PAGS: A Probabilistic Approach to Genome Similarity

Varun Radhakrishnan

Junhao Xiong

Ziang Song

Abstract

Researchers are often interested in determining the similarity between two genomes. However, this is often a costly and time consuming process. Our project aims to provide a fast and relatively accurate measurement for genome similarity through probabilistic means. Building upon prior probabilistic approaches, we extended the “sketch-distance” approach which shows promises in methods such as MASH and created PAGS - a Probabilistic Approach to Genome Similarity. Our main contributions are in: (1) creating sketches that are more representative of the original genomes (2) devising more accurate distance calculation (3) experimenting with new sampling techniques e.g. subsequences. To show the validity of our result, we calculated distances among archaea genomes and showed a correlation between the PAGS distance and ANI, which is a common measurement of genome similarity.

Introduction

Determining relationship among genomes constitutes an important part of computational genomics. A natural question that one might first encounter in this endeavor might be “How similar are they?” As simple as the question may sound, it has wide ranging applications in whole genome clustering, building phylogenetic tree to study evolution, studying relationships between specific genomes and many other important areas of downstream genomics research. The demand for efficient and accurate genome similarity measurement becomes even more urgent as a growing number of whole genome data are becoming available for analysis, due to the spread of second generation sequencing technology. This need motivates the study of this paper.

There are currently two main types of method in computational genomics to measure genome similarity - the alignment-based approach and the alignment-free approach. The alignment-based methods utilize information from sequence alignment to make judgement on genome similarity. Popular techniques such as BLAST uses local alignment determine similarity. The similarity measurements of alignment-based approaches are usually fairly accurate, largely due to the fact that they use information from most parts of the genomes. As an example, local alignment looks at every pair of substrings to determine the most optimal global alignment value. However, this accurate result also comes with a large time and space overhead. As an example, to perform local alignment on genomes, a large dynamic programming problem needs to be solved. Other methods rely on large data structures. MUMmer, for instance, MUMmer takes O(size of both genomes) time and space to create a generalized suffix tree, with genome sizes potentially being millions or billions base pairs long.

To prevent accruing such large time and space overhead, alignment-free methods arise specifically to address this issue. In a nutshell, most alignment-free methods only keep essential information of the genomes. This makes alignment-free methods more efficient in practice, especially when analyzing a large number of genomes. However, since alignment-free methods “throw away” a significant portion of genomic information, they usually produce less accurate results compared to the alignment-based ones.

Considering the tradeoff between accuracy and efficiency, we look to devise an upgraded alignment-free method to improve the accuracy of similarity measurement, while still maintaining most of the advantages in efficiency. In particular, we choose to focus on a probabilistic approach to measure genome similarity. Our work was inspired by MASH, which creates “sketches” of genomes by sampling k-mers and calculates “distance” between the genome by building a probabilistic model using the sketches.

Our program, PAGS, builds on the MASH algorithm with some improvements on the sampling process and distance calculation. Regarding sampling, PAGS allows repetition when sampling k-mers, a feature not included in MASH because of the use of MinHash, to take into the account of genomes’ repetitive nature and build more representative sketches. As regards to distance, calculation, PAGS utilizes Poisson random variables as well as the sketches to approximate the point mutation rate, using it to approximate the distance between two genomes. Moreover, PAGS allows users to extract k-mers from totally consecutive substrings to subsequences gapped with spaces in order to improve specificity in k-mer sampling.

Prior Work

MASH

The probabilistically-based method PAGS relies on is based on MASH, which creates a sketch of each genome to estimate the jaccard index, a calculation used to compare the similarity of two sets. In order to create the sketch, MASH relies on the MinHash procedure, which stores the k-mers of a specific genome with the bottom n hash values. Note that since MASH takes only the bottom n hash values, it does not allow repetitive k-mers in the sketch, because otherwise the sketch would be filled with k-mers that are largely the same. These sketches are compared to estimate the MASH Distance, a measure of the point mutation rate among two genomes.

The MASH Distance relies on the Poisson random variable, a distribution used to model rare events like point mutations. By estimating the jaccard index, MASH calculates the probability that a k-mer in one genome will be found in the other with zero mutations. By assuming that a match implies zero mutations, it can easily find the point mutation rate. It was also shown that MASH Distance has a clear correlation with ANI, proving that it can successfully approximate the similarity between two genomes.

KMACS

KMACS uses suffix arrays to measure genome similarity by calculating the average length of the longest common words between the two genomes. Rather than finding the exact values, KMACS relies on a greedy algorithm to probabilistically provide sufficiently close words.

Methods and Software

Methods

Similar to MASH, we wish to approximate the point mutation rate to calculate distance between genomes. We assume that the probability of a point mutation is rare. Consequently, we can use a Poisson random variable to model the number of mutations in a given kmer.

Let $k =$ kmer length, $d =$ point mutation rate, *G1, G2* represent two genomes respectively; *s1, s2* represent the sizes of the corresponding sketches; w = the number of common k-mers found in both sketches;.

then $e^{-kd} = P(kmer\ has\ 0\ mutations)$ by PDF of Poisson random variable.

