University of South Bohemia in České Budějovice

Faculty of Science

and

Johannes Kepler University in Linz

Faculty of Engineering and Natural Sciences

De novo genome assembly of Arthrobacter sp. isolated from arctic permafrost soil

Bachelor Thesis

Mariana Šatrová

Supervisor: Ing. Jiří Bárta, Ph.D.

České Budějovice 2017

Šatrová, M., 2017: De novo genome assembly of Arthrobacter sp. isolated from arctic permafrost soil. Bc. Thesis, in English. – 34 p., Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic and Faculty of Engineering and Natural Sciences, Johannes Kepler University, Linz, Austria.

ANNOTATION

A draft genome was assembled from newly sequenced Arthrobacter species isolated from arctic soil. The raw sequences were analyzed and their statistics discussed in great depth. Several assemblers were tested and compared for results. The best assembly (the final draft genome) was then uploaded to RAST server and annotated.

I hereby declare that I have worked on my bachelor thesis independently and used only the sources listed in the bibliography. I hereby declare that, in accordance with Article 47b of Act No. 111/1998 in the valid wording, I agree with the publication of my bachelor thesis, in full form to be kept in the Faculty of Science archive, in electronic form in publicly accessible part of the STAG database operated by the University of South Bohemia in České Budějovice accessible through its web pages.

Further, I agree to the electronic publication of the comments of my supervisor and thesis opponents and the record of the proceedings and results of the thesis defense in accordance with aforementioned Act No. 111/1998. I also agree to the comparison of the text of my thesis with the Theses.cz thesis database operated by the National Registry of University Theses and a plagiarism detection system.

In České Budějovice 19.4.2017

.....
Signature

ACKNOWLEDGEMENTS

I would like to first and foremost thank my supervisor Jiří Bárta for providing the data and for his guidance. My big gratitude also goes to Alois Regl who gave me the solid background in genomic assembly and to Jan Petrásek for technical support with Hyper-V and Windows server. Finally, I want to thank the whole bioinformatics office at JKU, especially Sepp, Gundi, Tom, and Boyang for their help with navigating through my studies and Ulrich for his inspiring lectures - they all have contributed to my enthusiasm for bioinformatics.

Contents

1. Introduction

In this work, the draft genome of Arthrobacter species that was isolated in arctic permafrost soil was assembled and annotated. The provided data consisted of forward and reverse reads sequenced with Illumina sequencer. The reads were inspected and analyzed with FastQC and low quality reads were trimmed. For the assembly, the MyPro and A5-miseq pipelines that are designed for de novo prokaryotic assembly were used. A5-miseq has its own assembler, automatic pre-processing of the reads and can work with paired-end data. In contrast to A5-miseq, MyPro is a virtual machine with Bio-Linux in which following assemblers are installed: SOAPdenovo, Abyss, SPAdes, Velvet, and Edena. When running the Autorun.py in MyPro, it performs quality trimming of the reads, runs the assemblers, integrates the resulting assemblies, makes a post-processing on the resulted integrated assembly (filling in gaps in the assembly) and finally, performs annotation. In this study only the best resulting assembly (456 contigs, N50: 48,680) was annotated and the annotation was done on RAST server.

This work also offers a brief overview of the process of assembly. My motivation to learn about this subject stems from the fact that the data produced in biology nowadays greatly exceeds our capacity for analyzing them. The number of sequenced genomes grows rapidly each year as the sequencers become more time and cost efficient. Though there are efforts to create fully automated pipelines that are simple to use without any knowledge of bioinformatics, as I learned during this work a lot of obstacles can arise, overcoming of which requires some skills, experience, and knowledge of the process.

The Arctic offers possibilities for many new discoveries and its crude environment hides ingenious ways of survival. One way we can learn about a microbe's life is through its genetic information, through understanding its metabolic pathways, proteins and RNAs with special functions whose blueprint is encoded in DNA. By reconstructing the genome of the Arthrobacter species I hope to give biologists the basis for better understanding of its biology and perhaps even contributing to building a better picture about the ecology of the Arctic.

2. Literary Review

2.1. Introduction to whole genome sequencing and assembly

The elementary problem of sequencing a whole genome is that today's technologies are not capable of sequencing it in one piece. One of the defining properties of different sequencers is their read length, i.e. the number of consecutive bases that it can sequence. This read length of commercial second-generation technologies is much shorter, than is the length of even the smallest genome (Miller, Koren, & Sutton, 2010). This problem is solved by splitting the genome into many smaller fragments that can be then sequenced. To reconstruct the information about the genome's sequence, these sequenced parts need to be put back together in the correct order by a process called assembly. On a very basic level, assembly can be thought of as putting together a large jigsaw puzzle with the help of complex computer algorithms.

The method most used for sequencing of genomes is whole genome shotgun sequencing. This method results in creation of many fragments with randomly selected positions on the chromosome. To make the assembly possible, two key conditions must be met. Firstly, the fragments must be overlapping and secondly, there must be a high coverage (Miller, Koren, & Sutton, 2010). High coverage means that each position on the chromosome must be present in more reads. Formally, coverage is defined as follows:

$$c = \frac{N \cdot r}{G}$$

Where c = Coverage, N = Number of reads, r = Mean read length, and G = Haploid genome length ("Estimating Sequencing Coverage", 2017). During the assembly, reads are aligned to find overlaps between them so that they can be connected into longer contiguous sequences (contigs).

Which sequencer is used to acquire the reads is an important information for genomic assembly. Different sequencers correspond to different read lengths and different sequencing errors that assemblers need to work with. Ideally, a sequencing project should choose the sequencer that best fits its needs. By the method that sequencers rely on, the sequencers can be divided into following groups: First generation, second generation, and third generation.

The first-generation sequencing uses Sanger chemistry, or the "chain termination method". Radioactively labeled dideoxy ribonucleotide triphosphates (ddNTPs) are used to

terminate the DNA strand synthesis at random moments producing copies of the original strand of different lengths, each ending with one labeled ddNTP. The DNA sequence is then inferred from electrophoresis of the differently long strands (Sanger, Nicklen, & Coulson, 1977). Reads produced by Sanger sequencing are up to 1000 bp long and the accuracy is very high. Modern Sanger machines are fully automated with capillary electrophoresis and use fluorescent labeling (Heather, & Chain, 2016). Although they can be used to sequence whole genome, they are more commonly used for smaller sequencing projects for their high per-base cost. They are often used for sequencing of single genes and genotyping. Sanger machines also find their use when the sequencing accuracy is crucial, for example when studying microsatellites, single nucleotide polymorphisms or generally to verify data produced by second generation sequencing.

Second-generation sequencers no longer rely on Sanger chemistry and are massively parallel. More methods of sequencing belong to second-generation, most notably pyrosequencing (454 Roche) and Solexa (Illumina). Another technology that exist is by the SOLiD platform (sequencing by ligation).

Pyrosequencing takes advantage of the fact that a pyrophosphate is released when a new base is incorporated. Pyrophosphate enters cascade of enzymatic reactions which result in emission of light whose intensity is proportional to the amount of pyrophosphate (Ronaghi, 2001). The principle of pyrosequencing in 454 is that fragments of the genome are fixed onto beads via universal adaptor sequences, then amplified by emulsion PCR to create polonies of $\sim 10^7$ identical amplicons. These polonies are then sequenced by attaching primers that are elongated in step-by-step synthesis by washing it in cycles with deoxyribonucleotide triphosphates (dNTPs) where each cycle contains one kind of dNTP. After each cycle, signal is detected in the polonies where a certain dNTP was incorporated. This method struggles with correct identification of the number of same consecutive bases in homopolymer, where its length should be proportional to the emitted light but noise can introduce an error when the homopolymer is longer than around 4 or 5 bases (Ronaghi, Uhlén, & Nyrén, 1998).

In Solexa, fragments are attached to flowing cells, amplified by bridge PCR to again create polonies of distinct fragments. During the sequencing, each cycle offers all 4 dNTPs with different color codes. However, every synthetized strand is elongated just by one nucleotide in each cycle as the 3' OH group of each dNTP contains a blocking group that is removed at the end of the cycle, after the current position of the polonies had been interrogated for its nucleotide (Heather, & Chain, 2016).

The main advantage of second-generation sequencing is that it is a real-time sequencing that gets rid of the lengthy process of electrophoresis, is cheaper than sanger method and has high throughput. It is now most widely used for genome sequencing and most assemblers that exist are designed to handle reads produced by these methods. Downside of it is that except for 454 they produce much shorter reads[1] and to create assembly from shorter reads, bigger coverage is required which increases complexity of the computation of assembly (Miller, Koren, & Sutton, 2010).

Third-generation sequencing is still quite new but very promising for the future as it no longer requires amplification like previous generations but rather sequences single DNA molecule in real time. In comparison to second generation, most platforms also have following properties: (i) Interrogation of bases proceeds at the full speed of DNA polymerase, (ii) fewer reactions and material is required, (iii) simpler or no labeling of nucleotides is used (Munroe, & Harris, 2010). Following technologies are commercially available today: Single Molecule Real Time (SMRT) sequencing (PacBio platform), nanopore sequencing (Oxford Nanopore), and Illumina Tru-seq Synthetic Long-Read technology (Lee et al., 2016). These long-range sequencing platforms have size of reads in order of tens of thousands. From the point of view of assembly, the longer the read the better, not only because it decreases the complexity of computation, but also because of the biggest obstacle of genomic assembly – long repetitive areas of DNA that second-generation sequencers cannot reconstruct if the length of repetitive sequence is bigger than their read length.

