Interim Report #1 - 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**.

(colored) 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 colored to distinguish from which genome each node originates. Such a colored 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 $5000 - 4997 = 3$.

UF 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.

Background

The following is the first interim report for my work this summer at Caltech. The field of research is Bioinformatics / Computational Biology, and my work will be a continuation and expansion of a software tool I started to develop last summer, when working for Professor Páll Melsted at the University of Iceland, from whom the concept originates.[1] The working name of the method and software is MinCE.

MinCE

A genome is an immensely information-rich molecule but that information is completely coded with only 4 types of nucleotides; A,C,G,T for DNA and A,C,G,U for RNA. The human genome consists of $\sim$ 3 billion base pairs of DNA, and due to a bottleneck in the human population approximately 10,000 years ago, the genomes of every pair of individuals are highly similar. Genomes of other organisms vary in length and divergence in genome sequence generally correlates with divergence in phenotype. This presents various problems (and some opportunities), from which stems the field of Bioinformatics. Two such problems are comparison between genomes and identifying the species and/or variant of an unknown individual solely from sequencing data. Both problems have to deal with the issue that no single, whole genome can be said to represent the species as a whole, since variation within species is always present. As a result, a direct comparison of 'letter-by-letter' between genomes is not only extremely slow, but in fact doomed to fail.
Therefore another approach is required. The idea of representing genomes as sketches was first proposed in Mash, a bioinformatics software tool which MinCE builds upon.[2] By creating a sketch of every genome in a given database, raw sequencing data of unknown organisms can be processed into a sketch and paired to its closest sketch match within the database.
This approach proves insufficient when comparing very related genomes, as the sketches will prove identical for multiple species or variants within species. MinCE proposes a solution to this, by sketching a large database of whole genomes available through NCBI[3] and subsequently cataloging the specific cliques of nearly identical sketches which result from this process. We create a colored deBruijn graph from the members of each clique and algorithmically select k-mers from that deBruijn graph that can distinguish between every member in the clique, with room for some redundancy. By searching for these selected k-mers in given sequencing data, we can extend the accuracy of the sketches. A MinCE dataset contains one large file which maps hash values to its respective sketches `hash_locator`, an index file mapping genome IDs in the MinCE dataset to NCBI IDs `indices` and then one file per each clique, describing which selected k-mers belong to which genome.
The utilization potential of this program is enormous. Raw sequencing data can be piped directly into MinCE, without the need for any aligning of reads or extra processing, and the result will in theory detail every species and/or strain from the database found therein. MinCE will therefore allow for quantification of large metagenomics datasets along with species and strain abundance, a task that is currently extremely challenging. For the food industry, public health sectors and medical diagnoses, this has the potential to radically expedite processes which until now have been very time consuming.

The work achieved last summer

Prior to my work here as a Caltech SURF student, I had made a working prototype of MinCE, which was written in C++ and Python. The functions for sketching individual genomes and creating UF clusters were written in C++, while the remaining functions needed for the basic creation of a MinCE dataset were written in shell script. The main function of MinCE, as well as the algorithms to create colored deBruijn graphs from cliques and subsequently choose distinguishing k-mers from those graphs and the algorithm iterating over that selection of k-mers when users provide new sequencing data to the software, were all written in Python. I had also compiled a MinCE dataset from the 258,339 genomes in the Genome Taxonomy Database with a sketch size of $s = 1000$. The entire set of data needed to run MinCE on that dataset amounted to roughly 3 Gb.

[1] https://github.com/mannaudur/MinCE
[2] https://github.com/marbl/Mash
[3] https://www.ncbi.nlm.nih.gov/

My aim for this summer

When work on MinCE had finished last summer, it was clear that a better approach was needed for the clustering of sketches and a more rigorous approach to the selection of distinguishing k-mers was also preferable. My undertaking this summer starts with refining every area of the prototype of MinCE written a year ago. This includes a complete redesign of the algorithm which chooses distinguishing k-mers from genomes in cliques, the algorithm iterating over those k-mers when the program is run, and the algorithm responsible for creating cliques from a given dataset. This also includes writing the whole thing in C++, i.e. translating those steps written in Python or shell script into C++ and writing every addition to the software in C++.