If we a draw a random k-mer from *G1* to be in the first sketch, the probability that it has 0 mutations and its counterpart in *G2* can be found in the other sketch can be given by: $\frac{2w}{s1 + s2}$.

Then we make the following assumption:

Assumption 1

If a given k-mer in a sketch has an exact match in the other sketch, the match corresponds to its ideal alignment in the optimal global alignment between the two genomes.

An important caveat for this assumption is that a sufficiently large k is chosen so that any given k-mer has a high specificity, meaning that it is likely to occur randomly in the genome. Consequently, if there is a match between k-mers in both sketches, we can claim with large likelihood that it represents an optimal alignment of the two genomes.

Now if we let I_{match} be an indicator random variable for whether a k-mer in the sketch of *G1* has a match in the sketch of *G2*. This means that $I_{match} = 1$ when the k-mer has a match in sketch two. Therefore, $E[I_{match}] = P(kmer\ is\ in\ sketch\ 2)$. This gives us the desired probability of a given k-mer in *G1* being in the sketch of *G2*.

One caution is that we must take more than one value to get an accurate measure of I_{match} since it has an extremely high variance:

$Var(I_{match}) = P(kmer\ is\ in\ sketch\ 2) - P(kmer\ is\ in\ sketch\ 2)^2$

If we take n trials, the variance drops to $\frac{Var(I_{match})}{n}$. Hence, a higher sketch size would lead to more accurate results.

The average of the samples of I_{match} for *G1* equals $\frac{w}{s1}$. By the same procedure, the average of the samples for *G2* equals $\frac{w}{s2}$. The cumulative average between the two samplings would be $\frac{2w}{s1+s2}$, since we had *2w* matches from *s1 + s2* k-mers.

Now that we have calculated $P(kmer\ is\ in\ sketch\ 2)$. We can attempt to compute the $P(kmer\ has\ 0\ mutations)$. If we invoke *Assumption 1*, we can assume that the match in the other sketch is the ideal alignment of the k-mer.

Let "k.s.g." represent "k-mer in sketch of the other genome"
If we partition the sample size,

$P(k.s.g.) = P(k.s.g.,\ kmer\ has\ 0\ mutations) + P(k.s.g.,\ kmer\ is\ mutated)$

$P(k.s.g.,\ kmer\ is\ mutated) = 0$, since we are assuming a match between two k-mers implies that they are aligned together.

Consequently,

$P(k.s.g.) = P(k.s.g.,\ kmer\ has\ 0\ mutations)$

$P(k.s.g.) = P(kmer\ has\ 0\ mutations) * P(k.s.g. \mid kmer\ has\ 0\ mutations)$

Since we are assuming a match between two k-mers implies that the two are aligned together, we have:

$P(k.s.g. \mid kmer\ has\ 0\ mutations) = P(kmer\ is\ selected)$, since each k-mer is unique by its position.

We allow the user to specify the portion of the genome to be included into the sketch as *p*.

Therefore,

$P(k.s.g. \mid kmer\ has\ 0\ mutations) = p$

$P(k.s.g) = P(kmer\ has\ 0\ mutations) * p$

$P(kmer\ has\ 0\ mutations) = \frac{P(k.s.g)}{p}$

If we plug in the appropriate functions for the probabilities,

$$e^{-kd} = \frac{\frac{2w}{s1 + s2}}{p}$$

Solving for d:

$$d = \frac{-ln(\frac{\frac{2w}{s1 + s2}}{p})}{k}$$

We now have a measure for the point mutation rate, d, which we use to approximate the distance between two genomes.

<u>Software:</u>

The program needs to solve three problems: gathering k-mers, storing k-mers, and calculating the distance. We have elaborated the distance calculation above, so we shall now introduce the way we gather and store k-mers. After explaining the basic procedures, we will also illustrate several other features of PAGS, namely taking subsequences as k-mers, using bloom filters for storage and allowing user the flexibility to choose from several hash functions for k-mer storage.

<u>Gathering K-mers:</u>

The first task is to gather k-mers that would be stored in sketch for each genome. Unlike Mash, we wanted to ensure that the size of both sketches would be indicative of the size of the corresponding genome. Consequently, we allowed the users to specify a parameter *p*.

Then, we would cycle through every k-mer of a genome and generate a random number between 0 and 1. If the number is less than *p*, we include the kmer in the sketch. To allow more flexibility, we enable users to additionally dictate the length of the gaps spaces- for extracting subsequences.

To show that the sketch sizes resulting from random sampling are relatively stable, we can treat each k-mer as a bernoulli random trial where a success would be its inclusion in the sketch and a failure its absence. Consequently, the number of successes would represent the number of k-mers included in the sketch.

Let N be the number of kmers in the sketch. We can model N through a binomial distribution, since each trial is independent and has a fixed probability *p*. A given genome with size G will have nearly G kmers in total; in other words, there will be G bernoulli trials. Therefore, $E[N] = p * G$. However, we can also approximate the binomial distribution with the normal.