2.2. Analysis of raw reads

The standard format in which reads are stored is the FASTQ file. FASTQ file always contains 4 lines per entry and its format is strictly defined so that it can be always correctly parsed by any software that needs to work with the reads (Figure 1). The file contains the sequences and quality score. The quality score (Q score, or Phred score) is an integer value and it is logarithmically related to the probability of an error in base call. Formally, quality score is defined as follows:

$$Q = -10 \log_{10}(P)$$

[1] As an example, SOLiD 5500 W Series Genetic Analyzer V2.0 offers 50 and 75 nucleotides-long reads ("v2.0 Specification Sheet", 2017). Illumina Miseq Reagent Kit v2 offers lengths 36, 25, 150, and 250 ("MiSeq Specifications | Key performance parameters", 2017).

Where Q = quality score, P = probability of an error ("FASTQ files", 2017). The quality scores in the FASTQ are encoded in ASCII, the encoding is not standardized and depending on the machine it can belong to two groups: ASCII_BASE 33 and ASCII_BASE 64, they can further differ by the value range (see Appendix C for Q scores to ASCII translation).

FASTQ is the standard output of Illumina, but other formats exist, for instance Standard Flowgram Format (SFF) used by 454 and Ion-Torrent, FAST5 used by Oxford Nanopore, or Hierarchical Data Format HDF5 used by PacBio (Deanna Church, 2017; "PoreCamp2016: Understanding your MinION data", 2017; "HDF5 Data Format for PacBio Sequences", 2017).

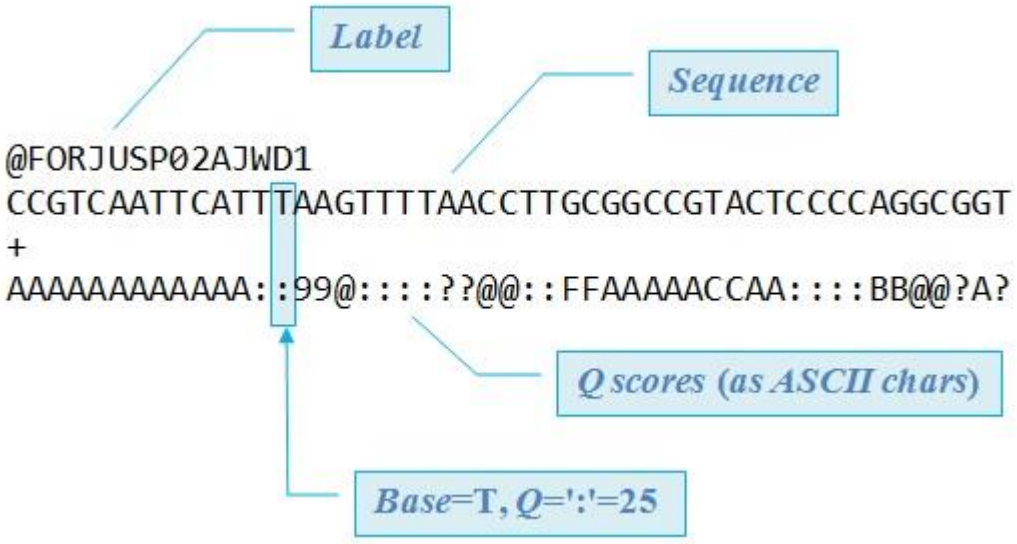

Figure 1: Structure of FASTQ format. One entry is shown, i.e. one sequence (read) with its label and quality score. Source: "FASTQ files", 2017.

The first step before the assembly is to create basic statistics of the reads and analyze their quality. The data received from sequencing lab can sometimes be poor in quality so checking and running an analysis can avoid future problems in assembly, or even reveal that it is completely unusable (Ekblom, & Wolf, 2014). The most widely used tool for this analysis is FastQC, a quality control tool for high throughput sequence data. FastaQC runs several tests on the data which report the basic statistics – number of sequences, sequence length, GC content etc., and then performs more complex statistics that can reveal or suggest there is a problem with adapter contamination, plasmid contamination, unbalanced coverage, etc. (more information in FastQC documentation; "Index of /projects/fastqc/Help", 2017). Most of the

tests are based on the idea that there should be no structure, or bias in the data, given that it consists of randomly fragmented genome. If there is some bias, it is possible that there is some contamination in the data.

Depending on the errors reported by FastQC, additional step before assembly called pre-processing can be carried out. The most obvious reason for pre-processing is when the reads contain adapter sequences that were used during the sequencing. In such case the reads need to be rid of the adapters. Another correction depends on the sequencer. For example, in Illumina the quality of bases is often declining towards the 3' end. An example of bad Illumina data (Figure 2) shows such case. Example of commonly used pre-processing software that can trim sequences based on the quality of bases and treat adapter contaminations is Erne-filter (Del Fabbro, Scalabrin, Morgante, & Giorgi, 2013) and Trimmomatic (Bolger, Lohse, & Usadel, 2014). The problem with trimming is that too much of it will come at the cost of losing a lot of data, so it should be done in a way that is a reasonable trade-off between reads' quality and amount of data left. Quality and contamination need not be the only reasons for pre-processing. Short repetitive sequences that are part of a genome are problematic for assemblers and it can be useful to remove them e.g. with RepeatMasker. In some cases, assemblers or assembly pipelines have their own requirements for read quality and include their own pre-processing.

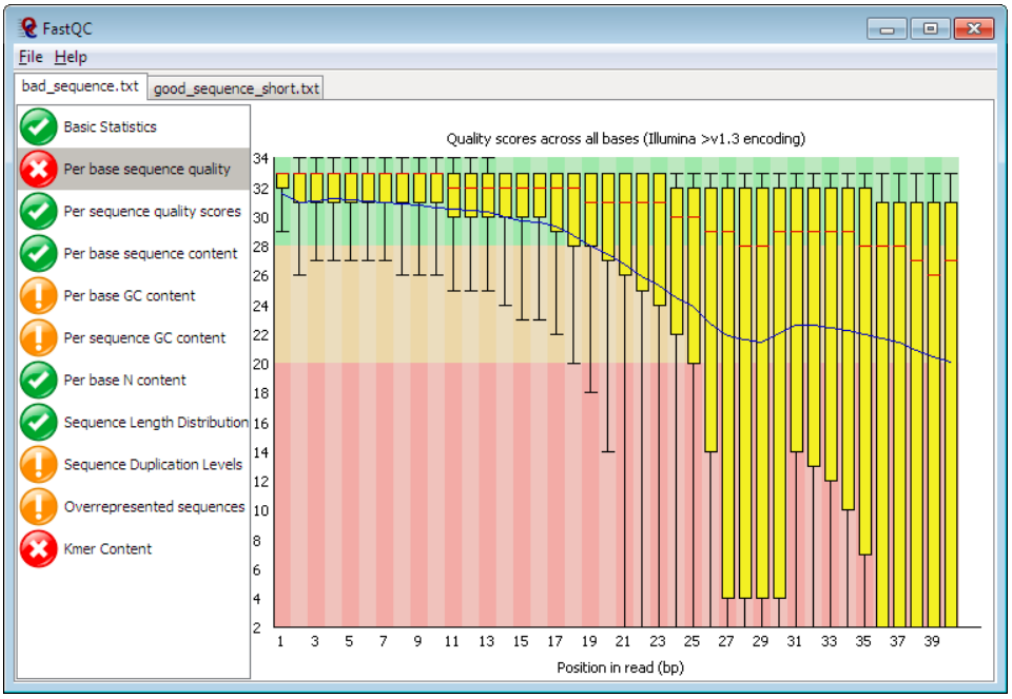

Figure 2: Example of Per base sequence quality of bad Illumina data (green area means good quality, orange area slightly bad, red area bad quality). Source: "Babraham Bioinformatics - FastQC A Quality Control tool for High Throughput Sequence Data", 2017.

2.3. De novo genome assembly

The paradigm of assembly is cleaning up reads, assembling them into contigs, putting the contigs in context by creating scaffolds and finishing it by filling in the gaps. Reads are assembled into longer sequences (contigs) by assemblers which work with the overlaps the reads have. The product of assembly usually comprises several contigs for which the direction and position on the genome is unknown. In the case where a genome of previously sequenced organism is assembled, the scaffolding step is easy. The already finished genome serves as a reference sequence to which the contigs are mapped to figure out their orientation and order. In the case of de novo assembly of a new organism, additional information is required, such as that obtained from paired-end and mate-pair reads.

Depending on the way the library of fragments for sequencing was prepared, fragments can either contain single read, paired-end reads or mate-pair reads. In case of paired-end reads, the fragment is sequenced from both ends to create forward and reverse read separated by an insert whose length can be estimated from fragment length distribution. Forward and reverse reads can also overlap. Mate-pairs span a longer region than paired-end reads (~2–20 kb long inserts) and they usually face outwards (Ekblom, & Wolf, 2014). Paired-end reads and mate-pair reads can be used during scaffolding to bridge over gaps and connect contigs if one read

is present in one contig and the second read is present in the other contig. They can also be used to resolve repetitive sequences if two sets of paired-end reads are available: (i) pairs where one read is in the repetitive sequence, the other in an area that flanks this sequence and (ii) pairs where both reads surround the repetitive sequence (Miller, Koren, & Sutton, 2010).

The finishing step is carried out if the goal of the assembly is to produce a final genome. Filling in the remaining gaps can include a lot of additional work in lab resequencing parts of the genome, expertise, and money and thus many genomes are published as drafts since that is already enough for many analyses. This issue was explored in a study (Nagarajan et al., 2010) which elaborates on how often genomes are left unfinished, but also shows that a prokaryotic genome can still be finished with limited resources in small labs and it points out that finished genomes could get more common place in the future with the rise of technologies with longer reads.

