Optimal sequencing by hybridization in rounds

Alan M. Frieze* Bjarni V. Halldórsson†
Department of Mathematical Sciences,
Carnegie Mellon University,
Pittsburgh, PA 15213,
USA

June 19, 2001

Abstract

Sequencing by hybridization (SBH) is a method for reconstructing a sequence over a small finite alphabet from a collection of probes (substrings). Substring queries can be arranged on an array (SBH chip) and then a combinatorial method is used to construct the sequence from its collection of probes. Technological constraints limit the number of substring queries that can be placed on a single SBH chip. We develop an idea of Margaritis and Skiena and propose an algorithm that uses a series of small SBH chips to sequence long strings while the number of probes used matches the information theoretical lower bound up to a constant factor.

DNA sequencing, Sequencing by Hybridization, Probabilistic Analysis.

*alan@random.math.cmu.edu, T: 412-268-8476, Fax:412-268-6380
†corresponding author: bjarni@cmu.edu, T: 412-268-5190, Fax:412-268-6380

1 Introduction

Consider the following problem. Let Σ be an alphabet with α letters. Given a string s drawn uniformly at random from Σ^n and the ability to ask queries of the type: "Is x a substring of s?". What is the minimum set of such questions one can ask such that with high probability one can reconstruct s.

The problem is an abstraction of a problem that occurs in the sequencing of DNA molecules. DNA strands can be seen as sequences drawn from the four letter alphabet of nucleotides $\{A, C, G, T\}$.

Sequencing by hybridization (SBH) (Bains and Smith, 1988; Drmanac et al., 1989; Lysov et al., 1998) has been proposed as an alternative to the traditional Gilbert-Sanger method of sequencing by gel electrophoresis; The surveys (Chetverin and Cramer, 1994; Pevzner and Lipshutz, 1994) give an overview of both the technological and algorithmic aspects of the method. The method applies the complementary Watson-Crick base pairing of DNA molecules. A given single stranded DNA molecule will hybridize with its complement strand. SBH is based on the use of a chip, fabricated using photolithographic techniques. The active area of the chip is structured as a matrix, each region of which is assigned to a specific oligonucleotide, biochemically attached to the chip surface. A solution of fluorescently tagged target DNA fragments are exposed to the chip. These fragments hybridize with their complementary fragment on the chip and the hybridized fragments can be identified using a fluorescence detector. Each hybridization (or lack thereof) determines whether the fragment is or is not a substring of the target string. In our formulation we will assume that the the hybridization chips can give us an answer to a ternary query: whether a string does not occur, occurs once or occurs more than once.

The classical sequencing chip design $C(m)$, contains all α^m single stranded oligonucleotides of some fixed length m. Pevzner's algorithm (Pevzner, 1989) for reconstruction using classical sequencing chips interprets the results of a sequencing experiment as a subgraph of the *DeBruijn graph*, such that any Eulerian path corresponds to a possible sequence. The reconstruction therefore is not unique unless the Eulerian path is unique.

Examples can be found that show that in order to uniquely reconstruct all members of Σ^n using a classical sequencing chip, $C(m)$, m needs to be greater than $\frac{n}{2}$ (see (Skiena and Sundaram, 1995)). Pevzner et al. (1991) show experimentally that the classical $C(8)$ chip which contains 65.536 oligonucleotides suffices to reconstruct 200 nucleotide sequences in only 94 out of 100 cases.

Dyer et al. (1994) and Arratia et al. (1996) have shown independently that for $C(m)$ to be effective on random strings of length n, m needs to be chosen greater than $2\log_\alpha n$. In other words, for there to be a constant probability that we can reconstruct a string of length n drawn uniformly at random from Σ^n using a classical hybridization chip, the chip must contain at least n^2 substrings. This compares to the information theoretical lower bound on the number of ternary queries needed to distinguish between α^n elements of $\Omega(n)$.

A variety of different methods have been suggested to overcome these negative results for the classical SBH chips. Using the assumption that *universal DNA bases* can be synthesized Preparata et al. (Preparata et al., 1999; Preparata and Upfal, 2000) give a scheme for which the size of the chip is optimal i.e. $O(n)$. Broude et al. (1994) suggest generating positional information along with the hybridization information (PSBH). PSBH was analyzed algorithmically by Hannenhalli et al. (Hannenhalli et al., 1996) which show NP-hardness of the general problem and give an efficient algorithm if the positional information is only off by a constant. This model was further analyzed by Ben-Dor et al. (Ben-Dor et al., 1999). Drmanac et al. (1989) suggested sequencing large sequences by obtaining spectra of many overlapping fragments. This model was analyzed algorithmically by Arratia et al. (1996) giving bounds on the probability of unique reconstruction. Shamir and Tsur (2001) recently improved on the analysis of Arratia et al. (1996) and furthermore gave an algorithm for the case when false negative errors occurred in the hybridization.

Sequencing by hybridization in rounds or interactive sequencing by hybridization (ISBH) was first considered by Margaritis and Skiena (1995). The assumption here is that the sequencing queries can be done adaptively and once the results of one hybridization round are known a new chip can be constructed. In their original paper Margaritis and Skiena give a number of upper and lower bounds on the number of rounds needed dependent on the number of probes allowed in each round. Among their results was an algorithm that reconstructs a sequence with high probability using $O(\log n)$ chips each containing $O(n)$ queries. The main result of this paper improves on this result. A few other papers have been written on ISBH. Skiena and Sundaram (1995) showed that if only a single query could be asked in each round $\frac{n\alpha}{4}$ rounds are necessary and $(\alpha - 1)n$ rounds are sufficient for string reconstruction. Kruglyak (Kruglyak, 1998) gave an algorithm with a worst case performance guarantee which shows that $O(\log n)$ rounds are sufficient

if n^2 queries are placed on a chip in each round.

The following theorem is the main result of this paper.

Theorem 1.1 *With high probability a string s drawn uniformly at random from Σ^n can be reconstructed by sequentially using seven hybridization chips each containing $O(n)$ substring queries.*

Notice that this result is optimal for the number of queries in the information theoretical sense, up to a constant multiple. Our algorithm proceeds in the following manner. In its initial step we ask substring queries corresponding to the classical SBH chip. We then construct the DeBruijn graph in the way suggested by Pevzner. We then proceed to ask targeted queries in order to unravel the string.