The $Var(N) = G * p * (1 - p)$. So, 95 percent of the sketches will fall within the following range:

$G * p \pm 2 * \sqrt{n * p * (1 - p)}$

It is worth noting that while it is possible that the entire genome will be included in a given sketch, it is highly unlikely. However, if the genome is large, the variance would increase and a larger range of sizes could occur.

Storing K-mers:

The next problem that needed to be solved is the implementation of the data structure to store the k-mer information. This class would be called *sketch* as it would serve as a representation of the genomes. In order to save space, we decided to simply use one class that would store the sketch information for both genomes. The first genome would add elements to the data structure in sketch, while the second would remove elements and update counts. This method would allow us to calculate the desired probability with less memory.

We primarily created two sub-classes that would implement the desired properties of this sketch abstract class: *Simple Sketch* and *Hash Sketch*.

Simple Sketch

Simple Sketch stores each k-mer with its number of occurrences in a python dictionary. Because the size of each k-mer is k, each entry in the hash map takes O(k) space. If we let p be the probability a kmer is included in the sketch and g the size of the genome, the entire dictionary should be $O(gpk)$ size. This implementation provides all the necessary information to compare two sketches with each other. However, it takes a significant amount of space as it grows linear with respect to k.

Rather than storing an entire byte for each character in the kmer, which may be more depending on the machine, the above implementation could store 2 bits for each nucleotide. If we assume a 4 letter alphabet (A,C,G,T), each character in the kmer can only take 4 values, so 2 bits should be sufficient. Consequently, if we assume the size of k is less than or equal to 16, the entire kmer can be stored in one integer through the use of bitwise operations. In fact, this may speed up the entire use of the python dictionary, because hash collisions could be resolved faster with simple integer comparisons than with string comparisons. Unfortunately, we did not have the time to implement and benchmark the above implementation.

In order to find matches between two genomes, it's imperative to compare the k-mers amongst each other. As a result, it may seem necessary to store the entire k-mer in memory, but by making sacrifices with respect to accuracy it may be possible to lower the space bound for the dictionary.

<u>Hash Sketch</u>

Each k-mer would take at least k bytes to store in the above Simple Sketch implementation; however, by mapping each kmer to an integer, a hash, every k-mer can be stored within 4 bytes.

Hash Sketch builds on this principle by storing the k-mer's hash rather than its actual value. Consequently, the size of the dictionary changes from $O(gpk)$ to $O(gpk)$, independent of k. In addition to the more optimal space bound, the dictionary as a whole should operate with better speed, since integer comparisons are much faster than string comparisons.

This implementation has its weaknesses, since it trades guaranteed accuracy for better performance. We can store at most $2^{32} = 4294967296$ possible values within 4 bytes. However, if we let k = 16, there are clearly $4^{16} = 2^{32}$ unique 16-mers. Consequently, by the pigeonhole principle, there cannot be a one to one function between the kmers of size larger than 16 and the possible integers, so there must be collisions.

Fortunately, genomes are highly repetitive and do not contain nearly as many unique kmers; however, larger genomes like Paris Japonica, which contains 149 billion base pairs, may test this assumption.

On the other hand, most hash functions do not guarantee that kmers of size 16 or less would even be hashed to unique integers. One potential solution to optimize our current hash sketch implementation could lie in the use of cryptographic hashes to create a unique hash for each observed k-mer, but our current implementation relies on the probability of collisions being low due to the size and repetitiveness of most genomes.

<u>**Subsequences**</u>

When collecting k-mers, PAGS by default will take consecutive substrings, a typical way of constructing genome sketches. Additionally, it is able to take in space lengths from the user inputs to attain gapped subsequences. Considering the tiny size of genome "vocabulary", "A", "C", "G", "T", we suppose that the correlations among nucleotides separated by a fixed distance should be rather apparent and structured. It also caters to the fact that even genomes that are highly resembling cannot match each other verbatim. Structural correlations are likely to be more tangible than correlation of consistent sequences under this circumstance. Theoretically, as we have explored in class, using subsequences as k-mers improves the specificity of k-mers and consequently makes a match between sketches more likely to indicate an actual match in alignment, which improves the validity of *Assumption 1.* Hence, storing subsequences that are “spread out” by gaps into the sketches might provide us better results in calculating similarity of both genomes.

Singletons and Bloom Filter

As we construct sketches from a collection of k-mers, we prefer them to be composed of constituents with high occurrences rather than with low ones. The reason is that k-mers with exceedingly low occurrences are likely to result from incorrect reads or other errors, which will jeopardize the representativeness of the sketch. The most suspicious of them are the singletons - k-mers that occur only once. Therefore, it is to our benefits to remove them from the hash table. To check single-copy k-mers, we would use Bloom Filter, a space-efficient probabilistic data structures that examines the membership of an item in respects to a set. Our counting bloom filter is based on a 64-bit bit array and hashes each input string to a couple of values stored in different slots. In the system, it collects every k-mer the first time it appears, and removes those that are seen a second time, which indicates that they are not singletons. Thus, we’ll attain a bloom filter of singletons. Compared with storing the single-copy k-mers in a set, which grows linearly in size, the size of a Bloom filter is fixed. The hash map also compares integers rather than strings, which is fast. The drawback is that sometimes it would result in increasing false positive rates, meaning that it will report that the key is a singleton but actually it’s not. But such probability is low and it never generates false negative results, reporting that a key is not a singleton but it actually is. This trait helps to remove any misreported, likely erroneous, singleton to be mixed into our hash table.

