# Utilizing Probabilistic Subsequence Methods and Markov Chains: Identifying Relationships in Protein Sequences

Kris Campbell, Elycia Finch,
Evan Mackay

04/09/2019

## 1 Abstract

This paper presents a comparison of two methods of sequence alignment algorithms using the probabilistic data structures: Bloom Filters and Markov chains. When comparing Bloom filters and Markov chains, versus typically used algorithms we were able use Bloom filters to show a reduction in time complexity from multiplicative to linear. Using Markov chains, we show how match, mismatch, and indel scores can be inferred from empirical data. Results from implementation show how these probabilistic data structures could be used to filter possible alignments before applying conventional global alignment algorithms.

## 2 Introduction

. The purpose of this study was to determine the feasibility of using probabilistic subsequence methods to identify sequence relationships and to compare the versatility of these methods with the subsequence search tools currently used. As paralogs are likely to have similar genes and functions, sub-string searching is necessary to properly identify these similarities. In view of the challenges faced in the industry, our team aimed to identify how probabilistic methods compare with Needleman-Wunsch and BLAST methods respectively. Currently, it is not yet possible to compare full length sequences with current technologies. As computation power is limited, there is a high demand for a quick, clever way to compare 2 or more sequences.

Markov chains are a way of preserving states and their transitions. We used Bloom Filters to see the probability of transitioning from one subsequence to another. From this we can generate the likelihood of transition from one sequence to the next.

Bloom filters are characterized by performing several different quick hashing methods on each subsequence. This results in several different integer values. These values are then stored in a single bit vector. In order to accurately test for membership in the sequence, the same hashes are then performed to the subsequence that is being searched. If there is not an complete match on the bit vector it can be determined that the subsequence does not exist in the given sequence. When all of the bit vectors match, this indicates the subsequence exists in the sequence given an accuracy dependent on the length of k and the number of hashes used.

We hypothesize that probabilistic methods of sequence alignment can demonstrate a marked improvement over conventional algorithms.

## 3 Methods

### 3.1 Markov Chain

Markov Chains are a way of representing a sequence of possible system events or states and the potential of transition from one state to another. A Markov Chain is created from the entire sequence for each k-mer with the probabilities of the next k-mer appearance. The subsequence to be searched is then converted into k-mer segments, using Markov chains the probabilities of the entire subsequence are then calculated. For the purpose of our research, our Markov Chain is built from the k-mer sub-sequences of a long protein sequence. This implementation requires a O(2n) time complexity, which can be scaled down to O(n) respectively.

#### 3.1.1 The Algorithm

The K-Mer Markov Chain is created from the entire sequence for each k-mer with the probabilities of the next k-mer appearance. The subsequence to be searched is then converted into k-mer segments, using

Markov chains the probabilities of the entire subsequence are then calculated. The probabilities are calculated by counting the number of occurrences that a k-mer appears after another k-mer.

#### 3.1.2 Experimentation and implementation

We implemented k-mer Markov chains using Python 3.6. The first step in creating the Markov chain is to engender a dictionary of all words in the sequence of length k. This dictionary also stores the occurrences for every following word in the sequence.

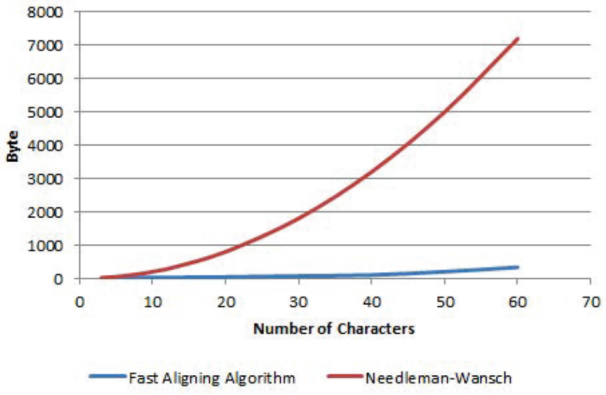
For example, with a k-mer length of 3, 'LCVNQEY' would increment the count of 'NQE' after 'LCV' in the dictionary. It would also increment the count of 'QEY' after 'CVN' in the dictionary, this is called the sliding window approach.

### 3.2 Bloom Filters

Bloom filters are a bit vector where all fields are initially set to 0. Using k independent hash functions with each of the hash functions outputting a number between 0 and the length of the bloom filter. Elements are added to the bloom filter by applying each hash function and setting the bit in the position generated by each hash to 1.

