Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Vladimír Boža

Autoreferát dizertačnej práce

Algorithms for high-throughput sequencing data

na získanie akademického titulu philosophiae doctor

v odbore doktorandského štúdia:

9.2.1. - informatika

Bratislava
2017

Dizertačná práca bola vypracovaná v dennej forme doktorandského štúdia na Katedre aplikovanej informatiky Fakulty matematiky, fyziky a informatiky Univerzity Komenského v Bratislave.

Predkladateľ: Vladimír Boža
Katedra aplikovanej informatiky
Fakulta matematiky, fyziky a informatiky
Univerzita Komenského
Bratislava

Školiteľ: Mgr. Tomáš Vinař, PhD.
Katedra aplikovanej informatiky
Fakulta matematiky, fyziky a informatiky
Univerzita Komenského
Bratislava

Odbor: 9.2.1. - informatika

Predseda odborovej komisie:
prof. RNDr. Rastislav Kráľovič, PhD.
Katedra informatiky
Fakulta matematiky, fyziky a informatiky
Univerzita Komenského
Bratislava

1 Overview

Knowledge of DNA sequences has become indispensable for basic biological research and in numerous applied fields like diagnostics, forensic biology, etc. It is also important for understanding cancer, fighting antibiotic resistant bacteria, etc. With decreasing cost of the DNA sequencing we are able to sequence more organisms, but consequently we have to handle much bigger amounts of data.

Usual length of DNA sequence of an organism is between millions and billions of characters. For example, human DNA has approximatelly three billions bases. Unfortunately, we are still not able to read the whole DNA sequence at once, we can only read it in small pieces called *reads*. Usual length of reads vary between hundred to tens of thousands of bases depending on the sequencing technology. Thus, our only option is to try to computationaly reconstruct original sequence, using overlap information between reads. This process is called *sequence assembly*.

Theoretically formulating DNA genome assembly leads usually to NP-hard problems (Kececioglu and Myers, 1995), thus we resort to heuristic approaches. To make matters worse, there are also multiple technologies (Quail et al., 2012; Liu et al., 2012) for reading DNA with very different properties like read length, error rate, error types, etc. Some technologies produce paired reads with known distance between pairs. Usually assembly algorithms are tailored to a specific type of data and thus do not work with every combination of data we can get. For example, ALLPATHS-LG (Gnerre et al., 2011) assembler requires paired reads with specific distance between them. As one of our contributions we present GAML (Genome assembly by maximum likelihood) (Boža et al., 2014), a genome assembly framework, which allows to seamlessly combine multiple data types. GAML is based on optimizing genome likelihood score (Ghodsi et al., 2013; Clark et al., 2013; Rahman and Pachter, 2013), which was shown to strongly correlate with the quality of the assembly. In GAML, we optimize the likelihood score by using simulated annealing, while we speed up likelihood evaluation by reusing results from the previous evaluations. We experimentally evaluate our algorithm and show that it produces comparable results to other assemblers, which are tailored to specific datasets. We also show that GAML is able to correct and improve previously created assembly by an other assembler.

Eventhough assembly algorithms are heuristics, they need to use many theoretically interesting data structures and algorithms to scale to large genomic datasets. Examples include probabilistic data structures for de Bruijn graphs (Chikhi et al., 2012), minhashing approaches for overlaps (Koren et al., 2017) and fast and memory efficient algorithms for string indexing (Li and Durbin, 2009; Ferragina and Manzini, 2000). In GAML, we encountered a problem of indexing large collections of short reads. We present our solution to this problem, a compressed index for genomic data, called CR-Index (Boža et al., 2015). In CR-Index, we exploit the fact that reads are usually derived from some unknown superstring, which we approximately reconstruct and use for indexing. We also achieve a large memory savings by carefully handling sequencing errors. CR-Index uses much less space that other solutions, while maintaining fast querying speed.

Another traditional problem is indexing of a long DNA string and finding positions of short substrings of prespecified size. Typical solutions include hash tables, which are fast, but use more memory, and FM-Index (Li and Durbin, 2009; Ferragina and Manzini, 2000), which uses less memory, but is slower. Inspired by success of minhashing for other genomic tasks (Koren et al., 2017; Wood and Salzberg, 2014), we present our own idea for fast and practical

data structure called MH-index for searching for fixed-length small substrings in a long string, which uses minhashing. Our new data structure has fast speed comparable to hash-tables and uses small memory amount of memory comparable to FM-indices.

We also explore problems relate to Oxford Nanopore MinION, which is one of the newest sequencing technologies showing a great promise in clinical applications and other areas. MinION produces long but very noisy reads. An interestring part of MinION from computer science point of view is that it does not perform exact base detection directly in hardware, but instead only collects electrical signal, which needs to be translated into DNA bases using a special software called base caller. We present our own base caller for MinION, called DeepNano (Boža et al., 2016), which uses recurrent neural networks. On older versions of MinION, DeepNano had much better accuracy and base calling speed than the original base caller. On newer versions, DeepNano has a slightly worse accuracy, but slightly better speed than the original base caller.

One interesting ability of MinION is to produce results while DNA is being sequenced and to reject reads while they are being sequenced, so we can focus sequencing on interesting parts of the DNA (Loose et al., 2016). Unfortunately, base calling algorithms are slower than the sequencing hardware, so we either have to use large computers, or come up with fast approach for rejecting reads. One of such approaches, is working directly with electrical signal coming from MinION, using algorithms like dynamic time warping (Sankoff and Kruskal, 1983). We experimentally show that the original version of DTW is not suitable in our case and propose a suitable improvements and experimentally demonstrate that our improvements lead to better sensitivity and specificity.

2 Probabilistic Sequence Assembly

Perhaps the oldest formulation of the assembly task as a computer science problem is the shortest common superstring problem.

Definition 1 (The shortest common supersting) *Given a set of strings $\mathcal{P} = \{S_1, S_2, \ldots, S_k\}$, the shortest common superstring is the shortest string S that contains every string from $\mathcal{P}$ as a substring.*

Unfortunately, finding the shortest common superstring is NP-hard (Gallant et al., 1980; Garey and Johnson, 1979). There is a known 2.5-approximation algorithm (Sweedyk, 2000).

Over the years, it has been shown that the shortest common superstring formulation does not represent the assembly problem very well. On one hand, the problem is difficult to solve computationally, and the approximation algorithms do not yield very practical results. On the other hand, this formulation does not consider some specifics of the underlying data, like: repeated regions in DNA, complementary DNA strands, errors in reads, paired reads and variety of the sequencing technologies (see Table 1).

A wide variety of technologies presents an additional challenge to the assembly software. Ideally, we would handle multiple combinations of read libraries from possibly different technologies (like Illumina + PacBio, or multiple libraries of Illumina reads with different insert sizes).

Due to these problems, it is obvious that finding the shortest common superstring is not a good formulation for the sequence assembly. In practice, our goals are usually more humble and we are satisfied with reconstructing unambiguous parts of the DNA sequence. Assembly

Technology	Read length	Error rate	Paired reads	Cost per million bases	Reads per run	Time per run
Sanger	900	0.1%	Yes	\$2400	≈ 100	few hours
454	700	0.1%	Yes	\$10	1 million	one day
Illumina	50 - 300	2%	Yes	$\$0.05 - \$0.15$	3 billion	few days
PacBio	20000	14%	No	$\$0.13 - \$0.60$	50000	few hours
Oxford Nanopore	30000	$10\% - 20\%$	No	\$0.5	20000	one day

Table 1: **Overview of current sequencing technologies**

tools usually produce substrings from the original sequence called *contigs*. Sometimes it is possible to detect approximate distances between contigs, but the DNA sequence between them remains unknown. A *scaffold* is a sequence of contigs with known approximate distance between them. In genome assemblies unknown areas are often represented as a strings of several Ns.