The main result of this paper is mathematical, although it may eventually have some practical relevance. Sequencing chips similar to the classical chips are already in production by Hyseq Inc., which holds several patents on the procedure (Drmanac, Crkvenjakov, 1993). These chips have been used for successfully for De-Novo sequencing (Drmanac et al., 1993) (sequencing when sequence is unknown). Given that many organisms have been sequenced another problem of practical importance is resequencing by hybridization (Drmanac et al., 1989; Pe'er and Shamir, 2000), in this problem a template sequence is known and the goal of the sequencing is to determine the specific mutational variants of the sequence. Machines for producing oligonucleotide arrays using ink-jet printer technology have been pioneered by Blanchard et al. (1996) and are currently being manufactured by Agilent Technologies. This technology may prove to be particularly useful for interactive sequencing by hybridization. For a review of different technologies for DNA array manufacturing see (Blanchard, 1998; Schena, 1999). Other relevant technologies include Affymetrix type arrays (Lockhart et al., 1996; Fodor et al., 1991) and Southern Array Makers developed by Oxford Gene Technologies (Southern, 1996). We note that technological constraints need to be considered before a practical implementation of the method developed in this paper. In particular, the more realistic case when false positive and negative errors occur in the experiments needs to be considered.

The organization of the paper is as follows. In the next section we will give overview of previous work. This will motivate our algorithm and we will give a simplified version of it. In Section 3 we will give our complete algorithm and verify its correctness. In Section 4 we will prove our complexity result and in Section 5 we demonstrate our computational experience.

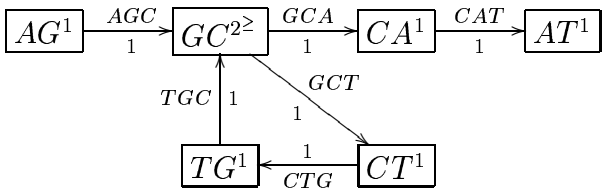

Figure 1: The DeBruijn-graph constructed if the the substrings of s of length 3 are AGC, GCT, TGC, CTG, GCA and CAT and each one occurs once.

2 Motivation and basic algorithm

2.1 DeBruijn graphs

In this section we review the DeBruijn graph construction first considered by Pevzner (89) Let s be our unknown target string. Given the answer of a ternary query for all strings r of length m, whether r occurs once in s, more than once, or not at all, we can construct an associated edge-labeled digraph, D_m^s, in the following manner. The vertex set of D_m^s consists of $[\alpha]^{m-1}$. There is no edge from $x_1x_2\ldots x_{m-1}$ to $y_1y_2\ldots y_{m-1}$ unless $x_{i+1} = y_i$ for $i \in \{1, 2, \ldots, m-2\}$ in which case the edge is labeled 1 if $x_1x_2\ldots x_{m-1}y_{m-1}$ occurs once and $2^{\geq}$ if $x_1x_2\ldots x_{m-1}y_{m-1}$ occurs more than once in s. In what follows we will call this the DeBruijn graph of s. Figure 1 shows the construction of a DeBruijn graph.

We will also label the nodes of D_m^s, a node x will be labeled 0 if it has no in- or out-edges, it will be labeled 1 if it has at most one in-edge and at most one out-edge both labeled 1, and labeled $2^{\geq}$ otherwise. Let $\lambda(x)$ denote the label of a node/edge x.

We note that there is a unique path in D_m^s for any substring $x = x_1x_2\ldots x_k$ of s where $k \geq m-1$, namely the path starting at $x_1x_2\ldots x_{m-1}$ and ending at $x_{k-m+2}x_{k-m+3}\ldots x_k$ that traverses all of the edges $x_ix_{i+1}\ldots x_{i+m-1}$ $\forall i \in \{1\ldots k-m+1\}$. We will denote this path by $\mathcal{P}(x)$ and refer to it as the path in D_m^s *corresponding* to x. In the special case where k is $m-1$ or m we will refer to it as the node/edge *corresponding* to x.

Pevzner (89) showed that s *corresponds* to an Eulerian path in this graph. Where we define an Eulerian path in this graph to be any walk that traverses the edges with label 1 exactly once and the edges with label $2^{\geq}$ at least twice. From the graph in Figure 1 we can tell that the original string s is AGCTGCAT.

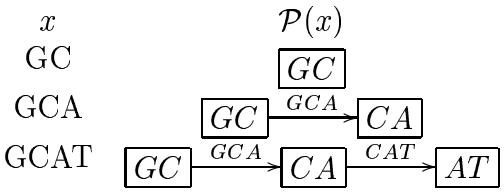

Figure 2: Examples of the mapping $\mathcal{P}(x)$, when $m = 3$.

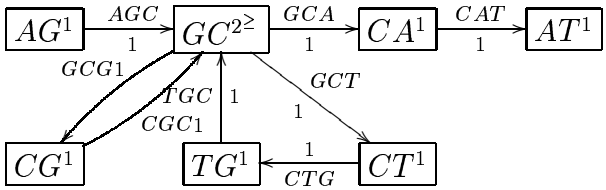

Figure 3: The DeBruijn-graph constructed if the substrings of s of length 3 are AGC, GCT, CGC, GCG, TGC, CTG, GCA and CAT and each one occurs once.

The DeBruijn graph may however have more than one Eulerian path. In this case the construction of string s is ambiguous (see Figure 3).

2.2 Simplified algorithm

Our algorithm proceeds by constructing the DeBruijn graph for all substrings of some fixed length m. We then use information from that graph to construct a set of substring queries that enable us to determine all substrings of s of length m', where m' is some number larger than m. We then iterate this process noticing that the probability that D_m^s is a path increases with m. I.e. we will attempt to elongate the strings *corresponding* to the nodes in the DeBruijn graph.

To motivate our algorithm let us look at where the ambiguities are in this elongation process. Notice that if a node x in the DeBruijn graph has label 1, then each of the strings *corresponding* to the in- and out-edges occurs only once in s. The elongation of the string *corresponding* to the in-edge of x is hence unambiguous and can be determined by appending to it the last character of the string corresponding to the out-edge. For example from the graph $CATA \xrightarrow{CATAG} ATAG \xrightarrow{ATAGT} TAGT$ we know that the string

CATAGT occurs in s. However if a node x has more than one in- or out-edge we need to pair the in-edges with the out-edges.

