MinCE: fast quantification of large metagenomics datasets along with species and strain abundance

Student: Thorhallur Audur Helgason

Mentor: Professor Lior Pachter

Definitions

metagenomic sample - a sample from 'the real world', such as a water sample from a lake or a fecal sample from a patient, as opposed to a sample from bacteria cultured on a petri dish. Metagenomic samples are thus expected to contain multitudes of different species of organisms.

fastq sequencing data - an unordered collection of short 'reads' from a sequenced sample, with each read being a string representing a snippet of genetic material found in the sample. For a metagenomic sample, these reads would be expected to be from multiple different species, with no readily available way to name the number of different species present or categorize reads into their respective species of origin. The method of sequencing is somewhat error prone, which has to be factored into any analysis.

k-mer - a substring of genomic sequencing data of length k. These k-mers overlap, such that for a given k-mer k_i, the subsequent k-mer, k_{i+1}, starts on the second letter of k_i. We'll call this a **k-mer overlap**.

(coloured) deBruijn graph - a directed graph created by breaking one or several genomes into its constituent k-mers. The nodes of the graph are said k-mers, with a directed edge going from k_i to k_j if k_i k-mer overlaps with k_j. For a single genome, the graph represents a compacted version of the entire genome. For multiple genomes, the graph can be coloured to distinguish from which genome each node originates. Such a coloured deBruijn graph can be used to identify variability between its different constituent genomes.

sketch - a collection of s integer values, here $s = 5000$. A sketch is produced by treating every k-mer of a genome with a specific hash function, resulting in a numeric hash value for each k-mer. The lowest s hash values resulting from this process become the sketch for said genome, functioning as a representative for the genome as a whole. This sketch is effectively a random subset of the k-mers comprising the genome and can be used as a quick substitute for comparison between genomes, the reliability of which is mathematically supported.

sketch distance - the number of hash values differing between two sketches. With sketches of size $s = 5000$, if two sketches share 4997 hash values, their sketch distance would be $5000 - 4997 = 3$.

union-find cluster - a tentative structure in the clustering of sketches into cliques. These UF clusters are formed by calculating the sketch distance between every sketch in the corresponding data set. UF clusters are formed with respect to a given threshold, which at the moment is 5. If the sketch distance is less than or equal to this threshold value of 5, the sketches are merged into one cluster using a union-find approach. This approach can lead to a runaway process, where large UF clusters have a tendency to get larger, because the only criteria to merge into a cluster is having a sketch distance of 5 or less to any member within the cluster. The advantage to this approach is that every sketch is only a member of one cluster and within each cluster every sketch is at most at distance 5 from its closest sketch within the same cluster. Thus every cluster has a void surrounding it of at least width 6, making it a good first clustering step.

clique - the final step of the clustering process. Cliques are currently formed within each UF cluster with a bottoms up approach, which is described in further detail below.

1 Abstract

Dramatic cost reductions in genome sequencing coupled with improvements in accuracy during the past two decades have facilitated broad sequencing efforts to catalog all genomes of living organisms. In particular, large public databases now house hundreds of thousands of bacterial genomes, most of which have been obtained through sequencing of cultured bacteria. However many bacteria are difficult to isolate and/or culture. To study them, "metagenome" sequencing approaches have been developed that rely on sequencing of short fragments from environmental samples to identify microbes in their natural habitat. The associated computational problems are manifold and complex, starting with the need for algorithms to align hundreds of millions, or even billions of DNA fragments to large existing databases.

We introduce MinCE, a method for quickly identifying bacterial species and strains in metagenomic samples. MinCE preprocesses a reference genome database to facilitate rapid lookups. Subsequently, the relevant genomes serving as the source for a collection of DNA fragments can be identified. At present, MinCE can be used to identify genomes from a 13.5Gb reference database containing 258,339 genomes of Eubacteria and Archaea. We present results on simulated data that suggest that MinCE has high sensitivity and specificity, making it suitable for metagenomics analyses.

2 A short introduction to Bioinformatics and Metagenomics

Since the publication of the first complete bacterial genomes in 1995 [1], the scope of sequencing efforts has grown exponentially. Today, online libraries such as RefSeq [2] and GenBank [3] house hundreds of millions of isolated sequences from hundreds of thousands of different species. The fastest growing kingdom of sequenced organisms is that of *Eubacteria*, which stands to reason as it is the biggest kingdom with respect to number of member species. The revolution of genome sequencing has opened new avenues of research and from them stems the field of Bioinformatics. To understand the nature of the data, a researcher needs a background in genetics and other sectors of biology. To glean information from the data, a thorough grounding in mathematics and computer science is also needed, as sequencing data is comprised of monotonous string segments of four characters - A, C, G and T or U - representing the four nucleotides of DNA or RNA, respectively. This provides challenges unique to this field, involving optimization of genome-specific search algorithms, advances in graph-theory as a representation of genomes and statistical inference methods based on genetic principles.

When sequencing a sample of genetic material, the optimal output would simply be a continuous print of the whole genomes within the sample. If this sounds too good to be true, it is because it currently is. There have been recent advancements in long-strand sequencing [4] but the most reliable method available today - i.e. the least error-prone one - involves breaking the genetic material into short strands prior to sequencing. The output is thus a long text file, printing each of the short strands into a string. The length of these strands is normally between 70-150 bp long and the order of the *reads* in the output file of a sequencer is random. To obtain the whole genome, these reads need to be properly aligned to form a continuous segment for each chromosome. A proper sample will undoubtedly contain several individuals of each species

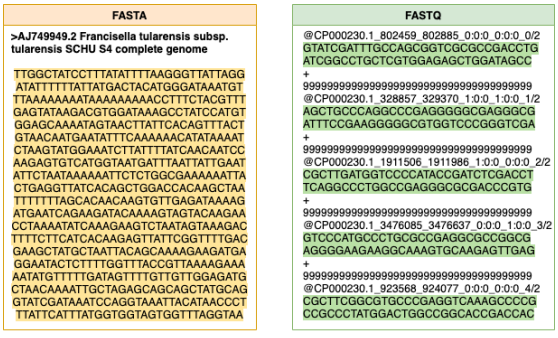


Figure 1: Aligned genomes are stored in .fasta files. Raw data from sequencers is stored in .fastq files, each *read* (green) representing a short strand originating somewhere in the sample.

[1] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4361730/
[2] https://www.ncbi.nlm.nih.gov/refseq/
[3] https://www.ncbi.nlm.nih.gov/genbank/
[4] https://nanoporetech.com/

- be they single cell bacteria or simply cells from the same individual - so one should assume a fair redundancy in these reads, i.e. each segment of the genome should appear multiple times. While this method is the least error-prone of those currently available, one should assume that one in every 100 or 200 bases read would be faulty on average. Such cases can be screened for, by looking only at those sequences which appear often - as the likelihood of identically faulty reads appearing often quickly becomes very low. For a sample of a single small species, e.g. a pure bacterial culture in a petri dish, a compilation of the true, whole genome is achievable through bioinformatic methods presently available. One can for instance construct a single deBuijn graph from every k-mer in every read and find a path through the said graph which corresponds to its whole genome, either a single continuous segment or several chromosomes or plasmids. For a sample from the plant kingdom, genomes can be very large and full of repeated segments, i.e. long mostly-identical segments interspersed throughout the whole genome. That makes this undertaking much more difficult, as every step towards compiling the whole genome needs to be tentative, as we can't be sure how many these repeating segments are. The corresponding deBruijn graph would be huge and likely full of loops, making it difficult to decide on a one true path through our graph. This paper focuses in on a case with commonalities with both of those previously mentioned; the sequencing of metagenomic samples.

Metagenomics refers to the study of microbial samples obtained from real life situations, as opposed to pure cultures. These could be swabs from a restaurant kitchen counter, a soil sample from a forest bed, water samples from an aquifer or a stool sample from a human gut. All of these are of course anything but pure - they would contain multitudes of microbial species, some in high concentration and others only minimally present. When sequencing such a sample, several passes are required to be sure to cover each associated genome adequately with redundant reads. Thus, the total size of the sequencer files is typically very large - easily tens of gigabytes - and the number of different genomes present is unknown. To identify the genomes present in our sample, we would need to concurrently align the expansive set of reads from our sequencer output to the genomes of every species possibly present in said sample. We have these genomes available online through the libraries previously mentioned. However, keeping all of these different genomes available for alignment would prove a very time-consuming and computationally demanding task and is therefore not a feasible approach. In addition, many bacteria share large sections of their genome across distantly related species and the specificity of variation within a species is often more important than a distinction of species alone. The last part of this section will detail how this comes about and why it matters.