In general, assembly algorithms proceed to "glue" reads which can be unambiguously glued together, building the assembly step-by-step without following clear optimization criteria. They use efficient representation of overlaps between reads and try to resolve ambiguous regions using paired reads and long reads.

A good review of assembly algorithms can be found in Miller et al. (2010). The algorithms differ mainly in overlap representation. In our work, we build mostly on top of the de Bruijn graphs (Pevzner et al., 2001). We do not work directly with reads, but with sequences of k consecutive bases (k-mers) which occur in reads. The nodes in de Bruijn graph represent k-mers and edges represent adjacencies between k-mers in reads. Note that time required to construct this graph is linear in the length of reads, but on the other hand we lose long range connectivity information since we are not working with complete reads but instead we cut each read to smaller k-mers. After constructing de Bruijn graph we can join k-mers which be unambiguously joined and then apply various heuristics for resolving repeats. A typical example of de Bruijn assembler is Velvet (Zerbino and Birney, 2008).

2.1 Probabilistic Models of Sequence Assembly

In many cases, we have to compare multiple assemblies and decide which one is the best. Ghodsi et al. (2013) showed that a very simple and theoretically sound probabilistic model can provide a good indication of the assembly quality.

We will consider a probabilistic model that defines probability $\Pr(R|A)$ that a set of sequencing reads R is observed assuming that the assembly A is the correct assembly of the genome. Since the sequencing itself is a stochastic process, it is very natural to characterize concordance of reads and an assembly by giving a probability of observing a particular read.

The model assumes that individual reads are independently sampled, and thus the overall likelihood is the product of likelihoods of the reads: $\Pr(R|A) = \prod_{r \in R} \Pr(r|A)$. To make the resulting value independent of the number of reads in set R, we use as the main assembly score the log average probability of a read computed as follows: $\mathrm{LAP}(A|R) = (1/|R|) \sum_{r \in R} \log \Pr(r|A)$. Note that maximizing $\Pr(R|A)$ is equivalent to maximizing $\mathrm{LAP}(A|R)$. Individual read probabilities can be computed using dynamic programming. In practice, this dynamic programming is too time consuming. To approximate the probability, we will instead align reads to the as-

Assembler	LAP	N50 (thousands)	NA50 (thousands)
Reference sequence	−23.509	2873	2873
ALLPATHS-LG	−23.760	1092	1092
SOAPdenovo	−23.862	332	288
Velvet	−23.925	762	126
AbySS	−24.584	34	28

Table 2: **Comparison of several assemblers on *Staphylococcus aureus* dataset using LAP and NA50.** Data from Ghodsi et al. (2013).

sembly and compute likelihood from these alignments.

To illustrate usufulness of likelihood score we present several results from Ghodsi et al. (2013) where they compare several assemblers and resulting $NA50$ and the assembly likelihood, and show that there is a good correlation between these measures (Table 2). More data can be found in Ghodsi et al. (2013).

2.2 Genome Assembly by Maximum Likelihood

We propose a new framework GAML (Genome Assembly by Maximum Likelihood), that allows systematic combination of diverse datasets into a single assembly, without requiring a particular type of data for specific heuristic steps. We build GAML on top of probabilistic model described above, where we have shown, that probabilistic models are very successful in evaluating the quality of genome assemblers. Here, we use likelihood of a genome assembly as an optimization criterion, with the goal of finding the assembly with the highest likelihood. Even though this may not be always feasible, we demonstrate that optimization based on simulated annealing can be very successful at finding high likelihood genome assemblies.

Our goal is to find the highest likelihood assembly directly. Of course, the search space is huge, and the objective function too complex to admit exact methods. Here, we describe an effective optimization routine based on the simulated annealing framework (Eglese, 1990).

Our algorithm for finding the maximum likelihood assembly consists of three main steps: preprocessing, optimization, and postprocessing. In *preprocessing*, we decrease the scale of the problem by creating an assembly graph, where vertices correspond to contigs and edges correspond to possible adjacencies between contigs supported by reads. In order to make the search viable, we will restrict our search to assemblies that can be represented as a set of walks in this graph. Therefore, the assembly graph should be built in a conservative way, where the goal is not to produce long contigs, but rather to avoid errors inside them. In the *optimization step*, we start with an initial assembly (a set of walks in the assembly graph), and iteratively propose changes in order to optimize the assembly likelihood. Finally, *postprocessing* examines the resulting walks and splits some of them into shorter contigs if there are multiple equally likely possibilities of resolving ambiguities. This happens, for example, when the genome contains long repeats that cannot be resolved by any of the datasets.

To find a high likelihood assembly, we use an iterative simulated annealing scheme. We start from an initial assembly A_0 in the assembly graph. In each iteration, we randomly choose a *move* that proposes a new assembly A' similar to the current assembly A and using simulating annealing rules we decide whether to use the new assembly or to keep the old one.

To further reduce the complexity of the assembly problem, we classify all contigs as either

long (more than 500bp) or *short* and concentrate on ordering the long contigs correctly. The short contigs are used to fill the gaps between the long contigs. Recall that each assembly is a set of walks in the assembly graph. A contig can appear in more than one walk or can be present in a single walk multiple times.

Proposals of new assemblies are created from the current assembly using moves like:

- *Walk extension.* We start from one end of an existing walk and randomly walk through the graph, in every step uniformly choosing one of the edges outgoing from the current node.
- *Local improvement.* We optimize the part of some walk connecting two long contigs s and t. We first sample multiple random walks starting from contig s. In each walk, we only consider nodes from which contig t is reachable. Then we evaluate these random walks and choose the one that increases the likelihood the most.
- *Repeat optimization.* We optimize the copy number of short tandem repeats. We do this by removing or adding a loop to some walk.
- *Disconnecting.* We remove a path through short contigs connecting two long contigs in the same walk, resulting in two shorter walks.
- *Repeat interchange.* If a long contig has several incoming and outgoing walks, we optimize the pairing of incoming and outgoing edges.

At the beginning of each annealing step, the type of the move is chosen randomly; each type of move has its own probability. We also choose randomly the contig at which we attempt to apply the move.

Note that some moves (e.g. local improvement) are very general, while other moves (e.g. joining with advice) are targeted at specific types of data. This does not contradict a general nature of our framework; it is possible to add new moves as new types of data emerge, leading to improvement when using specific datasets, while not affecting the performance when such data is unavailable.

The most time consuming step in our algorithm is evaluation of the assembly likelihood, which we perform in each iteration of simulated annealing. This step involves alignment of a large number of reads to the current assembly. However, only a small part of the assembly is changed in each annealing step, which we can use to significantly reduce the running time. We only align reads to the part of the assembly which was changed and also prefilter the candidate reads for alignment using read indexing.

2.3 Experimental Evaluation of GAML

To evaluate the quality of our assembler, we have adopted the methodology the GAGE project (Salzberg et al., 2012), using metrics on scaffolds. We have used the same genomes and libraries as Salzberg et al. (2012) (the *Staphylococus aureus* genome) and Deshpande et al. (2013) (the *Escherichia coli* genome). The overview of the datasets is shown in Table 3. An additional dataset EC3 (long insert, low coverage) was simulated using the ART software (Huang et al., 2012).