Our implementation serves more as a proof of concept or experiment; for reliable use in PAGS, the bloom filter’s size would need to scale according to the size of the sketch and multiple hash functions would need to be used to reduce false positive rates. Regardless, it is clear that bloom filters could help reduce singletons and lower the space bounds of our application.

Hash Functions

Hash functions play a significant role in indexing kmers, as we intend to generate and store them with minimum space and time tradeoff. Assuming the size of k-mer is not larger than 16 characters, the Python builtin function may serve well using 2 bits to store each nucleotide. The __hash()__ that hash() returns the integer value from allows 8 bytes on 64 bits-build, which is totally capable of this task. However, it does little to address hash collisions: according to the Pigeonhole Theorem, the Python builtin hash function can save at most $2^{32} = 4294967296$ possible values within 4 bytes. For kmers over the size of 16, it is inevitable that collisions will occur. To boost the hashing accuracies lost from the HashSketch, we would like to implement a series of hashing functions.

The intuition for the improving hash functions is based on XORing. It is used for combining two values. In our methods, the input kmer is converted to a binary string in representative of its ASCII code. Then the bit string will be split into two portions that are converted to integers. At each bit of the two integers to combine, a 0 is returned if two bits are equal and 1 otherwise. Namely, 1 will be the outputs in 50 percent of the combinations. If each of two input bits has 50% chances of being either 0 or 1, their outputs will also have the 50%-50% distribution. The direct statistical implication is that it presents a more uniform distribution of combinations than other boolean operators (AND, OR, NOT). This method is believed to yield a relatively random distribution of hash values where they are less likely to result in collisions.

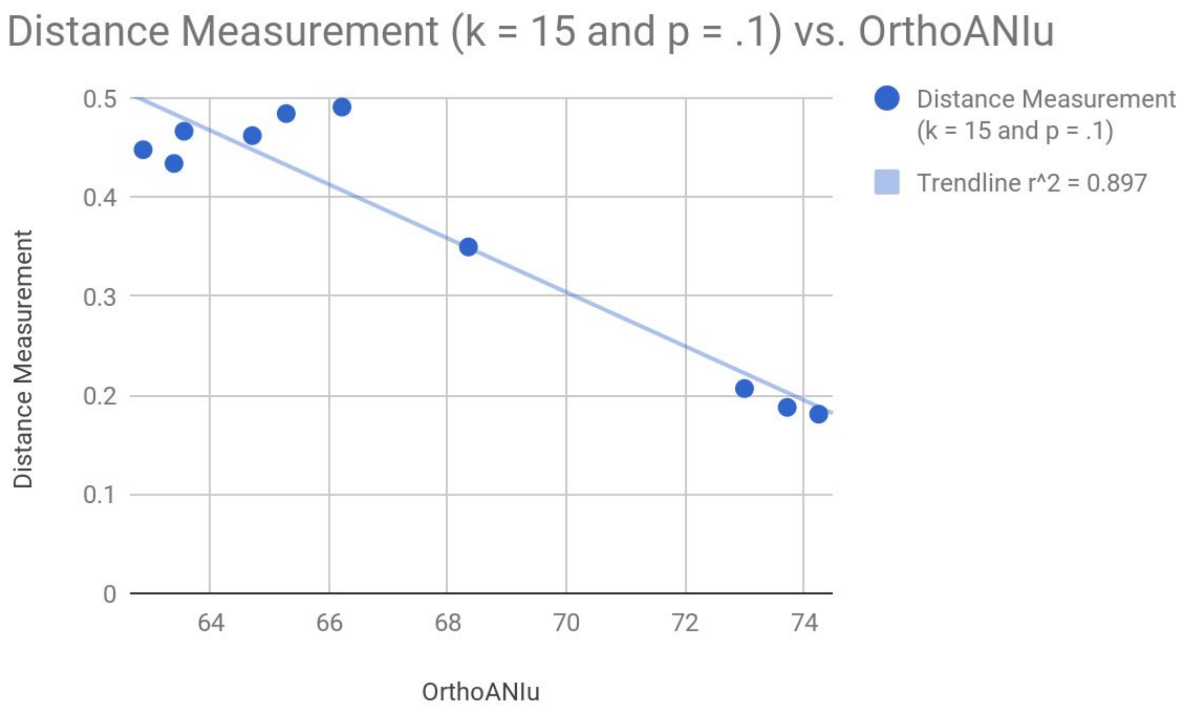
The binary_hash function dutifully implements this mechanism of XOR by comparing the converted ascii binary strings bit by bit. AT least to the level of characters, each character is guaranteed to have its unique bit strings. This makes their concatenated results unique as well. However, one discovery we found in practicing binary_hash is that the lengths of its hash values are, understandably, more random than the builtin hash function. Sometimes the hash values for 16-mer are more than 19 digits as typical of Python hash(), which requires more than 2 bits to store each nucleotide. Hence, it is expected that it will take more space to perform the hashing function. But on the other hand, it makes collisions of kmers longer than 16 much less likely.

