View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Dagstuhl Research Online Publication Server

Orthogonal Vectors Indexing*

Isaac Goldstein†1, Moshe Lewenstein‡2, and Ely Porat§3

1 **Bar-Ilan University, Ramat Gan, Israel**
goldshi@cs.biu.ac.il
2 **Bar-Ilan University, Ramat Gan, Israel**
moshe@cs.biu.ac.il
3 **Bar-Ilan University, Ramat Gan, Israel**
porately@cs.biu.ac.il

Abstract

In the recent years, intensive research work has been dedicated to prove conditional lower bounds in order to reveal the inner structure of the class P. These conditional lower bounds are based on many popular conjectures on well-studied problems. One of the most heavily used conjectures is the celebrated Strong Exponential Time Hypothesis (SETH). It turns out that conditional hardness proved based on SETH goes, in many cases, through an intermediate problem - the Orthogonal Vectors (OV) problem.

Almost all research work regarding conditional lower bound was concentrated on time complexity. Very little attention was directed toward space complexity. In a recent work, Goldstein et al. [17] set the stage for proving conditional lower bounds regarding space and its interplay with time. In this spirit, it is tempting to investigate the space complexity of a data structure variant of OV which is called *OV indexing*. In this problem n boolean vectors of size $c \log n$ are given for preprocessing. As a query, a vector v is given and we are required to verify if there is an input vector that is orthogonal to it or not.

This OV indexing problem is interesting in its own, but it also likely to have strong implications on problems known to be conditionally hard, in terms of time complexity, based on OV. Having this in mind, we study OV indexing in this paper from many aspects. We give some space-efficient algorithms for the problem, show a tradeoff between space and query time, describe how to solve its reporting variant, shed light on an interesting connection between this problem and the well-studied SetDisjointness problem and demonstrate how it can be solved more efficiently on random input.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases SETH, orthogonal vectors, space complexity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.40

1 Introduction

Recently, there is an intensive research work aimed at understanding the complexity within the class P (decision problems that are solved by polynomial time algorithms). Specifically, many conditional lower bounds have been proven on many polynomial algorithmic problems. These lower bounds are based on some conjectures on well-studied problems, especially

* A full version of the paper is available at https://arxiv.org/abs/1710.00586.
† This research is supported by the Adams Foundation of the Israel Academy of Sciences and Humanities
‡ This work was partially supported by an ISF grant #1278/16
§ This research is supported by the Adams Foundation of the Israel Academy of Sciences and Humanities

© Isaac Goldstein, Moshe Lewenstein and Ely Porat;
licensed under Creative Commons License CC-BY

notable are 3SUM, APSP and SETH. The Strong Exponential Time Hypothesis (SETH) [18, 19] states the following:

▶ **Conjecture 1** (Strong Exponential Time Hypothesis). *There is no $\epsilon > 0$ such that kSAT can be solved in $O(2^{(1-\epsilon)n})$ for all k.*

Many conditional lower bounds for both polynomial and exponential time solvable problems are based on this conjecture. A partial list includes [25, 21, 14, 28, 4, 6, 8, 16, 7, 2, 9, 1, 5, 22]. For polynomial time solvable problems many of the conditional lower bounds are proven through the use of an intermediate problem called *Orthogonal Vectors* (OV) which is defined as follows.

▶ **Definition 1** (Orthogonal Vectors). Given a set S of n input vectors from $\{0,1\}^d$, decide if there are $u, v \in S$ such that u is orthogonal to v.

If SETH is true then there is no $O(n^{2-\epsilon})$ solution for OV for any $\epsilon > 0$ (see [29, 30]). This conditional lower bound on OV was heavily used to obtain conditional lower bounds on the time complexity of a long list of algorithmic problems. This includes graph problems [28, 22], dynamic problems [4], string problems [6, 7, 2, 9] and many other important problems from a variety of research fields.

A recent work by Goldstein et al. [17] set the stage for proving conditional lower bounds on space-time tradeoffs. Specifically, it was suggested that we can achieve space lower bounds by considering a data structure variant of SAT. Given a formula ϕ in a CNF format and a list of variables L from ϕ, we need to preprocess ϕ and L and create a data structure to support the following queries. Given an assignment to all variables not in L we are required to answer if this assignment can be completed to a full assignment that satisfies ϕ. A closely related problem is *Orthogonal Vectors Indexing* (OV Indexing) that is defined as follows.

▶ **Definition 2** (Orthogonal Vectors Indexing). Given a set S that contains n d-length boolean vectors, preprocess S and answer queries of the following form: given d-length boolean vector v, is there a vector in S which is orthogonal to v.