We have evaluated GAML in the following scenarios:

1. combination of fragment and short insert Illumina libraries (SA1, SA2),

ID	Source	Technology	Insert length	Read length	Coverage	Error rate
Staphylococus aureus (2.87Mbp)						
SA1	Salzberg et al. (2012)	Illumina	180bp	101bp	90	3%
SA2	Salzberg et al. (2012)	Illumina	3500bp	37bp	90	3%
Escherichia coli (4.64Mbp)						
EC1	Deshpande et al. (2013)	Illumina	300bp	151bp	400	0.75%
EC2	Deshpande et al. (2013)	PacBio		4000bp	30	13%
EC3	simulated	Illumina	37,000bp	75bp	0.5	4%

Table 3: **Properties of datasets used.**

Assembler	Number of scaffolds	Longest scaffold (kb)	Longest scaffold corr. (kb)	N50 (kb)	Err.	N50 corr. (kb)	LAP
(1) *Staphylococus aureus*, read sets SA1, SA2							
GAML	28	1191	1191	514	**0**	514	**−23.45**
ALLPATHS-LG	**12**	1435	**1435**	1092	**0**	**1092**	−25.02
MSR-CA	17	**2411**	1343	**2414**	3	1022	−26.26
(2) *Escherichia coli*, read sets EC1, EC2							
PacbioToCA	55	1533	1533	**957**	**0**	**957**	**−33.86**
GAML	29	1283	1283	653	**0**	653	−33.91
Cerulean	**21**	**1991**	**1991**	694	**0**	694	−34.18
(3) *Escherichia coli*, read sets EC1, EC2, EC3							
GAML	**4**	**4662**	**4661**	**4662**	**3**	**4661**	**−60.38**
Celera	19	4635	2085	4635	19	2085	−61.47

Table 4: **Comparison of assembly accuracy.**

2. combination of a fragment Illumina library and a long-read high-error-rate Pacific Biosciences library (EC1, EC2),
3. combination of a fragment Illumina library, a long-read high-error-rate Pacific Biosciences library, and a long jump Illumina library (EC1, EC2, EC3),

In each scenario, we use the short insert Illumina reads (SA1 or EC1) in Velvet with conservative settings to build the initial contigs and assembly graph. For the LAP score, we give all Illumina datasets weight 1 and the PacBio dataset weight 0.01. The results are summarized in Table 4. Note that none of the assemblers considered here can effectively run in all three of these scenarios, except for GAML.

3 Read Indexing and Related problems

In this chapter, we explore problems of indexing either a long string or collection of short strings in such way that we can quickly search for all positions of a short fixed length k-mer (typically with $k \leq 32$).

3.1 Fast String Matching Using Min-Hashing

In this section, we explore a problem of indexing a long string of length ℓ to quickly search for all locations of a fixed length k-mer in this string. Here we assume, that ℓ is very large and k is relatively small ($k \leq 32$).

Traditionally, this problem is solved in two ways:

- Directly indexing all k-mers (and their positions). All k-mers are stored either in a sorted array or in a hash table. The sorted array is usually more compact and hash table has faster retrieval time. We usually get fast constant query time in case of hash tables (if we assume k fits into the machine word), or logaritmic time in case of sorted arrays.

- Using suffix arrays or FM-index (Li and Durbin, 2009; Ferragina and Manzini, 2000). This approach usually takes much less space than indexing all k-mers. On the other hand, querying is slower. The advantage of FM-index is a possibility to trade query time for memory by adjusting the sampling rate s.

Even though all of these solutions have comparable asymptotic time and space complexities, a practical performance differs significantly (see Tables 5 and 6).

In this section, we propose an alternative index structure, *MH-index* (minhash-index), which is mainly inspired by the use of minimizers in KRAKEN (Wood and Salzberg, 2014) and minhashing used in Canu (Koren et al., 2017). Minimizers were also used for speeding up the de Bruijn graph construction (Li et al., 2013; Chikhi et al., 2014). MH-index does not exhibit good theoretical properties in the worst case, but is very efficient in practice.

We start first by definition of a minimizer of a k-mer and then describe how to use the minimizers for indexing k-mers in a string.

Definition 2 (Minimizer of a k-mer) *Given k-mer x, number m, and ordering O, a* minimizer of x *is a substring of x of length m with the minimal value in O. If there are multiple possible minimizers, we select the first one.*

One of the possible orderings is lexicographic, but this leads to undesirable properties, like very skewed distribution of minimizers (Wood and Salzberg, 2014). Instead, we use exclusive-or (XOR) operation with predefined constant c before comparing ordering of k-mers.

In our work, we exploit a crucial property of minimizers that adjacent k-mers are likely to share a common minimizer. First, we find a minimizer for each k-mer in the original string. Then we find a union of these minimizers and construct index only for the minimizers; for each minimizer value, we record positions of their occurences. We also store the original string in addition to the index.

Querying in MH-index is straighforward. Given query q of length k, we first find the minimizer of the query and then check each position indexed for the minimizer for the query match. Note that we use the position of the minimizer in the original query, thus we only examine a single substring for each minimizer hit.

In fact, checking the minimizer hits may seem like a computational bottleneck. For example, consider a string A^n and a query `AAAAAAAC`, where `AAAA` is the minimizer. In this string we would have to check almost every position in the original string and all the positions of minimizers are false positives.

Index	Memory consumption (MB)	Time per query (μs)
Hash table	426	0.4
Sorted array	72	0.5
FM-index (sampling 1)	15	3.6
FM-index (sampling 4)	5.2	4.0
MH-index ($m = 12$)	4.7	0.5

Table 5: **Comparison of indices on *Echerichia coli* dataset (4.6 Mb)**

Index	Memory consumption (MB)	Time per query (μs)
Hash table	7217	0.4
Sorted array	1293	0.5
FM-index (sampling 1)	325	3.6
FM-index (sampling 4)	107	4.0
MH-index ($m = 12$)	89	2.5
MH-index ($m = 15$)	100	1.4
MH-index ($m = 19$)	237	1.1

Table 6: **Comparison of indices on *Human* chromosome 14 dataset (88 Mb)**

In reality, typical DNA sequences do not exhibit this behaviour. Moreover, if k is small, checking the minimizer hit essentially requires only comparison of two 64-bit numbers.

Similarly to FM-index, MH-index exhibits time vs. memory tradeoff. By making the length of the minimizer larger, we use more memory, since k-mers would share less minimizers, but the querying time is faster, since there are less false positive hits from the minimizers.

Experiments We tested the MH-index on two DNA sequences: *Escherichia coli* K12 MG1665 (4.6 Mb long) and *Human* chromosome 14 (88Mb long). We compared indexes based on the hash table, the sorted array, the FM-index with sampling rates 1 and 4, and MH-index with several values of m. We recorded memory consumed by the index and query time for executing million queries of length $k = 30$. We chose half of the queries randomly from original string and the other half of the queries as random strings. Results are sumarized in Tables 5 and 6.

Experimental results show, that for smaller strings, we can achive small memory footprint comparable with FM-index and very fast query time comparable with hash tables. On larger string we are slightly slower than hash tables, but consume much smaller amount of memory. We are also faster than FM-index using comparable amount of memory. We also believe, than further research and better implemention can speed up MH-index even futher.