The invert_hash function develops from the idea of XOR, that it additionally applies reversible actions to make hashes even less possible. Our idea is to reverse Thomas Wang's integer hashing function. First, the kmers are first converted to binary bit strings. Then the hashes are mixed into 64-bits. Each single bit of difference in the input will make about 1/2 of the output bits be different. This complicated computation will effectively prevent the reversible values to collide, since it establishes one-to-one mapping from each input to output. In the meantime, its space remains aligned with that of the original bit string. The complex nature of this function makes it more suitable for hashing kmers that are longer than 16, since python hash function is far more efficient in dealing with shorter inputs.

We also implement SHA-256, a cryptographic hashing function supported on Python hashlib module. An collision-resistance algorithm, it is computed with 32-bits words and returns a hexadecimal digest (hash value) in the end. All of these hash functions aim to resolve collision issues. Since we no longer consider each nucleotide as being stored with 2 bits, more runtime and space is expected when running these functions. It is a tradeoff between runtime and space and accuracy. Unless it is really imperative to examine the similarity of certain pair of genomes, we would recommend using the default Python hash function, which is enough considering its efficiency in processing kmers under the size of 16.

Results

Relationship Between ANI and Distance Measurement

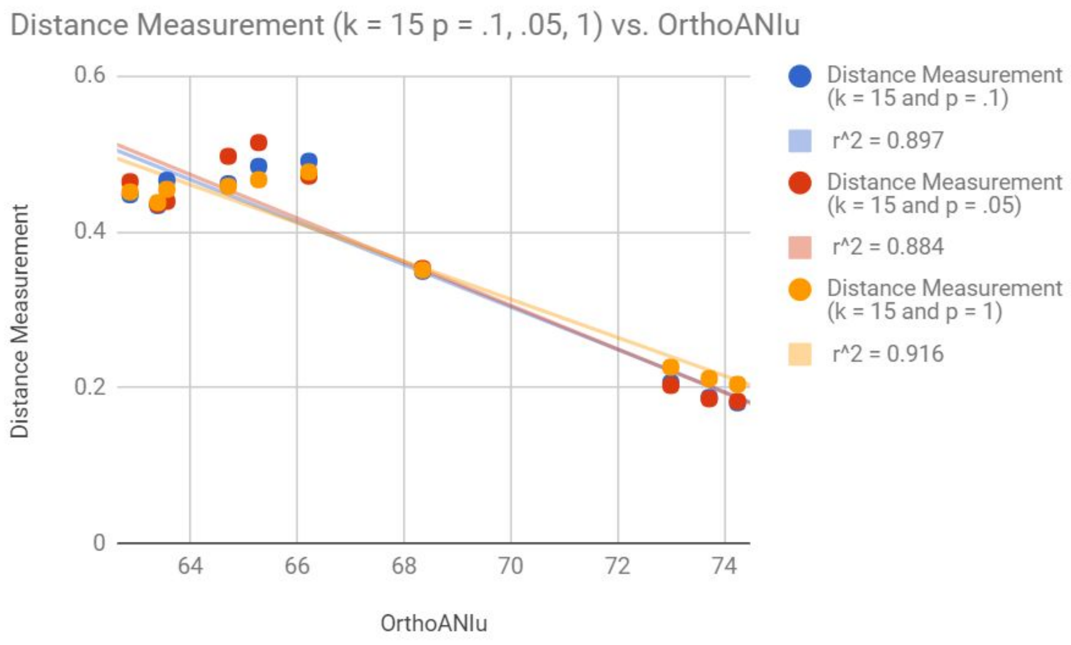


In order to test the validity of the distance measurement, the OrthoANIu values between two genomes was mapped with their corresponding distance measurement. From the plot, it is clear that there is an inverse relationship between OrthoANIu scores and our calculated distance measurement. This relationship is expected since higher OrthoANIu scores indicate more similar genomes and higher point mutation rates indicate more dissimilar genomes.

Furthermore, the observed r^2 value for the data, .897, indicates a strong relationship between the two scores. Consequently, it appears our distance values can provide a valid measurement for genome similarity.

Because the poisson random variable relies on point mutations being rare, less genome similarity would lead to a more inaccurate measurement. Consequently, this may explain why the points with more dissimilarity, lower OrthoANIu scores, appear to show more variability with respect to the distance measurement.