2.3.1. Algorithms

Assembly algorithms are based on graphs and employ one of the following methods: Overlap-Layout-Consensus (OLC) which is based on overlap graph, de Bruijn Graph method which is based on a k-mer graph, and greedy assembly. A graph is a data structure with a set of points (vertices) that are connected by lines (edges). The important property of a graph is the connectivity between vertices (Jones, & Pevzner, 2004). In bioinformatics, the most used graphs are directed graphs where each edge connects the source node with sink node. The composition of graph differs depending on the algorithm. But universally, creating an assembly is a problem of finding the best path through the graph that represents the reads and the overlaps.

The OLC approach consists of 3 steps: (i) Creating overlap graph, (ii) finding path through the graph which represents the sequence of reads connected by overlaps i.e. the layout, and (iii) creating the final consensus sequence. First, the assembler finds all overlaps in the set of reads. Overlaps are computed by pair-wise comparison between all reads using a Blast-like seed and extend algorithm (Miller, Koren, & Sutton, 2010). This information is then organized into overlap graph, where vertices represent the reads and edges represent the suffix-to-prefix overlaps between the reads (example of overlap graph shown is in Figure 3; El-Metwally, Hamza, Zakaria, & Helmy, 2013). Each vertex can be connected to multiple other vertices with edges of different overlaps. In the layout step, a path through the graph is found in such way that every vertex is visited exactly once (every read is used) while some edges are left

out. Such path is called the Hamiltonian path and is NP-complete[2]. In the final step, the consensus is created from the overlaps to construct the final sequence. The OLC method is typically used for Sanger data and 454 data and is considered computationally too intensive for data from sequencers with short read length and huge depth of coverage like Illumina and SOLiD. However, there are short-read assemblers like Edena and Shorty that also implement this method (Ekblom, & Wolf, 2014). Another assemblers that use this method include the famous Celera assembler, Newbler, and CABOG.

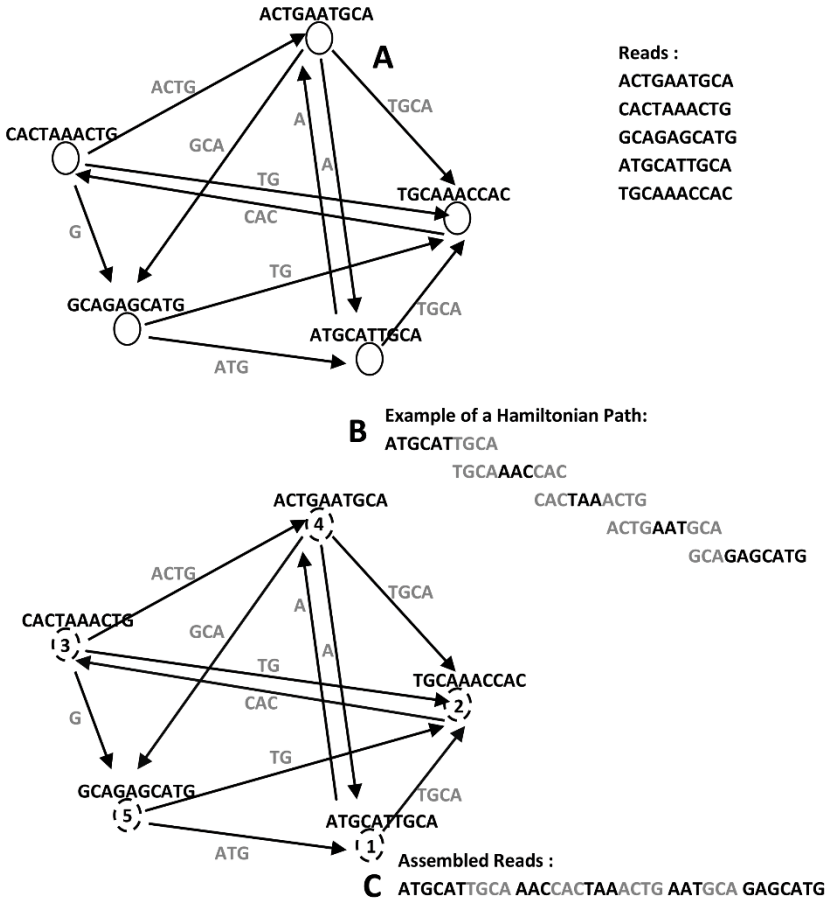

Figure 3: An example of overlap graph created from few short reads. Source: El-Metwally, Hamza, Zakaria, & Helmy, 2013.

[2] NP-complete problems lie with their complexity between exponential and polynomial problems and are difficult to solve (Jones, & Pevzner, 2004). Assemblers have to use heuristic and approximation algorithms to overall simplify the graph in order to increase efficiency of finding a solution (Miller, Koren, & Sutton, 2010).

A variation of OLC is the string graph approach (Myers, 2005) which focuses on the graph reduction. Large number of reads (typically 40%) is contained in a path spelling a sequence of reads. Removing these redundant reads and their overlaps reduces the memory requirements. This approach is used in next-generation whole genome assembler BOA (Berkeley Open Assembler).

Good approximation of the DNA sequence can be obtained by finding the shortest superstring that explains all reads. A superstring of the reads could be easily constructed by concatenating the reads. That would however be a useless assembly. Instead, we are interested in finding a superstring which is the shortest common string that contains all the reads (Jones, & Pevzner, 2004). The greedy assembly is the way to solve this problem. This method merges the reads whose overlap scored the highest (scoring can be based on length of overlap and number of mismatches). Solution is found in the overlap graph by finding such path that visits each node exactly once and maximizes the score. Finding this path is known as the Travelling Salesman Problem and is NP-hard. The major drawback of this method is that it tends to get stuck at local maxima. Greedy approach is more suitable for small genome assembly and is used in short-read assemblers such as SSAKE, SHARCGS, and VCAKE (El-Metwally, Hamza, Zakaria, & Helmy, 2013).

De Bruijn graph method is the most used approach for assembly of short-read Illumina and SOLiD data. Unlike in the previous methods, it does not rely on overlap graph but on a k-mer graph. That means that the time-consuming step of matching all-against-all to find overlaps is entirely skipped and instead, the k-mer spectrum of the reads is created. The graph's vertices represent the k-mers and the edges are the k-1 long overlaps between the vertices (example of overlap graph shown in Figure 4; El-Metwally, Hamza, Zakaria, & Helmy, 2013). With ideal data where the k-mers are error-free, provide full coverage, and span every repeat, solving of the problem is a matter of finding the Eulerian path, i.e. a path that traverses the graph in a way that every edge is visited exactly once. In contrast to the Hamiltonian path this is a simpler problem (Miller, Koren, & Sutton, 2010). Real life data is usually not that perfect and repeats and errors are problematic in the graph as they cause "tangles", or "bubbles". Assemblers use reads, paired information and different trimming strategies to resolve these problems. Another disadvantage is that the assembler is sensitive to the selection of parameter k and it mostly needs to be simply tested which k will perform the best. Selection of k is a trade-off between sensitivity (small enough to consider true overlaps) and specificity (large enough to avoid false overlaps). K cannot logically be larger than the

read length as that would create no k-mer spectrum. Velvet assembler also only accepts odd k because odd k-mer of palindrome cannot match its reverse complement and create loop in the graph. The software that implements de Bruijn graphs includes Euler, Velvet, ABySS, and SOAPdenovo.

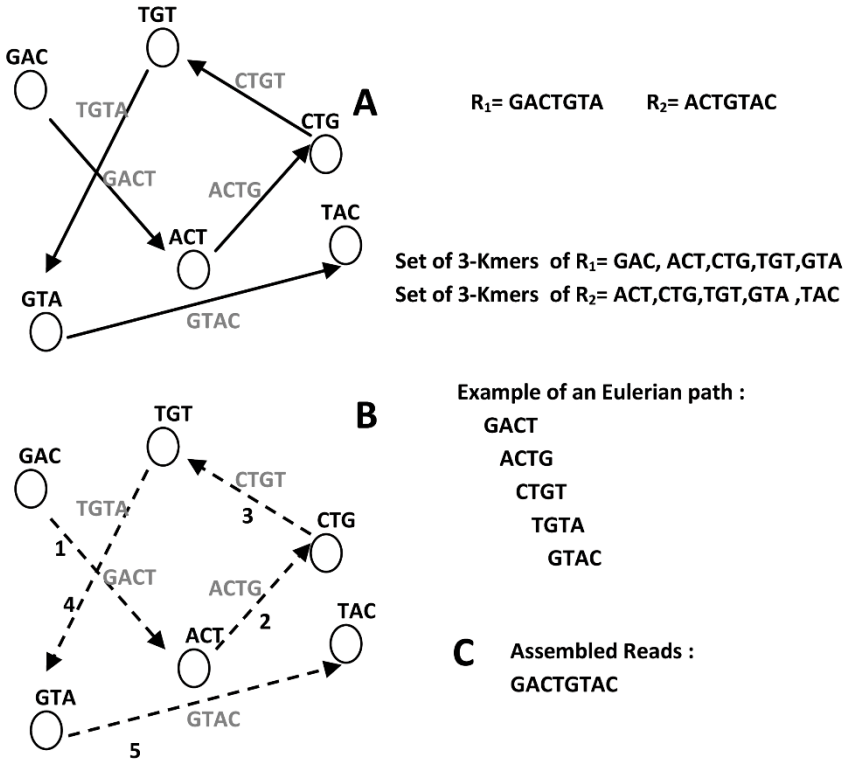

Figure 4: Example of k-mer graph, created from few short reads. Source: El-Metwally, Hamza, Zakaria, & Helmy, 2013.