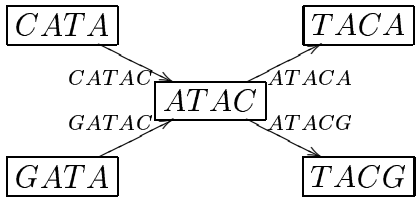

Figure 4: A node with two in- and out-edges

From the graph in Figure 4 we can tell that two of the strings CATACA, CATACG, GACACA, GACACG occur in s but not which ones. To determine all substrings of length six we would ask substring queries for each of these strings. The central question in the remainder of this paper is to determine conditions such that the number of queries generated in this way is not too large.

If each of the edges in this graph was labeled 1 they would have a unique elongation. To determine the elongation of the string CATAC by k characters it would be sufficient to determine whether CATAC elongates to CATACA or CATACG and then determine the elongation of ATACA or ATACG by $k-1$ characters.

However if CATAC elongates to CATACG and ATACG occurs two or more times in s then ATACG will have two elongations of length $k-1$ and we cannot determine which is the elongation of CATAC. We see that to determine the elongation of a particular edge e by k characters it is sufficient to determine all paths from e that either (1) have length k or (2) are shorter and end at an edge of multiplicity one.

This motivates the definition of a *cluster*, a collection of nodes and edges that all have label $2^{\geq}$. Only when the Eulerian path passes through those nodes and edges that have label $2^{\geq}$ is the determination of the string ambiguous. The set of *clusters* in the graph is the set of ambiguous parts of the graph.

Definition 2.1 *The* cluster *containing x, $Cl(x)$, is the maximal connected subgraph of D_m^s containing x such that $\lambda(y) = 2^{\geq}$ for all nodes and edges $y \in Cl(x)$.*

Our task is to determine s which can be thought of as determining an Eulerian path from the *start* node of D_m^s to the *end* node of D_m^s. Notice that any internal node or edge labeled 1 has a unique occurrence in s, and will therefore have a unique elongation. Assuming that we know which nodes are the *start* node and the *end*, node we can reconstruct s by determining a path from the *start* node to an edge labeled 1. We can then determine the continuation of the Eulerian path from that edge by its unique elongation to either another edge of label 1 or to the *end* node.

This motivates the following algorithm for reconstructing s. Let c be a positive constant and Q be the set of queries, to be placed on the DNA chip.

Algorithm 1

Step 1. *Classical SBH chip.*
$Q = \{x_1 x_2 \dots x_m \mid x_i \in \Sigma, m = \lceil \log_\alpha n \rceil + c\}$
`Ask the queries in` Q `and construct` D_m^s.

Step 2. *Resolve ambiguities.*
`Let` $C_m^s = \{\text{nodes } v \in D_m^s \mid \lambda(v) = 2^{\geq}\}$
While $C_m^s \neq \emptyset$ `choose a node` x `from` C_m^s
`Let` $C \leftarrow Cl(x)$. $C_m^s \leftarrow C_m^s - C$.
`Let` Q `be the set of strings` $\{x_1 \dots x_{m+k}\}$ `where`
i) $\mathcal{P}(x_1 \dots x_{m-1})$ `=` ***start*** `or` $\lambda(x_1 \dots x_m) = 1$.
ii) $\lambda(x_i \dots x_{i+m-1}) = 2^{\geq}, \ \forall i \in \{2 \dots k\}$.
ii) $\mathcal{P}(x_i \dots x_{i+m-2}) \in C, \ \forall i \in \{2 \dots k+1\}$.
iv) $\mathcal{P}(x_{k+2} \dots x_{m+k})$ `=` ***end***
`or` $\lambda(x_{k+1} \dots x_m) = 1$.

Step 3. `Reconstruct` s `from the DeBruijn graph and the answers to the queries` Q.

2.3 Potential pitfalls

Let us look at the complications we may face in the analysis of the algorithm. If there is a cycle in the DeBruijn graph we cannot determine whether a given string passes through the cycle or past it and we will add queries for both possible strings. This may cause us to ask a large number of queries for each such cycle.

As an example of this in Figure 5 we have a loop in AAA and the edge from AAA to AAC has label $2^{\geq}$. We cannot determine from the graph

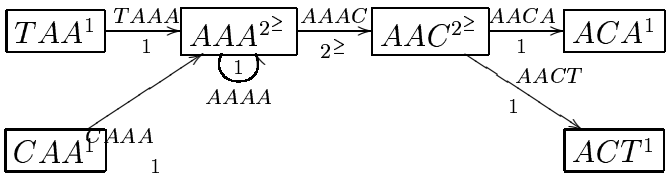

Figure 5: Example of a *cluster* along with its in and out edges, the *cluster* being those nodes and edges having label $2^{\geq}$.

which two of the strings TAAACA, TAAACT, TAAAACA, TAAAACT, CAAACA, CAAACT, CAAAACA or CAAAACT occur in s. The algorithm will add the queries TAAAA, CAAAA, TAAACA, TAAACT, CAAACA, TAAACA, AAAACA, AAAACT. As the edge AAAA occurs only once in the graph we can first determine whether TAAAA or CAAAA occurs in s and then determine the occurrence of TAAAACA, TAAAACT, CAAAACA and TAAAACT from which of the strings AAAACA or AAAACT occurs in s.

If the *cluster* contains no cycles the number of queries generated by this algorithm will grow as the number of in-edges times the number of out-edges of the *cluster*. If the *cluster* contains cycles we may not be able to determine how often a given path traverses the cycle. If it contains multiple cycles the same holds true for each one of them, the number of queries generated by the algorithm may therefore grow exponentially with the number of cycles in the *cluster*. Notice that the occurrence in s of a string *corresponding* to a node in the graph is highly correlated with the occurrence of the strings *corresponding* to its neighbors. This interdependence makes the algorithm difficult to analyze. Complex *clusters* that require a large number of queries have a reasonable probability of occurring and the average number of queries generated by the algorithm may in fact be large.

3 Modified Algorithm

We modify the previous algorithm so that we only make a limited number of queries initiating at any given node in the graph. Using two rounds of queries we may hence not be able to determine s, but we will show that with high probability seven rounds will be sufficient. We will use the following modification of **Step 2**. Notice also that this modified version doesn't assume

prior knowledge of the *start* and *end* nodes as we will add queries starting at any node in the *cluster* and terminating at any node. Let $k_0 = \lceil \frac{1}{5} \log_\alpha n \rceil$.