What follows is building a connection from MinCE to kallisto, a program for quantifying abundances of transcripts from bulk and single-cell RNA-Seq data, written in part by both my Caltech mentor Lior Pachter and Páll Melsted, my mentor last summer when starting work on MinCE. [4] If the resolution of Mash can be said to be roughly on the species level and the resolution of MinCE can be said to extend down to the strain level, then kallisto's resolution would be on the level within the individual. Within each cell, the entire genome of the organism is present. The specialization of cells into liver cells, skin cells etc. stems from different transcripts from the whole genome being produced in different cells. The entire set of every transcript produced within a members of a species is called the species' transcriptome. By using sequencing data from multiple cells of individuals of the same species, kallisto creates an index which maps transcripts to equivalence classes, which are a collection of cell types known to produce said transcripts. Therefore, cell types can be inferred from sequenced data. Moreover, this can give an indication of the processes underway in the organism at the time of sequencing. However, the size of each species' transcriptome makes this endeavour quite memory-intensive. For a metagenomic sample, indexing the transcriptome of every prospective species therein would quickly overload the memory capacity of a standard laptop computer, diminishing the use-case potential of the software.

When kallisto was introduced in 2017,[5] the authors noted that "[as] the number of highly-similar strains increases, the need for new algorithms that can distinguish strains on the basis of a few differentiating reads becomes more pressing." The concept of MinCE comes from Professor Páll Melsted, one of the authors of the quoted paper, and it is intended to meet this need. My main objective this summer is to implement MinCE as a sieve for kallisto, such that a metagenomic sample can be analyzed by MinCE to reduce the number of possible species within the sample. The results of MinCE would then be piped onward into kallisto, which would load the respective index. The nature of the connection implies that false positives from MinCE are acceptable but false negatives are a serious drawback.
The first step in building this connection will be to fully flesh out MinCE's feature to make so-called **mega-sketches** from metagenomic samples and finesse the algorithm which finds multiple matches within the MinCE data set. A normal sketch has a fixed size *s* and is produced by reading a pre-aligned genome representing a certain species, processing every k-mer into a hash value and picking the lowest *s* values as our sketch. For a metagenomic sample, such a fixed size sketch would be useless because we might be choosing the lowest hash values produced by hashing k-mers from multiple different species. Thus, we create a mega-sketch instead, a sketch of unfixed size that contains every hash value lower than the max hash value in our database. In effect, we place every hash value *possibly* in our database into the mega-sketch. This functionality was present in the prototype written last summer but it has yet to be tested. When the function is fully realized, I will run sequenced data from authentic metagenomic samples on MinCE to analyze this feature further and subsequently finish work on piping the results from MinCE to kallisto in a smooth manner.

[4] https://pachterlab.github.io/kallisto/about

[5] Schaeffer L, Pimentel H, Bray N, Melsted P, Pachter L. Pseudoalignment for metagenomic read assignment. Bioinformatics. 2017 Jul 15;33(14):2082-2088. doi: 10.1093/bioinformatics/btx106. PMID: 28334086; PMCID: PMC5870846.

The work so far this summer

The first few days of work were spent getting oriented within the scope of the project; understanding the use-case of kallisto, the rough outline of the field of metagenomics and clarifying the problems I was meant to solve. By the end of June 16th I had translated the main function of MinCE into C++ and rewritten the function called triangulate, which compares the sketch from an input file with every other sketch in the database and so returns its nearest match. For a database with sketch size $s = 1000$, the total runtime for triangulation was 72 seconds, thereof 71 seconds for loading the `hash_locator` file used for quick comparison. I assume the final version of the software will be implemented on a server, where this load time would not factor into each run, so the more important time is the time spent on everything else; around 1 second.