The general concept of a species has historically been a contested one. Attempts to define it based on reproductive isolation (the inability of two organisms to produce fertile offspring based on a variety of factors), while an effective distinction for the animal kingdom, prove insufficient for other kingdoms, in particular *Eubacteria* and *Archaea*. A key reason is that micro-organisms display a variety of pathways to exchange genetic material, apart from having genetically mixed offspring (often dubbed *vertical gene transfer*). One such option is *horizontal gene transfer*, where segments of genetic material can flow from one bacterium into another, where it becomes integrated into the latter's whole genome. This alone provides a significant obstacle in determining a comprehensive species-mapping of the kingdom, and other such pathways complicate the undertaking even further.

As the definition of the species-concept is a compromise between a fair representation of the living world and the facilitation of dialogue between scientists on these matters, the concept of a species at the microbial level should be understood as something closer to a functional shorthand, rather than a robust truth. This does not mean that the concept is in no way applicable, as evolutionary forces do of course act on bacterial populations as any other and there are general commonalities to be found within groups of related individuals. As such, it is very useful to group large sections of the expansive bacterial kingdom under a common species, such as *Escherichia coli*, and discuss general characteristics of those members. However, the concept of a strain becomes very important as a subdivision within a species. In contrast to the human genome, in which 98.5% of the 6 billion base-pair genome is made up of non-coding genes[5], the average bacterial is only about 5 million base-pairs and, in general, every segment plays an important, functional role. This lean nature of the genome - likely a result of natural selection for smaller genomes - means that small changes to bacterial genomes are

[5] Alberts, p. 29

more likely to result in significant changes in its overall nature, than are similar changes in larger organisms with more redundant genomes. Looking again to the example of *E. coli*, several strains of the bacterium are a vital part of a healthy human digestive system, while other strains - which differ only minimally from those present in our gut - are responsible for a variety of infections, ranging from mild to life-threatening in some cases.

3 Origins of MinCE

Work on MinCE began in June 2021 as part of a summer work program for students at the University of Iceland. Professor Páll Melsted hired me and Jóhannes Nordal, both students of computer science, to realize his concept for a software tool to identify species and strains in metagenomic samples. The architecture of the project would build on the *sketch*-objects introduced with Mash[6], but extend the specificity of the tool beyond the capabilities present in Mash. A prototype was finished by end of summer with much appreciated support from Prof. Melsted.

Mash's *sketch* refers to a numerical representation of the entire genome. The process of sketching involves treating every k-mer in the genome file with a hash function, which turns the string of 31 characters into a number. The hash function should map every unique sequence of 31 nucleotides to a unique number, which requires a very large span for the hash function. The total number of variations for a string of length 31 - where each character can take 4 different values; A, C, G, or T - is 4^{31} or roughly 4 billion billion possibilities. Thus, the resulting hash value of a 31 character string could be an 18 digit number, which doesn't seem like a large reduction in size for the genome. However, this large set of possible 31-mer forms is an indicator of how a set of those could uniquely identify a genome. A well defined subset of the resulting hash-value number set could therefore be a fairly reliable representative of genomes and, in particular, serve as a quick medium for comparison between genomes, provided the sketches are created using the same hash function and the same definition of a hash-value subset. In Mash, this subset is simply defined as the lowest s values produced by the hashing process, where s is the desired size of the resulting sketch. The hash function maps irrespective of the string's pattern, so similar strings should not (necessarily) map to similar hash values. Therefore, the lowest s values constitute a random subset of sequences from the entire genome.

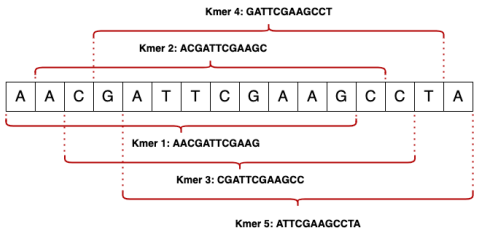


Figure 2: A string of length 16 broken up into 5 k-mers with $k = 12$. Overlapping k-mers share a subsequence of k-1.

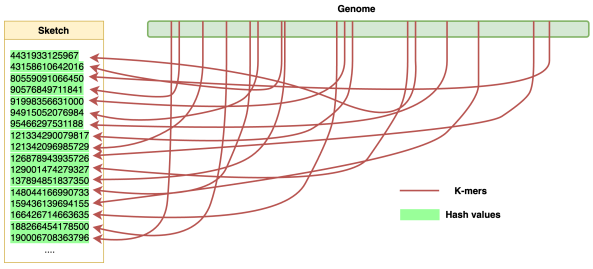


Figure 3: Hash values are not correlated with string pattern. The lowest s hash values thus constitute a random subset of sequences from genome.

As the size of our k-mers and the size of our sketch increases, so does the accuracy of our representation but so also does the size of our sketch file. The k-mer size in MinCE is 31 and the sketch size was chosen to be 5000. Two bacteria of the same species would almost assuredly yield identical sketches, as the tiny variation between them is unlikely to be in the reserved size of 5000 subsequences of length 31. More distantly related species would share a smaller percentage of sketch values. This facilitates easy identification of species for unknown samples, provided a reference database of catalogued sketches. However, the sketches' limitations are revealed when trying to discern between very closely related species or strains of the same species. For a real life sample

[6] https://mash.readthedocs.io/en/latest/

containing *E. coli* strain O157:H7, we would expect tens or hundreds of other *E. coli* strains to match perfectly to our sketch. Even related species could in some cases have perfectly identical sketches to our specific strain, which severely diminishes the use-case of this representation.

Professor Melsted's concept was to create a huge database with representatives of fully sequenced *Eubacteria* and *Archaea*, including fully sequenced strains within each species. With this static library of sketches, one could map the distribution of the sketches and identify the *clusters* of sketches which proved too similar to reliably distinguish through the sketch alone. These clusters of species and strains could be analyzed at the full-genome level to extract several distinguishing sequences from each genome, which would differentiate between the members of the cluster. Genomes relating to sketches in those clusters could be piped into a colored deBruijn graph and translated to a matrix, describing the variability of the cluster's genomes. This would allow a deterministic algorithm to identify the variability between the genomes - the variability which did not find its way into the sketches - which could be extracted and saved in specific files relating to each cluster. By pre-processing our dataset in this way, we could provide these prepared sequences in the event of unsatisfactory results from sketch comparison and discard those candidates whose sequences weren't found in the sample. This static library, with a small retinue of front end functions for users, could therefore identify species and strains in metagenomic samples with specificity and sensitivity.

A final thing to address, before getting into the inner workings of MinCE, is how we can create a sketch from a sample containing multiple species. For a pure sample of a single species, the sequencer's output - a file with the extension *.fastq* - will contain only genetic information for that single species, as well as some faulty reads which do not represent the true genome. For such a sample, we would scan every k-mer found in the .fastq file and log how often we have seen each one. Seeing the same erroneous k-mer twice or thrice in the sample by chance is very unlikely, as the pattern of k-mers is quite unique. For instance, two randomly chosen k-mers will have a $\frac{1}{4^{31}}$ chance of being identical. Thus, we can screen for these faulty reads by only accepting k-mers which we see a given number of times throughout the entire sequencing file. From those accepted k-mers, we create the sketch by choosing the lowest s corresponding hash values. For a sample containing multiple species, with the specific number of differing genomes unknown, a different approach is required. Conveniently, the MinCE library of sketches is static, so we know every hash value we could possibly want to discover. Looking every hash from our sequenced k-mers up in a database of MinCE's sketches would slow our process down considerably but we can utilize the fact, that our sketch values are chosen by their property of being *low*. Thus, we can discard every hash value that exceeds the maximum hash value in MinCE's library of sketches. This saves quite some time, as the max hash value is $\sim 1/3$ of the total span of the hash function, 4^{31}, so we only need to inspect every third hash value on average. Every hash value lower than our max value is used to create our sketch. This is a *MinCE-specific* sketch, tailored for our specific library. Its size is no longer fixed s, but should be expended to grow as a function of the number of different genomes present. When comparing this larger sketch - currently called a *mega-sketch* - to every sketch in our library, we could expect to find multiple perfect matches to distantly sketches throughout our database. An analysis of how this technically unbounded sketch size affects the likelihood of matches to fixed size sketches in our database should be carried out in later stages of research.