3.2 CR-index

Previous approaches (Philippe et al., 2011; Välimäki and Rivals, 2013) were optimized for collections that consist of randomly generated strings. Yet, in many applications (like our GAML framework), the collection contains reads that have large overlaps, and there can even be many identical reads. In this section, we propose a new data structure CR-index targeted at read collections that are randomly selected short substrings of a given template, sampled

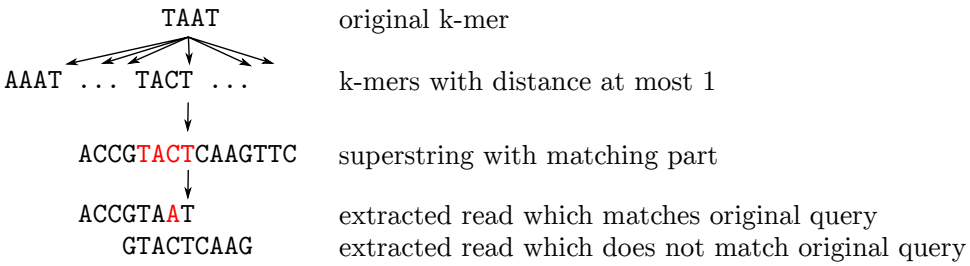

Figure 1: **Overview of the algorithm for answering CR-index query.**

to high coverage, with only a few differences compared to the template (e.g., Illumina reads from a given genome at 50× coverage and 1% error rate).

The main idea of CR-index is to use a *guide superstring* G, which contains all reads r_i from the collection R as substrings. The guide superstring is supplemented by additional structures allowing identification of IDs of all reads that align to a particular position in G. The guide superstring will be generally much shorter than the concatenation of all reads and since representation of this string accounts for most of the memory used in the previous indexing structures, it will be possible to reduce the memory footprint significantly on real data.

However, there is a problem with errors contained in the reads. Any read that has been changed compared to the original template will likely not align to the original template and thus the guide string would have to be significantly enlarged to also include all the reads with errors. On the other hand, allowing too many differences between guide string and reads would complicate the querying, since during query time we would have to test all possible differences. As a tradeoff, we decided that the best way is to allow only one difference per each k-mer, which gives following definition of the guide string:

Definition 3 (k-guide superstring) *For a given read collection R and number k, a* k-guide superstring *is a string G such that for each read $r \in R$ there exists a substring of G or a reverse complement of a substring of G, denoted s_r, such that any two differences between s_r and r are located more than k bases apart.*

Note that in this work, we allow only substitutions as differences between r and s_r. The query algorithm is illustrated in Figure 1. For a given query k-mer x, we search the guide string for all strings at Hamming distance at most one from x and from the reverse complement of x. This bound on Hamming distance is sufficient, because differences between r and s_r are more than k bases apart, and thus the query will overlap at most one difference between the guide string and the target read. After recovering all potential matching reads, we verify that each of them actually contains the original query x as a substring.

Even though this search algorithm is somewhat complicated due to the relaxed definition of the k-guide superstring, we gain significant improvements in memory. For example on *E. coli* dataset, we were able to construct exact superstring of length 224 Mbp and k-guide string of length 108 Mbp.

Experiments We compare the performance of our data structure with compressed Gk-arrays (Välimäki and Rivals, 2013) on two datasets. The first data set is the set of 151bp Illumina reads from *E. coli* strain MG1655 (genome length 4.7 Mbp, 184× coverage after

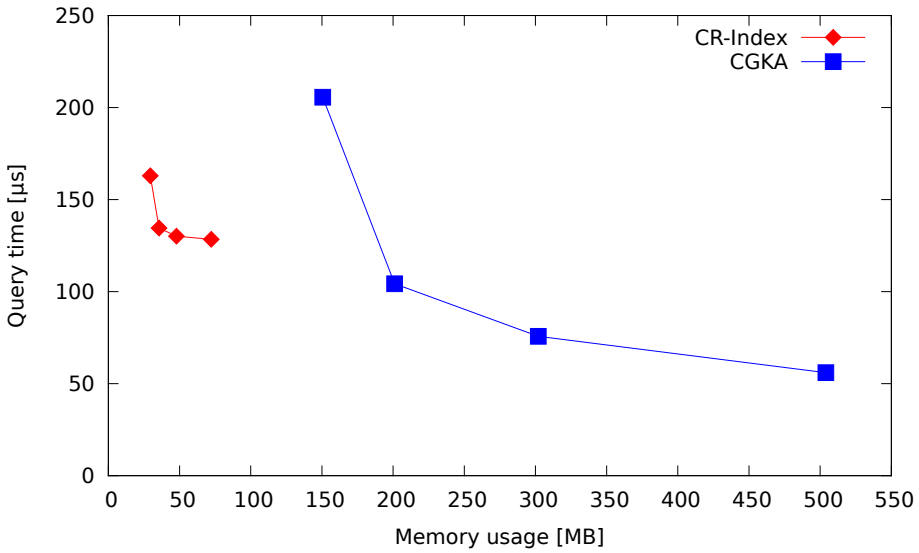

Figure 2: **Comparison of query time and memory usage of CR-index and compressed Gk-arrays on the *E. coli* dataset with coverage 50 for varying suffix array sampling rates.**

removal of low-quality reads, 0.75% error rate) (Illumina). The second data set is a set of 101bp Illumina reads from human chromosome 14 (sequence length 107 Mbp, 23× coverage, 1.5% error rate) from Genome Assembly Gold-Standard Evaluations (library 1; Salzberg et al. (2012)). In all experiments, we have used query length $k = 15$.

On the human chromosome 14 (23× coverage of 107 Mbp sequence, 1.5% error rate), the CR-index requires only 571 MB of memory, while Gk-arrays require ≈ 1.7 GB. On the other hand, a typical query takes 214 ms in CR-index, or roughly 3× longer than in Gk-arrays (71 ms). We believe that this is due to higher error rate in the data which may cause the Bloom filter to filter out fewer queries.

Both CR-index and compressed Gk-arrays use sampling of the suffix array. This allows time vs. memory tradeoff, since by sampling more values, we get a better running time at the expense of higher memory usage. Figure 2 shows this tradeoff at coverage 50 in the *E. coli* dataset. CR-index is much less sensitive to sampling parameters than compressed Gk-arrays. This is due to compressed superstring representation which results in a smaller FM-index and a smaller number of hits in FM-index during the search.

4 Base Calling and Indexing Nanopore Reads

In this chapter, we study problems directly related to DNA sequencing, in particular to MinION nanopore sequencing device. The MinION device by Oxford Nanopore (Mikheyev and Tin, 2014), weighing only 90 grams, is currently the smallest high-throughput DNA sequencer.

Although MinION is able to produce long reads, data produced by the platform exhibit a rather high sequencing error rate.

From the computer science perspective, we are mostly looking at problems of sequence to sequence prediction and indexing and comparing sequences of real valued numbers.

	E. coli training	*E. coli* testing	*K. pneumoniae* testing
# of reads	3,803	3,942	13,631
# of events	26,403,434	26,860,314	70,827,021

Table 7: **Sizes of experimental data set.**

In the MinION device, single-stranded DNA fragments move through nanopores. At each pore electric current,affected by the passing DNA fragment, is measured thousands times per second. The electric current depends mostly on the context of several DNA bases passing through the pore at the time of measurement. As the DNA moves through the pore, the context shifts, and measured signal changes. Based on these changes, the sequence of measurements is split into *events*, each event ideally representing the shift of the context by one base. Each event is summarized by mean, variance and duration. This sequence of events is then translated into a DNA sequence by a base caller.

