[MP80] W.J. Masek and M.S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20, 1980.

[MP94] S. Muthukrishan and K. K. Palem. Non-standard stringology: Algorithms and complexity. In *STOC*, pages 770–779, 1994.

[Pat11] Mihai Patrascu. Unifying the landscape of cell-probe lower bounds. *SIAM J. Comput.*, 40(3):827–847, 2011.

[PT06] Mihai Patrascu and Mikkel Thorup. Higher lower bounds for near-neighbor and further rich problems. In *FOCS*, pages 646–654. IEEE, 2006.

[PTW08] Rina Panigrahy, Kunal Talwar, and Udi Wieder. A geometric approach to lower bounds for approximate near-neighbor search and partial match. In *FOCS*, pages 414–423, 2008.

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

[Riv74] Ronald Linn Rivest. *Analysis of Associative Retrieval Algorithms.* PhD thesis, 1974.

[RV13] Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *STOC*, pages 515–524, 2013.

[Smo87] Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *STOC*, pages 77–82, 1987.

[SW81] T.F. Smith and M.S. Waterman. The identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

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

[Wil14] Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *STOC*, pages 664–673, 2014.