4 A short overview of the processes of MinCE on runtime

MinCE is a twofold software and can be separated into a front-end and a back-end. The back-end contains all of the information needed for MinCE to run and is comprised of sketches of size $s = 5000$, clusters containing identical or nearly-identical sketches, and distinguishing sequences relating to sketches in each cluster. The number of genomes currently represented in MinCE's library is 317,542; 311,480 *Eubacteria* genomes and 6062 *Archaea* genomes. It is built upon the Genome Taxonomy Database release v207[7]. The prototype of MinCE contained each sketch as a file and also specific files for the distinguishing sequences. Such an architecture is however redundant, since both parts can be summed up in a single file each. Rather than load each of the

[7] https://gtdb.ecogenomic.org/

$\sim 317K$ sketches into memory and compare each sequentially to our input sketch, we create a dictionary which maps hash values to sketches. In MinCE's framework, these dictionaries are called hashmaps and, given a hash value, will return the indices of those sketches which contain said hash value. Therefore, an input which produces a sketch of size s will only require s look-up procedures to compare the input to all members of MinCE's library. The resulting `sketch.hashmap` file is 17 Gb in size and contains most of the information needed to run MinCE. The same principle is applied to the distinguishing sequence files, where each sequence is only stored as its first constituent k-mer and hashed in the same way as normal sketch values. This produces the much smaller `seq.hashmap` file, only taking about 12 Mb of space. Having introduced this framework, we will now discuss MinCE's front end processes.

When MinCE is given a .fastq output file from a sequencer, the program iterates through the reads, breaking each one into its constituent k-mers. The k-mers are then treated with the same hash function used to create the sketches in MinCE's library. We can name these resulting hash values of k-mers *hash-mers*. If a hash-mer is lower than MinCE's maximum hash value, it looked up in the `sketch.hashmap` file, which will return a list of sketches containing that hash-mer - which might be none. Each sketch in this list is logged as having a hash-mer in common with our input file and given a point. By the end of the process, a sketch with a perfect match will have received 5000 such points.

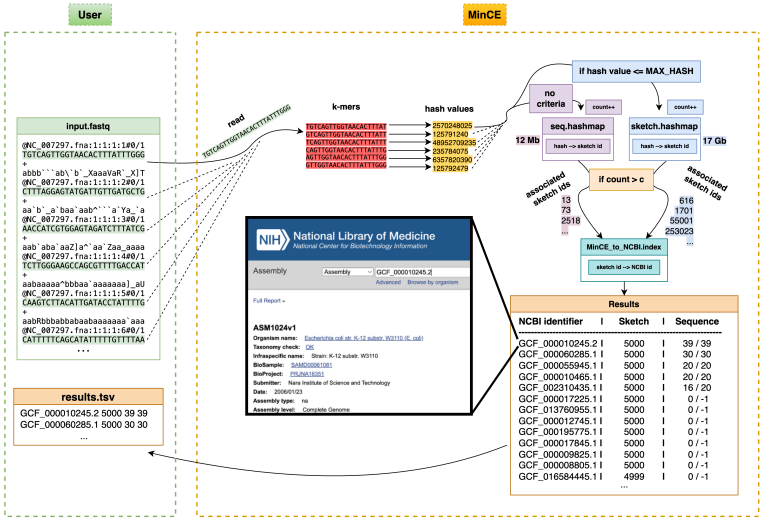


Figure 4: The MinCE front-end process. Reads are broken into k-mers, which are then treated with a hash function. Resulting numbers are compared with the `sketch.hashmap` and `seq.hashmap` files to get final results. Results are presented with either GenBank (GCA_*) or RefSeq (GCF_*) identifiers, which can be searched for in the NCBI database to get further information on the organism.

When we have our results from sketch comparison, the next step should be to inspect whether any of the top results are members of a cluster, i.e. whether the top matches contain identical sketches. If this turned out to be the case, we would ordinarily have to iterate through the entire .fastq file again to look for those distinguishing sequences. Iterating through a .fastq file is a time consuming process, as they can easily grow very large. However, this step has already been run concurrently with our first pass through the .fastq file. After we've looked any hash-mer up in our `sketch.hashmap` file, it is subsequently fed into the `seq.hashmap` file - regardless of its size with respect to the maximum hash value in MinCE - and sketches with matching distinguishing sequences get a point logged, separate from the sketch value points. Since the distinguishing sequences are selected by whether they contain necessary variability within each cluster, they will not necessarily have the property of being low. Therefore, we cannot disregard any interval of our total span of hash values, as our distinguishing sequences will map to a wide range of values. This means that every hash-mer will need to be looked up in `seq.hashmap`. Experimentation on whether the deterministic algorithm selecting distinguishing sequences could select sequences based on low hash values as well as variability might reveal ways to minimize the span of these sequence hash-mers.

Once the entire .fastq file has been iterated over, the results are delivered via command line and a tabular result file with extension .tsv. Results are displayed through the GenBank or RefSeq database identifiers, with a prefix of 'GCA' or 'GCF' respectively. Those identifiers are accepted by the NCBI website, which can offer further

details on the specifics of the organism in question.

5 Clustering the sketches

As might be apparent, the runtime processes of MinCE are quite simple. This is made possible by extensive pre-processing of the dataset before any user enters the picture. By anticipating the cases where the resolution of the sketch object will fail to deliver concrete separation of candidate genomes, we can prepare the characteristics of each genome necessary to resolve the issue in each case. The first step in this pre-processing stage is identifying the clusters of sketches which will require such an intervention. This, it turns out, is no simple task.

A simpler case might be helpful to visualize the problem. If the distance between two sketches were merely a two-dimensional property, the problem could be presented as Figure 5. The red circles represent the position of sketches in this 2D space and their relative distance is simply the Euclidean distance between them. If two sketches are two close, a cluster should be formed to include both of them. The true distance metric is however very far from two-dimensional. The distance between two sketches is simply the number of shared sketch values subtracted from the total sketch size, $s = 5000$. A glimpse into the multi-faceted nature of this distance metric, we can think about the ways two sketches can have a distance of 1 between them. Such two sketches, $\boldsymbol{A}$ and $\boldsymbol{B}$, would share 4999/5000 hash values. The possible combinations of which hash values are different correspond to one for each sketch value, i.e. we have 5000 different possibilities of differing hashes. For each of those values, we then have a myriad of possible hash values to choose from that would be the differing one. The total span of the hash function covers 4^{31} or roughly 4 billion billions. The true span of the hash values in our MinCE library is but a third of that and the total number of hash values in our library is only around a third of a billion; 366,053,393 in total. This still exemplifies how densely packed these sketches might considerably be. For each pair of sketches to differ by a single value, we have around 350 million different values which could be located in any of the 5000 locations of the sketch value list.

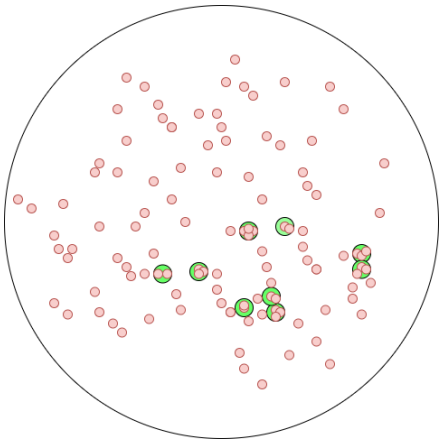


Figure 5: A two-dimensional representation of the clustering problem. Green circles identify clusters of red circled sketches.

The true density of the sketch distribution is thankfully milder than a worst case scenario of this might suggest. On average, with the 366,052,393 sketch values distributed on 317,542 sketches, each sketch has 1153 unique hash values. This would seem quite high, as these should be unique identifiers for each species. It is important to keep in mind, however, that the genetic code is only made out of 4 distinct nucleotides, so repetition is sure to occur. This is made even more probable in the bacterial kingdom, where horizontal gene flow makes it possible for bacteria to exchange genetic material across species. The true distribution does however present some challenge, as will now be further detailed.