SETH can be reduced to OV indexing (see the details in the full version of this paper). As a consequence of this reduction there is no polynomial time preprocessing algorithm for OV indexing that achieves truly sublinear query time.

The main question that we consider is what the space requirements of OV indexing are. In this paper we examine this question in detail from various aspects for the case that $d = c \log n$ for some constant $c > 1$ (if c is non-constant its seems hard to achieve any improvement due to the connection to SETH). On one hand, solving OV indexing for input vectors of length $c \log n$ can be done easily using a lookup table of size n^c. Using this table, queries can be answered in constant time. On the other hand, without any preprocessing queries can be answered in linear time. It is interesting to figure out what can be done in between these two extremes. Can we achieve truly sublinear query time with less than n^c space? Is there a clear tradeoff between time and space? What can we say about the reporting version of this problem? In this paper we investigate all these questions and more.

Understanding the space requirements of OV indexing is interesting in its own right, but it can have many implications on other problems. Along the lines of Goldstein et al. [17] OV indexing can serve as a basis for proving conditional hardness in terms of space for other algorithmic problems. Specifically, as OV is a standard tool in demonstrating conditional hardness of problems in terms of time it is likely that understanding the space hardness of its data structure variant - OV indexing - can be applied to many problems shown to be hard based on OV. In the work by Goldstein et al. [17] there was an attempt to state a general hardness conjecture for OV indexing. However, as no solution to neither OV indexing nor the data structure variant of SAT was suggested in [17] (other than the trivial ones), a more fine grained conjecture was out of reach. One major motivation for this paper is to state such a conjecture based on improved upper bounds for OV indexing (see more detailed discussion in the last section of this paper).

Related Work. The Partial Match problem and its variants were extensively studied for decades. These problems are related to our OV indexing problem (see, for example, [3]). One of the first works regarding Partial Match is by Rivest [26, 27]. However, his work focused on the average case analysis of several solutions for the problem that in the worst case do not achieve an improvement over the trivial solution, unless the number of "don't cares" symbols (corresponding to the zeroes in the OV indexing problem) in the query is not too large. Many works on the Partial Match problem and its variants focus on improving the time complexity rather than the space complexity which is the main concern of this paper. Other works that do consider space complexity deal with the case of very large dimension d that can be even linear in n [10, 13, 20]. This case admits very different behaviour from the case we handle in this paper in which $d = \Theta(\log n)$.

Our Results. In this paper we present the following results regarding OV indexing. We suggest 3 algorithms that solve OV indexing with truly less that n^c space and truly sublinear query time. We show how to use the second and third algorithms we present to get a tradeoff between space and query time. A variant of the first algorithm is used to prove the connection between OV indexing and SetDisjointness, a problem which was considered by several papers as the basis for showing space conditional hardness. We also solve the reporting variant of OV indexing in which we need to report all input vectors that are orthogonal to our query vector. Finally, we show that, on random input vectors, OV indexing can be solved more efficiently in terms of space.

2 DivideByOnes: First Space-Efficient Solution for OV indexing

Our goal is to achieve an algorithm that has truly sublinear query time and requires $O(n^{c-\epsilon})$ space for some $\epsilon > 0$. This is an improvement over the trivial algorithm that uses n^c space. We note that in this solution and throughout this paper the notations $\tilde{O}$ and $\tilde{\Omega}$ (almost always) suppress not just polylogarithmic factors as usual, but also all factors that are smaller than n^ϵ for any $\epsilon > 0$.

2.1 DivideByOnes Algorithm

Preprocessing. The first step is to save a set S_1 of all vectors from S with at most $c_1 \log n$ ones for some constant $0 < c_1 \leq c/2$. There are at most $\sum_{k=0}^{c_1 \log n} \binom{c \log n}{k} \leq c_1 \log n \binom{c \log n}{c_1 \log n}$ vectors in S_1. We have that $\binom{c \log n}{c_1 \log n} \approx n^{c \log c - c_1 \log c_1 - (c-c_1) \log (c-c_1)}$. We choose the largest c_1 such that the number of vectors in S_1 will be $\tilde{O}(n^{1-\epsilon})$ for some $\epsilon > 0$.

Let S_2 be the set of vectors from S with more than $c_1 \log n$ ones. Assume that c is an integer. We split each vector in S_2 into c parts each of length $\log n$ bits. As all the vectors in S_2 have at least $c_1 \log n$ ones, we are guaranteed that at least one of the c parts of each vector has at least $\frac{c_1}{c} \log n$ ones.