On June 17th I met with Lior and we discussed a few tricky areas of the software yet to be implemented properly, in particular an appropriate clustering algorithm for our purposes in MinCE and a more robust, rigorous approach to selecting distinguishing k-mers within cliques. Lior recommended I look into the k-nearest neighbours implementation of annoy[6] and to look into selecting k-mers with a variation on deterministic column subset selection, which he and his students had written a paper on.[7]

Clustering of sketches

So why do we need an algorithm to cluster the sketches? The reason is central to how MinCE improves upon Mash; its ability to discern between members of closely related species or different strains within species. We want to catalogue how similar the sketches in our dataset are and pin-point the clusters of sketches that are either identical or *where the pairwise sketch distance is lower than some threshold of reliable results.* This threshold represents our error margin for the reliability of sketch comparison and incorporates a few unknowns. The sketch provided in our database is made by hashing a single .fasta file, which is the genome of a single individual. We have to account for variability within the species as well as sequencing errors. Variability within species may find its way into one of our smallest hash values and subsequently change the sketch, and so might sequencing errors, though appropriate steps are taken to minimize that likelihood. Now, to be clear, we don't know what this threshold value should be. That is something to be refined with empirical testing and could turn out to be different for different species. At present, this threshold value is 5, which is likely way too large.

The function of this clustering would be to sort these sketches into separate groups, which would be treated with an algorithm to select k-mers distinguishing between every member of the group. The union-find clustering implemented last summer creates huge clusters of sketches, which is a useful first step but if we want to divide them up further while still respecting our threshold value, it would seem we would need overlapping cliques. The challenge is to find a way to do this that **a)** doesn't result in huge cliques, **b)** doesn't result in constant overlap, making the number of cliques rise exponentially and **c)** doesn't lump sketches of distantly related genomes in with sketches of closely related genomes. This last point becomes clear when you think about the process of choosing distinguishing k-mers. The more closely related members of a clique are, the more specificity we get from the resulting k-mers. An exaggerated example is to pick 10 k-mers which distinguish between a human and a fruitfly - it's not that difficult and doesn't tell us very much. A closely packed group of different strains within the same species would thus yield a fine distinction of the variability within the species. If we were to throw several members of distantly related species in with the group, we would need to expand our set of k-mers to get the same distinction and optimally we would want to keep that number fairly low.

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

[7] McCurdy SR, Ntranos V, Pachter L. Deterministic column subset selection for single-cell RNA-Seq. PLoS One. 2019 Jan 25;14(1):e0210571. doi: 10.1371/journal.pone.0210571. PMID: 30682053; PMCID: PMC6347249.

Multi-dimensional scaling

I looked into annoy and - true to its name - discovered that it could not accept a simple distance matrix as its input, which is a problem for this case. The sketches are in essence bit vectors in a 4^k-dimensional space, where $k = 31$ is our k-mer value for the sketches. This stems from the fact that their distance is calculated as the number of hash values they don't share, so we can in theory think of them as having value 1 in the dimension corresponding to hash values that they take, otherwise 0, and in theory the hash values have a span of 4^k. Of course, we don't have all those hash values in our dataset, so we could sensibly reduce this to $|M|$-dimensional space, where $|M|$ is the number of different hash values in our dataset. For the dataset with sketch size $s = 1000$, that number M is just south of 53 million. This approach is of course ludicrous, but it highlights the intricacies of the distances between sketches. As I could neither feed the annoy algorithm a simple distance matrix detailing the pairwise distances between sketches nor seemingly implement my own distance metric function within the source code, Lior advised I look into multidimensional scaling as a way of approximating the relationship of the sketches to a lower dimensional space.