The initial step towards clustering the dataset is a tentative one, but nevertheless quite useful. The sketches are joined to groups using a union-find algorithm, which accepts a threshold value t as input. This threshold value indicates the minimum distance between two sketches that triggers a joining of the two, so for a threshold value of $t = 3$, two sketches having 4997/5000 or more shared hash values will be joined to a group. These groups persist in memory as the process continues, so each groups grows as we look at more sketches and more are joined to the evolving group. This reveals the major drawback to the union-find approach. For a large group of 15 sketches, the current sketch under consideration only needs to have distance $d \leq t$ to *any* member of the group to be joined to the group entire. This is analogous to a party, in which you have to have the number of a present guest saved in your phone to be granted entry. As the party grows, it becomes ever easier for new

people to join, as the chances of having any members phone number saved increases with the size of the party. Therefore, some of the smaller union-find clusters will prove useful, but others will grow obscenely large and require further subdivision to separate the larger 'party' into cliques of those truly well acquainted.

A few approaches were explored to further subdivide these union-find clusters. Using union-find with a threshold of $t = 3$ produced 10,231 clusters, which mostly were very small. The median size was 2 with a mean of ~ 3.44. The largest of those clusters were however not suitable in their current state, with the largest one counting 489 sketches and the mean of the largest 50 clusters being 93.2. Professor Pachter suggested I look into *Annoy*[8] for approximate nearest neighbor clustering for these larger ones. Regrettably, Annoy could only accept vectors in n-dimensional space, whereas our data was not well fitted to such a representation. Each sketch could certainly be treated as a vector of 0's and 1's, where 1 in cell j meant that the sketch contained hash value j. A truly naïve approach would place these vectors in the space of all possible hash values, a 4^{31}-dimensional space with each vector maxing out the memory of a normal desktop computer. This space could be reduced by only looking at the values present in the dataset or trying to map a distance matrix of the clusters onto a set of vectors in lower-dimensional space, a process known as *multi-dimensional scaling*. Even for relatively small clusters of ~ 20 sketches, the strain of this multi-dimensional scaling - a metric for how well the lower-dimensional simplification represents the characteristics of the original distribution - far exceeded what was considered acceptable, from what I had read about the process. Therefore, I concluded that another approach was needed to generalize the process for all clusters.

Continuing on with the party analogy, it is reasonable to assume that the network of people admitted to the party could be broken up into isolated friend groups. In other words, there might be single edges in the graph between two vertices which, when cut, might divide the whole graph in two. $Charlie$ might be friends with people from group A and group B, but he is the only person linking those two groups together, so by removing that connection, we could have two isolated groups and place Charlie in both of them. One mathematical way of identifying such *weak links* in a graph is called a *minimum cut algorithm*. I implemented a minimum cut approach which found the weakest link in the graph, assuming that one traversed between the furthest points in the graph (corresponding to the maximum distance in the respective distance matrix for the cluster). Once the weakest link had been found and cut, the vertices connected to that edge would need to be added to both resulting graphs. This is crucial, since we need to keep sketches connected in our graph, if their distance is less than our threshold t. Therefore, each step in this algorithm would identify the weakest link in the graph, cut it and divide the neighbours of the cut into both sub-graphs. Then each sub-graph would receive the same treatment as its parent-graph, being further subdivided. The process would halt when a sub-graph contained fewer members than some number k - here placed at $k = 7$ - or when the subdivision and subsequent backtracking by adding the members adjacent to the cut would result in either sub-graph being the same full graph as the original one. This was a task undertaken at the beginning of my summer at Caltech and I spent some time trying different variations on this approach but all of them halted in the first steps of the iteration, due to the fact that the subdivision and subsequent reallocation of neighboring vertices return the original graph. Since the data could not be visualized in three dimensions, it became quite difficult to define the problem in a concrete way.

Since much of the remaining work relied on *some* sort of subdivision being available for these larger clusters, I wrote a custom algorithm which prioritized the grouping of identical sketches and did in fact return a set of smaller clusters from the larger union-find clusters. These smaller sub-clusters were called *cliques* and the algorithm was dubbed *bottom-up cliques*, as it starts with the smallest distances and works its way up. The 0-cliques were the first to be formed, consisting of sketches having 0 distance. These groups of identical sketches were extrapolated from the distance matrix of the corresponding distance matrix. When those 0-cliques had been formed, those sketches were removed from consideration for the next iteration. Thus, a smaller distance matrix was analyzed to find the clusters of sketches having distance 1 and those would form 1-cliques. The largest drawback of this approach is that some sketches might lose out on their best pairing. Let's say sketch j has distance 1 to a cluster of identical sketches and distance 2 to a cluster of sketches with distance 1. During the first step, the closest match to j will form a 0-clique and be removed from consideration. The next step will form a 1-clique from j's next-best match and so on and so forth. It is therefore possible, that j will in

[8] https://github.com/spotify/annoy

fact be paired with its *least favourable* pairing, if this process continues on in the worst case possible. The algorithm halts when it reaches a threshold of q, in which step it should start forming q-cliques. For this current version of MinCE, this was set at $q = 4$. The largest cliques were therefore 3-cliques and once the algorithm reached step $q = 4$, any remaining sketches not already paired to a clique (those who missed out on their previous matches due to them being removed) are added to the clique of their closest sketch in the cluster. This is of course also problematic, as some sketches might be added to a distant 3-clique in the last step and their closest member might be added to another clique in the remainder-step. This approach was a temporary work-around and was meant to be replaced later on. I experimented with multiple, overlapping cliques - similarly based on prioritizing low distance - but the resulting number of cliques was over 60,000. The are nevertheless some benefits to this approach, as it prioritizes the clustering of identical sketches and does not "muddy" the 0-cliques with more distantly related genomes. When selecting for distinguishing sequences, it is preferable to choose from a smaller set of sequences, which carry key variation in an otherwise similar genome-set, rather than waste some of those sequences by picking something more separating between dissimilar genomes. It also raises the chance of false positives, but should in theory not raise false negative rates. The presence of false positives in MinCE's results is acceptable but missing genomes that should be present is more serious. If some sketch a wind up in a sub-optimal clique, the resulting distinguishing sequences might prove more general than otherwise, i.e. the sequences separating them from their closest relatives have a lower likelihood to be chosen. As there are more available sequences to distinguish genome a from those present in this sub-optimal clique, some of the chosen sequences for a might be present in other genomes in the clique best suited for a. Therefore, when searching for a we might get false positives in said clique, but we are unlikely to miss a entirely in our results.

I presently consider this to the biggest unsolved issue with MinCE. The work this summer required me to focus on multiple disjointed areas of the software tool, so they could all come together in the end. This bottom-up approach was a temporary fix for an issue which I hoped to solve later on, returning to it often throughout the summer. I started work on multiple variations but was ultimately not able to solve it before end of summer. A focal issue is that of the *role* of the threshold value. The threshold for clique-forming should reflect the resolution of the sketch object. If two sketches of size $s = 5000$ for the same organism can differ by 5, then the threshold should be larger than $t = 5$. Working on the dataset this summer, multiple clusters with 0 distance formed from same-species strains, so perhaps the clique-threshold should in reality be closer to 1 or 2. This is ultimately something that empirical work with MinCE will shed more light on. The first steps in developing the software tool further will focus on this issue.

6 Selecting distinguishing sequences from cliques

The function of a clique is to group together the sketch representations of genomes which are too similar to reliable distinguish between. By this categorization, we can anticipate instances where additional information is needed to eliminate some contenders and confirm others. MinCE does this by supplying sequences of genomic information which are present in some genomes in the clique and not in others. A set of such distinguishing sequences for a clique should be able to reliably distinguish between each and every member of said clique. Our dataset is built from .fasta files, containing the whole genome of a representative of each species and strain. Since we do have to account for variability between individuals, we cannot use our representative .fasta file as gospel, as individuals in the wild might not be completely identical to our template. Therefore, we would also like to select multiple sequences which distinguish between the members of the clique with some redundancy.