We have n possible vectors of size $\log n$, so we create c arrays $A_1, A_2, ..., A_c$ of length n each, such that the ith entry in each array represents the $\log n$-length boolean vector that its numerical value is i. In the ith entry of an array A_j we create a list that contains each vector $v \in S_2$ such that: (i) The number of ones it has in its jth part is the maximum among all its parts (ties are broken arbitrarily). (ii) The value of its bits in its jth part is orthogonal to the value of the $\log n$-length vector whose numerical value is i (the numerical value of an m-length vector is the value of this boolean vector that is parsed as an m-length boolean number).

To analyse the space consumed by these arrays one should notice that each vector $v \in S_2$ appears only in one array. Moreover, as v appears only in the array that represents the part in which v has the maximum number of ones, the number of lists in this array that contain v is at most $\frac{n}{2^{\frac{c_1}{c}\log n}} = n^{1-\frac{c_1}{c}}$. Therefore, the total size of all arrays is no more than $n \cdot n^{1-\frac{c_1}{c}} = n^{2-\frac{c_1}{c}}$ which is truly subquadratic.

Query. When we get a query vector u we first check in S_1 if there is a vector that is orthogonal to u. Then, we partition u to c equal parts. For each part j if the numerical value of all bits in this part is i we check all the vectors in the ith list of A_j and verify if one of them is indeed orthogonal to u. The problem with this process is that the length of the list we check may be $\tilde{\Omega}(n)$, so our query time will be $O(n)$ which is trivial. To overcome this and obtain a constant query time for long lists, we need to treat lists whose length is $\tilde{\Omega}(n)$ differently in the preprocessing phase.

Additional Preprocessing. For each entry i in some array that the length of the vectors list in it is not truly sublinear, we store a bitmap that tells for all possible values of the other $(c-1)\log n$ bits whether there is a vector in S that is orthogonal to these bits and the $\log n$ bits represented by i. The size of the bitmap is $2^{(c-1)\log n} = n^{c-1}$. As calculated before, the total number of vectors in all lists of the array is $n^{2-\frac{c_1}{c}}$. Consequently, the number of lists that have $\tilde{\Omega}(n)$ vectors in them is no more than $n^{1-\frac{c_1}{c}}$. Therefore, the space needed for all bitmaps is $n^{c-\frac{c_1}{c}}$.

2.1.1 Generalization to klogn

We can generalize the above solution by partitioning the vectors to parts whose size is $k\log n$ for some $k > 0$. First we consider the case that k divides c. In this case, the algorithm continues in same way as for the case that $k = 1$. The number of lists in each array is n^k. Each input vector v has at least $\frac{c_1}{c}k$ ones in the part with the largest number of ones. Consequently, each input vector v occurs in $n^{k-\frac{c_1}{c}k}$ lists in the array corresponding to the part with most ones in v. The total size of all arrays and lists is $O(n^{k+1-\frac{c_1}{c}k})$. The number of long lists is at most $O(n^{k-\frac{c_1}{c}k})$. Each bitmap has size n^{c-k}. Therefore, the space usage for handling long lists is $O(n^{c-\frac{c_1}{c}k})$. The total space of the data structure is $O(n^{k+1-\frac{c_1}{c}k} + n^{c-\frac{c_1}{c}k})$. By setting $k = c-1$ (if possible, otherwise see the next paragraph) we get the lowest space complexity, which is $O(n^{c-c_1(1-\frac{1}{c})})$.

In case k does not divide c, we can partition each vector to $\lfloor \frac{c}{k} \rfloor$ parts of length $k\log n$. However, we are left with one part P whose length is smaller than $k\log n$. It can be the case that for some input vector v the number of ones in each of the parts of length $k\log n$ is smaller than $\frac{c_1}{c}k$, as there can be many ones in P. In order to solve this problem we can do the following. Let $c_1' = c_1 - \epsilon$ for any $\epsilon > 0$. We define $k' = \lfloor \frac{k}{\epsilon} \rfloor \epsilon$ and $c' = \lfloor \frac{c}{\epsilon} \rfloor \epsilon$. It is clear that $k' > k - \epsilon$ and $c' > c - \epsilon$. Each input vector v can be partitioned to $m_1 = \lfloor \frac{c}{\epsilon} \rfloor$ parts