2.4. Microbial life in arctic soil

The arctic permafrost that exists in extreme conditions that can seemingly only hardly sustain life shelters variety of well adapted microorganisms. Microorganism in permafrost need to battle with very low temperatures, dehydration, thawing and freezing of soil or extremes in pH and salinity. To survive they can produce pigments, protective layers, have different structure of cytoplasmic membrane to ensure it remains fluid (Tehei, & Zaccai, 2005) and their enzymatic proteins can catalyze reactions even in lower temperatures (Stibor & Králová, 2000).

Example of a commonly occurring resilient bacterium is Arthrobacter. Arthrobacter belongs to the phylum Actinobacteria whose common trait is gram positivity and GC content of 57 – 75% (Lo et al. 2002). It has a simple life cycle and can be found in different types of

soil in all kinds of climates and can survive in extreme condition such as arctic ice and chemically or radioactively contaminated areas. For instance, a study on *Arthrobacter aurescens* strain TC1 (Mongodin et al., 2006), a bacterium that can degrade herbicide atrazine, found that it contains two large plasmids that encode for great number of proteins involved in stress response of the cell. The study attributed its resilience to its metabolic versatility caused by duplication of catabolic genes and to its ability to incorporate plasmid-derived intermediates into chromosome-encoded pathways. Metabolic versatility is a key ability for survival in arctic areas where the sources of nourishment are scarce.

The arctic permafrost stores big reserves of carbon (C) and can potentially greatly influence the Earth's global C cycle with the warming up of polar areas and the release of CO_2 and CH_4 into the atmosphere (McGuire et al., 2009). With thawing of permafrost, rising activity of microbial activity is expected. Microbes decompose the organic soil matter and convert it to the greenhouse gases which will further contribute to the global warming and thawing of permafrost, i.e. create the so-called permafrost carbon feedback (Schaefer, Zhang, Bruhwiler & Barrett, 2011). Better understanding of the microbial composition of permafrost and their metabolizing capabilities which can also be inferred from annotated genomes of the microbes is part of understanding the global issues that the arctic areas are part of.

3. Materials and Methods

All work and analyses were done in silico, on the Bio-Linux 8 platform which is based on Ubuntu Linux 14.04 LTS to which bioinformatics packages were added (Field et al., 2006). In the whole process of getting the final assembly, bioinformatics software, pipelines, and scripts were used, which are all mentioned at each step for which they were employed.

The data provided by my supervisor consists of two FASTQ files with reads from whole genome sequencing project (see "1189_TTAGGC_L001_R1_001.fastq" and "1189_TTAGGC_L001_R1_001.fastq" in Appendix A). The sequencing was done by Illumina machine from libraries of approximate size 180 bp. Paired-end sequencing was used, so the two files are complementary, where one contains the forward reads and the other reverse reads. Prior to this work the 16S ribosomal RNA of the specimen was isolated and from its sequence it was inferred that the bacterium is of genus Arthrobacter.

3.1. Analysis of raw reads

To get first familiar with the data set, simple bash script (see Appendix B) was written to check the FASTQ file format and find out basic information about the reads. The script reported the format to be invalid (number of lines was not divisible by 4). After inspection of the file, redundant empty line at the end was discovered. The last line was erased and the script was run anew, this time reporting no error in formatting. To investigate the quality of reads both FASTQ files were analyzed with FastQC (version 0.11.5).

3.2. Genome assembly

Pre-processing and assembly of reads was carried out by two pipelines that specialize on prokaryotic assembly. The first, A5-miseq (version for Linux from 25. 08. 2016), is specifically designed for de-novo prokaryotic assembly of Illumina paired-end data (Coil, Jospin, & Darling, 2014). It is simply downloaded as an archive that comprises several pearl scripts. It doesn't require any installation and can be run either in Linux or Mac OS. It was run with raw paired reads on which it performed pre-processing with Trimmomatic and then created the assembly. The second pipeline, MyPro (Liao, Lin, Sabharwal, Haase, & Scannapieco, 2015), is a Bio-Linux virtual machine for VirtualBox that can be downloaded in OVA file. MyPro is not an assembler on its own, it consists of scripts that automate the entire process of assembly using already made software. MyPro has following assemblers installed:

SOAPdenovo, Abyss, SPAdes, Velvet, and Edena. In those assemblers where k-mer graph is used, it automatically runs assemblies with several different values of k and then keeps the best one. MyPro's scripts for each step in the genome assembly are: Preprocesss.py, Assemble.py, Integrate.py, Postassemble.py, and Annotate.py. It also includes the script Autorun.py, which runs all these scripts in sequence.

MyPro was used on a computer with Intel(R) Core(TM) i7-4702MQ CPU @ 2.20GHz, 4 physical cores, with 8 GB DDR3 RAM and Windows 10. The OVA file was imported to Virtual Box (Version 5.1.2) and Preprocess.py and the pipeline Autorun.py in the virtual machine was run. Virtual machine specifications in VirtualBox for successful run: 6GB RAM, 2 cores.

Virtual disk in format VMDK was exported to virtual disk format VHD and MyPro was run on Hyper-V on a windows server 2016 (Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHZ, 4 physical cores, 16 GB DDR4 RAM). Virtual machine specifications in Hyper-V: Dynamic RAM 2048 MB – 12 GB, 8 virtual cores, virtual disk was connected through controller SCSI. Both read files were trimmed with Preprocess.py with two different values for parameters -l (quality cutoff value) and -min (minimal length of read) to produce quality trimmed reads. For the first pre-processing task, default value for quality cutoff was chosen (0.01) together with minimum length 40 bp. For the second pre-processing task, reads were trimmed using quality cutoff value of 0.05 with minimum length 55 bp. Trimmed reads were analyzed with FastQC and the ones trimmed with -l = 0.05 were chosen for assembly because the first trimming caused a bigger loss of information. Another FASTQ file was created from the two trimmed FASTQ files by combining them with Flash (version 1.2.11.; Magoc, & Salzberg, 2011) that quality trimmed the sequences and merged the corresponding forward and reverse reads together, where they overlapped. Assemble.py was run with (i) raw reads, (ii) trimmed reads, (iii) extended reads.

The partial results of assembly allowed to run script Integrate.py that uses CISA contig integrator (version 1.3) only on assemblies from raw reads and extended reads. Postassembly.py was used to further join contigs that were overlapping in the integrated assembly of extended reads.

3.3. Genome annotation

The result of extended reads' post-assembly ("Bridged.ctg.fa" in Appendix A) was uploaded to the RAST server (Aziz et al., 2008) for annotation. In "genome information" on RAST, Arthrobacter was specified as the genus of the organism and the genetic code was automatically selected accordingly. In the options, Classic RAST annotation was selected as the annotation scheme, RAST as the gene caller, Release70 of FIGfam was chosen and the option for automatic fixing of errors that automatically resolves problems it encounters during annotation was checked.

4. Results

4.1. FastQC analysis of raw reads

	forward reads	reverse reads
Basic Statistics	PASS	PASS
Per base sequence quality	PASS	PASS
Per tile sequence quality	WARN	WARN
Per sequence quality scores	PASS	PASS
Per base sequence content	WARN	WARN
Per sequence GC content	FAIL	WARN
Per base N content	PASS	PASS
Sequence Length Distribution	PASS	PASS
Sequence Duplication Levels	PASS	PASS
Overrepresented sequences	WARN	WARN
Adapter Content	PASS	PASS
K-mer Content	FAIL	FAIL

Table 1: FastQC summary of statistics. (See full FastQC reports in Appendix A)

Forward and reverse reads performed similarly in the FastQC analysis (Table 1). The basic statistics showed that both files have the same number of reads (1,072,349) which they should have because they contain paired-end reads (one file has forwards reads, the second reverse reads). No reads were flagged for bad quality and they had uniform length (151 bp) in both files. Since the libraries had about 180 bp it means that the paired-end reads were overlapping. The GC content was 62% which corresponds to the higher GC content of Actinobacteria.

Both FASTQ files had overall good per base quality of reads with declining quality towards the end (especially in the case of reverse reads). The comparison of FastQC analysis of per base quality of reads before and after the second trimming (with cutoff value 0.05) is showed in figure 3 and 4. Some of the reads lost a lot of bases during trimming and were discarded for having length smaller than 55 bp. The number of reads left after trimming was 1,071,417 for forward reads and 1,065,860 for reverse reads, meaning 932 and 6,489 reads were lost, respectively.

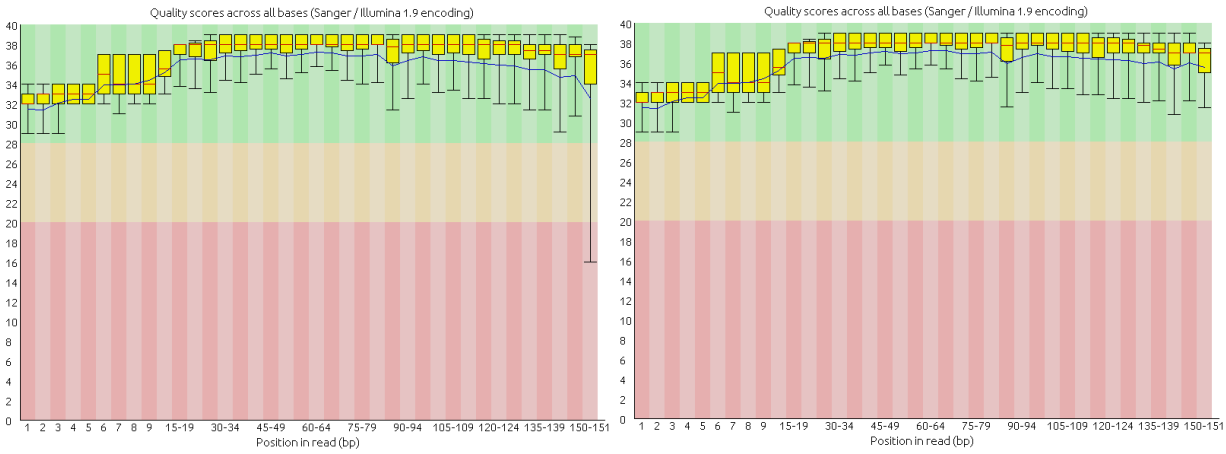

Figure 5: Per base sequence quality of forward reads before trimming and after (trimming with cutoff =0.05).