4.1 Base Calling Using Recurrent Neural Networks

Original basecaller for MinION was based on Hidden markov models. We developed our own basecaller DeepNano, based on recurrent neural networks.

Interesting problem, was generating correct labels for supervised learning, since we do not know the correct output sequence; more specifically, we only know the region of the reference sequence where the read is aligned, but we do not know the exact pairs of output bases for individual events. We solve this problem in an EM-like fashion. First, we create an approximate alignment between the events and the reference sequence using a simple heuristic. After each hundredth pass through the whole data set, we realign the events to the reference sequence. We score the alignment by computing the log-likelihood of bases aligned to each event in the probability distribution produced by the current version of the network. Each event can get zero, one or two bases aligned to it. Score of an alignment for each event is sum of log of output probabalities (either for aligned bases or for empty bases). To find the alignment with maximum likelihood, we use a simple dynamic programming.

For experimental evaluation, we have used existing data sets from *Escherichia coli* (Loman et al., 2015a) and *Klebsiella pneumoniae* (WTC Human Genetics, 2016) produced by the SQK-MAP006 sequencing protocol with R7.3 flow cells. We have only used the reads that passed the original base calling process. We have also omitted reads that did not map to the reference sequence.

We have split the *E. coli* data set into training and testing portions; the training set contains the reads mapping to the first 2.5 Mbp of the genome. We have tested the predictors on reads which mapped to the rest of the *E. coli* genome and on reads from *K. pneumoniae*. Basic statistics of the two data sets are shown in Table 7.

We have compared our base calling accuracy with the accuracy of the original Metrichor base caller and with Nanocall. The main experimental results are summarized in Table 8. We see that our base caller is significantly better in both data sets.

	E. coli	*K. pneumoniae*
Metrichor	71.3%	68.1%
Nanocall	68.3%	67.5%
DeepNano	77.9%	76.3%

Table 8: **Accuracy of base callers on two testing data sets for R7.3 MinION data.**

4.2 Fast Indexing and Alignment of MinION Data

One of the MinION applications mentioned is selective sequencing, where we only sequence "interesting" reads by rejecting all other reads after reading the first few hundred bases. The idea was mostly explored in Readuntil tool (Loose et al., 2016).

Hard part of selective sequencing is deciding which reads to reject. Standard algorithm for selective sequencing would be to base call the first few hundreds of events from the read and then align them to the reference sequence. After aligning, we would accept the read if we find a match with the acceptable similarity.

Unfortunately, the speed of the base calling algorithm on a reasonable computer is much lower than the rate at which the sequencer produces the data. This means that we need to use other approaches than base calling followed by alignment. We will explore an approach which does not translate electric signal into DNA, but instead it works directly with the electric signal data.

Standard approach for comparing two signal sequences, is Dynamic time warping (DTW) (Sankoff and Kruskal, 1983). We demonstrate that using DTW and application of thresholding on its matching cost does not have enough discriminative power to distinguish false matches from true matches. We hypothesize that the main problem lies in determining correct scaling and shift of the data (which is hard on small number of events). We test this experimentaly by using scaling and shift parameters determined from the whole read and our experiments agree with the hypothesis. Then we propose a variant of DTW algorithm and demonstrate that it has much better discriminative power than the original DTW algorithm.

Applying DTW to Selective Sequencing Before running DTW on MinION data, we first have to determine correct scaling and shift of the data. Standard procedure for this is Z-score normalization, i.e. normalizing data mean to zero and variance to one, as used in Readuntil (Loose et al., 2016).

After finding the best matching, we have to decide based on the score, whether the read matches the target sequence or whether the score only represents a spurious match of the query to the target sequence (a false positive). This is usually done by simply thresholding on the score. We will call this setup a *naive scaling*.

In the following sections, we demostrate that the naive scaling has very low specificity when higher sensitivity is required. This causes problems especially when we are trying to match data to a longer reference sequence, where many false positive matches can occur.

Here, we propose a simple heuristics, which greatly improves the tradeoff between sensitivity and specificity. If the alignment of the query was fixed, we could easily find better scaling parameters by looking at the sequence of matched pairs and setting scaling and shift to minimize the matching cost.

Once we have better scaling parameters, it is possible to improve the matching by running

Algorithm variant	Sensitivity	Specificity
Baseline	90%	99.72%
Naive	90%	99.88%
Two iteration rescaling	90%	99.92%
Baseline	95%	99.64%
Naive	95%	99.45%
Two iteration rescaling	95%	99.86%
Baseline	98%	99.46%
Naive	98%	90.74%
Two iteration rescaling	98%	99.46%
Baseline	99%	98.88%
Naive	99%	47.30%
Two iteration rescaling	99%	97.20%

Table 9: Comparison of specificity of several scaling variants on prespecified sensitivity levels.

the DTW algorithm on newly rescaled signal. In fact, we could run several iterations of rescaling and DTW to improve the matching and the scaling parameters. We call this approach *iterative scaling*.

In the next section, we compare the performance of the naive scaling and our new iterative rescaling method.

4.3 Experimental Evaluation

In our experiments we have used R9 Ecoli dataset from Loman Labs (Loman, 2016) using only reads with and alignment to the reference and longer than 3000 bases.

As a query, we have used events number 100 to 350 from each read. In each experiment, we run each method for each read against aligned location and against hundred random 500 bp long locations from the genome. After running each method, we study specificity-sensitivity tradeoff of each method by varying the threshold for the score. For each method, we set the threshold to achieve a predefined sensitivity (how many times DTW accepted a match of the read to the correct location) and measure specificity (how many times DTW rejected a match of the read to a random location).

We compared the following methods:

- Z-score normalization using complete reads (on more than 3000 events), followed by DTW (on 250 events). This is an unrealistic scenario in which we effectively know the correct scaling parameters. We will use this method as a *baseline* for comparison.
- Naive scaling, which represents the state of the art, consisting of Z-score normalization, followed by DTW.
- Two iteration rescaling, consisting of Z-score normalization, followed by DTW, rescaling, and another DTW.

Table 9 shows comparison of these scaling variants.

We can see that at higher sensitivities, the naive variant performs much worse than rescaled variants of DTW. This is especially visible at 99% sensitivity level, where naive variant treats

every second false match as a good match. Also at 99% sensitivity we see benefit of two iteration rescaling, which discards many more false positives than simple rescaling. A consistently strong baseline performance suggests that the scaling is a big problem when using DTW and need to be addressed before the method can be widely applicable. Our approach seems like a step in a good direction to solve the problem.