$P_1, P_2, ..., P_{m_1}$ whose length is ϵ and another optional part P whose length is less than ϵ. If we ignore the bits of any vector in P, we are still guaranteed that there are at least c_1' ones in the rest of the vector. We can choose $m_2 = \lfloor \frac{k}{\epsilon} \rfloor$ parts from the m_1 parts $P_1, P_2, ..., P_{m_1}$. This will give us exactly $k' \log n$ bits. There are $m_3 = \binom{m_1}{m_2}$ options of how to choose m_2 parts out of the m_1 parts. The number m_3 is constant as k, c and ϵ are all constants. Therefore, we can create m_3 arrays $A_1, A_2, ..., A_{m_3}$ each one of them represents some $k' \log n$ bits from our input vectors. We handle these arrays as in the regular case explained above. The crucial point one should observe is that for each input vector v there must be $k' \log n$ bits among these m_3 options that contains at least $\frac{c_1'}{c}k$ of the ones in v. Let A_i be the array representing $k' \log n$ bits out of the m_3 options that contains the maximum number of ones in v. We are guaranteed that v will appear in at most $n^{k' - \frac{c_1'}{c}k'}$ lists in A_i. We continue the solution as in the regular case. Following the analysis of the regular case, we have that the total space of the data structure will be $O(n^{k'+1-\frac{c_1'}{c}k'} + n^{c-\frac{c_1'}{c}k'})$. As $k - \epsilon < k' \le k$ and $c_1' = c_1 - \epsilon$, we get that the total space is $O(n^{k+1-\frac{c_1-\epsilon}{c}(k-\epsilon)} + n^{c-\frac{c_1-\epsilon}{c}(k-\epsilon)})$. Setting $k = c - 1$ as before, we get that the space is $O(n^{c-\frac{c_1-\epsilon}{c}(c-1-\epsilon)})$. We can make this space complexity as close as we wish to the space complexity for the case k divides c by choosing ϵ whose value is very close to 0. Consequently, we have the following result (c_1 is the largest number that satisfies $\binom{c \log n}{c_1 \log n} = \tilde{O}(n^{1-\delta})$ for some $\delta > 0$):

▶ **Theorem 3.** *For every $\epsilon > 0$ the DivideByOnes algorithm solves OV indexing with truly sublinear query time using $O(n^{c-\frac{c_1-\epsilon}{c}(c-1-\epsilon)})$ space.*

3 TopLevelsQueryGraph: Second Space-Efficient Solution for OV indexing

There are two problems with the previous solution. The first one is the sharp separation between long lists (having $\tilde{O}(n)$ vectors) and short lists. For long lists we use a large amount of space and answer queries very quickly in constant time, while for short lists we just save the vectors in the lists and spend time in the query stage. The second problem is that each input vector is saved many times in different lists.

3.1 Query Graph

In order to improve the space requirements for sublinear query time we introduce the notion of a *query graph*. The idea of the query graph is to create a tradeoff between query time and space and save each vector just once. We are now ready to define the query graph. A query graph is a directed acyclic graph $G = (V, E)$ such that each vertex v_i in V represents a boolean vector α_i of length $k \log n$. There is an edge $(v_i, v_j) \in E$ if the vectors α_i and α_j differ on exactly one element which is 0 in α_i and 1 in α_j. Following this definition the query graph can be viewed as a layered graph with $k \log n$ layers. The jth layer in this graph contains all the nodes v_i such that the number of ones in α_i is exactly j. All the edges from the vertices in the jth layer are directed to vertices in the $(j+1)$th layer. We call the layers for small values of j *top* layers and the layers with high values of j *bottom* layers.

Let W be a set of indices such that $W \subseteq [c \log n]$ and $|W| = k \log n$. We want each vertex v_i that represents a vector α_i to contain a list L_i of input vectors such that their elements in the indices specified by W are orthogonal to α_i. This is the same as we did in the previous construction as each entry in an array contains all input vectors that are orthogonal to the value of this entry in indices of the relevant part. However, instead of saving all input vectors that are orthogonal to α_i in the indices specified by W, we just pick all the input vectors that their elements in the indices specified by W are exactly the complements of the elements in α_i. All these vectors are saved in the list L_i in vertex v_i. Using these lists, we have the following easy observation:

▶ **Observation 1.** *Given a set $W \subseteq [c \log n]$ such that $|W| = k \log n$, the complete list of input vectors such that their values in the indices specified by W are orthogonal to some α_i can be recovered by concatenating all lists of vectors in the vertices that are reachable from vertex v_i in the query graph G.*

3.2 TopLevelsQueryGraph Algorithm

We start the preprocessing phase by constructing a query graph G. Now, following the last observation, instead of saving each vector many times in all the lists that their index is orthogonal to our query in the relevant indices (as suggested by the previous solution), we can save each vector in just one list and recover the original list by traversing G. We start the traversal from the vertex v_i such that the values of the query vector in the indices specified by W are equal to α_i. We can use any standard graph traversal algorithm to obtain all the input vectors that are orthogonal to the query vector in the indices specified by W. The number of vertices that we visit in the traversal of the query graph for a query vector q that have $k' \log n$ ones in the indices specified by W is $2^{k \log n - k' \log n} = n^{k-k'}$.