Step $2'$ **Repeat** `six times`
  `Let` Q `be the set of strings` $\{x_1 \dots x_{m+k}\}$ `where`
    `i)` $k \in \{1, \dots, k_0\}$
    `ii)` $\mathcal{P}(x_1 \dots x_{m+k}) \in D_m^s$
    `iii)` $\lambda(x_i \dots x_{m+i-2}) = 2^{\geq}, \forall i \in \{2 \dots k-1\}$
    `iv)` $\lambda(x_i \dots x_{m+i-1}) = 2^{\geq}, \forall i \in \{2 \dots k-1\}$
  `Ask the queries in` Q.
  $m \leftarrow m + k_0$. `Construct` D_m^s.

3.1 Correctness of algorithm

Let us clarify the statement of Theorem 1.1.

Definition 3.1 *We say that an event occurs with high probability (whp) if it occurs with probability* $1 - o(1)$ *as* $n \to \infty$.

Lemma 3.1 *The number of substring queries generated by an algorithm satisfying the conditions of Theorem 1.1 is optimal in the information theoretical sense, up to a constant multiple.*

Proof: There are α^n strings of length n and for there to be high probability that we can sequence all the strings we must be able to distinguish between $(1 - o(1))\alpha^n$ strings. There are 3^m possible answers to m ternary queries. Therefore $3^m \geq (1 - o(1))\alpha^n$ or $m \geq \Omega(n)$. Our algorithm generates $O(1) \cdot O(n) = O(n)$ queries. □

We now verify that the algorithm is correct, i.e. whp it reconstructs s.

Lemma 3.2

(a) *A single iteration of* **Step** $2'$ *on* D_m^s *will allow us to construct* $D_{m+k_0}^s$.

(b) *After applying* **Step** $2'$ *we can whp reconstruct* s.

Proof: (a) For each substring x of length m we add a query for all possible elongations either of length k_0 or from x to a string y that has multiplicity one, in which case the elongation of x can be determined from the unique elongation of y. **(b)** follows from the result in Arratia et al. (1996) and Dyer et al. (1994) that whp the DeBruijn graph D_m^s, $m \geq \frac{11}{5} \log_\alpha n$, is a path for random $s \in \Sigma^n$. □

4 Complexity Analysis

We now proceed to estimate the expected number of queries in each iteration. The main goal of this section is to prove the following lemma.

Lemma 4.1 *The expected number of queries in Q generated by a single iteration of* **Step** $2'$ *is $O(n)$.*

We will start by defining *normal* nodes in Section 4.1. We will then show that the queries generated originating at a *normal* node form a tree. In Section 4.2, we will bound the number of such trees. In Section 4.3, we will consider the relationship between trees and substrings of s. In Section 4.4, we summarize and upper bound the expected number of queries generated originating at *normal* strings. In Section 4.5 we show that it is rare that a node is not *normal*, and hence prove Lemma 4.1. Finally, in Section 4.6, we show concentration of the expectation of the number of queries.

4.1 *Normal* substrings

Definition 4.1 *For every node $x \in D_m^s$ we define a subgraph L_x of D_m^s. Its edge and vertex sets are the sets of edges and vertices reachable from x by a path $x^1x^2 \dots x^k$ where $x = x^1$, $k \leq k_0$, $\lambda(e) = 2^{\geq}$ if $e = (x^i, x^{i+1}), i \in \{2, \dots, k-2\}$ and $\lambda(x^i) = 2^{\geq}, i \in \{2, \dots, k-1\}$.*

We say that x and L_x are normal *if L_x is a tree that does not contain* end *and all substrings y of s that $\mathcal{P}$ maximally maps[1] to L_x occur disjointly in s.*

Notice that the definition of *normal* refers to substrings of s, i.e. a node will be *normal* depending on the string s. In the example in Figure 5 L_{TAA} is the graph shown except for the node CAA and edge CAAA. L_{TAA} is not *normal* since it contains a cycle. Figure 6 shows L_{AAC}.

4.2 Counting the number of trees

We will now bound the expected number of queries in Q whose initial string is x if L_x is *normal*. Notice that in this case the algorithm will in **Step** $2'$ generate one query with initial string x for every node in L_x, other than x

[1] I.e. the collection $\{y \prec s \mid \mathcal{P}(y) \in L_x, \mathcal{P}(z) \notin L_x, \forall z : y \prec z\}$, where $\prec$ denotes substring relation.

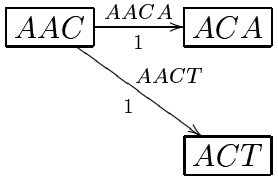

Figure 6: L_{AAC} for example in Figure 5. We consider L_{AAC} to be *normal* if s is of the form *AACA*AACT*, where * denotes any string and not *normal* if s is of the form *AACAACT* since here the two strings overlap.

and its children. Also notice (by Definition 4.1) when L_x is *normal* it does not contain a cycle and must therefore be a tree.

Definition 4.2 *We say that an α-ary tree (each node has at most α children) is a (b, i, l)-tree if it has b branching nodes (nodes with more than one child), i single child nodes and l leaves. The children of a node will be considered to be ordered and we will make a distinction between two children of a node based on their ordering.*

We can now count the expected number of queries whose initial string is x by counting the number of different (b, i, l)-trees and then in Section 4.3 estimating the probability that L_x is such a tree.

Lemma 4.2 *The number of distinct (b, i, l)-trees is at most*

$$\frac{1}{(\alpha-1)k+1}\binom{\alpha k}{k}\alpha^i\binom{i+k-1}{i},$$

where $k = b + l$.

Proof: The number of trees with b branching nodes and l leaves is at most $\frac{1}{(\alpha-1)k+1}\binom{\alpha k}{k}$, where $k = b + l$, since it is less than the number of α-ary trees of size k (see (Knuth, 1968), Ex. 2.3.4.4.11). We now insert the internal non-branching nodes into the tree by subdividing one of the existing edges in the tree or adding a new single child root node. The choice of where to put the internal nodes can be done in $\binom{i+k-1}{i}$ ways and the out-edges of the internal nodes can be chosen in α^i ways. □

4.3 From trees to strings

We will now estimate the probability that L_x takes the form of a tree T, and multiply by the number of queries generated if L_x takes this form. To avoid significant over-counting of the number of queries with initial string x, we count only the number of queries terminating at leaf nodes of T; The queries with initial string x that terminate at internal nodes of T can then be counted when estimating the number of queries generated when L_x takes the form of one of the subtrees of T.

We will now define a partial ordering of trees. This partial ordering ensures that we count all the queries to the internal nodes as well.