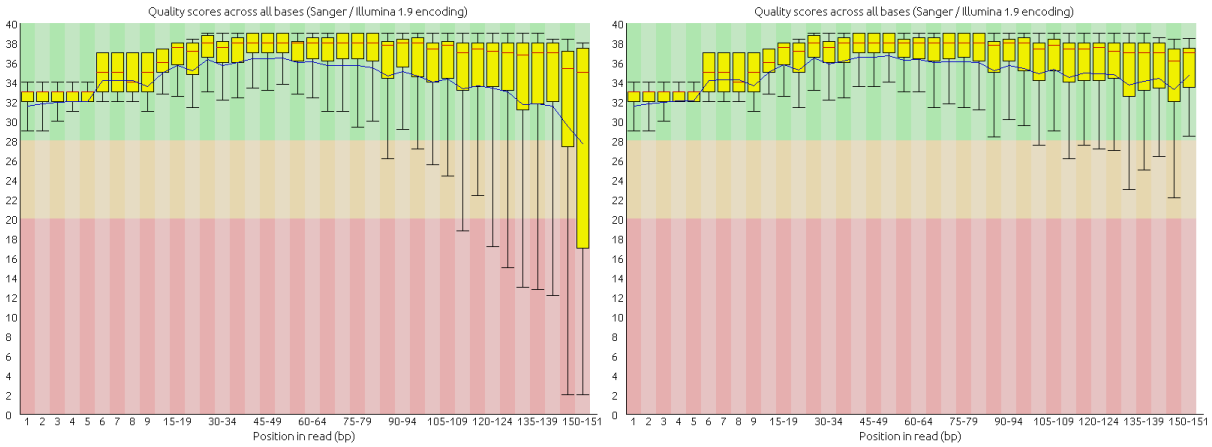

Figure 6: Per base sequence quality of reverse reads before trimming and after (trimming with cutoff =0.05).

4.2. MyPro pipeline assembly

With default settings of VirtualBox, the first trials ended up unsuccessful. After starting Assemble.py script, deadlock occurred and was followed by Windows being stopped with power error ID 41. After trials with different settings in Virtual box, some results were achieved when 4GB of RAM and two cores were allocated to MyPro.

The results of the Assemble.py were partial in all runs with different reads and allowed to run Integrate.py only in the case of raw reads' and extended reads' assembly. I decided to run the same assemblies on the Windows server with more RAM available in case the problem was in insufficient amount of memory but the results were the same (see the results in supplementary material). Table 2 summarizes the assembly statistics for all runs on the Windows server.

	N50	Number of contigs	Longest contig	Whole genome
Raw paired				
SOAPdenovo	NA	NA	NA	NA
Abyss	5,134	1,898	37,654	6,530,368
SPAdes	NA	NA	NA	NA
Velvet	2,459	2,780	15,393	5,772,789
Edena	4,236	2,135	24,934	6,679,958
Integrate (CISA)	8,019	1,130	108,065	6,292,931
Trimmed paired				
SOAPdenovo	NA	NA	NA	NA
Abyss	6,302	5,377	75,206	9,865,384
SPAdes	NA	NA	NA	NA
Velvet	NA	NA	NA	NA
Edena	NA	NA	NA	NA
Integrate (CISA)	NA	NA	NA	NA
Extended reads				
SOAPdenovo	NA	NA	NA	NA
Abyss	4,918	1,455	51,220	5,748,845
SPAdes	33,729	655	158,947	9,084,313
Velvet	4,459	1,552	29,013	5,801,602
Edena	NA	NA	NA	NA
Integrate (CISA)	38,220	525	152,725	7,796,365
Post-assembly	48,680	456	159,598	7,787,369

Table 2: Assembly statistics of all genomes assembled in MyPro

The worst assembly came out of reads that were pre-processed with the Pre-process.py script in MyPro (Trimmed paired). The only assembler that produced any results was Abyss and since the minimal requirement for integration is having results from at least 3 assemblers, no other work was done with that data.

Both raw reads and extended reads assembly was successful with three assemblers and the results could be used for integration. The integrated assembly of extended reads was superior to that of raw reads with higher N50 value (38,220) and fewer contigs (525). The following post-assembly further improved the assembly with N50 value being 48,680 and number of contigs 456 (assembly stored as “Bridge.ctg.fa” in Appendix A).

4.3. A5-Miseq pipeline assembly

The resulting assembly is in the file "Final.scaffolds.fastq" in Appendix A and the assembly statistics are summarized in Table 3. A5-miseq created a lot of contigs but from all assemblies, it created the longest contig (177,031) and the N50 statistics also belongs to the higher ones (25,977) especially given the fact that it has the longest genome length (~ 10 Mbp).

	N50	Number of contigs	Longest contig	Whole genome
A5-miseq	25,977	2,367	177,031	10,491,617

Table 3: Assembly statistics of genome assembled with A5-miseq

4.4. Genome annotation

The annotation ran successfully on the RAST server (summary of run in Table 4) and the annotation result in Genbank format is available in Appendix A (file "1663.48.gbk"). In addition to already known N50, SEED also computed a similar statistic, L50, which was 46 (Table 5). This value represents the smallest number of contigs whose combined length is 50% of the total contigs' length, or the assembly size (Bradnam, 2017). From this follows that the smaller the value, the better as that would mean the assembly consists of very large contigs. For our assembly that comprises 456 contigs it means that 50% of the assembled genome is represented by 46 contigs and the other 50% is represented by 410 contigs of smaller size.

Number of features	7279
Number of warnings	1
Number of fatal problems	0
Possibly missing genes	123
Same strand overlaps	1 Warning

Table 4: Summary of the annotation run on RAST server.

L50	46
Number of contigs with protein encoding genes (PEGs)	456
Number of subsystems	430
Number of coding sequences	7196
Number of RNAs	83

Table 5: Selected information from SEED-Viewer's organism overview

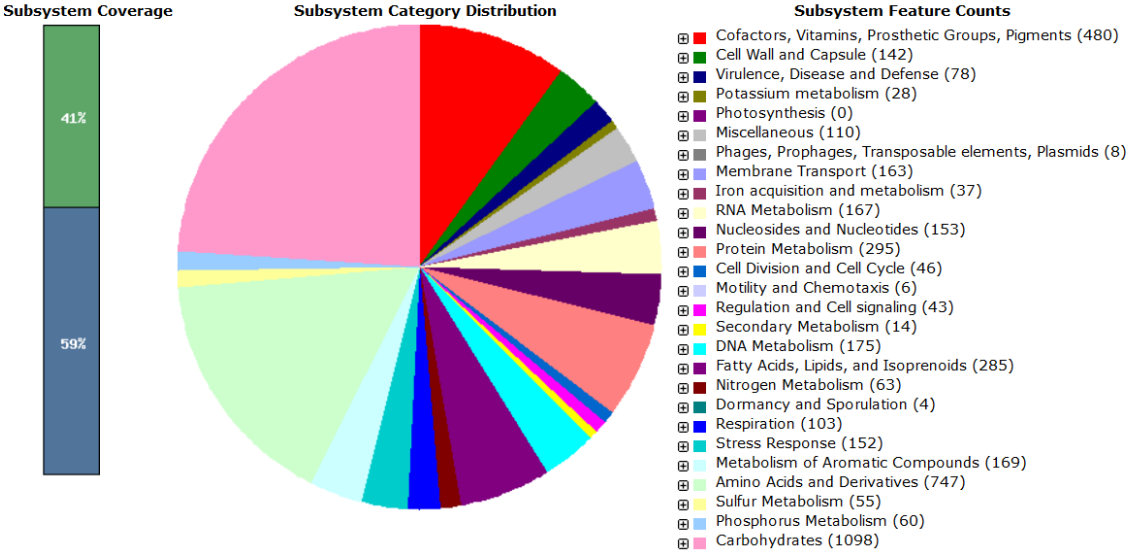

Figure 7: Subsystem statistics of Arthrobacter's genome. Statistics and the chart were created by SEED-Viewer.

Subsystem coverage in Figure 7 shows that 59% of the coding sequences are not in subsystems (of which 50% is hypothetical function assignment) and 41% of sequences are in subsystems (of which mere 4% are hypothetical). The pie chart shows the distribution of subsystem categories in the genome. The most prominent categories are Carbohydrates (1098 sequences), Amino Acids and Derivatives (747 sequences), and Cofactors, Vitamins, Prosthetic Groups, Pigments (480 sequences). As a bacterium that must survive in Arctic, unsurprisingly significant portion of coding sequences are in the category Stress Response (152 sequences) and 4 sequences were also identified in the Dormancy and Sporulation category.

5. Discussion

5.1. Quality of reads

The uniformity of length and good per base quality of reads indicates that some initial preprocessing was already done by the sequencing lab. However, the FastQC reported several warnings and errors. To continue with the assembly, the possible sources of these results and their implications for assembly had to be first considered. In the following paragraphs, I will discuss the results of the first file (forward reads) only, because the second one had very similar results.