We can identify two types of nodes in the query graph. A node v_i that has an empty list L_i is considered a *black node*, otherwise it is considered a *white node*. We note that the number of white nodes is at most n and it can be $O(n)$ if the input vectors are split between many lists. In order to achieve a truly sublinear query time we would like the number of nodes we visit during the traversal in the query graph to be truly sublinear. Moreover, as the number of white nodes can be $\Theta(n)$ we need to make sure that the total number of white nodes we visit is truly sublinear even if we know how to avoid black nodes. As mentioned before, the number of nodes we visit during our traversal is $n^{k-k'}$ which is truly sublinear if we set $k - k' < 1$. This means that we need to handle queries that match some vertex v_i in the top levels of the query graph differently. For all vertices v_i in the x top levels of the graph we create a list L'_i of *all* input vectors that are orthogonal to α_i. Then, for each list L'_i we create a bitmap to quickly identify if there is a vector in the list that is orthogonal to our query. The size of each bitmap is n^{c-k}. The total number of bitmaps we create is $\tilde{O}(\binom{k \log n}{x \log n})$ for $x \leq k/2$ as the number of vectors in the jth level of the query graph is $\binom{k \log n}{j \log n}$ (we choose $j \log n$ positions for the ones in α_i out of $k \log n$ positions). Moreover, the number of layers is logarithmic in n. Thus, the total required space for handling the top layers of the query graph is $\tilde{O}(n^{c-k}\binom{k \log n}{x \log n})$. The binomial coefficient $\binom{k \log n}{x \log n}$ can be approximated by $n^{k \log k - x \log x - (k-x) \log (k-x)}$ using Stirling's approximation. So, the total space for the top layers is approximately $\tilde{O}(n^{c-k+k \log k - x \log x - (k-x) \log (k-x)})$.

Now, a query vector q that matches a vertex v_i in the x top levels can be answered in constant time by just looking at the proper entry in the bitmap of v_i. Otherwise, the number of vertices we need to traverse in the query graph will be at most n^{k-x} which is truly sublinear if $k - x < 1$. The problem is that the total number of vectors in the lists of these vertices can be $\theta(n)$. To overcome this problem, we change the way we handle any list L_i in the $(k - x) \log n$ bottom levels according to the number of elements in it. If the number of elements in the list is $O(n^{1-k+x})$ we do nothing - the elements are kept in the list with no special treatment. Otherwise, we save a bitmap over all the possibilities of the other bits in the query vector. The size of the bitmap, as before, is n^{c-k}. The number of lists that have more than $O(n^{1-k+x})$ elements is at most n^{k-x}. Therefore, the space for all the bitmaps of the long lists is n^{c-x}. We have that the total space of our data structure is $\tilde{O}(n^{c-k+k\log k-x\log x-(k-x)\log(k-x)}+n^{c-x})$. To obtain the best space complexity (while preserving the truly sublinear query time), we set k very close to 1.3 and x to 0.3. The space complexity of this solution using these values is approximately $\tilde{O}(n^{c-0.3})$. To conclude, we obtain the following result:

▶ **Theorem 4.** *The TopLevelsQueryGraph algorithm solves OV indexing with truly sublinear query time using approximately $\tilde{O}(n^{c-0.3})$ space.*

4 BottomLevelsQueryGraph: Third Space-Efficient Solution for OV indexing

We can use the query graph to obtain another solution to the OV indexing problem. This time we focus on the x bottom levels of the query graph. For each vertex v_i in the x bottom levels of the query graph we save a bitmap to quickly identify if there is an input vector such that (a) Its bits in the indices specified by W are the complements of α_i and (b) It is orthogonal to our query vector. The space we invest in these bitmaps is $\tilde{O}(n^{c-k}\binom{k\log n}{x\log n})$. Then, for every vertex v_i which is not in the x bottom levels of the query graph we save in its list L_i all the input vectors that are *orthogonal* to α_i, but *do not appear* in the any of the lists of the vertices in the x bottom. For every list L_i that its length is $\tilde{\theta}(n)$ we save a bitmap to get the answer in $\tilde{O}(1)$ time. Because we do not include in any list L_i vectors from the lists in the x bottom levels, we are guaranteed that each input vector appears in at most n^{k-x} lists. In our view of the query graph, this means that if an input vector appears in the list L_i of some vertex v_i it will be duplicated in the lists of all vertices that v_i is reachable from them. Consequently, the total number of vectors in all lists above the x bottom levels is at most n^{1+k-x}. Therefore, the number of bitmaps we will save for lists of size $\tilde{\theta}(n)$ is at most n^{k-x}. Each bitmap is of n^{c-k} space, so the size of all bitmaps is n^{c-x}. The total size of the data structure is again $\tilde{O}(n^{c-k+k\log k-x\log x-(k-x)\log(k-x)}+n^{c-x})$.