Varying the Parameter p:

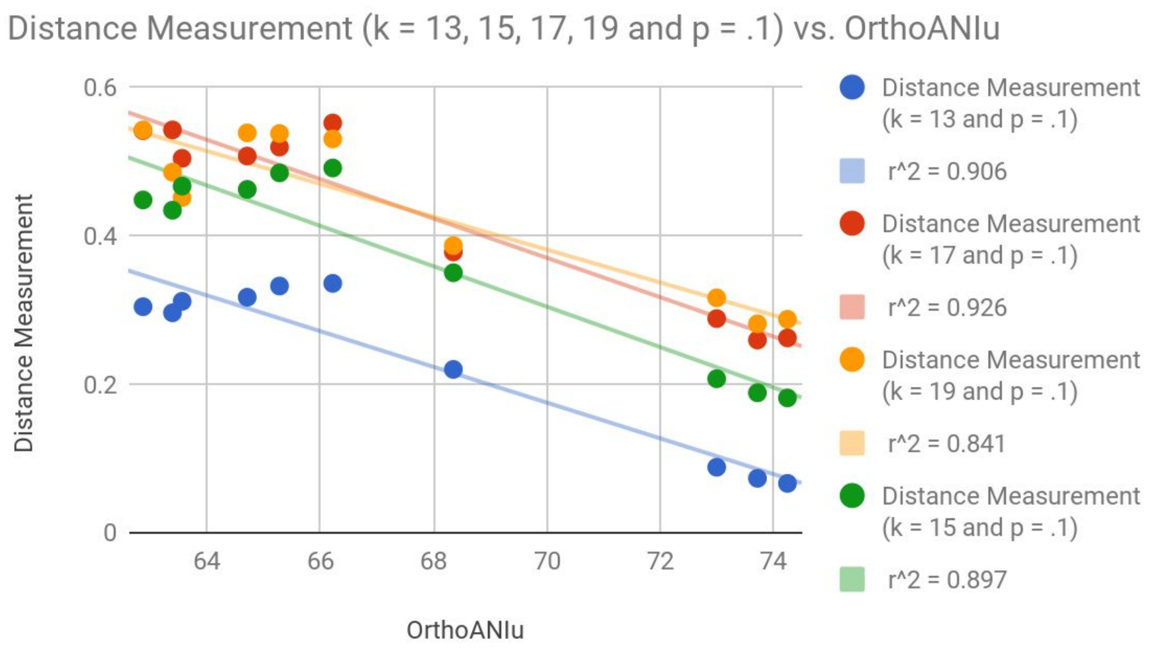


In order to determine the effect of the sketch size on the distance calculation, we varied the size of p to create different sized sketches. From the scatterplot above, it is clear that the size of the sketch seems to have little effect on the distance measurement : the same inverse relationship is shown. This shows that an accurate calculation can be found despite small sketch sizes.

Although the trend lines for all sketch sizes show strong, inverse relationships, the r^2 values seem to decrease as p decreases. This inherently makes sense, since lowering p would decrease the amount of information available to determine the $P(kmer\ in\ sketch\ of\ other\ genome)$.

Consequently, if the sketch size is reduced drastically, the calculation for the $P(kmer\ is\ in\ sketch\ of\ other\ genome)$ would have more variance, leading to less stable results. This could be counteracted by creating multiple small sketches of the genomes and averaging the results together to estimate the probability with less variance. In order to reduce the dependence between sketches, k-mers should not be re-chosen for other sketches; however, the probability of a k-mer being included in the sketch would have to be revised.

Varying the Size of K:



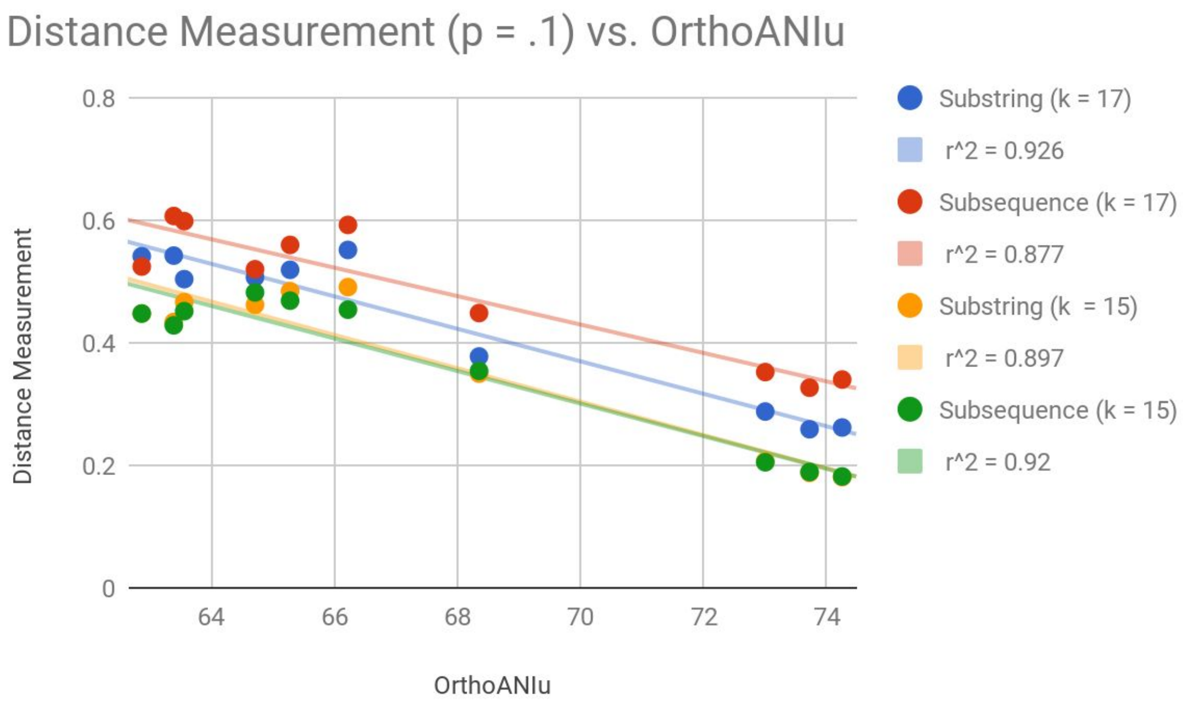
The distance measurement relies on *Assumption 1*: a k-mer match in the sketches is the k-mer's ideal alignment. In order to test the effect of k-mer size on this assumption, we plotted the associated distance measurement with varying *k* values fixed at the *p* value of .1.