I tried this approach out for some time with mixed results. I obviously have an initial clustering for the entire dataset in the UF clusters, so I could work from there. The smaller UF clusters yielded good results with MDS, but those weren't the problem to begin with. In the case of the larger UF clusters, the calculated strain of those MDS approximations turned out to be on the scale of several hundreds. This was all based on the original dataset that I had created last summer with sketch size $s = 1000$, so I decided to update my dataset with a sketch size of $s = 5000$ this time. Parallel to this, I rewrote the code to compile a new MinCE dataset to only use C++ and joined any loose ends into one single command. I compiled my new dataset through a server in Iceland called Mimir and decided to test out some variability in the threshold value for joining sketches into UF clusters. With a dataset of sketch size $s = 5000$, I didn't really have a good criteria for my new threshold value for UF cluster formation and therefore I generated UF clustering with threshold values 15, 10 and 5. For the smallest value of 5, the resulting clusters were still fairly large (the largest containing 486 sketches), so I considered that an indication that this was an appropriate starting value to work from. For this new sketch size of $s = 5000$, the `hash_locator` file grew from its previous size of 2.5Gb (for $s = 1000$) to 13Gb, a considerable increase in size. The process of loading this file, previously around 70 seconds, now took around 10 minutes.

Mininum cut approach

On this new MinCE dataset I ran the aforementioned analysis for MDS again, with similar results - the resulting strain from MDS indicated that the largest clusters were too large to approximate to a manageable number of dimensions. I then turned to other approaches to further subdivide these UF clusters and targeted the runaway problem of the union-find approach, which would make larger clusters even larger because the criteria for merging into a cluster is only based on being a sketch distance less than 5 away from any member. We can frame this as a graph, where nodes represent sketches and there is an edge between sketches if the sketch distance is less than or equal to 5. I could imagine this resulting in several nodes within the graph of degree 2 linking together, forming a weak chain between more tightly connected sections of the graph. Thus this tangle of sketches could be partitioned into smaller graphs with only a few cuts. I used a minimum-cut algorithm from NetworkX[8] to implement this and defined the source and sink inputs to be the nodes with highest values in a distance matrix generated from the cluster's pairwise sketch distance. I implemented a recursive algorithm to subdivide clusters until a desired size was reached or iterations failed to generate new results. This recursion had a caveat that is important to note: for each subdivision from cluster A into subclusters B and C, it is necessary to look at the edges which were cut. Every node in B connected to a cut edge would need to be added to cluster C and similarly every node in C connected to a cut edge would need to be added to cluster B. This is necessary to retain the connections between sketches; if the sketch distance between two sketches is below a certain threshold, they should be grouped into the same cluster. This effectively makes the clusters overlapping. Therefore, every iteration included one step forward and an uncertain fraction of 1 step backwards. The results were very disappointing. These UF clusters turned out to be very densely connected, a result of the myriad of options for any

[8] https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.flow.minimum_cut.html

pair of sketches to have for example a sketch distance of 1 (not to mention 2, 3 and so on), so the minimum-cut approach either simply resulted in the algorithm cutting out either the source or the sink or halting in the first steps because the caveat step after the cuts resulted in one or both of the generated subcluster being the same as the original cluster. I tried lowering the threshold to 3 or even 2, but the problem persisted.

Increasing sketch sizes

Furthermore, it became apparent that the sketch size of $s = 5000$ still produced a considerable amount of identical sketches, with one cluster of size 99 containing 10 separate clusters of identical sketches, the largest of which counted 15 and 12 identical sketches. I entertained the idea of increasing the size of sketches to $s = 10000$, but that would again increase the size and loading time of the `hash_locator` file. Additionally, this felt somehow wrong, because the strength of the sketches is their small size and their weakness is the accuracy of their representation. The clusters are meant to carry the role of this accurate representation and thus increasing the size of sketches to the detriment of their strengths to move the responsibilities of the clusters onto the sketches felt counter to the whole concept. I therefore tabled this idea for later review and set to work finding a better way to define these clusters.