Upon receiving a query vector q, if it matches a vertex in one of the x bottom levels, we immediately get the answer by looking at the right entry in the bitmap in that vertex. Otherwise, we need to look not just at the bitmap of the vertex that matches our query, but rather we have to go over all the vertices v_i in the $(k-x)\log n$ level (the top level of the x bottom levels) such that α_i is orthogonal to q in the positions specified by W. In all these vertices we check in their bitmap if there is an input vector that is orthogonal to q. If $k-x<1$ we ensure that the query time is sublinear in n. All in all, we obtain a solution that has the same query time and space complexities as the previous one using a different approach, as summarized in the following theorem:

▶ **Theorem 5.** *The BottomLevelsQueryGraph algorithm solves OV indexing with truly sublinear query time using approximately $\tilde{O}(n^{c-0.3})$ space.*

5 Space and Query Time Tradeoff for Solving OV indexing

In all the solutions we presented so far we tried to minimize the space usage and still achieve a sublinear query time. However, obtaining a tradeoff between the space and query time would be of utmost interest. We know how to obtain constant query time by using n^c space. But can we obtain, for example, $O(\sqrt{n})$ query time using just $n^{c-\epsilon}$ space for some $\epsilon>0$? In the first method we have suggested there is an inherent problem to achieve this as all lists can have more than $O(\sqrt{n})$ vectors. In the second and third solutions we can improve the query time by choosing larger x. However, as x becomes $k/2$ the space of the data structure becomes $\tilde{O}(n^c)$. The following theorem demonstrates how to obtain any polynomial query time while consuming $O(n^{c-\gamma})$ space for some $\gamma > 0$.

▶ **Theorem 6.** *For any $\epsilon > 0$ there is a solution to OV indexing that its query time is $O(n^\epsilon)$ and the space complexity is $O(n^{c-\gamma})$ for $\gamma > 0$.*

Proof. The idea is to combine the second and third solutions. We can save bitmaps for both the x top levels and the x bottom levels of the query graph using $\tilde{O}(n^{c-k}\binom{k \log n}{x \log n})$ space. Then, for every vertex that is not in the x top or bottom levels we do the same as in the second solution - save a bitmap for every node whose list is of length $\theta(n^\delta)$ or more for some $\delta > 0$. The total cost of these bitmaps is $\tilde{O}(n^{c-k+1-\delta})$. When we get a query vector q that matches a vertex v_i in our query graph. If v_i is on the x top or bottom levels, we just check the right entry in the bitmap of v_i. Otherwise, we start a traversal from v_i to all the vertices that are reachable from it except those in the x bottom levels. The number of vertices we visit is at most $\binom{(k-x)\log n}{x \log n}$ if $k/3 < x$. This is approximately $\tilde{O}(n^{(k-x)\log k-x-x\log x-(k-2x)\log(k-2x)})$. It is easy to verify that as x gets close to $k/2$ the exponent of this expression is very close to 0. Therefore, the total query time is $\tilde{O}(n^{\delta+(k-x)\log(k-x)-x\log x-(k-2x)\log(k-2x)})$ as the query time in each vertex we visit is at most n^δ. By choosing suitable value of $k \geq 1$, $x < k/2$ and $\delta > 0$, we can obtain a query time of $\tilde{O}(n^\epsilon)$ for any $\epsilon > 0$ using a data structure that consumes $\tilde{O}(n^{c-\gamma})$ space for some constant $\gamma > 0$. ◀

6 The Reporting Version of OV indexing

In the reporting version of OV indexing, given a query vector q we are required not just to decide if there is a vector in S that is orthogonal to q, but rather we are required to report all input vectors in S that are orthogonal to q.

To solve this version we can use the same methods as we have described for the decision version. However, the only part of these solutions that does not support reporting is the use of bitmaps. Using a bitmap we can answer the query quickly if there is an input vector that is orthogonal to our query vector, but we are unable to discover the list of input vectors that are orthogonal to the query if there are such vectors. The following lemma demonstrates how to construct a data structure that uses almost the same space as a bitmap, but supports efficient reporting.

▶ **Lemma 7.** *Given n $c\log n$-length boolean vectors, there is a data structure that uses $\tilde{O}(n^c)$ preprocessing time and upon receiving a query vector v report on all t input vectors that are orthogonal to v in time $O(t \log n)$*

6.1 Improving The Query Time

We can remove the dependency on n in the query time as shown by the following theorem[1].

▶ **Theorem 8.** *Given n $c\log n$-length boolean vectors, there is a data structure that uses $\tilde{O}(n^c)$ space and upon receiving a query vector v report on all t input vectors that are orthogonal to v in $O(t)$ time.*

[1] All missing proofs appear in: https://arxiv.org/abs/1710.00586

We can plug in the data structure from the previous theorem into any of the three solutions for OV indexing and get solutions for the reporting version of OV indexing that have the same space usage (up to logarithmic factors) and just an additive $O(t)$ to the query time.

