Max-Minhash Clustering Algorithm of RNA sequences

Mehdi Asser Husum Nadif

April 4, 2015

Contents

Midway Report

Problem Statement

I will attempt to create a clustering algorithm, that can compete with, or even surpass, uClust in terms of speed, without losing precision, when working with a dataset of 500000 RNA/DNA sequences of length 500-1500.

Analysis

The very high number of sequences that have to be compared means that the speed of the algorithm that i develop is extraordinary. uClust is at the moment number one contender in terms of speed without loss of much precision.

Studies have shown that the implementation of Minhash in [16] have approached both the speed and precision of uClust. This method is quite different from the centroid based approach of uClust, and therefore spurred my interest.

My attempt will be to apply some improvements to the Minhash implementation above, and see whether i can achieve the same speed and precision as that of uClust. This implementation i will call Max-Minhash.

Status

Currently i have finished the following tasks:

1. Finished a prototype of my Max-Minhash algorithm. It has the following functionalities
   (a) Singlethreaded implementation of a Max-Minhash algorithm
   (b) A hashfunction based on the one in [16]
   (c) Reads sequences from .seq files.
2. Finished a draft of the following two sections: Universal Hashing and Minhash

What is left to be done are the following tasks

1. A comprehensive test of several universal hashing schemes to find a method that might be faster than the currently used one.
2. Developing the prototype so that it has the following improvements
   (a) The newfound quickest hashfunction.
   (b) Programmed to work in a Mapreduce framework, so that it can be run in Hadoop
   (c) Able to read .fasta files. Has to be able to handle sequences that contain the character N.
3. Comprehensive tests of my algorithm in comparison to the uSearch algorithm, to see which is faster and most precise

4. As my algorithm uses K-mers as the basis for its Minhash input, a comprehensive test of which k-mer size is the most optimal for precision and speed.

5. Documenting all the tests that i perform above.

6. Writing a conclusion.

7. Improving the sections i wrote about universal hashing, and the Minhash section

It should be noted that quite a few of these tasks can be completed without having 2.b ready, read. 1, 2.c, and 4. These will therefore be done parallel to implementing the Mapreduce framework, some using the current prototype.

Time plan

I will estimate each of the above remaining tasks and insert them into a Gantz diagram.

1. Test of universal hashing schemes.
   (a) **Product:** Imlpementation of, and a comprehensive test of several hashing schemes. Includes graphs and data. Includes conclusivee findings.
   (b) **Resources:** Java, .seq test data
   (c) **Dependencies:** Section on Universal hashing
   (d) **Workload:** 2 days

2. Implement IO handler that can read Fasta files
   (a) **Product:** Implementing IO handler that can read fasta files.
   (b) Resources: Java, fasta files
   (c) Dependencies: Defining how to handle sequences containing the character N, Java
   (d) **Workload:** 1 day

3. Mapreduce Framework
   (a) **Product:** Improving the prototype to be runnable in a Mapreduce Framework.
   (b) **Resources:** Java, Hadoop
   (c) **Dependencies:** Knowledge of Hadoop, Researching Mapreduce
   (d) **Workload:** 10 days

4. Test of K-mer sizes
   (a) **Product:** Documented tests of k-mer sizes to find the appropriate k-mer size for RNA/DNA sequences. Includes data and graphs of results.

(b) **Resources:** Java, .seq files

(c) **Dependencies:** A Gold standard, Java, .seq test data

(d) **Workload:** 2 days

5. Final comparative test

    (a) **Product:** Documented final comparative test of Max-min implementation alongside uClust. Includes data and graphs of results.

    (b) **Resources:** Java, Hadoop, usearch

    (c) **Dependencies:** A finished implementation of Max-min, A sorted .fasta file for smallmem usearch.

    (d) **Workload:** 3 days.

6. Section about my algorithm

    (a) **Product:** Writing a section describing my algorithm. Includes examples from code, etc.

    (b) **Resources:** Latex

    (c) **Dependencies:** Final implementation of Max-Min hash algorithm.

    (d) **Workload:** 1 day.

7. Finetuning the report.

    (a) **Product:** Writing the conclusion and finetuning the other sections. Making a red thread between all sections. Assuring the scientific correctness of introductory sections about minhash and universal hashing.

    (b) **Resources:** Latex, Articles about previous sections, feedback

    (c) **Dependencies:** Section about algorithm, Section about Minhash, Section about universal hashing, The documented tests described above.

    (d) **Workload:** 10 days.

The timeplan will be describe so that the number above will be completed within the week it is set into:

Week 15: (1), (2)
Week 16: (3), (4)
Week 17 - 19: (3),(6)
Week 20: (5)
Week 21-23: (7)

Additionally, the sections about Minhash and about Universal Hashing have been written and included in this midway report for review.

Minhash

Hashing

In the implementation of randomized algorithms, a strong need for data representation is needed. Truly independent hash functions are extremely useful in these situations. It allows generalization of arrays, where it is possible to access an arbitrary position in an array in *O(1)* time.