Definition 4.3 *A subtree T' of a labeled tree T, is an* incubating subtree *if, for each node v in T, either all or none of the children of v occur in T'.*

Note that by this definition all children of a given node must be removed at the same time. Rephrasing, T descends from T' through a series of *incubation* operations where all children of any given node appear in the same operation. For the purpose of our proof, the important thing to note here is that each node in T is a leaf node of some *incubating subtree* of T.

The following observation is immediate from Definition 4.1 of *normal* and Definition 4.3.

Lemma 4.3 *Given a collection C of nodes of a* normal *L_x such that for all $y, z \in C$ where y is not a predecessor z then there exist disjoint substrings of s corresponding to each of the nodes in C. In particular the collection of leaves of any (incubating) subtree of L_x is such a collection.*

We will now relate the trees L_x to substrings of s. The following definition gives a minimal requirement on the occurrence in s of the strings *corresponding* to the nodes of L_x.

Definition 4.4 *We say that a collection C of strings is a* string decomposition *of a tree T if each string* corresponding *to an edge occurs as a substring in C and the strings* corresponding *to the edges that are not incident to the root or the leaves occur twice as substrings in C.*

A string requirement labeling *of a tree T is defined by labeling $l(e) = 1$ if e is a root edge or an edge between a branching node and a leaf, otherwise $l(e) = 2$.*

Let $R_T = \sum_e l(e)$ and $l(b) = \sum_{e \text{ out of } b} l(e)$.

We now show an upper bound on the probability that L_x takes form T in four steps. First we give an algorithm that returns a particular type of *string decomposition*. Then we bound the number of possible *string decompositions* generated by the algorithm for any fixed tree T. We then go on to bound the probability that s contains disjointly a given *string decomposition* of s. Finally we show that when L_x is *normal* and takes form T or has T as an *incubating subtree* then s contains disjointly substrings that form one of the *string decompositions* of T that are generated by the algorithm.

Let us now fix a node x in D_m^s and fix T to be some given (b, i, l)-tree, for some fixed integers b, i and l. Furthermore, let us define mapping from T to a subgraph of the complete DeBruijn graph. We will name the root node of T x and let us name all the nodes of the graph T, as follows. Now notice that in our definition of (b, i, l)-trees that each node has at most α children and we make a distinction between two based on a predefined ordering of the children, we can therefore talk about the k^{th} child of a node, where k is some number between 1 and α, possibly greater than the number of x's children. If z is the k^{th} child of a node named $y_1 y_2 \ldots y_{m-1}$ name z $y_2 \ldots y_{m-1}\alpha_k$ and the edge between them $y_1 y_2 \ldots y_{m-1}\alpha_k$, where α_k is the k^{th} letter of the alphabet Σ.

The following algorithm generates a *string decomposition* of T.

Algorithm 2

Label T `using the string requirement labeling of Def. 4.4`
Preorder `the nodes of` T.
Initialize C `as an empty collection.`
While $\exists$ e `such that` $l(e) > 0$
  `Let` v `be the lowest ordered node of` T `with a`
    `positively labeled out-edge,` e.
  `Choose` y `such that` $\mathcal{P}(y) = e$.
  $l(e) \leftarrow l(e) - 1,\ v \leftarrow \text{out}(e)$.
  While v `is non-leaf`
    `Choose` e `as one of the out-edges of` v.
    `Append to` y `the character` ***corresponding*** `to` e.
    $l(e) \leftarrow l(e) - 1,\ v \leftarrow \text{in}(e)$.
  `Add` y `to the collection` C.
Return C.

We will first upper bound the number of *string decompositions* generated by this algorithm.

Lemma 4.4 *The number of possible* string decompositions *generated by Algorithm 2 is bounded above by* α^{R_T-2i+1}, *where* i *is the number of internal non-branching nodes in* T *and* R_T *is defined in Definition 4.4.*

Proof: Let us count the number of choices made by the algorithm. Let v be some branching node and k be the sum of the labels of v's out-edges. Algorithm 2 will arrive at most $k+1$ times at v. Whenever v is chosen in the outer loop one of v's out-edge labels gets decreased by one. Since the in-edge of v has label at most two we will arrive at most twice at v in the inner loop. The first time we arrive at v in the inner loop all out-edges of v have positive labels and one of them will be decreased. The second time we arrive at v in the inner loop we may choose an edge we have chosen before. The last time we arrive at v in the outer loop we have no choice of which edge to traverse. All other times we have some choice of edge to traverse, the number of choices clearly being less than α. When we arrive at a node that has a single child we have no choice as to which edge to traverse next. The number of *string decompositions* is therefore at most

$$\Pi_{v\in B}\alpha^{l(v)} \leq \alpha^{R_T-2i+1},$$

where B is the set of branching nodes. The inequality can be verified by noting that the labels of out-edges of nodes that have one child are always two except in the case when the root has only one child. □

We will now upper bound the probability of any *string decomposition.*

Lemma 4.5 *If* C *is a collection of strings generated by Algorithm 2 then the probability that the strings of* C *occur disjointly as substrings of* s *is upper bounded by* $\xi^l\alpha^{-R_T}$, *where* $\xi = \frac{n}{\alpha^{m-1}}$.

Proof: Let D be our set of strings. By Lemma 4.3 $|D| \geq l$. When originally chosen each of the strings in D has length $m-1$. Furthermore we will append to them at least R_T extra characters. Note that the probability of a string of length j occurring in s is at most $\frac{n}{\alpha^{j-1}}$. We then have that the probability of all of the strings in D occurring in s is bounded by:

$$\Pi_{x\in D}\frac{n}{\alpha^{|x|}} \leq \xi^l\Pi_{x\in D}\alpha^{m-1-|x|} \leq \xi^l\alpha^{-R_T}$$

□

We now relate L_x to the string s.

Lemma 4.6 *If T is an* incubating subtree *of a* normal L_x *then s contains disjointly one of the collections generated by Algorithm 2.*

Proof: Based on the actual string s we will show how the Algorithm 2 can be made to construct a collection of disjoint substring s. By the definition of *normal*, if T is an *incubating subtree* of L_x and L_x is *normal* and e is an edge of T labeled k by Algorithm 2 then s must contain at least k disjoint substrings *corresponding* to e. In the outer loop of Algorithm 2 we can hence always choose y to be some substring of s disjoint from those previously chosen. Now let $s_{j-m+1} \dots s_j$ *correspond* to y. In the inner loop we make the choice of e based on the next characters (s_{j+1} and onwards) of s, i.e. if s_{j+1} is k then e will be chosen as the k-th child of v. Since L_x is *normal* the *end* string will not occur in L_x and the choice of the child is hence always well defined. Since T is an *incubating subtree* of L_x this choice will never return an edge not in T and will terminate at a leaf node of T. □

4.4 Number of *normal* queries

Combining the results of Lemmas 4.2, 4.4 and 4.5 gives the following lemma.