5 Author's Publications and Citations

1. Boža, Vladimír, Broňa Brejová, and Tomáš Vinař. "DeepNano: deep recurrent neural networks for base calling in MinION nanopore reads." arXiv preprint arXiv:1603.09195 (2016). Cited in:

    - Cao, M. D. - Ganesamoorthy, D. - Elliott, A. G. - Zhang, H. H. - Cooper, M. A. - Coin, L. J. M.: Streaming algorithms for identification of pathogens and antibiotic resistance potential from real-time MinION (TM) sequencing. In: GigaScience, Vol. 5, 2016, Art. No. 32 - SCI ; SCOPUS
    - Chua, E. W. - Ng, P. Y.: MinION: A novel tool for predicting drug hypersensitivity?. In: Frontiers in Pharmacology, Vol. 7, 2016, Art. No. 156 - SCI ; SCOPUS
    - Jain, M. - Olsen, H. E. - Paten, B. - Akeson, M.: The Oxford Nanopore MinION: delivery of nanopore sequencing to the genomics community. In: Genome Biology, Vol. 17, 2016, Art. No. 239 - SCI ; SCOPUS
    - Taylor, D. - Powers, D.: Teaching artificial intelligence to read electropherograms. In: Forensic Science International: Genetics, Vol. 25, 2016, S. 10-18 - SCI ; SCOPUS
    - Walter, M. C. - Zwirglmaier, K. - Vette, P. - Holowachuk, S. A. - Stoecker, K. - Genzel, G. H. - Antwerpen, M. H.: MinION as part of a biomedical rapidly deployable laboratory. In: Journal of Biotechnology, 2016,DOI:10.1016/j.jbiotec.2016.12.006
    - Cao, M. D. - Nguyen, S. H. - Ganesamoorthy, D. - Elliott, A. G. - Cooper, M. A. - Coin, L. J. M.: Scaffolding and completing genome assemblies in real-time with nanopore sequencing. In: Nature Communications, Vol. 8, 2017, Art. No.14515 - SCI ; SCOPUS
    - Carr, Ch. E. - Mojarro, A. - Hachey, J. - Saboda, K. - Tani, J. - Bhattaru, S. A. - Smith, A. - Pontefract, A. - Zuber, M. - Finney, M. - Doebler, R. - Brown, M. - Herrington, K. - Talbot, R. - Nguyen, V. - Bailey, R. - Ferguson, T.- Church, G. - Ruvkun, G.: Towards In Situ Sequencing for Life Detection. In: 2017 IEEE Aerospace Conference. New York : IEEE, 2017, S. 15
    - Chu, J. - Mohamadi, H. - Warren, R. L. - Yang, Ch. - Birol, I.: Innovations and challenges in detecting long read overlaps: an evaluation of the state-of-the-art. In: Bioinformatics, Vol. 33, No. 8, 2017, s. 1261-1270
    - Kamenova, S. - Bartley, T. J. - Bohan, D. A. - Boutain, J. R. - Colautti, R. I. - Domaizon, I. - Fontaine, C. - Lemainque, A. - Le Viol, I. - Mollot, G. - Perga, M. E. - Ravigne, G. - Massol, F.: Invasions toolkit: current methodsfor tracking the spread and impact of invasive species. In: Networks of Invasion: A Synthesis of Concepts : Advances in Ecological Research, Vol. 56. Dordrecht : Elsevier, 2017, S. 65

2. Boža, Vladimír, et al. "Fishing in Read Collections: Memory Efficient Indexing for Sequence Assembly." International Symposium on String Processing and Information Retrieval. Springer International Publishing, 2015.
3. Boža, Vladimír, Broňa Brejová, and Tomáš Vinař. "GAML: genome assembly by maximum likelihood." Algorithms for Molecular Biology 10.1 (2015): 18. Cited in:
    - Rahman, A. H.: Statistical models for genome assembly and analysis. Berkeley : University of California, 2015, S. 89
    - Luhmann, N. - Thevenin, A. - Ouangraoua, A. - Wittler, R. - Chauve, C.: The SCJ Small Parsimony Problem for Weighted Gene Adjacencies. In: Bioinformatics Research and Applications : Lecture Notes in Bioinformatics, Vol. 9683. Cham :Springer, 2016, s. 200-210 - CPCI-S ; SCOPUS
    - Muhindira, P. V.: A novel approach for the assembly of complex genomic DNA cloned into bacterial artificial chromosome vectors: Assembly and analysis of Triticum aestivum chromosome arm 7DS. Brisbane : University of Queensland, 2016,S. 145
    - Luhmann, N.: Phylogenetic assembly of paleogenomes integrating ancient DNA data. Bielefeld : University, 2017, S. 121
    - Simpson, J. T. - Pop, M.: The Theory and Practice of Genome Sequence Assembly. In: Annual Review of Genomics and Human Genetics, Vol. 16, 2015, s. 153-172 - SCOPUS
4. Boža, Vladimír. "Experimental Comparison of Set Intersection Algorithms for Inverted Indexing." (2013). Cited in:
    - Kudo, M.: Attacks against search Poly-LWE. In: Cryptology ePrint Archive, Art. No. 1153, 2016, s. 28
    - Fort, M. - Sellares, J. A. - Valladares, N.: Intersecting two families of sets on the GPU. In: Journal of Parallel and Distributed Computing, Vol. 104, 2017, s. 178

References

Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

Austin Appleby. Murmurhash. `https://code.google.com/p/smhasher/wiki/MurmurHash`, 2008.

Joel Baker. De bruijn graphs and their applications to fault tolerant networks. Master's thesis, Oregon State University, 2011. 1, 2011.

Timo Beller, Simon Gog, Enno Ohlebusch, and Thomas Schnattinger. Computing the longest common prefix array based on the burrows–wheeler transform. *Journal of Discrete Algorithms*, 18:22–31, 2013.

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, pages 3–10, 2010.

Konstantin Berlin, Sergey Koren, Chen-Shan Chin, James Drake, Jane M Landolin, and Adam M Phillippy. Assembling large genomes with single-molecule sequencing and locality sensitive hashing. *bioRxiv*, page 008003, 2014.

Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

Avrim Blum, Tao Jiang, Ming Li, John Tromp, and Mihalis Yannakakis. Linear approximation of shortest superstrings. *Journal of the ACM (JACM)*, 41(4):630–647, 1994.

Vladimır Boža, Brona Brejová, and Tomáš Vinař. Gaml: Genome assembly by maximum likelihood. In *Algorithms in Bioinformatics (WABI)*, volume 8701 of *Lecture Notes in Bioinformatics*, page 122. Springer, 2014.

Vladimír Boža, Jakub Jursa, Broňa Brejová, and Tomáš Vinař. Fishing in read collections: Memory efficient indexing for sequence assembly. In *International Symposium on String Processing and Information Retrieval*, pages 188–198. Springer, 2015.

Vladimír Boža, Broňa Brejová, and Tomáš Vinař. DeepNano: Deep Recurrent Neural Networks for Base Calling in MinION Nanopore Reads. Technical Report arXiv:1603.09195, arXiv preprints, 2016.

James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. Quasi-recurrent neural networks. *arXiv preprint arXiv:1611.01576*, 2016.

Andrei Z Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29. IEEE, 1997.

Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659, 2000.

Mark J Chaisson and Glenn Tesler. Mapping single molecule sequencing reads using basic local alignment with successive refinement (blasr): application and theory. *BMC Bioinformatics*, 13(1):238, 2012.

Rayan Chikhi, Guillaume Rizk, et al. Space-efficient and exact de bruijn graph representation based on a bloom filter. In *WABI*, pages 236–248, 2012.

Rayan Chikhi, Antoine Limasset, Shaun Jackman, Jared T Simpson, and Paul Medvedev. On the representation of de bruijn graphs. In *International Conference on Research in Computational Molecular Biology*, pages 35–55. Springer, 2014.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. Technical Report 1412.3555, arXiv, 2014.

David Clark. *Compact Pat Trees*. PhD thesis, University of Waterloo, 1998.