The first step in choosing these sequences is to construct a deBruijn graph from all genomes represented in the clique. A deBruijn graph is effectively a network of k-mers, where k-mers which share a subsequence of length $k - 1$ (in other words, which *k-mer overlap*) are connected by directed edges. Therefore, the deBruijn is not a single row of k-mers, ranging from the beginning of the genome to its end, but rather a representation of how the constituent k-mers connect. Each k-mer only appears once in the graph, with all its k-mer overlapping variants exiting or entering it. Therefore, a deBruijn graph is only useful if its constituent k-mers are large enough to be fairly unique, such as $k = 31$. A deBruijn graph containing k-mers from multiple origins can be

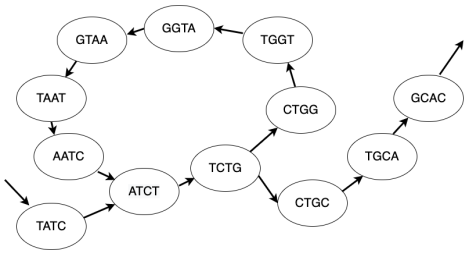


Figure 6: A deBruijn graph made from k-mers with $k = 4$. Subsequent k-mers *k-mer overlap*. The loop in this case is not ambiguous, but another loop with same entry and exit points would result in two equivalent paths.

Figure 7: A colored deBruijn graph represented as a subway map. Stations where multiple lines stop equate to genomes having identical sequences. The brown and yellow "genomes" share a long, identical sequence.

colored to distinguish these origins and retain that information. A good analogy is to imagine a subway map, such as the one in Figure 7. The lines are colored by each genome and those stations which many lines visit are analogous to share sequences between the genomes. The subway representation seems to represent 6 fairly dissimilar genomes, though the yellow and brown one share a large, contiguous sequence. In the case of cliques, the genomes are pre-selected by of their similarity, so the representation of that deBruijn graph would be more of a tangle. Working with such a visual representation is not well suited for a deterministic algorithm, so we opt instead to output the deBruijn graph as a matrix of 0's and 1's. The columns of the matrix represent the different constituent genomes in the graph and each line represents a *unitig* in the graph. A unitig is simply an unbroken sequence of k-mers in the graph, in other words a path that does not split and is not ambiguous. One such unitig would be the inside of the loop in Figure 6, "CTGGTAATC". By definition, a unitig is at least of length a k-mer, k, and each k-mer within a unitig is unique in the entire graph - otherwise the unitig would not be unambiguous.

This bit-matrix $\boldsymbol{B}$ of 0's and 1's contains all the information of the deBruijn graph, apart from the specific sequences of the unitigs. $\boldsymbol{B}[i, j] = 1$ indicates that genome j contains the unitig in row i, otherwise $\boldsymbol{B}[i, j] = 0$. We can write the sequences for each unitig to a different file with an index matching the order of rows in our matrix. Then the problem is simply to choose a set of row-vectors from our bitmatrix $\boldsymbol{B}$ which cover our set of constituent genomes. For a clique of 3 members, we would have three columns representing members m_1, m_2 and m_3. A row-vector $r_i = [0, 1, 1]$ would discern m_1 from both m_2 and m_3 but could not disambiguate between members m_2 and m_3. We can define the characteristic of how many 1's the row-vectors contain to be their $degree$. It follows, that the best row-vectors to choose would be those with only a single 1, i.e. those with degree 1.

Professor Pachter suggested I look into Deterministic Column Subset Selection, an algorithm to select a subset of columns from a matrix while retaining the original matrix's spectral qualities. The algorithm was outlined in a paper[9] written by Prof. Pachter and his students and the code was available online[10]. The subset of columns output by the algorithm (with the input matrix transposed) was unfortunately not in line with the purposes of the sequence selection, as the discarded column-vectors were most often those having low degree. Neither were the discarded columns perfectly in line with what was needed, so DCSS could not be implemented as a negative definition of which columns to choose. I decided against pursuing to amend the algorithm to suit my needs, as I was not very familiar with some of the steps involved.

[9] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6347249/
[10] https://github.com/srmcc/dcss_single_cell

Instead, I chose to re-write the prototype algorithm from last summer and optimize it where possible. That prototype had been entirely written in Python and was not very streamlined, so I restructured it and wrote an updated version in C++. The algorithm, dubbed the *greedy sequence selection algorithm* or GSS, starts by reducing the entire bit-matrix B to a smaller matrix R, which only contains a single row-vector of each *vector-form*. A vector-form is here used to mean the distinct pattern of 0's and 1's in a row-vector. A clique with 3 members would therefore have 8 different vector-forms:

$$[0,0,0] \quad [0,0,1] \quad [0,1,0] \quad [0,1,1] \quad [1,0,0] \quad [1,0,1] \quad [1,1,0] \quad [1,1,1]$$

The first vector-form is of course never present, as each unitig in the deBruijn graph has to appear in at least 1 constituent genome. The last vector-form is one of purely 1's, which of course does not distinguish any members. Many such row-vectors appear in these deBruijn graphs made from related genomes but for the purposes of distinguishing sequence selection they can all be removed. The reduced matrix for the example of a clique of 3 members would therefore at most have 6 rows - a dramatic reduction indeed. For each row-vector of a certain vector-form, we attach a list of the unitigs which take that vector-form. This list is simply the indices which point to the unitig's sequence in a separate file. As this reduced matrix R is constructed from the larger one, B, we take care to order the row-vectors in order of increasing degree value. A final object to create is then a connection matrix, C, which is an $n \times n$ matrix for a clique with n members. This matrix logs how many times the members have been selected as part of a row-vector which does not distinguish between them. For the example above of $r_i = [0, 1, 1]$, the connection matrix C would be updated by adding the vector r_i to rows 2 and 3. A high number in $C[i, j]$ indicates that members i and j have been paired often together and a subsequent choice for either should avoid unitigs which appear in both. By contrast, a low number in $C[i, j]$ indicates that choosing a unitig present in i and j is acceptable, as the unitigs chosen so far do not appear often in both i and j.

GSS accepts a parameter N for how many sequences should be selected for each member. For the current version of MinCE this was set to $N = 20$ but it is entirely possible that not 20 sequences were available. For a clique containing only two genomes of differing strains within the same species, the total variation between them could be located in 2 or 3 nucleotides. It is even possible that one strain is a subset of the other, meaning that the smaller one cannot be distinguished from the larger one except by absence of a few key sequences. In other cases, a recombination or duplication of the genome might have created a new strain and the total set of k-mers would remain identical. Therefore, not all cliques will result in a set of distinguishing sequences for all members.

I will now go over the process involved in the GSS algorithm. GSS is given the input N and proceeds to create the reduced matrix R and an empty connection matrix C (initialized with all cells as 0). Having created R, we have information on how many unitigs are available for each member of the clique. We define the total number of available unitigs for member m_i as T_i. The algorithm starts with examining the top row-vector of R. The top rows will be all vector-forms having degree 1, as R is ordered by that metric. For each vector-form, GSS picks as many as N unitigs uniquely defining each member of the clique. These unitigs are chosen by looking at the list associated with each row-vector form and choosing the first index in that list, which points to a sequence in a separate file. That index is then removed from the list. In many cases, N or more such unitigs will be available for each member of the clique and the algorithm will conclude. In the case that the number those unique unitigs does not reach N, we move over to unitigs of higher degrees. This next step is iterative and goes as follows: From the connection matrix, find the member j which has most unitigs not yet chosen. The corresponds to choosing the member j, where $T_i - C[j, j]$ is the largest. From the set of row-vectors in R having $r_i[j] = 1$, choose the row-vector r_b which produces the lowest scalar product when vector multiplied to line j in C. This corresponds to picking the vector-form which pairs member j to the other members in the clique that have most infrequently been paired with j in previous iterations. If multiple row-vectors produce equal scalar products, the first one to be inspected will be chosen, which will be the vector-form of lowest degree (owing to the ordering of R). From the list of unitig indices associated with row-vector r_b, pop the first one (choosing it and removing from the list) and add that index to our list of chosen unitigs. Finally, update the connection matrix C by adding