Lemma 4.7 *The probability that L_x contains a (b,i,l)-tree as an* incubating subtree *is at most*

$$\frac{1}{(\alpha-1)k+1}\binom{\alpha k}{k}\binom{i+k-1}{i}\xi^l\alpha^{-i+1}$$

where $k = b + l$ and $\xi = \frac{n}{\alpha^{m-1}}$.

Lemma 4.8 *Let $\xi = n/\alpha^{m-1}$ and assume that m is large enough so that $\frac{e\alpha}{1-\alpha^{-1}}\sqrt{\xi} < 1$. Then the expected size of the set of queries Q_N added to Q* corresponding *to* normal *strings x is bounded by*

$$\frac{ne\alpha}{(1-\alpha^{-1})^2}\frac{1}{1-\frac{e\alpha\sqrt{\xi}}{1-\alpha^{-1}}}$$

Proof: We can estimate the expected number of queries by multiplying the number of nodes x, in D_m^s with the probability that L_x has a (b,i,l)-tree as an *incubating subtree* and with the number of leaves l and then summing over all (b,i,l).

Let $k = l + b$ and notice that $l \geq b + 1$. To get the expected number of queries in Q we multiply the number of nodes in D_m^s with the sum over all possible k's and i's of the number of queries added for each tree ($b \leq k$) multiplied with the probability of the tree over all possible k's and i's and the number of ways to choose the initial node x.

$$
\begin{aligned}
E(|Q_N|) \leq & \\
& \alpha^{m-1} \sum_{k=1}^{\infty} \sum_{i=0}^{\infty} k \frac{1}{(\alpha-1)k+1} \binom{\alpha k}{k} \binom{i+k-1}{i} \xi^{(k+1)/2} \alpha^{-i+1} \\
& \leq \frac{n\alpha}{\alpha-1} \sum_{k=1}^{\infty} \sum_{i=0}^{\infty} (e\alpha)^k \xi^{(k-1)/2} \binom{i+k-1}{i} \alpha^{-i} \\
& = \frac{n\alpha}{\sqrt{\xi}(\alpha-1)} \sum_{k=1}^{\infty} (e\alpha\sqrt{\xi})^k \left(\frac{1}{1-\alpha^{-1}} \right)^k \\
& = \frac{ne\alpha}{(1-\alpha^{-1})^2} \frac{1}{1 - \frac{e\alpha\sqrt{\xi}}{1-\alpha^{-1}}}
\end{aligned}
$$

The second inequality follows from a series of algebraic manipulations and noting $\binom{\alpha k}{k} \leq (e\alpha)^k$. The second equality is a well known identity for geometric series. The first equality is less well known but can be observed by differentiating the identity for geometric series k times (Slomson, 1991). □

We have now shown that only $O(n)$ queries are generated for nodes x in the graph where L_x is *normal*.

4.5 Remaining Cases

We will now show that it is unlikely that L_x is not *normal* and then we can use the fact that the maximum number of queries generated by the algorithm with initial string *corresponding* to any given node in the graph is bounded by $\sum_{i=2}^{k_0} \alpha^i = O(n^{\frac{1}{5}})$.

Let us introduce some terminology.

Definition 4.5 *1. We say that a string, $x = x_1 x_2 \ldots x_p$ is* k-periodic *if $x_i = x_{i+k}$, $i = 1, 2, \ldots, p-k$.*

2. *The* core *of a string that corresponds to a node (node string)* $x = x_1x_2\ldots x_{m-1}$ *is the substring of* x *that occurs as a substring in all node strings of* L_x*, i.e.* $x_{k_0}\ldots x_{m-1}$.

The cases when L_x is not *normal* are when *end* occurs in L_x, L_x has a cycle or the *string decomposition* of L_x consists of strings that are non-disjoint in s.

End will occur in L_x for at most $\sum_{i=0}^{k_0+1}\alpha^i = O(n^{\frac{1}{5}})$ nodes x. The number of queries containing the string *corresponding* to *end* as a substring is hence bounded by $O(n^{\frac{2}{5}}) = o(n)$.

As the depth of L_x is bounded by k_0+1 then if L_x contains a cycle the period of the cycle must also be bounded by k_0+1. The *core* of x must therefore have period at most k_0+1 as it occurs in all the node strings of L_x. If the *core* of x has period less than k_0+1 we will call x a *low-periodic core* string. To simplify presentation we will also consider strings which are periodic with period less than $\lceil\frac{1}{2}\log_\alpha n\rceil$ to be *low-periodic core* strings.

All extensions of a node string x will appear disjointly if the *core* of x does not occur twice in s, starting at positions i and j, where $|i-j| \leq m+k_0+1$. Here $m-1$ is the length of the node strings in the current iteration of **Step** $2'$, the condition may therefore be rewritten as $|i-j| \leq O(\log_\alpha n)$. If this happens we say the *core* of x is *self-repetitive*.

To count the expected number of queries in Q we will consider four cases. First we will count the expected number of queries stemming from strings with a *low-periodic core*. Then the expected number of elongations of strings with a *self-repetitive core* given that the *core* is not *low-periodic*. The remainder of the strings are *normal* or have the terminal string of s occurring in L_x.

We will now count the number of queries in Q that originate at *low-periodic core* strings. The number of node strings with a *core* of period k is determined by the degrees of freedom outside the *core* (k_0+1) plus the degrees of freedom inside the *core* ($\leq k$) and is therefore α^{k_0+k+1}. The number of extensions of *low-periodic core* node string is hence at most

$$\sum_{k=1}^{\lceil\frac{1}{2}\log_\alpha n\rceil} \alpha^{\lceil\frac{1}{5}\log_\alpha n\rceil+k+1}O(n^{\frac{1}{5}}) = o(n).$$

Now we look at the number of query strings originating at node strings whose *core* is *self-repetitive* but not *low-periodic*. The expected number of

cores that are *self-repetitive* and not *low-periodic* is at most

$$\begin{array}{r}(\text{number of places } i \text{ for the first } \textit{core} \text{ to start})\times \\ (\text{number of places for second } \textit{core} \text{ to start})\times \\ P(\text{second } \textit{core} = \text{first } \textit{core})\end{array}$$

which is bounded above by $n \times O(\log_\alpha n) \times \alpha^{-\lceil \frac{1}{2} \log_\alpha n \rceil} = O(n^{\frac{1}{2}} \log_\alpha n)$. The expected number of strings with a *self-repetitive core* is hence bounded by $O(n^{\frac{7}{10}} \log_\alpha n)$. The expected number of queries added to Q in this case is hence bounded by $O(n^{\frac{9}{10}} \log_\alpha n) = o(n)$.