The first warning was reported for per tile sequence quality. The headers of FASTQ files of each sequence contain information about the flow tile the read came from. The graph shows the deviation from the average quality score for each tile at every base. Cold colors mean average or better quality and hotter colors worse than average. Therefore, from the patterns we can see whether some tile was associated with bad quality (we would see a horizontal line with hot color) and is responsible for bad quality in the data. In graph for forward reads (Figure 8) we can see that the tiles 2114 and 2113 had a quality of the 6th base worse than the 6th base at other tiles. The quality of other bases in these tiles were otherwise average or better, indicating that there was not a major problem with the tiles so there is no need for any correction.

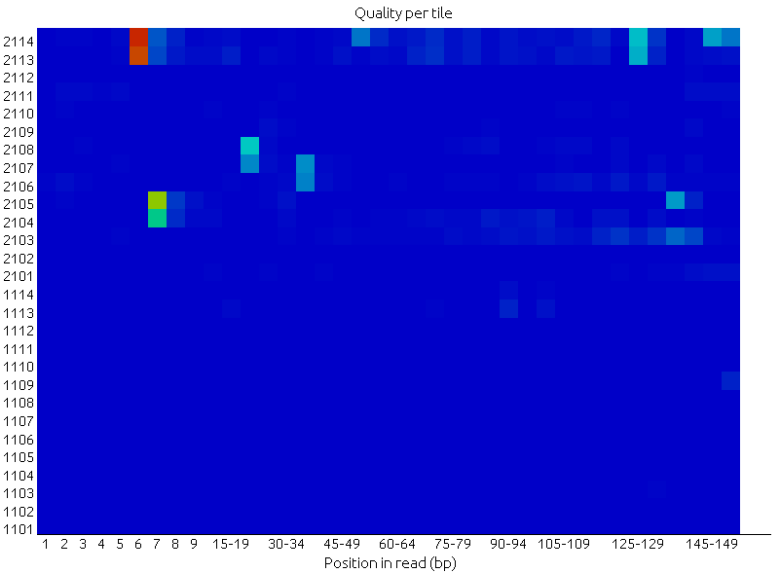

Figure 8: Per tile sequence quality for forward reads

Error in per sequence GC content is most likely the result of plasmid presence in the data. The theoretical distribution is built from the data provided. Unfortunately, FastQC documentation doesn't provide details on how this is achieved. The red line in the graph (Figure 9) shows the actual distribution of GC count per read. Generally, a peak indicates some contamination. Since plasmids often have different GC content than the chromosome of the bacterium, I concluded that the likely source of this deviation from theoretical distribution is caused by plasmids being included in the whole genome sequencing project.

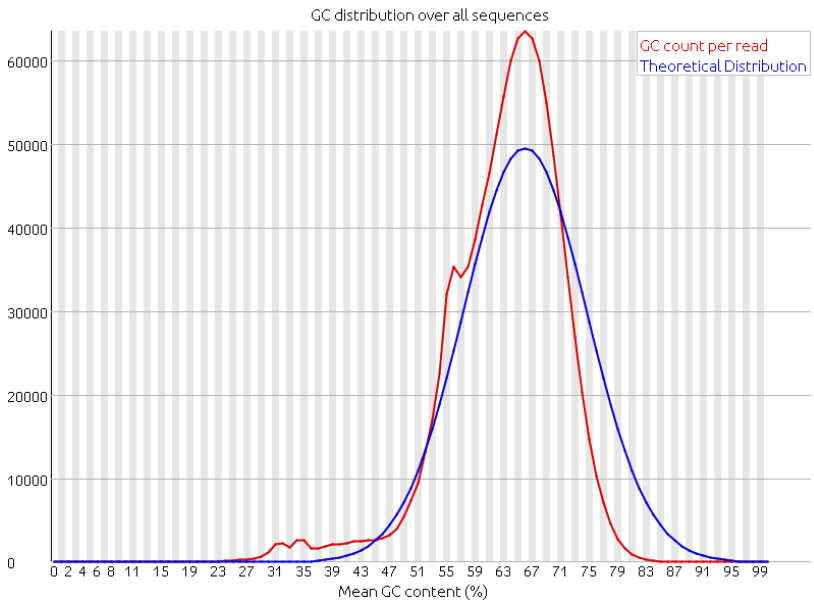

Figure 9: GC distribution over all forward reads.

The interpretation of other warnings is a little bit more complicated. If we look at them separately, they could mean several things. Error in k-mer content could be a result of adapter contamination. However, as seen in the summary (Table 1), both files passed the adapter content test. Moreover, we would expect adaptor contamination at the end of the sequence as the result of read-through adapter contamination (when the read is shorter than read length, the sequencer starts to sequence the adapter), not the start of the sequence as in this case (Figure 10a). Another warning was issued for overrepresented sequences which can either indicate that the library was contaminated or that the sequences are highly biologically significant. Per base sequence content also shows a bias at the beginning of reads just like the k-mer content.

When I looked at possible connection between the warnings and errors, I found that they most probably stem from one source – overrepresented sequences. The obvious connection is, as mentioned before, between k-mer content and per base sequence content (Figure 10b). If per base sequence content was unbiased the plot would show 4 straight horizontal lines (each line for one base) meaning that the base content has the same ratios across all positions in reads. Although the lines oscillate across all positions, the biggest peaks are in the beginning. It is logical that overrepresented k-mers in the beginning of the sequences will create this unbalance, hence both statistics reflect the same problem. Finally, I compared the lists of most frequent k-mers with most overrepresented sequences and found that the most common k-mers are present in those sequences. FastQC didn't list any source of common contamination (by finding matches in its database) so it can either be some less common contamination or the sequences are biologically significant. I tried blasting 3 full sequences from the file that start with the 50 nucleotides that FastQC reported as overrepresented and all three of them found best match with bacterial rRNAs. Though only three sequences are not representative enough to be certain that all overrepresented sequences are not some less common contamination, I find the possibility that they all come from the sequenced genome more likely and decided to move on to assembly.

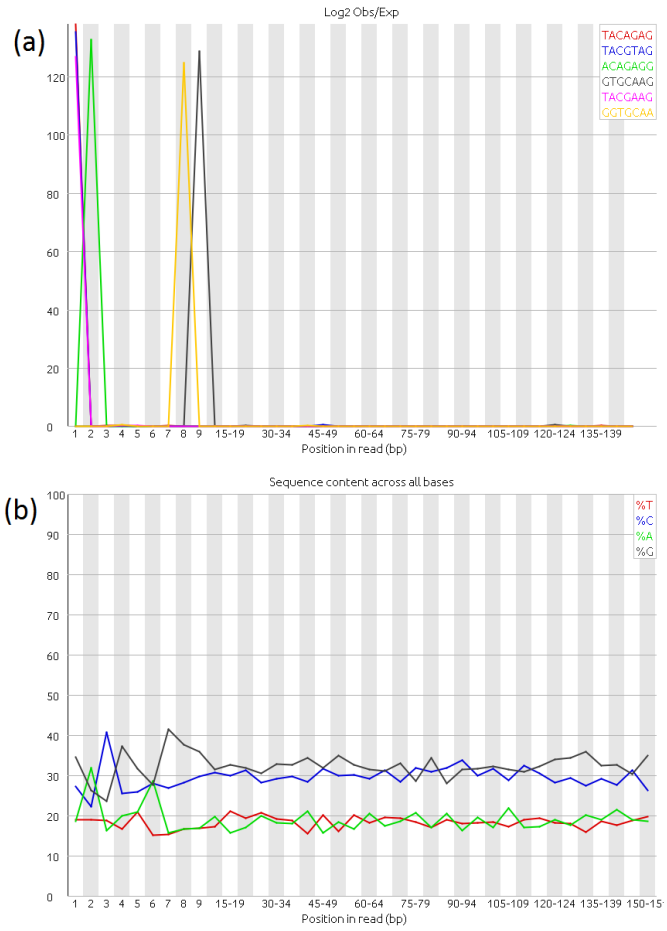

Figure 10: Distribution of 6 most represented 7-mers in forward reads (a) and per base sequence content of forward reads. (b)

5.2. Genome assembly

MyPro turned out to be a bit more complicated for use than the "Seamless pipeline for automated assembly" part of the name of its published article would indicate. First problems were already encountered with settings of VirtualBox where the user must have some basic knowledge on how to optimally allocate resources such that the scripts do not take too long and also enough memory and processing power is left for the host system in which the virtual machine is running. It would also be better if MyPro was available for download in VHD format which can be imported in VirtualBox and also in Hyper-V which is better integrated with the operating system, in the case user has Windows (for instance Hyper-V makes sure that the virtual computer always lefts enough resourced for the host system and therefore any deadlock in the system should not occur). While the intention of authors for MyPro to be used in VirtualBox is understandable as VirtualBox is cross-platform software, the very idea of creating a whole Linux-based virtual machine just for assembly suggests that the intended users of MyPro are biologists with no bioinformatics background who mainly work with Windows or Mac OS. Therefore, providing MyPro in OVA format seems illogical to me.

The most troubling part of the results is that in all instances of assembly runs, several assemblers did not produce any assemblies. Ironically, when Pre-process.py which is part of the MyPro pipeline was used, the trimmed sequences apparently posed some difficulties to the assemblers of which only Abyss produced results. In contrast to that, even when no pre-processing was used (in the "Raw paired" run) the assembly was successful in case of Abyss, Velvet and Edena. SOAPdenovo and Velvet did not produce any log so I could not search for possible errors, but in the case of SPAdes and Edena, the logs included error message that was responsible for the failure. In all cases where SPAdes failed, the error message was: Error in malloc(): out of memory. In cases where Edena failed the error was: all reads within a file must be of the same length. Since quality trimming in "Trimmed paired" and "Extended reads" trims low quality bases on the ends of sequences resulting in non-uniform lengths of reads, it is understandable why Edena failed in these two cases. As for SPAdes, running out of memory is a problem of limited RAM of the hardware, part of which is in addition unavailable to the virtual computer as it has to share it with the host system. The authors of MyPro provide a short guide where they demonstrate usage of their software on examples that were run in a VirtualBox with 16GB RAM @ Dell Precisions Workstations T1600 Computer Workstation (Quad Core Xeon E3-1245, 3.30 GHz with 32GB RAM). 32GB RAM is a very decent memory

which is quadruple of what I have on my laptop and double of what I had available on the Windows server. It is therefore possible that the rest of assemblers also failed due to insufficient memory.