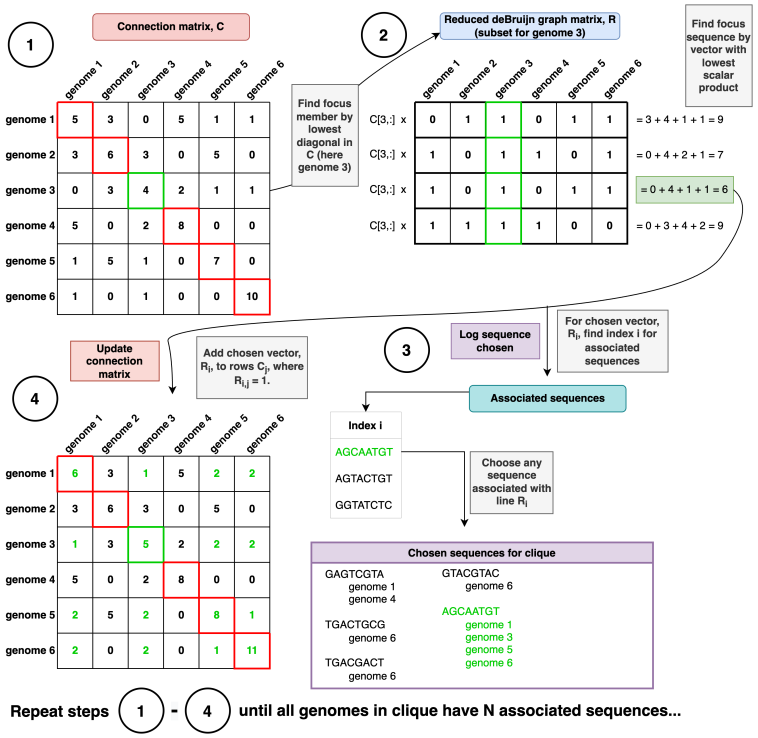


Figure 8: The iterative process of GSS. **1)** The member with fewest sequences is selected as the focus for this iteration. **2)** The vector-form joining the focus member to other members least frequently paired with focus member is chosen. **3)** The index of a unitig corresponding to the chosen vector-form is popped from the list and added to our chosen list. **4)** The connection matrix is updated with the chosen vector-form.

the vector r_b to each row C_q, where $r_b[q] = 1$. If any diagonal cell in C is still not equal to $\min(T_i, N)$, repeat the iterative process. From each unitig we can simply extract the first k-mer available, since each k-mer in the unitig is unique. Keeping the sequences as k-mers will allow us to hash them as we do the sketch values.

When this iterative loop terminates, each member of the clique should have N associated sequences or the maximum amount of sequences available from the deBruijn graph. Since each step always looks at only a single member, rather than holistically at the total distribution of sequences, it is possible for the iterative step to conclude with some members being always paired together, despite some sequences being present to distinguish between them. This can happen if no uniquely identifying unitigs are available for either member and neither is ever chosen as the lowest member in the connection matrix. This is of course very unlikely, but nevertheless the algorithm concludes by iterating over each row of the connection matrix and finding the maximum value in each one. If the maximum value appears twice in the row, then that every chosen unitig for that member also appears in another individual. To examine whether this can be rectified, the algorithm finds all sequences present in one member and subtracts those present in the second one. If any remain, the algorithm chooses 2 of those to add to our list of chosen distinguishing sequences.

7 Benchmarks of MinCE

The entire MinCE program takes 17 Gb of space. Almost all of that belongs to the $\sim$ 17 Gb `sketch.hashmap` file. This could be brought down by saving the hashmap as something other than plaintext. MinCE needs access to the entire file to run, so loading this file also accounts for the majority of MinCE's runtime. On a 16 Gb RAM M1 MacBook Air 2020, the loading of the `sketch.hashmap` file takes around 22 minutes. Provided a set of

two 4.6 Gb .fastq files (complementary files from the same sample), MinCE iterates over the reads, compares the resulting mega-sketch to every sketch in the MinCE library and delivers results in around 7.5 minutes. The maximum RAM usage, obtained by using zsh's `time` function[11], was measured to be $\sim$ 4.4 Gb. This RAM should be fixed irrespective of .fastq file size and is made possible by storing the counts for each k-mer in the hashmap object loaded into the program, rather than a separate object counting every k-mer seen. To screen for faulty reads, the user is able to set a threshold value c as the minimum number of times k-mer has to be seen to be considered valid. Originally, these counts were stored in an object called a *candidate set*, which would grow very large over the construction of a sketch. Since the dataset of MinCE is static, we know beforehand which sketch values would be of use to us, since they are already stored in the hashmap files. The hashmap files are loaded into a khash hash table[12], where the keys are hash values and values are vectors of corresponding sketch IDs, both read from the plaintext file when running MinCE. By appending a zero at the front of each value vector, the count of each k-mer in our database can be logged within the hash table, eliminating the need to log every instance of k-mers in our .fastq file. In addition, by iterating over the .fastq files as a stream, we only need to keep one read in memory at a time and don't need to load the entire .fastq file into memory.

8 Results on simulated data

Testing the reliability of MinCE is difficult, owing to the fact that the software is meant to solve the problem of identifying species and strains in metagenomic datasets. If there were an easy way to double check the results of MinCE, then there would be little need for the software itself. However, Professor Pachter did supply *simulated* metagenomic datasets, generated from a predetermined set of 10, 100 and 400 genomes. The datasets and lists detailing their supposed contents were available online, so I downloaded datasets for each size of group, simulated as results from Illumina sequencers.[13]

10 genome simulated fastq file

Figure 9 shows the top results from mincing two fastq files, each 4.6Gb, generated from 10 different genomes. Of the top 11 results, 10 were present in the sample. Result 7, red as it should not appear in the sample, refers to *Methanococcus maripaludis* strain C8. Result 5 refers to *Methanococcus maripaludis* strain C7, green as it should appear in the sample. Neither genome has any distinguishing sequences in the MinCE-library, due to their set of k-mers being identical and thus no way to distinguish between them through deBruijn graphs.

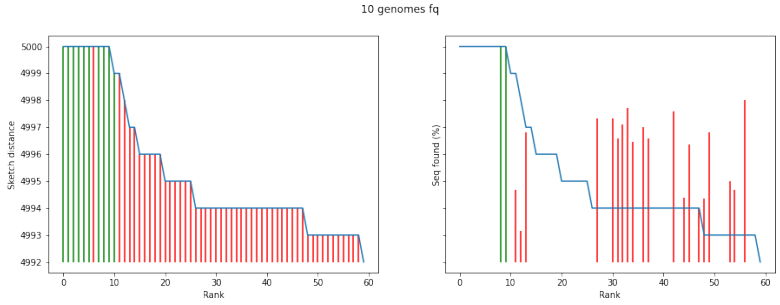


Figure 9: Green bars represent genomes present in sample, red results should not be present. **Left:** Blue line and bar height indicate number of shared sketch values. **Right:** Bar height indicates ratio of found distinguishing sequences, with range 0% at bottom to 100% at top, overlayed on blue line. 0% could also mean no sequences available in MinCE-library.

The 11th result should be present in the graph but only had 4999 of its MinCE representative's 5000 sketch values. The fastq files were minced with a threshold value of $c = 2$, indicating that each k-mer would need to be observed twice to be eligible for consideration.

[11] https://ohmyz.sh/
[12] https://github.com/attractivechaos/klib/blob/master/khash.h
[13] https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0031386

These results granted some optimism, as both the sketch values and the corresponding distinguishing sequences clearly delimit the true positives against the rest, aside from one false positive from a different strain of a species present in the sample.

100 genome simulated fastq file

The results of mincing the next largest set of genomes, containing 100 genomes, proved results. Figure 10 shows the results where the threshold value was set to $c = 2$. On the right side, we can see that the majority of genomes which should be identified within the sequenced data come in around places 13,000-22,000. Ahead of them we have more than 10,000 genomes supposedly not found within the sample. Additionally, the right graph shows we have multiple erroneous matches finding 100% of their distinguishing sequences. This representation does not reveal the number of those sequences, whether 20 as is preferred or only 1 or 2 in cases of cliques formed from near-identical genomes. In any case, this is case for concern.

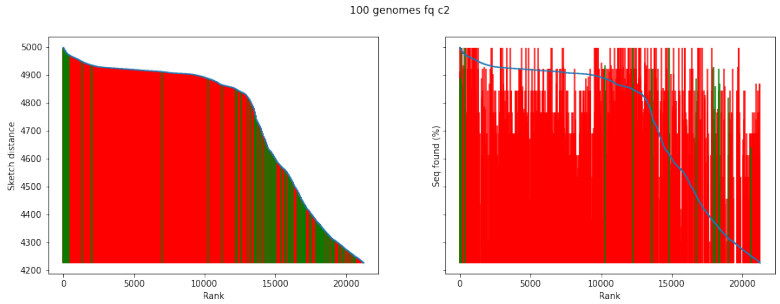


Figure 10: Results from mincing a simulated data from 100 genomes with threshold value $c = 2$.