Using Lemma 4.8 have now shown Lemma 4.1. The expected number of queries generated in **Step** $2'$ is bounded by

$$\frac{ne\alpha}{(1-\alpha^{-1})^2} \frac{1}{1 - \frac{e\alpha\sqrt{\xi}}{1-\alpha^{-1}}} + o(n) + o(n) + o(n) = O(n).$$

4.6 Concentration of Expectation

We now use Azuma's inequality (see (Alon and Spencer, 1992)) to show that with high probability **Step** $2'$ has only a linear number of queries.

Lemma 4.9 *With high probability no more than $O(n)$ queries are added to Q in each iteration of* **Step** $2'$.

Proof:

We can view s to be a sequence of n independent random trials, one for each of its characters. We want to bound the number of queries that may be added to or removed from Q if we change one character. The changing of one character may effect at most $(m + \lceil \frac{1}{5} \log_\alpha n \rceil + 1) \sum_{i=0}^{\lceil \frac{1}{5} \log_\alpha n \rceil + 2} \alpha^i = O(n^{\frac{1}{5}} \log_\alpha n)$ strings, where m denotes the length of the node strings in the current iteration of **Step** $2'$. This can be seen by first choosing the position of the character change in the query string and noting that the length of each query string is at most $m + k_0 + 1$. Now let c be the character that was changed in s and let $r_{-m+2} r_{-m+3} \ldots r_{m-2}$ be the characters immediately preceding and following c with $c = r_0$. Let q be a query string that is affected by the character change and $m + k$ the length of q. The path in D_m^s that q *corresponds* to must pass through one of the nodes corresponding to substrings of length $m - 1$ containing c. In particular if the character

change occurred in the jth position of q then if $j \geq m-1$, q has characters $r_{-m+2}r_{-m+3}\ldots r_0$, in positions $j-m+2, j-m+3 \ldots j$ the characters in position $1, 2, \ldots j-m+1$ and $j+1, \ldots m+k$ can then be any character. If $j \leq m-1$ q has characters $r_{-m+2+j}r_{-m+3+j}\ldots r_j$ in its first $m-1$ positions and the characters $m \ldots m+k$ can be any character. In both cases there are at most $\sum_{i=0}^{k_0+2} \alpha^i$ strings that can be affected.

By Azuma's equality we have

$$P(|Q| > E(|Q|) + t) \leq 2\exp\left\{-\frac{2t^2}{2n(O(n^{\frac{1}{5}}\log_\alpha n))^2}\right\}.$$

Putting $t = n^{9/10}$ completes the proof. □

5 Computational Results

The choice of k_0 as $\lceil \frac{1}{5}\log_\alpha n \rceil$ in the previous section was done for ease of presentation and may be chosen slightly larger to decrease the number of rounds. To test the practicality of our method we implemented a variant of the algorithm presented that is not as stringent as the algorithm analyzed (**Step** $2'$) and closer to the original algorithm with **Step** 2. In this variant, we limited the length of the queries to the largest l such the total number of queries in each round is limited to $O(n)$, instead of limiting their length to a fixed k_0. In other words **Step** 2 is modified to:

Step $2''$ **Repeat** `until` D_m^s `is a line.`
`Let` Q `be the set of strings` $\{x_1 \ldots x_{m+k}\}$ `where`
`i)` $k \in \{1, \ldots, l\}$
`ii)` $\mathcal{P}(x_1 \ldots x_{m+k}) \in D_m^s$
`iii)` $\lambda(x_i \ldots x_{m+i-2}) = 2^{\geq}$ `where` $i \in \{2 \ldots k-1\}$
`iv)` $\lambda(x_i \ldots x_{m+i-1}) = 2^{\geq}$ `where` $i \in \{2 \ldots k-1\}$
`v)` l `is chosen as large as possible with` $\mid Q \mid \leq \alpha^c n$.
`Ask the queries in` Q.
$m \leftarrow m+l$. `Construct` D_m^s.

Table 1 shows the number of SBH chips used when the base pairs are generated randomly. The number of query rounds is significantly lower than the number of rounds guaranteed by the worst-case performance guarantee of

the algorithm. We see that if we initially use a classical SBH chip containing all oligonucleotides of size $\lceil \log_4 n \rceil + 1$ we can finish the sequencing of the DNA in less than $4n$ extra queries, using a single chip, for all of our examples.

Table 2 shows the number of SBH chips used to sequence arbitrarily chosen virus sequences. For all but one of our examples our algorithm will sequence their DNA by using the classical chip containing all strings of length $\log_4 n + 1$ and then an extra round of at most $4n$ queries.

6 Acknowledgements

The authors would like to than R.Ravi, Magnús M. Halldórsson, Daníel F. Gudbjartsson and the anonymous referees for reviewing this paper. Alan M. Frieze was supported in part by NSF grant CCR9818411. Bjarni V. Halldórsson was supported by a Merck Computational Biology and Chemistry Program Graduate Fellowship from the Merck Company Foundation.

References

Alon, N. and Spencer, J., 1992. *The probabilistic method*. John Wiley and Sons, New York, NY.

Arratia, R., Martin, D., Reinert, G., and Waterman, M., 1996. Poisson process approximation for sequence repeats, and sequencing by hybridization. *Journal of Computational Biology*, 3, 425–463.

Bains, W. and Smith, G. C., 1988. A novel method for nucleic acid sequence determination. *J. Theor. Biol.*, 135, 303–307.

Ben-Dor, A., Pe'er, I., Shamir, R., and Sharan, R., 1999. On the complexity of positional sequencing by hybridization, 88–100. In Crochemore, M. and Paterson, M., editors, *Proceedings of the 10th Annual Symposium on Combinatorial Pattern Matching*, Springer-Verlag, Berlin.