Scott C. Clark, Rob Egan, Peter I. Frazier, and Zhong Wang. ALE: a generic assembly likelihood evaluation framework for assessing the accuracy of genome and metagenome assemblies. *Bioinformatics*, 29(4):435–443, 2013.

Matei David, Lewis Jonathan Dursi, Delia Yao, Paul C. Boutros, and Jared T. Simpson. Nanocall: An Open Source Basecaller for Oxford Nanopore Sequencing Data. Technical Report doi:10.1101/046086, bioRxiv, 2016.

Nicolaas Govert de Bruijn and Paul Erdos. A combinatorial problem. *Koninklijke Nederlandse Akademie v. Wetenschappen*, 49(49):758–764, 1946.

Arthur L Delcher, Adam Phillippy, Jane Carlton, and Steven L Salzberg. Fast algorithms for large-scale genome alignment and comparison. *Nucleic Acids Research*, 30(11):2478–2483, 2002.

Viraj Deshpande, Eric DK Fung, Son Pham, and Vineet Bafna. Cerulean: A hybrid assembly using high throughput short and long reads. In *Algorithms in Bioinformatics (WABI)*, volume 8126 of *LNCS*, pages 349–363. Springer, 2013.

RW Eglese. Simulated annealing: a tool for operational research. *European Journal of Operational Research*, 46(3):271–281, 1990.

Adam C English, Stephen Richards, et al. Mind the gap: upgrading genomes with Pacific Biosciences RS long-read sequencing technology. *PLoS One*, 7(11):e47768, 2012.

Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 390–398. IEEE, 2000.

Paolo Ferragina, Rodrigo González, Gonzalo Navarro, and Rossano Venturini. Compressed text indexes: From theory to practice. *Journal of Experimental Algorithmics (JEA)*, 13:12, 2009.

John Gallant, David Maier, and James Astorer. On finding minimal length superstrings. *Journal of Computer and System Sciences*, 20(1):50–58, 1980.

Michael R Garey and David S Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979. ISBN 0716710447.

Mohammadreza Ghodsi, Christopher M Hill, Irina Astrovskaya, Henry Lin, Dan D Sommer, Sergey Koren, and Mihai Pop. De novo likelihood-based measures for comparing genome assemblies. *BMC Research Notes*, 6(1):334, 2013.

Sante Gnerre, Iain MacCallum, et al. High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proceedings of the National Academy of Sciences*, 108(4):1513–1518, 2011.

Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri. From theory to practice: Plug and play with succinct data structures. In *Symposium on Experimental Algorithms (SEA)*, volume 8504 of *Lecture Notes in Computer Science*, pages 326–337, 2014.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Szymon Grabowski and Marcin Raniszewski. Sampling the suffix array with minimizers. In *International Symposium on String Processing and Information Retrieval*, pages 287–298. Springer, 2015.

Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer-Verlag, Heidelberg, Germany, 2012.

Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 841–850. Society for Industrial and Applied Mathematics, 2003.

Alexey Gurevich, Vladislav Saveliev, Nikolay Vyahhi, and Glenn Tesler. Quast: quality assessment tool for genome assemblies. *Bioinformatics*, 29(8):1072–1075, 2013.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Dark knowledge. *Presented as the keynote in BayLearn*, 2014.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. Technical Report 1503.02531, arXiv, 2015.

Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Weichun Huang, Leping Li, Jason R Myers, and Gabor T Marth. ART: a next-generation sequencing read simulator. *Bioinformatics*, 28(4):593–594, 2012.

Martin Hunt, Taisei Kikuchi, Mandy Sanders, Chris Newbold, Matthew Berriman, and Thomas D Otto. Reapr: a universal tool for genome assembly evaluation. *Genome Biology*, 14(5):R47, 2013.

Illumina. *E.coli* MG1655 Illumina sequencing dataset. ftp://webdata:webdata@ussd-ftp.illumina.com/Data/SequencingRuns/MG1655/MiSeq_Ecoli_MG1655_110721_PF.bam, 2015. Accessed: 2015-03-03.

Guy Joseph Jacobson. *Succinct static data structures*. PhD thesis, 1988.

G David Forney Jr. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

Kim Judge, Simon R Harris, Sandra Reuter, Julian Parkhill, and Sharon J Peacock. Early insights into the potential of the Oxford Nanopore MinION for the detection of antimicrobial resistance genes. *Journal of Antimicrobial Chemotherapy*, 70(10):2775–2778, 2015.

M Frans Kaashoek and David R Karger. Koorde: A simple degree-optimal distributed hash table. In *International Workshop on Peer-to-Peer Systems*, pages 98–107. Springer, 2003.

Haim Kaplan and Nira Shafrir. The greedy algorithm for shortest superstrings. *Information Processing Letters*, 93(1):13–17, 2005.

Juha Kärkkäinen and Peter Sanders. Simple linear work suffix array construction. In *International Colloquium on Automata, Languages, and Programming*, pages 943–955. Springer, 2003.

John D Kececioglu and Eugene W Myers. Combinatorial algorithms for dna sequence assembly. *Algorithmica*, 13(1-2):7–51, 1995.

John Dimitri Kececioglu. *Exact and approximation algorithms for DNA sequence reconstruction*. PhD thesis, 1991.

David R Kelley, Michael C Schatz, Steven L Salzberg, et al. Quake: quality-aware detection and correction of sequencing errors. *Genome Biology*, 11(11):R116, 2010.

W James Kent. Blat—the blast-like alignment tool. *Genome Research*, 12(4):656–664, 2002.

Sergey Koren, Michael C Schatz, Brian P Walenz, Jeffrey Martin, Jason T Howard, Ganeshkumar Ganapathy, Zhong Wang, David A Rasko, W Richard McCombie, Erich D Jarvis, et al. Hybrid error correction and de novo assembly of single-molecule sequencing reads. *Nature Biotechnology*, 30(7):693–700, 2012.

Sergey Koren, Brian P Walenz, Konstantin Berlin, Jason R Miller, Nicholas H Bergman, and Adam M Phillippy. Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *bioRxiv*, page 071282, 2017.

Peter Kuljovský. Efficient substring search in long sequence. slovak: Efektívne vyhľadávanie krátkych reťazcov v dlhej sekvencii. Bachelor's thesis, Comenius University, 2016.

Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature Methods*, 9(4): 357–359, 2012.

C Lee Giles, Gary M Kuhn, and Ronald J Williams. Dynamic recurrent neural networks: Theory and applications. *IEEE Transactions on Neural Networks*, 5(2):153–156, 1994.

Heng Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. Technical Report 1303.3997, arXiv, 2013.

Heng Li and Richard Durbin. Fast and accurate short read alignment with burrows–wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

Yang Li, Pegah Kamousi, Fangqiu Han, Shengqi Yang, Xifeng Yan, and Subhash Suri. Memory efficient minimum substring partitioning. In *Proceedings of the VLDB Endowment*, volume 6, pages 169–180. VLDB Endowment, 2013.

Yu Lin, Jeffrey Yuan, Mikhail Kolmogorov, Max W Shen, Mark Chaisson, and Pavel A Pevzner. Assembly of long error-prone reads using de bruijn graphs. *Proceedings of the National Academy of Sciences*, 113(52): E8396–E8405, 2016.

Dong C Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1-3):503–528, 1989.