Bottoms up approach

I drew the conclusion that I would need to rethink the whole approach of clustering, since these UF clusters proved so densely connected. Simulating UF clustering with threshold 3 did result in slightly smaller cluster sizes, but the larger ones were still a ways off from being manageable for the k-mer selection algorithm. I brainstormed for quite a while on new solutions to the clustering problem and realized that I had subconsciously connected the threshold for cluster formation to the threshold of viable results when a user runs the program. This is not a necessary interdependence. If the size of clusters can vary, then I can separate dense sketch clusters of closely related genomes from surrounding sketches and look at clustering the surrounding sketches later on, irrespective of the dense clusters. At this point, I will start to talk about these subdivisions of UF clusters as *X-cliques*, where X refers to the size of threshold used to form the cliques, or in general just *cliques*.

The bottoms up approach for clique formation is an iterative process that works in the following way for each cluster:
1) Translate all sketches into a distance matrix of pairwise sketch distances, D, and copy D_o. Set $v = 0$ and set *iter* value to desired threshold. Create list of cliques, C.
2a) For each column D_i: add all j satisfying $D_i[j] = v$ to set $S_{i,v}$.
2b) For each set s in S: if $s \subseteq r$ or $r \subseteq s$, $r \in S$, then merge s and r.
2c) For each set s in S: add s to C.
2d) Update D to be a submatrix of D, such that it only includes members not allocated to cliques in step **2c)**
2e) $v = v + 1$
2f) If $v < iter + 1$: go back to **2a)**
3) For all members not allocated to cliques, add to clique of nearest neighbour in D_o.

This method may seem crude, but it has some beneficial aspects that are particular to our use case. The *iter* value is the threshold for how large of a clique we want to generate through the algorithm until we reach step 3. We start by grouping together all of the identical sketches into respective 0-cliques and consider them fully clustered and not to be included in any other cliques. We then find the clusters of remining sketches that have a distance of 1 between them. We merge subsets into larger sets, if those cases appear, and then log those sets as 1-cliques and remove their sketches from further consideration. We iterate like this until we reach our *iter* value, at which time we look at our remaining members and lump them in with their nearest clique.

Now, this creates cliques of varying sizes and some chance of sketches not being grouped into a clique with their closest neighboring sketch. To illustrate such an incident, imagine a sketch s_i. It has sketch distance 1 from a cluster of identical sketches, which we shall call C_a, and distance 2 from sketch, s_j, and distance 3 from sketch s_k. Now through the steps of the algorithm above, we would create a 0-clique, $c_{0,0}$ (first subscript indicates size, second is ID of clique), from the members of C_a and then remove them from further consideration. With

$v = v + 1$ in step **2e)** we now look at a sketch distance of 1. We don't join s_i in with $c_{0,0}$, because they're off the table, but for s_j we find some matches, so we form the clique $c_{1,0}$ with s_j as a member and so it is removed from further consideration. When reaching distance 2 we thus don't do anything with s_i, because it's not at distance 2 from anyone open to consideration. We finally merge it with s_k through the step looking at sketch distance 3, creating the 3-clique $c_{3,0}$. If s_k had not been present, s_i would have been lumped in with $c_{0,0}$ in step 3.

Pros and cons to bottoms up approach

So, the sketch s_i was put into a clique with a sketch with 3 differing hash values, despite many closer matches being available. Why is this okay? Well, it honestly isn't but this is a work in progress. Furthermore, there are positive aspects. Firstly, a general inspection of the clusters suggests they are very tight, so this case is probably not common but that is something I will have to inspect further if I stick by this approach. Secondly, this prioritizes the joining of smaller-threshold cliques. For a metagenomic sample translated into a mega-sketch, the chance of a kind of "cross-contamination" - i.e. getting false positives in wrong sketches because the mega-sketch has so many more hash values than a normal sketch - is something to think about. This approach keeps cliques as small as possible, allowing for more redundancy within clique files which reduces risk of cross-contamination, while still covering the entire cluster in cliques. Thirdly, this pretty much eliminates the possibility of a bunch of distantly related sketches being grouped in with closely inter-related groups. That would put strain on the specificity of that clique's limited number of chosen k-mers, but this bottoms up approach prioritizes separation of densely packed cliques from ones more spread out. I say 'pretty much eliminates', because stage 3 adds leftover sketches to their nearest clique which could well be a 0-clique at distance 4. However, this last step is needed to ensure that we don't wind up isolating sketches and losing them from this k-mer comparison, just because of the order of the algorithm's steps.