Blanchard, A., 1998. Synthetic DNA arrays. *Genetic Engineering*, 20, 111–123.

Blanchard, A., Kaiser, R., and Hood, L., 1996. High-density oligonucleotide arrays. *Biosensors and bioelectronics*, 11, 687–690.

Broude, N., Sano, T., Smith, C., and Cantor, C., 1994. Enhanced DNA sequencing by hybridization. *Proceedings of the National Academy of Sciences*, 91:3072–3076.

Chetverin, A. and Cramer, F., 1994. Oligonucleotide arrays: New concepts and possibilities. *Bio/Technology*, 12, 1093–1099.

Drmanac; R. T., Crkvenjakov, R. B. *Method of sequencing of genomes by hybridization of oligonucleotide probes.* U.S. Patent No. 5,202,231

Drmanac, R., Labat, I., Bruckner, I., and Crkvenjakov, R. (1989). Sequencing of megabase plus DNA by hybridization. *Genomics*, 4, 114–128.

Drmanac, R. et al., 1993. DNA sequnence determination by hybridization: A strategy for efficien large scale sequencing. *Science*, 260, 1649–1652.

Dyer, M., Frieze, A. M., and Suen, S., 1994. The probability of unique solutions of sequencing by hybridization. *Journal of Computational Biology*, 1, 105–110.

Fodor, S., Read, J., Pirrung, M., Stryer, L., Lu, A., and Solas, D., 1991. Light-directed, spatially addressable parallel chemical synthesis. *Science*, 251, 767–773.

Hannenhalli, S., Feldman, W., Lewis, H., Skiena, S., and Pevzner, P., 1996. Positional sequencing by hybridization. *Comp. Appl. Biosci*, 12, 19–24.

Knuth, D., 1968. *The Art of Computer Programming: Fundamental Algorithms.* Addison-Wesley, Reading, MA.

Kruglyak, S., 1998. Multistage sequencing by hybridization. *Journal of Computational Biology*, 5, 165–171.

Lockhart, D., Dong, H., Byrne, M., Follettie, M., Gallo, M., Chee, M., Mittmann, M., Wang, C., Kobayashi, M., Horton, H., and Brown, E., 1996. Expression monitoring by hybridization to high-density oligonucleotide arrays. *Nature Biotechnology*, 14, 1675–80.

Lysov, Y., Florentiev, V., Khorlin, A., Khrapko, K., Shih, V., and Mirzabekov, A., 1998. Sequencing by hybridization via oligonucleotides. a novel method. *Dokl. Acad. Sci. USSR*, 303, 1508–1511.

Margaritis, D. and Skiena, S. S., 1995. Reconstructing strings from substrings in rounds, 613–620. In *36th Annual Symposium on Foundations of Computer Science (FOCS'95)*, IEEE Computer Society Press.

Pevzner, P.A. 1989. l-tuple DNA sequencing: Computer analysis. *Jour. Biomolecul. Struct. & Dynamics*, 7, 63–73.

Pe'er, I. and Shamir, R., 2000. Spectrum alignment: Efficient resequencing by hybridization, 260–268. In et al., R. A., editor, *Proceedings of the 8th Annual Symposium on Intelligent Systems for Molecular Biology*, AAAI Press, Menlo Park, CA.

Pevzner, P. and Lipshutz, R., 1994. Towards DNA-sequencing by hybridization. In *19th Symp. on Mathem. Found. of Comp. Sci.*, volume 841 of *Lecture Notes in Computer Science*, pages 143–158.

Pevzner, P., Lysov, Y., Khrapko, K., Belyavsky, A., Florentiev, V., and Mirzabekov, A., 1991. Improved chips for sequencing by hybridization. *Journal of Biomolecular Structure and Dynamics*, 9, 399–410.

Preparata, F., Frieze, A., and Upfal, E., 1999. Optimal reconstruction of a sequence from its probes. *Journal of Compututational Biology*, 6, 361–368.

Preparata, F. P. and Upfal, E., 2000. Sequencing-by-hybridization at the information-theory bound: An optimal algorithm, 245–253. In Shamir, R., Miyano, S., Istrail, S., Pevzner, P., and Waterman, M., editors, *Proceedings of the 4th Annual International Conference on Computational Molecular Biology (RECOMB-00)*, ACM Press, New York, NY.

Schena, M., editor, 1999. *DNA Microarrays*. Oxford University Press, Oxford, UK.

Shamir, T. and Tsur, D., 2001. Large scale sequencing by hybridization, 269–277. In Lengauer, T., Sankoff, D., Istrail, S., Pevzner, P., and Waterman, M., editors, *Proceedings of the 5th Annual International Conference on Computational Molecular Biology (RECOMB-01)*, ACM Press, New York, NY.

Skiena, S. and Sundaram, G., 1995. Reconstructing strings from substrings. *Journal of Computational Biology*, 2:333–353.

Slomson, A., 1991. *An Introduction to Combinatorics.* Chapman and Hall, London, UK.

Southern, E., 1996. DNA chips: analyzing sequence by hybridization to oligonucleotides on a large scale. *Genetics*, 10, 110–114.

Base Pairs	$c = 0$	$c = 1$	$c = 2$	$c = 3$	$c = 4$
1000	4	2	2	2	2
2000	3.6	2	2	2	2
5000	3	2	2	2	2
10000	4	2	2	2	2
20000	3	2	2	2	2
50000	4	2	2	2	2
100000	3	2	2	2	2

Table 1: Number of iterations needed to sequence random sequences. In the initial iteration we use the classical SBH-chip containing strings of length $\lceil \log_4 n \rceil + c$ and in subsequent iterations we allow at most $4^c n$ queries in each iteration, where n is the number of base pairs. Data is averaged over 5 trials.

Accession	Base Pairs	$c = 0$	$c = 1$	$c = 2$	$c = 3$	$c = 4$
NC001792	1767	4	2	2	2	2
NC001928	2581	4	2	2	2	2
NC000930	4316	8	3	3	2	2
NC000944	7405	4	2	2	2	2
NC001786	11488	4	2	2	2	2
NC001813	33213	4	2	2	2	2
NC001720	43804	4	2	2	2	2

Table 2: Number of rounds needed to sequence arbitrarily chosen virus sequences. In the initial iteration we use the classical SBH-chip with all strings of length $\lceil \log_4 n \rceil + c$ and in subsequent iterations we allow at most $4^c n$ queries in each iteration, where n is the number of base pairs.