Although all trend lines show a strong inverse relationship with generally high r^2 values, the trend lines themselves appear to have drastically different y-intercepts. By using smaller sizes of the k-mers, the specificity of each kmer in the genome is lower. Consequently, *Assumption 1* becomes weaker, since two k-mers in the sketches may match due to pure chance. There is no guarantee that they would occur together in the ideal alignment. The trend is still observed with $k = 13$, since two genomes that are highly similar will likely have more smaller k-mers in common. However, the actual distance calculation, intended to be the point mutation rate, becomes flawed.

When we reduce k to a small number such as two, the specificity is lowered to an extremely small number. This completely breaks *Assumption 1* and will show that two genomes are more similar than they actually are. For instance, when k was set to two, all probabilities were negative, since matches were overcounted in the sketches. Nevertheless, if a sufficiently sized k is chosen, the distance calculation can provide meaningful differentiation between genomes.

It is also worth noting that if a k that is too large was chosen, for genomes that are relative small in size as the ones we use, even if we include the entire genomes in the sketches, there is a very low probability to find a k-mer in both sketches that match. For example, for k = 19, we had to run the benchmark several time to get a distance calculation. For k that is larger than 20, PAGS appears to be unable to provide a distance calculation within several runs. Therefore, k > 20 might suit better for larger genomes with larger sketch sizes. As for the test data we have, we find k between 13 to 17 is suitable to calculate distances.

Subsequences vs. Substrings

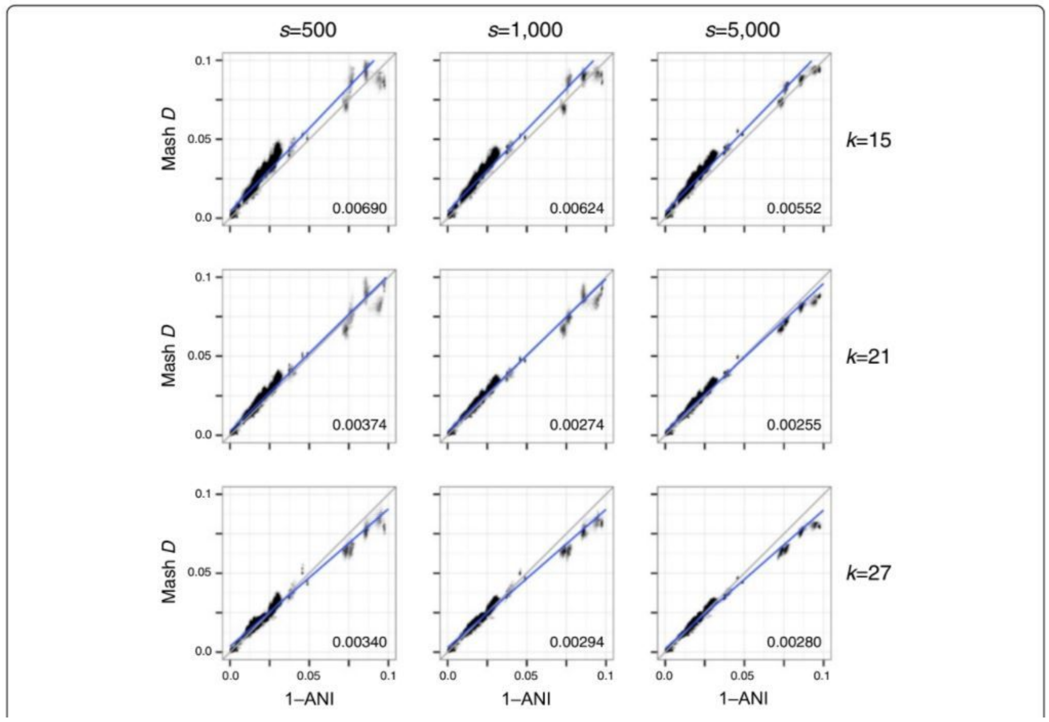


In order to test the effect of subsequences on the validity of *Assumption 1*: a k-mer match in a sketch implies the k-mer's are aligned together, we tested the distance calculation with similar values of k but varied the use of substrings and subsequences. The bitmask chosen was spaced-seed, like 101010.