To check if an element is present in a Bloom filter the same hashes are applied to the element in question. If all the bits in the positions generated from each hash are set to one we can confidently say the element probably exists in the set.

Using the quick lookup time of checking for presence in a Bloom filter we are able to quickly align sequences compared to the widely used Needleman Wunsch algorithm.




## 4 Results

We measured the integrity of our results using three criteria: Memory used, Quality of results, and total execution time. When compared against the Needleman-Wunsch Algorithm, our K-Mer Bloom filter outperformed in all three areas. In terms of time, bloom filters run at an impeccable O(n), far superior to Needleman-Wunsch, having a time complexity $\Omega(m * n)$. Taking quality into consideration, we found Bloom Filters to be consistent regardless of size, whereas Needleman-Wunsch's accuracy increases with size. The most significant finding is the memory used. Our results show bloom filters O(n) complexity to be substantially superior in memory usage, given Needleman-Wunsch's burdensome memory usage of $O(N^2)$. A linear regression shows that the Bloom filter indexing stage takes only 1.7 seconds, compared to 20.2 for the Integer-encoded and 11.9 for the string-based indexing. Similarly, the Bloom filter matches 1718 sequences per second, compared to 589 and 310 for the Integer and string based indexes, respectively. (Malde, OâSullivan, 2008.)

The K-mer Markov chain method posed quite apprehensive results. K-mer Markov Chains can be used to represent the states and transition probabilities of protein sequences, however, the memory and computational power required for any useful analysis is too high compared to other methods. Using a k-mer length of 5 with a sequence of 2.5 million proteins, the resulting Markov chain is already over 200 MB, with this size only increasing exponentially with larger k-mer values.

## 5 Conclusion

Needleman-Wunsch is more gradual, but can be applied to a far wider range of data sets. In comparison, Bloom Filters exceed Needleman-Wunsch substantially, but can only be applied to a niche set. Bloom filter could potentially be used as a preprocessing stage to eliminate unlikely candidates for a match.

For practical purposes, Markov chains are too memory-intensive. These Markov chains, not including probabilities of insertion/deletion, could be feasible used on a small scale. K-mer Markov chains is a great way to represent the states and transition probabilities of a small protein sequences, but becomes computationally infeasible with larger sequences.

# 6 References

[1]Benoit G., Lemaitre C., Lavenier D., et al. 2015. Reference-free compression of high throughput sequencing data with a probabilistic de Bruijn graph. BMC Bioinformatics 16, 288.

[2]Bloom B. 1970. Space/time trade-offs in hash coding with allowable errors. Commun. ACM 13, 422â426 Chor, B., Horn, D., Goldman, N., Levy, Y., Massingham, T. (2009). Genomic DNA k-mer spectra: Models and modalities. Genome Biology,10(10). doi:10.1186/gb-2009-10-10-r108 [3]Malde, K., OâSullivan, B. (2008). Using Bloom Filters for Large Scale Gene Sequence Analysis in Haskell. Practical Aspects of Declarative Languages Lecture Notes in Computer Science, 183-194. doi:10.1007/978-3-540-92995-$6_1$3

[4]Pell J., Hintze A., Canino-Koning R., Howe A., et al. 2012. Scaling metagenome sequence assembly with probabilistic de Bruijn graphs. Proc. Natl. Acad. Sci. U. S. A.

[5]Pellow, D., Filippova, D., Kingsford, C. (2017). Improving Bloom filter performance on sequence data using k-mer Bloom filters. Journal of Computational Biology, 24(6), 547-557.

[6]R/bioinformatics - Probability of generating a kmer? (n.d.). Retrieved from `https://www.reddit.com/r/bioinformatics/comments/7w3jrk/probability_of_generating_a_kmer/` Rozov R., Shamir R., and Halperin E. 2014. Fast lossless compression via cascading Bloom filters. BMC Bioinformatics

[7]Shi H., Schmidt B., Liu W., and MÃ$\frac{1}{4}$ller-Wittig W. 2009. Accelerating error correction in high-throughput short-read DNA sequencing data with CUDA, 1â8. In IEEE International Symposium on Parallel Distributed Processing, 2009 (IPDPS 2009) Rome, Italy

[8]Space/time trade-offs in hash coding with allowable errors. (n.d.). Retrieved from https://dl.acm.org/citation.cfm?doid=362686.362692

[9]Yuan, Z. (1999). Prediction of protein subcellular locations using Markov chain models. FEBS letters, 451(1), 23-26.