Quality of assembly was evaluated with assembly statistics, based on which the best assembly after integration of individual assemblies was achieved with extended reads. The most important statistics that was reported both by MyPro and A5-miseq is number of contigs and N50. The goal of assembly is to get the smallest number of contigs possible to be close to having one contiguous sequence as a result. The N50 statistics is the length of the contig which is at the point of half of the mass of the length distribution which means by its computation it is dependent on the combined length of all contigs[3] (reported as genome length). Therefore, it is not a statistic that can be really compared between different assemblies unless the genome sizes are the same, but still is an important parameter of assembly quality. In both regards, by far the best assembly was produced in the "Extended reads" run by SPAdes (655 contigs, N50: 33,729).

The result of A5-miseq assembly was disappointing in regard to number of contigs but otherwise created a good assembly. An advantage to A5-miseq is that it can use paired-end reads to create scaffolds from contigs. However, in this case our "paired-end reads" overlap, therefore do not provide any additional help by spanning longer distances. I would opt for this assembler again in the future if I would have paired-end reads with some longer insert.

Since paired-end reads overlapped, the reads in MyPro assembly were rather merged and that also turned out to be a smart step. Some assemblers may even have problems with overlapping paired sequences and it also simplifies and speeds up the process of assembly (Seemann, 2017).

[3] N50 statistics is computed in following way: The contigs are ordered from longest to shortest and then, starting with the longest, the lengths of contigs are summed until the sum equals 50% of combined contig length (length of assembled genome). The smallest length of the sequence which was still added to the summation is the resulting N50 value (Yandell, & Ence, 2012).

5.3. Annotation of the draft genome

The annotation run on RAST server issued one warning for “Same-strand overlaps”. From the SEED documentation for the RAST report ("RAST Quality Report - TheSeed", 2017), same-strand overlap is a pair of same-stranded PEGs oriented in the same direction, whose overlap is more than the threshold of 120 bp. Since the documentation does not state otherwise, I suppose the overlap is meant as a perfect overlap with zero mismatches. Occurrence of two sequences in the genome with identical 120 bases just by chance is a very unlikely scenario. In my opinion there are two possible sources of this: (i) it is not an error in assembly, but a gene duplication, (ii) it is an assembly error. The second option would be interesting - if we found the locations in assembly where these overlapping sequences occur, it could possibly reveal a problematic place for the assembly and allow us to make precautions in possible future assemblies.

The SEED estimated the number of missing genes (PEGs possibly present in gaps) to be 123. The estimation is very rough but also very conservative, so it is possible the number of actual missing genes is smaller. Given that the drat genome consists of many contigs, it is no surprise that some genes might not be present in the assembly.

6. Conclusion and future prospects

From inspection of the reads from Illumina sequencer, it was clear they had already undergone the basic pre-processing that trimmed the adapter sequences and discarded any low-quality reads. However, I think that playing more with the pre-processing step could greatly influence the assembly (just like the merging of the reads did). The data had a lot of overrepresented sequences. While those sequences are most likely not a product of contamination, their source should be more investigated and improving their representation in the data (deleting some portion of identical sequences) could potentially improve assembly.

The resulting assembly is broken up into many contigs and is far from being a finished genome. However, given the data the assembly will always be incomplete, even though it could still be improved. The constructed draft genome was good enough to make an informative genome annotation.

In the study, I got to test MyPro and A5-miseq pipelines which provided valuable lessons for the future. Overall, given that MyPro produced better results than the A5-miseq pipeline, I would say it is a very useful software which is unfortunately horribly degraded by not providing complete documentation which makes use of this software very difficult especially in terms of interpreting the results. One way to improve the assembly would be by running MyPro again on a computer with larger memory to gain complete results. The A5-miseq pipeline as an individual assembler also produced good results, but MyPro has the advantage that it integrates multiple assemblies.

The results of individual assemblers in MyPro provide information on which assemblers are best for this kind of data. I think the most promising way of making the assembly better would be by running the best performing assembler (SPAdes) and those that did not produce any assembly individually outside of MyPro to avoid the problem with insufficient memory. Then, CISA could be used for integrating the best assemblies, perhaps also with the A5-miseq assembly.

The future intentions are to resequence the genome using MinION from Oxford Nanopore Technologies with read length in order of kbp. MinION has a completely different error profile than second-generation sequencers and so completely different assemblers would need to be tested. Combination of MinION reads or their assembly could potentially be enough information to make a finished genome.

7. Bibliography

7.1. Publications

1. Aziz, R., Bartels, D., Best, A., DeJongh, M., Disz, T., & Edwards, R. et al. (2008). The RAST Server: Rapid Annotations using Subsystems Technology. *BMC Genomics*, *9*(1), 75. http://dx.doi.org/10.1186/1471-2164-9-75
2. Bolger, A., Lohse, M., & Usadel, B. (2014). Trimmomatic: a flexible trimmer for Illumina sequence data. *Bioinformatics*, *30*(15), 2114-2120. http://dx.doi.org/10.1093/bioinformatics/btu170
3. Coil, D., Jospin, G., & Darling, A. (2014). A5-miseq: an updated pipeline to assemble microbial genomes from Illumina MiSeq data. *Bioinformatics*, *31*(4), 587-589. http://dx.doi.org/10.1093/bioinformatics/btu661
4. Del Fabbro, C., Scalabrin, S., Morgante, M., & Giorgi, F. (2013). An Extensive Evaluation of Read Trimming Effects on Illumina NGS Data Analysis. *Plos ONE*, *8*(12), e85024. http://dx.doi.org/10.1371/journal.pone.0085024
5. Ekblom, R., & Wolf, J. (2014). A field guide to whole-genome sequencing, assembly and annotation. *Evolutionary Applications*, *7*(9), 1026-1042. http://dx.doi.org/10.1111/eva.12178
6. El-Metwally, S., Hamza, T., Zakaria, M., & Helmy, M. (2013). Next-Generation Sequence Assembly: Four Stages of Data Processing and Computational Challenges. *Plos Computational Biology*, *9*(12), e1003345. http://dx.doi.org/10.1371/journal.pcbi.1003345
7. Field, D., Tiwari, B., Booth, T., Houten, S., Swan, D., Bertrand, N., & Thurston, M. (2006). Open software for biologists: from famine to feast. *Nature Biotechnology*, *24*(7), 801-803. http://dx.doi.org/10.1038/nbt0706-801
8. Heather, J., & Chain, B. (2016). The sequence of sequencers: The history of sequencing DNA. *Genomics*, *107*(1), 1-8. http://dx.doi.org/10.1016/j.ygeno.2015.11.003
9. Jones, N., & Pevzner, P. (2004). *An introduction to bioinformatics algorithms* (1st ed., pp. 49 – 50, 247, 265). Cambridge, MA: MIT Press.

10. Lee, H., Gurtowski, J., Yoo, S., Nattestad, M., Marcus, S., & Goodwin, S. et al. (2016). Third-generation sequencing and the future of genomics. http://dx.doi.org/10.1101/048603
11. Liao, Y., Lin, H., Sabharwal, A., Haase, E., & Scannapieco, F. (2015). MyPro: A seamless pipeline for automated prokaryotic genome assembly and annotation. *Journal Of Microbiological Methods*, *113*, 72-74. http://dx.doi.org/10.1016/j.mimet.2015.04.006
12. Lo, C., Lai, N., Cheah, H., Wong, N., & Ho, C. (2002). Actinomycetes isolated from soil samples from the Crocker Range Sabah. *ASEAN Review Biodiversity And Environmental Conservation*, *9*, 1-7.
13. Magoc, T., & Salzberg, S. (2011). FLASH: fast length adjustment of short reads to improve genome assemblies. *Bioinformatics*, *27*(21), 2957-2963. http://dx.doi.org/10.1093/bioinformatics/btr507
14. McGuire, A., Anderson, L., Christensen, T., Dallimore, S., Guo, L., & Hayes, D. et al. (2009). Sensitivity of the carbon cycle in the Arctic to climate change. *Ecological Monographs*, *79*(4), 523-555. http://dx.doi.org/10.1890/08-2025.1
15. Miller, J., Koren, S., & Sutton, G. (2010). Assembly algorithms for next-generation sequencing data. *Genomics*, *95*(6), 315-327. http://dx.doi.org/10.1016/j.ygeno.2010.03.001
16. Mongodin, E., Shapir, N., Daugherty, S., DeBoy, R., Emerson, J., & Shvartzbeyn, A. et al. (2006). Secrets of Soil Survival Revealed by the Genome Sequence of Arthrobacter aurescens TC1. *Plos Genetics*, *2*(12), e214. http://dx.doi.org/10.1371/journal.pgen.0020214
17. Munroe, D., & Harris, T. (2010). Third-generation sequencing fireworks at Marco Island. *Nature Biotechnology*, *28*(5), 426-428. http://dx.doi.org/10.1038/nbt0510-426
18. Myers, E. (2005). The fragment assembly string graph. *Bioinformatics*, *21*(Suppl 2), ii79-ii85. http://dx.doi.org/10.1093/bioinformatics/bti1114
19. Nagarajan, N., Cook, C., Di Bonaventura, M., Ge, H., Richards, A., & Bishop-Lilly, K. et al. (2010). Finishing genomes with limited resources: lessons from an ensemble of microbial genomes. *BMC Genomics*, *11*(1), 242. http://dx.doi.org/10.1186/1471-2164-11-242