7 Reducing OV indexing to SetDisjointness

In this section we present a connection between OV indexing and the problem of SetDisjointness. In the problem of SetDisjointness we are given m sets $S_1, S_2, ..., S_m$ such that the total number of elements in all sets is N and after preprocessing them we need to answer queries of the following form: given a pair of indices (i, j), decide whether $S_i \cap S_j$ is empty or not. The problem can be generalized to k-SetDisjointness in which we are given as a query a k-tuple $(i_1, i_2, ..., i_k)$ and we are required to answer if the intersection $S_{i_1} \cap S_{i_2} \cap ... \cap S_{i_k}$ is empty or not. The SetDisjointness problem was the first problem used to show conditional lower bounds on space complexity(see [12, 24, 15, 23]). Therefore, it should be interesting to see the connection between our OV indexing problem and the fundamental problem of SetDisjointness. Other problems connected to SetDisjointness are discussed in [17]. Currently, the best known space-query time tradeoff for k-SetDisjointness is $S \times T^k = O(N^k)$, where S is the space complexity and T is the query time [11, 17].

We begin by presenting a simple reduction from OV indexing to k-SetDisjointness for $k = c \log n$. Given an instance of OV indexing with n $c \log n$-length boolean input vectors we can create an instance of k-SetDisjointness in the following way. We create $c \log n$ sets. The set S_i contains all the vectors that have 0 in their ith element. Then, given a query vector q that has ones in the elements whose indices are $(i_1, i_2, ..., i_k)$ all that we need in order to answer this query is to verify if the intersection $S_{i_1} \cap S_{i_2} \cap ... \cap S_{i_k}$ is empty or not. If it is empty then we know that there is no input vector that has zeroes in all the position of the ones in q, which means that no input vector is orthogonal to q. Otherwise, there is an input vector which is orthogonal to q.

We would like to show this reduction to other values of k, especially small and constant. The idea is to use the first solution that we have suggested to obtain the following result:

▶ **Theorem 9.** *There is a reduction from OV indexing to k-SetDisjointness that can be used to solve OV indexing with truly sublinear query time and $O(n^{c-\gamma})$ space for some $0 < \gamma < 1$.*

8 OV indexing for Random Input

The solution to OV indexing that we have described in Section 3 is limited by the tradeoff between the bitmaps for the lists in the top levels of the query graphs (lists in vertices v_i such that α_i has a small number of ones) and the bitmaps for long lists in the bottom levels of the query graph. Therefore, we may improve the solution by making the lists in the bottom levels short, as for short lists we only save the elements themselves. We also note that we can also benefit from making the lists in the bottom levels very long, since their number is small. Consequently, the costly lists are those that are not too short and not too long.

In the solution we have presented in Section 3, we pick a set W of the indices for the query graph. Our solution works for any choice of W, but the question is whether there is a choice of W that will make the list shorter or longer, so we can utilize it for a more compact solution to OV indexing. In the following lemma we show that for random input vectors that are uniformly distributed the probability for choosing W such that there are lists that are not short is small.

▶ **Lemma 10.** *The size of every list L_i in the query graph of the second solution of OV indexing is at most logarithmic w.h.p. for any choice of W (the set of $k \log n$ indices) on random input vectors, where $k \geq 1$.*

The last lemma guarantees that on random input vectors for any choice of W the length of all lists in the query graph are supposed to be of length at most $4c \log n$ w.h.p. Therefore, instead of saving bitmaps for both lists in the top levels of the query graph and long lists in the bottom levels of the query graph, we need to save bitmaps just for the former as the latter do not exist. Consequently, for $c \log n$-length input vectors we just create the query graph with nodes representing $\log n$-length vectors and save bitmaps for the top $\delta \log n$ levels, for some $\delta > 0$. The space required by these bitmaps is $n^{c-1+\epsilon}$, for some $\epsilon > 0$ that can be as small as possible by choosing appropriate small value for δ. To conclude, we have obtained the following result:

▶ **Theorem 11.** *OV indexing on random input vectors can be solved in expected truly sublinear query time using $O(n^{c-1+\epsilon})$ space, for any $\epsilon > 0$.*

This improved space complexity for random input makes it tempting to think that the same property holds even for worst case input. More specifically, it is enough to have just one W that will map all input vectors to short or long lists. It turns out that for worst case input this cannot be achieved. In the following lemma we show how to create a worst case input vectors such that many lists in the query graph are neither too short nor too long.