In theory, for similar values of k, subsequences should have a higher specificity; consequently, a match in the sketch would be more convincing. This should make the core assumption more valid. However, from the above scatterplot, which was created by graphing the values for both subsequences and substrings for varying values of k, there appears to be negligible difference between using substrings and subsequences. Moreover, the r^2 values seem relatively independent from the use of substrings and subsequences. But more trials will need to be conducted to make us more confident with this conclusion.

Hence, to the best of our knowledge, using subsequences may not lead to a higher specificity after all in our case. The gapped subsequence failed to contain more implicit information than the k-length substring; however, it seems that a different bit mask, like 101101, may lead to results.

Distance Measurement vs. Mash Distance



The above scatter plots show the relationship between Mash Distance and ANI. The grey line represents the expected regression for Mash Distance and 1 - ANI; consequently, it is the line y = x.

Clearly, Mash provides a much more accurate of measurement of ANI than our probabilistic method. The RMS error for the data sets remains relatively low, and the lines of best fit all have slopes close to one. Furthermore, small sketch sizes, like 500, still accurately predict ANI values though with increasing inaccuracy.

Because we could not implement an automated method for collecting ANI scores for genomes, we needed to manually generate the ANI scores by using online genome comparison tools. Consequently, we could not generate the sufficient data points to truly compare the two methods. By measuring the similarity between more genomes, it is possible our solution will begin to model the line $y = -x + 1$; however, from the data collected, this seems unlikely.

Nevertheless, our measurement, while not accurately measuring the value 1 - ANI, still shows predictive behavior, since the above results show that there is a strong inverse relationship between our distance calculation and the measured ANI values.

Conclusions

Our probabilistic method for measuring genomic similarity seems to have a strong correlation with ANI values; a larger distance value indicates more dissimilarity and a lower ANI score. This relationship appears to be inverse and linear, and the measurement appears to be stable as p is varied to reasonable values; however, a larger sketch size should produce more accurate results. Although the measurement appears to be predictive, it still has some flaws that can be further refined and corrected. One of the considered optimizations includes finding a subtle balance between accuracy, the eventual goal of PAGS, and space efficiency, the expedient advantage which our program has. This can be resolved in data collection, by introducing collision-free hashing function and winnowing of underqualified (e.g. singletons) kmers. In the meantime, they need to be as economical with space as possible.

By using the Poisson random variable, the measurement will be inaccurate for highly different genomes. By switching to a binomial distribution to modeling the point mutation rate, this error may decrease.

Fundamentally, the main component that worths further investigation regarding the distance calculation method is its reliance on *Assumption 1,* that a match in the sketch implies that the two k-mers would match each other in the ideal alignment. This issue is compounded when a smaller value of k is chosen, since the two k-mers are more likely to occur in different areas. One method that could potentially increase the validity of the assumption would be the use of subsequences. From our benchmarking, the space-seeded subsequences failed to improve performance in the distance calculation, but a different bit mask may provide better results. For instance, if we took the subsequence with the bitmask 100111 for a k of size 4, this might have a higher specificity than taking a substring of size 4. Because of a higher specificity, a match in a sketch would be more convincing of an alignment pair.

Despite its limitations, PAGS shows a probabilistic approach can provide a relatively accurate similarity measurement with very low overhead of time and space, compared to alignment-based approaches. This means a more efficient approach in analyzing large data sets of genomes and allowing preprocessed data to reach downstream researches quicker. It could benefit research areas such as building phylogenetic trees to study evolutions, genomes clustering and studying relationships between specific genomes, etc. Thus, we see using probabilistic approach in analyzing genome similarity a method with potential for future researches.

Literature Cited

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

2. Indyk, Piotr, and Rajeev Motwani. “Approximate Nearest Neighbors.” *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing - STOC '98*, 1998, doi:10.1145/276698.276876.

3. Leimeister, Chris-Andre, and Burkhard Morgenstern. “Kmacs: the k -Mismatch Average Common Substring Approach to Alignment-Free Sequence Comparison.” *Bioinformatics*, vol. 30, no. 14, 2014, pp. 2000–2008., doi:10.1093/bioinformatics/btu331.

4. Ondov, Brian D., et al. “Mash: Fast Genome and Metagenome Distance Estimation Using MinHash.” *Genome Biology*, vol. 17, no. 1, 2016, doi:10.1186/s13059-016-0997-x

5. Lee, Imchang, et al. “OrthoANI: An Improved Algorithm and Software for Calculating Average Nucleotide Identity.” *International Journal of Systematic and Evolutionary Microbiology*, vol. 66, no. 2, Jan. 2016, pp. 1100–1103., doi:10.1099/ijsem.0.000760.

6. Pall Melsted and Jonathan K Pritchard. Efficient counting of k-mers in dna sequences using ´ a bloom filter. BMC bioinformatics, 12(1):333, 2011.

7. Chang, Donghoon, et al. “Indifferentiability of the Hash Algorithm BLAKE.” *Semantic Scholar*, Google Inc, 23 June 2011, www.semanticscholar.org/paper/Indifferentiability-of-the-Hash-Algorithm-BLAKE-Chang-Nandi/c93e51858fd7856ecfcf3c4192b2b63723e7a165.