20. Nyrén, P., & Lundin, A. (1985). Enzymatic method for continuous monitoring of inorganic pyrophosphate synthesis. *Analytical Biochemistry*, *151*(2), 504-509. http://dx.doi.org/10.1016/0003-2697(85)90211-8
21. Ronaghi, M., Uhlén, M., & Nyrén, P. (1998). A Sequencing Method Based on Real-Time Pyrophosphate. *Science*, *281*(5375), 363-365. http://dx.doi.org/10.1126/science.281.5375.363
22. Ronaghi, M. (2001). Pyrosequencing Sheds Light on DNA Sequencing. *Genome Research*, *11*(1), 3-11. http://dx.doi.org/10.1101/gr.11.1.3
23. Sanger, F., Nicklen, S., & Coulson, A. (1977). DNA sequencing with chain-terminating inhibitors. *Proceedings Of The National Academy Of Sciences*, *74*(12), 5463-5467. http://dx.doi.org/10.1073/pnas.74.12.5463
24. Schaefer, K., Zhang, T., Bruhwiler, L., & Barrett, A. (2011). Amount and timing of permafrost carbon release in response to climate warming. *Tellus B*, *63*(2), 165-180. http://dx.doi.org/10.1111/j.1600-0889.2011.00527.x
25. Stibor, M., & Králová, B. (2000). Psychrofilní a psychrotolerantní mikroorganismy, jejich adaptace a využití v moderních biotechnologiích. *Chemické Listy*, *95*(2), 91-97.
26. Tehei, M., & Zaccai, G. (2005). Adaptation to extreme environments: Macromolecular dynamics in complex systems. *Biochimica Et Biophysica Acta (BBA) - General Subjects*, *1724*(3), 404-410. http://dx.doi.org/10.1016/j.bbagen.2005.05.007
27. Yandell, M., & Ence, D. (2012). A beginner's guide to eukaryotic genome annotation. *Nature Reviews Genetics*, *13*(5), 329-342. http://dx.doi.org/10.1038/nrg3174

7.2. Internet resources

1. *Babraham Bioinformatics - FastQC A Quality Control tool for High Throughput Sequence Data*. (2017). *Bioinformatics.babraham.ac.uk*. Retrieved 27 March 2017, from https://www.bioinformatics.babraham.ac.uk/projects/fastqc/
2. Bradnam, K. (2017). *L50 vs N50: that's another fine mess that bioinformatics got us into*. *ACGT*. Retrieved 17 April 2017, from http://www.acgt.me/blog/2015/6/11/l50-vs-n50-thats-another-fine-mess-that-bioinformatics-got-us-into

3. Deanna Church, S. (2017). *Formats : Documentation : Trace Archive v4.2 : NCBI/NLM/NIH*. *Ncbi.nlm.nih.gov*. Retrieved 2 April 2017, from https://www.ncbi.nlm.nih.gov/Traces/trace.cgi?cmd=show&f=formats&m=doc&s=format#scf
4. *Estimating Sequencing Coverage*. (2017). *www.illumina.com*. Retrieved 5 April 2017, from https://www.illumina.com/documents/products/technotes/technote_coverage_calculation.pdf
5. *FASTQ files*. (2017). *Drive5.com*. Retrieved 30 March 2017, from http://drive5.com/usearch/manual/fastq_files.html
6. *HDF5 Data Format for PacBio Sequences*. (2017). *Homolog.us*. Retrieved 30 March 2017, from http://homolog.us/blogs/blog/2012/07/06/hdf5-data-format-for-storing-sequences/
7. *Index of /projects/fastqc/Help*. (2017). *Bioinformatics.babraham.ac.uk*. Retrieved 6 April 2017, from https://www.bioinformatics.babraham.ac.uk/projects/fastqc/Help/
8. *MiSeq Specifications | Key performance parameters*. (2017). *Illumina.com*. Retrieved 22 March 2017, from https://www.illumina.com/systems/sequencing-platforms/miseq/specifications.html
9. *PoreCamp2016 : Understanding your MinION data*. (2017). *https://porecamp.github.io*. Retrieved 24 March 2017, from https://porecamp.github.io/2016/tutorials/PoreCamp2016-02-MinIONData.pdf
10. *RAST Quality Report - TheSeed*. (2017). *Theseed.org*. Retrieved 17 April 2017, from http://www.theseed.org/wiki/RAST_Quality_Report
11. Seemann, T. (2017). *Tools to merge overlapping paired-end reads*. *Thegenomefactory.blogspot.cz*. Retrieved 14 April 2017, from https://thegenomefactory.blogspot.cz/2012/11/tools-to-merge-overlapping-paired-end.html
12. *Quality (Phred) scores*. (2017). *Drive5.com*. Retrieved 17 April 2017, from http://www.drive5.com/usearch/manual/quality_score.html
13. *v2.0 Specification Sheet*. (2017). *http://www.thermofisher.com*. Retrieved 17 April 2017, from https://tools.thermofisher.com/content/sfs/brochures/5500-w-series-spec-sheet.pdf

8. List of figures

9. List of tables

10. List of appendices

A. DVD with electronic version of this work, raw data, FastQC reports, results of assemblies and annotation.
B. Script for FASTQ file validation and basic statistics of reads.
C. Translation table between ASCII code, error probability in base call and Q score.

Appendix B: Script for FASTQ file validation and basic statistics of reads.

File name: fastaCheck.sh

Language: zsh

Description: The script check for validity of the FASTQ format in terms of number of lines, then it assesses the number of reads and gets the number of occurrences of different lengths of reads.

Input file: FASTQ file

Output file: Text file with results.

```
#!/bin/bash

#check for argument
if [ $# -eq 0 ]
  then
    echo "No argument supplied"
    exit
fi

#count the number of lines in the file
X=`wc -l $1 | cut -f1 -d' '`

#compute number of reads in the file
if [ $(( X % 4 )) == 0 ]; then
  readCOUNT=$(( X/4 ))
else
  echo "Error: the number of lines in $1 is not divisible by 4"
  exit
fi

#write number of reads into a file named similarly to input file
output_file=${1%.*}_fastaCheck.txt
printf "$1 fasta check and summary\n\n" > $output_file
printf "Number of sequences: $readCOUNT\n" >> $output_file


#starting from 2nd line extract every 4th line (i.e. the read),
#compute its length, list occurences of length
printf "Occurences of read length\n" >> $output_file
sed -n '2~4p' $1 | awk '{print length}' | sort | uniq -c >> $output_file
```

Appendix C: Translation table between ASCII code, error probability in base call, and Q score

ASCII_BASE=33 Illumina, Ion Torrent, PacBio and Sanger

Q	P_error	ASCII	Q	P_error	ASCII	Q	P_error	ASCII	Q	P_error	ASCII
0	1.00000	33 !	11	0.07943	44 ,	22	0.00631	55 7	33	0.00050	66 B
1	0.79433	34 "	12	0.06310	45 -	23	0.00501	56 8	34	0.00040	67 C
2	0.63096	35 #	13	0.05012	46 .	24	0.00398	57 9	35	0.00032	68 D
3	0.50119	36 $	14	0.03981	47 /	25	0.00316	58 :	36	0.00025	69 E
4	0.39811	37 %	15	0.03162	48 0	26	0.00251	59 ;	37	0.00020	70 F
5	0.31623	38 &	16	0.02512	49 1	27	0.00200	60 <	38	0.00016	71 G
6	0.25119	39 '	17	0.01995	50 2	28	0.00158	61 =	39	0.00013	72 H
7	0.19953	40 (	18	0.01585	51 3	29	0.00126	62 >	40	0.00010	73 I
8	0.15849	41)	19	0.01259	52 4	30	0.00100	63 ?	41	0.00008	74 J
9	0.12589	42 *	20	0.01000	53 5	31	0.00079	64 @	42	0.00006	75 K
10	0.10000	43 +	21	0.00794	54 6	32	0.00063	65 A			

ASCII_BASE=64 Old Illumina

Q	P_error	ASCII	Q	P_error	ASCII	Q	P_error	ASCII	Q	P_error	ASCII
0	1.00000	64 @	11	0.07943	75 K	22	0.00631	86 V	33	0.00050	97 a
1	0.79433	65 A	12	0.06310	76 L	23	0.00501	87 W	34	0.00040	98 b
2	0.63096	66 B	13	0.05012	77 M	24	0.00398	88 X	35	0.00032	99 c
3	0.50119	67 C	14	0.03981	78 N	25	0.00316	89 Y	36	0.00025	100 d
4	0.39811	68 D	15	0.03162	79 O	26	0.00251	90 Z	37	0.00020	101 e
5	0.31623	69 E	16	0.02512	80 P	27	0.00200	91 [	38	0.00016	102 f
6	0.25119	70 F	17	0.01995	81 Q	28	0.00158	92 \	39	0.00013	103 g
7	0.19953	71 G	18	0.01585	82 R	29	0.00126	93]	40	0.00010	104 h
8	0.15849	72 H	19	0.01259	83 S	30	0.00100	94 ^	41	0.00008	105 i
9	0.12589	73 I	20	0.01000	84 T	31	0.00079	95 _	42	0.00006	106 j
10	0.10000	74 J	21	0.00794	85 U	32	0.00063	96 `			

Source: "Quality (Phred) scores", 2017