▶ **Lemma 12.** *There exist n $c \log n$-length input vectors such that for all $W \subseteq [k \log n]$ there are $\Theta(n)$ vectors that are mapped by h_W to lists of size between $n^{1/6}$ and $n^{2/3}$*

In the last lemma we can obtain values other than $n^{1/6}$ and $n^{2/3}$ by changing the basic block size from $0.5 \log n$ to some other $r \log n$ for $r > 0$.

This demonstrates that for worst-case input vectors, as opposed to random input vectors, there can be $O(n)$ vectors that are mapped to lists that are neither too short nor too long. The exact size can be controlled by proper choices of block and group size.

9 Further Research

In this paper we presented several algorithms to solve OV indexing that obtain truly sublinear query time and require $O(n^{c-\gamma})$ space for some constant $0 < \gamma < 1$. For random input vectors we demonstrated in Section 8 how to obtain sublinear query time solution to OV indexing using $O(n^{c-\gamma})$ for *any* $0 < \gamma < 1$. We note that the preprocessing time of all algorithms is polynomial in n.

The main question regarding OV indexing, following this paper, is can one obtain a sublinear query time solution to OV indexing that requires only $O(n^{c-1})$ space. This question is interesting even if we allow an unlimited preprocessing time. We conjecture that there is no such solution to OV indexing:

▶ **Conjecture 2.** *There is no truly sublinear query time solution to OV indexing that requires only $O(n^{c-1})$ space.*

Even if that conjecture is false, it is of utmost interest to find the exact lower bound on the space requirements of OV indexing for both unlimited and polynomial preprocessing time. Finding the exact space requirements can be used to obtain conditional lower bounds on the *space* complexity of many problems known to be conditionally hard in terms of time based on OV.

References

1 Amir Abboud, Arturs Backurs, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Or Zamir. Subtree isomorphism revisited. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1256–1271, 2016.

2 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78, 2015.

3 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.

4 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443, 2014.

5 Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 377–391, 2016.

6 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.

7 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 51–58, 2015.

8 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 661–670, 2014.

9 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 79–97, 2015.

10 Moses Charikar, Piotr Indyk, and Rina Panigrahy. New algorithms for subset query, partial match, orthogonal range searching, and related problems. In *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, pages 451–462, 2002.

11 Hagai Cohen and Ely Porat. Fast set intersection and two-patterns matching. *Theor. Comput. Sci.*, 411(40-42):3795–3800, 2010.

12 Hagai Cohen and Ely Porat. On the hardness of distance oracle for sparse graph. *CoRR*, abs/1006.1117, 2010.

13 Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and don't cares. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 91–100, 2004.

14 Marek Cygan, Marcin Pilipczuk, and Michal Pilipczuk. Known algorithms for EDGE CLIQUE COVER are probably optimal. In *Proceedings of the Twenty-Fourth Annual*

ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013, pages 1044–1053, 2013.

15 Pooya Davoodi, Michiel H. M. Smid, and Freek van Walderveen. Two-dimensional range diameter queries. In *LATIN 2012: Theoretical Informatics - 10th Latin American Symposium, Arequipa, Peru, April 16-20, 2012. Proceedings*, pages 219–230, 2012.

16 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM J. Comput.*, 43(5):1541–1563, 2014.

17 Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. Conditional lower bounds for space/time tradeoffs. In *To appear in WADS 2017*, 2017.

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

19 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

20 Kasper Green Larsen and R. Ryan Williams. Faster online matrix-vector multiplication. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2182–2189, 2017.

21 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011.

22 Daniel Moeller, Ramamohan Paturi, and Stefan Schneider. Subquadratic algorithms for succinct stable matching. In *Computer Science - Theory and Applications - 11th International Computer Science Symposium in Russia, CSR 2016, St. Petersburg, Russia, June 9-13, 2016, Proceedings*, pages 294–308, 2016.

23 Mihai Patrascu and Liam Roditty. Distance oracles beyond the thorup-zwick bound. *SIAM J. Comput.*, 43(1):300–311, 2014.

24 Mihai Patrascu, Liam Roditty, and Mikkel Thorup. A new infinity of distance oracles for sparse graphs. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 738–747, 2012.

25 Mihai Patrascu and Ryan Williams. On the possibility of faster SAT algorithms. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1065–1075, 2010.

26 Ronald L. Rivest. *Analysis of Associative Retrieval Algorithm.* PhD thesis, Stanford University, 1974.

27 Ronald L. Rivest. Partial-match retrieval algorithms. *SIAM J. Comput.*, 5(1):19–50, 1976.

28 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 515–524, 2013.

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

30 Ryan Williams and Huacheng Yu. Finding orthogonal vectors in discrete structures. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1867–1877, 2014.