When an input file is sketched into a mega-sketch, we allow our triangulate function to locate multiple good matches. That is, we look at every sketch with distance less than some threshold t from our input sketch and fetch their corresponding clique files, if those sketches belong to cliques. Thus we aren't looking at any centers of clusters, but rather just looking at matches in sketches. We grab files from all cliques in a radius from those matches and should therefore be examining all of the possible candidates in one clique or another. By isolating those clusters with identical sketches from the rest, we preserve the most sensitive distinction between closely related genomes, but still map all sketches in our UF clusters to cliques. We also keep the clusters relatively small in size and compact, so the threshold t can be tuned by the user down to very small values without ever going beneath some 'hard-coded' resolution for the cluster size.

The drawback to this algorithm is the scenario described above concerning sketch s_i. Let's take a better look at what happens when a user sends in an input, which should find a match in the species and strain corresponding to sketch s_i. It will find a good match to the sketch s_i and some surrounding sketches, in particular members of $c_{0,0}$. Then MinCE would fetch the clique file for $c_{3,0}$, search for the associated k-mers in that file and almost definitely get a pretty much perfect score for s_i - after all, we're comparing it to a genome that's pretty distantly related. We don't mind that the members of this clique are distantly related, because it's formed late in the process and it's not going to disturb the accuracy of discerning between some identical sketches. Now, we might get a false positive within clique $c_{0,0}$, because we've missed out on some k-mers distinguishing s_i from members of that clique, but a false positive is preferable to a false negative.

When running the UF clustering with threshold value 5 on the MinCE dataset with sketch size $s = 5000$, we get 9724 clusters, the largest of which contains 486 sketches. Running a Python version of this bottoms up approach on those 9724 clusters subdivided them into 13926 cliques, the largest of which contained 63 sketches that were identical.

At the moment, this is the best implementation I have. There are other aspects which add to its case and further explain why this awkward approach is easier than some traditional clustering algorithms, but I feel I've already delved way too deep into the details of this for normal sensibilities. This description will however help me write my final report and in particular make it easier to revisit this problem at a later date, if needs be.

Selecting distinguishing k-mers

Finally, I've also started work on developing the algorithm to select k-mers from the colored deBruijn graph produced from members of cliques. The code to produce these graphs and translate them into bit matrices was originally written in Python and I've finished writing it fully in C++, importing the necessary functions and structures from the bifrost[9] library. This code should be able to get a path to all the cliques generated by the algorithm above and churn out bitmatrices describing the genomic variation within the clique. I'm currently in the process of getting acquainted with the DCSS algorithm Lior sent me and plan to use this current week working out how to implement it to suit our needs.

What's ahead

For the time being I will proceed as if this bottoms up approach works, but I might look at compiling the dataset again with a lower threshold of 3 for UF clustering. Given how little variability seems to translate into these sketches, I would think that the likelihood of two sketches from members of the same species differing by 3 is very low. Finally, it is worth the look to compile the same database again using a sketch size of $s = 10,000$. It is still my hope that such a solution will not be necessary.

If the DCSS implementation goes well, I would next get to work on the algorithm which searches for these selected k-mers in an input file from the user. As the target input file is a sequenced metagenomic sample, the algorithm to search for k-mers should be structured in such a way as to be able to work modularly with each clique file it receives. The algorithm should begin by gathering all the files needed and then not search linearly through every file independently, but rather scale in some better fashion with a growing number of files.

When that is ready, I can add a few tweaks to the main function of MinCE. As of now the UI is designed to output results based on an input file containing a single genome but I would like it to be designed towards making multiple, disjointed matches read more legibly. Then it's time to run some real life metagenomic samples on the software, which is incredibly exciting.

Thorhallur Audur Helgason

Pasadena, California, USA

July 5th 2022

[9] https://github.com/pmelsted/bifrost