The graphs in Figure 10 also omit much of the latter results for the sake of clarity. Among those omitted is a false negative, according to the information supplied by the authors of the simulated dataset. Figures 11 and 12 show MinCE's result on this worst match, a megaplasmid found in the genome *Ralstonia eutropha H16* - a heterotypic synonym of *Cupriavidus necator H16*, which is the name used in the MinCE library. This result was the 85,858th result in the total list of matches, with only 279 hashes shared with its MinCE-library representative.

Initially, this was thought to be caused by the threshold value c being too high. MinCE might have missed crucial k-mers found in this genome simply because their coverage was only 1 - i.e. the other 4721 sketch values associated with *Cupriavidus necator H16* only appeared once and thus were not considered reliable. This is of course very improbable, but the simulated dataset was run again with $c = 1$. *Ralstonia eutropha H16* still ranked 85,858 out of all matches, but this time with 316/5000 sketch value matches. Additionally, none of the distinguishing sequences were found in either run. The other genomes measured slightly better, with the majority of green results clustered between rank 10,000 and 14,000 having between 4800 and 4900 sketch values in common with the 5000 of the representative sketch.

There are several ways to explain these catastrophic results. Firstly, the genome in place 85,858 is that of a megaplasmid found in *Cupriavidus necator H16*, not its entire genome. NCBI houses sequence data on two additional chromosomes present in *C. necator H16*, which together with the plasmid comprise the .fasta file for *C. necator H16* used to build the sketch representative in MinCE's library. The megaplasmid only contributes around 6% of the total genome, which approximates the ratio of both 279/5000 and 316/5000.

This applies to some other members within the simulated dataset, as they are constructed from parts of the genome and not the whole. This does not suffice to explain the extremely low matches for all members of the dataset. Most red matches preceding the actual genomes present in the data have sketch values around 4900, which ordinarily would them exclude from further examination for being too low a match. Thus, the problem isn't false positives, but false negatives. This could possibly be explained by the fact that this data was generated in 2012, based on data available at that time. The number of sequenced genomes has increased drastically since

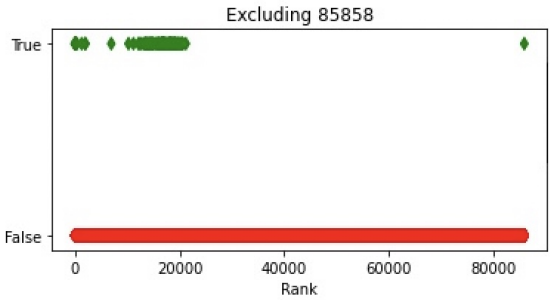


Figure 11: The distribution of genomes present in the simulated sequence-data (green) over the x-axis' ranking of all results. About 60,000 genomes not present in the sample (red) had a higher score than the last green results.

	NC_005241 Ralstonia eutropha H16
NCBI_id	GCF_000009285.1_ASM928v2
Sketch	279
Seq_found	0
Seq_total	10
In sample	True
Rank	85858
Ratio	0.0

Figure 12: The 85,858th result, a megaplasmid found in *Cupriavidus necator H16*, designated by its heterotypic synonym *Ralstonia eutropha H16* in the info file accompanying the simulated fastq data. Only 279/5000 sketch values were found and no distinguishing sequences were found.

then and so has the depth of each genome's sequencing. The data used to construct MinCE's library is based on the latest sequencing data available for each genome as of July 2021 and so might include large sections of genomic information not available to the researchers in 2012. If this were a real metagenomic sample from 2012, MinCE might have no trouble identifying these genomes as their *actual* genome has not changed in any real way since 2012.

400 genome simulated fastq file

Continuing the trend, the results from the simulated dataset of 400 genomes were not satisfactory. MinCE found indications of 399/400 genomes present, only missing *Candidatus Carsonella ruddii PV* which is not a member of MinCE's library. As the treshold $c = 2$ had been suspected of muddying the waters when mincing the 100 genome dataset, this was run with $c = 1$. Figure 13 shows these results, with the highest sketch match being 4837/5000 and most green matches situated between 3000 and 3500. It is important to note that each simulated dataset, regardless of size - 10 genomes, 100 genomes or 400 genomes - is stored in two 4.6Gb fastq files. Thus the 10 genome sample should have an estimated 10x coverage compared to the 100 genome sample and 40x relative coverage to the 400 genome one. Therefore, diminishing results are to be expected. With so many unknowns possibly contributing to the lacking nature of these results, some other measures of testing need to be taken.

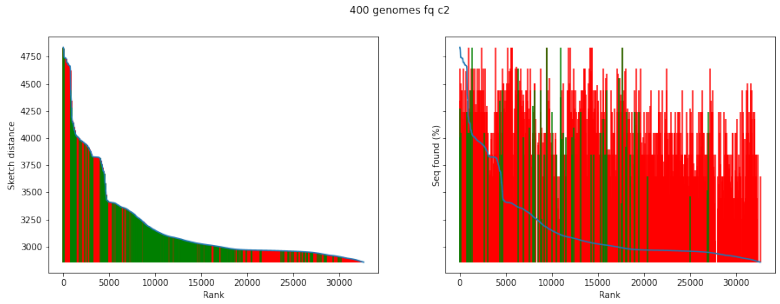


Figure 13: Results from mincing a simulated data from 400 genomes with threshold value $c = 1$. Most genomes supposedly present in the sample have between 3000 and 3500 matching sketch values out of 5000.

9 Verifying simulated results with kallisto

A roundabout way to verify the reliability of results from mincing this simulated data is to load the dataset into kallisto. Kallisto is a software tool developed by Professor Pachter and Professor Melsted, among others.[14]. MinCE was in part developed as a sieve for kallisto, to reduce the number of possible genomes in metagenomic samples before the latter is run, as kallisto is a very memory intensive software tool. If MinCE can be said to function on the level of species and strains, kallisto can be said to function of the level of each individual - even individual cells. Each cell contains the total genome of the organism but the specialization of each cell into liver cells, nerve cells or skin cells is achieved by control over which sections of the genome are translated into proteins and transcription factors. Different genes are translated in different cells and furthermore, the splicing of those genes can differ between cells. Splicing refers to the targeted removal of certain sections of the gene and subsequent recombination of those parts remaining. Such specialization can even be controlled by the state of the organism as a whole, whether a bacteria is in a acidic environment or an animal is sleeping. These different variants of genes, which together provide the malleability of an organisms genome, are called *transcripts*.

Kallisto is able to take a whole genome .fasta file and create an analog to a deBruijn graph, where this analog serves as a template for the possible transcripts for the organism. From this .fasta file, kallisto creates an *index* for the genome, onto which new reads can be aligned to map the sequenced file onto multiple different transcripts. For a metagenomic sample, such an index would need to be constructed from multiple whole genomes and this quickly becomes unfeasible if the number of possible genomes is not restricted in some way. Therefore, MinCE could be run on the sequenced data to narrow the scope of kallisto's index. This is yet another reason for false positives being preferable to false negatives.

When kallisto aligns the sequenced reads onto its index, it delivers an estimation of each genome's quantity in the sample - a measure of how many reads could be aligned to genes relating to any of the genomes provided for the index. Thus, by creating an index from the .fasta files in MinCE representing the 100 individuals purportedly present in the metagenomic sample simulated from 100 genomes, we can create a kallisto index to map the simulated reads onto. If kallisto determines that each genome's presence is high, we can conclude that MinCE is to blame for these unsatisfactory results. If, however, kallisto agrees with MinCE that these whole genomes are not adequately represented in the sample, we cannot place the blame purely on MinCE.