Lin Liu, Yinhu Li, Siliang Li, Ni Hu, Yimin He, Ray Pong, Danni Lin, Lihua Lu, and Maggie Law. Comparison of next-generation sequencing systems. *BioMed Research International*, 2012, 2012.

Nicholas J Loman. `https://s3.climb.ac.uk/nanopore/E_coli_K12_1D_R9.2_SpotON_2.tgz`, 2016.

Nicholas J Loman, Joshua Quick, and Jared T Simpson. `http://www.ebi.ac.uk/ena/data/view/ERR1147230`, 2015a.

Nicholas J Loman, Joshua Quick, and Jared T Simpson. A complete bacterial genome assembled de novo using only nanopore sequencing data. *Nature Methods*, 12(8):733–735, 2015b.

Matthew Loose, Sunir Malla, and Michael Stout. Real-time selective sequencing using nanopore technology. *Nature Methods*, 13(9):751–754, 2016.

Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. *Siam Journal on Computing*, 22(5):935–948, 1993.

Paul Medvedev, Son Pham, Mark Chaisson, Glenn Tesler, and Pavel Pevzner. Paired de bruijn graphs: a novel approach for incorporating mate pair information into genome assemblers. *Journal of Computational Biology*, 18(11):1625–1634, 2011.

Alexander S Mikheyev and Mandy MY Tin. A first look at the Oxford Nanopore MinION sequencer. *Molecular Ecology Resources*, 14(6):1097–1102, 2014.

Jason R Miller, Sergey Koren, and Granger Sutton. Assembly algorithms for next-generation sequencing data. *Genomics*, 95(6):315–327, 2010.

Satomi Mitsuhashi, Kirill Kryukov, So Nakagawa, Junko S Takeuchi, Yoshiki Shiraishi, Koichiro Asano, and Tadashi Imanishi. A portable system for metagenomic analyses using nanopore-based sequencer and laptop computers can realize rapid on-site determination of bacterial compositions. *bioRxiv*, page 101865, 2017.

Eugene W Myers. Toward simplifying and accurately formulating fragment assembly. *Journal of Computational Biology*, 2(2):275–290, 1995.

Eugene W Myers. Efficient local alignment discovery amongst noisy long reads. In *Algorithms in Bioinformatics*, pages 52–67. Springer, 2014.

Eugene W Myers, Granger G Sutton, Art L Delcher, Ian M Dew, Dan P Fasulo, Michael J Flanigan, Saul A Kravitz, Clark M Mobarry, Knut HJ Reinert, Karin A Remington, et al. A whole-genome assembly of drosophila. *Science*, 287(5461):2196–2204, 2000.

Niranjan Nagarajan, Timothy D Read, and Mihai Pop. Scaffolding and validation of bacterial genome assemblies using optical restriction maps. *Bioinformatics*, 24(10):1229–1235, 2008.

Alexis L Norris, Rachael E Workman, Yunfan Fan, James R Eshleman, and Winston Timp. Nanopore sequencing detects structural variants in cancer. *Cancer Biology & Therapy*, 17(3):246–253, 2016.

Daisuke Okanohara and Kunihiko Sadakane. Practical entropy-compressed rank/select dictionary. In *Workshop on Algorithms Engineering and Experiments (ALENEX)*, pages 60–70. SIAM, 2007.

Jaroslav Petrucha. Efficient representation of set of short strings. slovak: Efektívna reprezentácia množiny krátkych reťazcov. Master's thesis, Bachelor's thesis, Comenius University, 2016.

Pavel A Pevzner, Haixu Tang, and Michael S Waterman. An eulerian path approach to dna fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, 2001.

Nicolas Philippe, Mikael Salson, Thierry Lecroq, Martine Leonard, Therese Commes, and Eric Rivals. Querying large read collections in main memory: a versatile data structure. *BMC Bioinformatics*, 12(1):242, 2011.

Michael A Quail, Miriam Smith, Paul Coupland, Thomas D Otto, Simon R Harris, Thomas R Connor, Anna Bertoni, Harold P Swerdlow, and Yong Gu. A tale of three next generation sequencing platforms: comparison of Ion Torrent, Pacific Biosciences and Illumina MiSeq sequencers. *BMC Genomics*, 13(1):341, 2012.

Joshua Quick, Philip Ashton, Szymon Calus, Carole Chatt, Savita Gossain, Jeremy Hawker, Satheesh Nair, Keith Neal, Kathy Nye, Tansy Peters, Elizabeth De Pinna, Esther Robinson, Keith Struthers, Mark Webber, Andrew Catto, Timothy J Dallman, Peter Hawkey, and Nicholas J Loman. Rapid draft sequencing and real-time nanopore sequencing in a hospital outbreak of Salmonella. *Genome Biology*, 16:114, 2015.

Joshua Quick, Nicholas J Loman, Sophie Duraffour, and Jared T Simpson et al. Real-time, portable genome sequencing for Ebola surveillance. *Nature*, 530(7589):228–232, 2016.

Atif Rahman and Lior Pachter. CGAL: computing genome assembly likelihoods. *Genome Biology*, 14(1):R8, 2013.

Steven L Salzberg, Adam M Phillippy, Aleksey Zimin, Daniela Puiu, Tanja Magoc, Sergey Koren, Todd J Treangen, Michael C Schatz, Arthur L Delcher, Michael Roberts, et al. Gage: A critical evaluation of genome assemblies and assembly algorithms. *Genome Research*, 22(3):557–567, 2012.

David Sankoff and Joseph B Kruskal. Time warps, string edits, and macromolecules: the theory and practice of sequence comparison. *Reading: Addison-Wesley Publication, 1983, edited by Sankoff, David; Kruskal, Joseph B.*, 1, 1983.

Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45(11):2673–2681, 1997.

Jared T Simpson and Richard Durbin. Efficient construction of an assembly string graph using the FM-index. *Bioinformatics*, 26(12):i367–i373, 2010.

Jared T Simpson, Kim Wong, Shaun D Jackman, Jacqueline E Schein, Steven JM Jones, and Inanç Birol. Abyss: a parallel assembler for short read sequence data. *Genome Research*, 19(6):1117–1123, 2009.

Jared T Simpson, Rachael Workman, Philip C Zuzarte, Matei David, Lewis Jonathan Dursi, and Winston Timp. Detecting dna methylation using the oxford nanopore technologies minion sequencer. *bioRxiv*, page 047142, 2016.

Martin Smith. Nanoporetech community. `https://community.nanoporetech.com/posts/ultra-long-read-validation`, 2016.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1139–1147, 2013.

Z Sweedyk. A 2.5-approximation algorithm for shortest superstring. *SIAM Journal on Computing*, 29(3): 954–986, 2000.

Niko Välimäki and Eric Rivals. Scalable and versatile k-mer indexing for high-throughput sequencing data. In *Bioinformatics Research and Applications (ISBRA)*, Lecture Notes in Bioinformatics, pages 237–248. Springer, 2013.

Zhong Wang, Mark Gerstein, and Michael Snyder. Rna-seq: a revolutionary tool for transcriptomics. *Nature reviews genetics*, 10(1):57–63, 2009.

Derrick E Wood and Steven L Salzberg. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome Biology*, 15(3):R46, 2014.

WTC Human Genetics. MinION sequence data for clinical gram-negatives. `https://www.ebi.ac.uk/ena/data/view/SAMEA3713789`, 2016.

Daniel R Zerbino and Ewan Birney. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research*, 18(5):821–829, 2008.