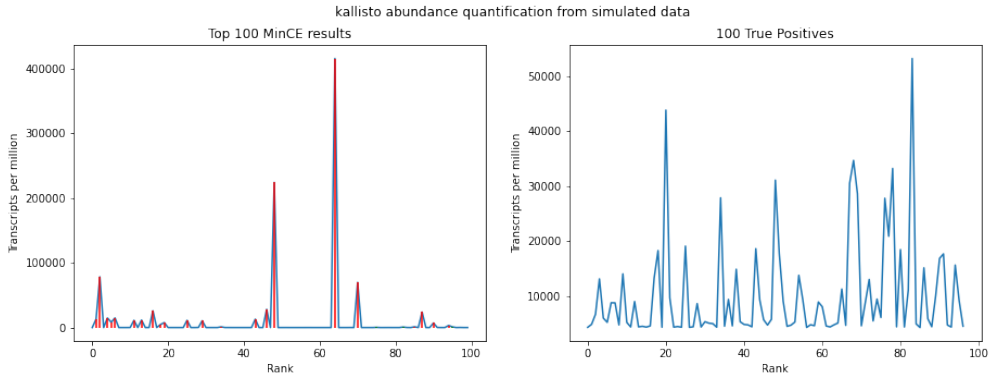
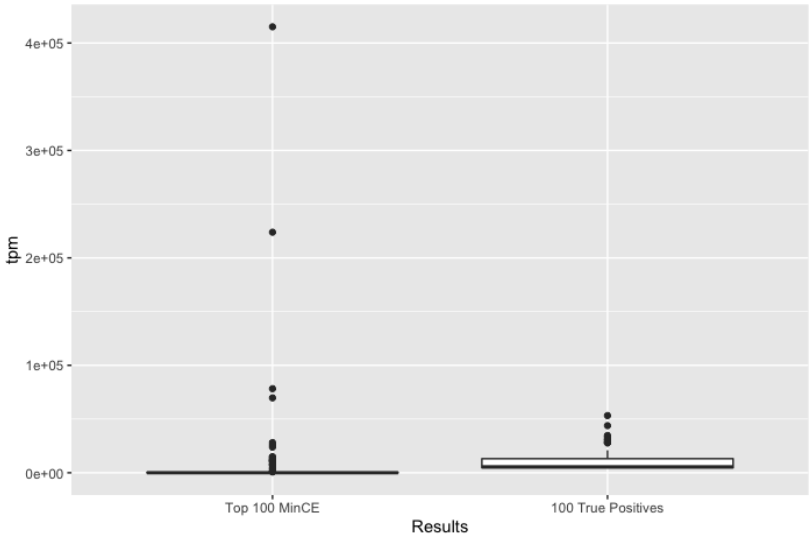


Figure 14: Results from aligning the simulated reads onto a kallisto index from the top 100 MinCE results (**right**) and from the true results in the simulated dataset's information file **(left)**. The difference in value span on y-axes is notable.

As a comparison, we can also load the index for the top 100 results from MinCE and query that index on the same simulated dataset. Figure 14 shows the results of querying the two kallisto indexes on the simulated reads. The y-axis shows the metric *transcripts per million* or *tpm*. This is a statistic where we correct for the varying

[14] https://github.com/pachterlab/kallisto

size of the genomes, so as not to overestimate the presence of larger genomes compared to smaller ones. The top 100 MinCE results show a maximum peak of $\sim 400,000$ tpm, whereas the true positives according to the information file accompanying the simulated datasets shows a peak result $\sim 50,000$ tpm. The different span of the y-axes is of note, as it shows that MinCE's estimation is in no way inferior to the purported *true* results of the simulated dataset.



Results	min	med	mean	max	sd
Top 100 MinCE	1.23000e-05	1.60679	10307.71	415086.70	48581.859
100 True Positives	4.34539e+03	5545.50000	10309.28	53203.59	9253.845

Figure 15: Results from mincing a simulated data from 400 genomes with threshold value $c = 1$. Most genomes supposedly present in the sample have between 3000 and 3500 matching sketch values out of 5000.

Figure 15 shows the boxplots for the results displayed graphically in Figure 14 and some general statistics for each one. The variance is considerably larger in the top 100 results from MinCE, with the lowest result being virtually negligible but the most significant one much more prominent than the top result from the information file. It is worth mentioning that kallisto divided the .fasta files from MinCE up into its constituent chromosomes when creating the index, treating each chromosome as an individual in a sense. After conferring with Delaney Sullivan, a student of Prof. Pachter, I felt confident that summing up the transcripts per million for each chromosome constituting the entire genome would not provide a biased estimate of the transcripts per million of the genome as a whole.

These results suggest that the simulated dataset might be one of questionable reliability. Small segments of the genome, such as the megaplasmid in *C. necator H16*, are unlikely to be found in isolation in true metagenomic samples. An overview of the details available in the information file seems to suggest that this applies to more organisms in the sample. Additionally, results from 10 years ago might be outdated or incomplete by today's standards. In any case, further testing is warranted on other datasets. True metagenomic samples could be run through MinCE and the results verified in the same manner with kallisto, though a true scan for any possible false negatives would indeed be time consuming, as every .fasta file in the MinCE library would need to be loaded into an index. Loading 100 genomes at a time would require more than 3000 passes, so the process would need to be automated.

10 Conclusions

The code for MinCE is currently available at https://github.com/mannaudur/MinCE. To compile the true program, one needs the .fasta files for the genomes comprising the MinCE library. A compiled version of each file needed to run MinCE in its current form is available upon request but presently not available online. MinCE shows promising results but the finer details of its reliability are still underdeveloped. The aspect most in need of improvement is the clustering step, which is not robust or rigid to pass scrutiny. Development is still ongoing and other areas of future research will involve faster ways to read in the `sketch.hashmap` file and ways to minimize the span of hash values for distinguishing sequences. Currently, the GSS algorithm chooses the first k-mer of each unitig, but for each vector-form, the algorithm has multiple unitigs to choose from. When a vector-form has been chosen in each step, we could scan each k-mer of each available unitig and choose the one that minimizes the resulting hash value. This might reduce the span of the resulting hash values, allowing us to implement a max hash value for the `seq.hashmap` file so we would no longer need to look every single k-mer up in the hash table. To minimize collisions with the sketch values in `sketch.hashmap` file, we might opt instead to use the *maximum* hash values available.

Once the software tool is ready, it can be used as a framework for other datasets of genomes. A viral MinCE library might be constructed to track the different strains of Covid or other viral pandemics in the future. Since MinCE should be able to identify the presence of these genomes in metagenomic samples, water samples from aquifers could be used to infer the presence and spread of varying strains throughout a community without the need to take swabs from people in person. With the current library, it could be used to analyze blood samples and the human gut microbiome, possibly leading to new avenues in diagnostic medicine.

I would like to extend the dearest of thanks and deepest gratitude to the following: Professor Lior Pachter for his help and insight this summer, Professor Páll Melsted for introducing me to the field of Bioinformatics, and his help and insight last summer, Jóhannes Nordal for his continuing cooperation in working on MinCE - both last summer and this past one, Delaney Sullivan for answering all of my poorly phrased questions on kallisto and C++ and Kristján Eldjárn for his hospitality during my stay at Caltech and for his endless capacity to formulate new ideas for MinCE from even more poorly phrased musings and half-thoughts on my part.

Thorhallur Audur Helgason

Reykjavík, Iceland

September 23rd 2022

11 References

1	"Insights from 20 years of bacterial genome sequencing"	https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4361730/
2	RefSeq database	https://www.ncbi.nlm.nih.gov/refseq/
3	GenBank database	https://www.ncbi.nlm.nih.gov/genbank/
4	Oxford Nanpore Technology	https://nanoporetech.com/
5	Alberts, Bruce (2017). *Molecular biology of the cell*, CRC Press	p. 29
6	Mash - Fast genome and metagenome distance estimation using MinHash	https://mash.readthedocs.io/en/latest/
7	Genome Taxonomy Database	https://gtdb.ecogenomic.org/
8	Annoy - (Approximate Nearest Neighbors Oh Yeah)	https://github.com/spotify/annoy/
9	Deterministic column subset selection for single-cell RNA-Seq	https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6347249/
10	Deterministic Column Subset Selection for single-cell RNA-Seq on GitHub	https://github.com/srmcc/dcss_single_cell
11	zsh command line tool	https://ohmyz.sh/
12	khash hash table on GitHub	https://github.com/attractivechaos/klib/blob/master/khash.h
13	Assessment of Metagenomic Assembly Using Simulated Next Generation Sequencing Data	https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0031386
14	kallisto on GitHub	https://github.com/pachterlab/kallisto