Introduction to hash functions

Imagine a universe of keys $U = \{u_0, u_1, ..., u_{m-1}\}$ and a range $[r] = \{0, 1, ..., m-1\}$ where u_i are integers module m. Given a hash function $h(x) \rightarrow [r]$ which takes any $u_i, i = 0, 1, ..m-1$ as argument, it should hold that $\forall u_i, \exists r_j \in [r]$ such that $h(u_i) = r_j$. What remains is to consider collisions, which we define as

$$\delta_h(x,y) = \begin{cases} 1 & \text{if } x \neq y \text{ and } f(x) = f(y) \\ 0 & \text{else} \end{cases} \tag{1}$$

for a given hashfunction h and two keys x and y. The goal of a hashing algorithm is then to minimize the number of collisions across all possible keys. A truly random hash function can assure that there are no collision at all. Unfortunately, to implement such a function would require at least $|U| \log_2 m$ bits[19], defeating the purpose of hash functions altogether. Fixed hashing algorithms have attempted to solve this problem. Unfortunately, its dependence on input causes a worst case average retrieval time of $\Theta(m)$.[6]

Universal hashing can circumvent the memory and computation cost of both random- and fixed hashing, without losing much precision. An introduction to universal hashing will follow, alongside two applications of said hashing which will be tested later in this paper.

Universal hash functions

The first mention of universal hashing was in [3], in which they define universality of hash functions as follows:

Given a class of hash functions $H : U \rightarrow [r]$, H is said to be universal if $\forall x \forall y \in U$

$$\delta_H(x,y) \leq |H|/|[r]|$$

where, with $S \subset U$

$$\delta_H(x,S) = \sum_{h \in H} \sum_{y \in S} \delta_h(x,y)$$

That is, H is said to be universal if

$$\Pr_h[h(x) = h(y)] \leq 1/m \tag{2}$$

for a random $h \in H$. In many applications, $\Pr_h[h(x) = h(y)] \leq c/m$ for $c = O(1)$ is sufficiently low.

Carter and Wegman[3]

Given a prime $p \geq m$ and a hash function $h^C_{a,b} : [U] \to [r]$,

$$h^C_{a,b}(x) = ((a * x + b) \mod p) \mod m \tag{3}$$

where a and b are integers mod m, where $a \neq 0$. We want to prove that $h^C_{a,b}(x)$ satisfies Eq. 2; thus proving that it is universal.

Let x and y be two randomly selected keys in U where $x \neq y$. For a given hash function $h^C_{a,b}$,

$$r = a \cdot x + b \mod p$$

$$q = a \cdot y + b \mod p$$

We see that $r \neq q$ since

$$r - s \equiv a(k - l)(\mod p)$$

must be non-zero since p is prime and both a and $(k - l)$ are non-zero module p, and therefore $a(k - l) > 0$ as two non-zero multiplied by each other cannot be positive, and therefore must also be non-zero module p. Therefore, $\forall a \forall b, h_{a,b}$ will map to distinct values for the given x and y, at least at the mod p level.

Dietzfelbinger et al.[7]

Also commonly refered to as **multiply-shift**, this state of the art scheme described in [7] reduces computation time by eliminating the need for the **mod** operator. This is especially useful when the key is larger than 32 bits, in which case Carter and Wegman's suggestion is quite costly[19].

Take a universe $U \geq 2^k$ which is all k-bit numbers. For $l = \{1, .., k\}$, the hash functions $h^D_a(x) : \{0, ..., 2^k - 1\} \to \{0, ..., 2^l - 1\}$ are then defined as

$$h^D_a(x) = (a \cdot x \mod 2^k) \div 2^{k-l} \tag{4}$$

for a random odd number $0 < a < 2^k$. l is bitsize of the value the keys map to. The following C-like code shows just how easy the implementation of such an algorithm is

```
h(x)=(unsigned) (a*x) >> (k-l)
```

This scheme only nearly satisfies Eq. 2, as for two distinct $x, y \in U$ and any allowed a

$$\Pr_{h^D_a}[h^D_a(x) = h^D_a(y)] \leq \frac{1}{2^{l-1}} = \frac{2}{m} \tag{5}$$

If Eq. 5 is not sufficently precise, Wölfel [21, p.18-19] modified this scheme so that it met the requirement in Eq. 2. The hash function is then

$$h^D_{a,b} = ((a \cdot x + b) \mod 2^k) \div 2^{k-l}$$

where $a < 2^k$ is a positive odd number, and $0 \leq b < 2^{k-l}$. This way Eq. 2 is met for $x \neq y \pmod{2^k}$. For a proof of this, consult [21][1]. The C-like implementation shown below reveals that the modifications are only minimal

```
h(x)=(unsigned)((a*x) + b) >> (k-l)
```

References

[1] R. C. D. Anil K. Jain. *Algorithms for Clustering Data.* First edition, 1988.

[2] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60:327–336, 1998.

[3] J. Carter and M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143 – 154, 1979.

[4] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 380–388, New York, NY, USA, 2002. ACM.

[5] W. Chen, C. K. Zhang, Y. Cheng, S. Zhang, and H. Zhao. A comparison of methods for clustering 16s rrna sequences into otus. *PLoS ONE*, 8(8):e70837, 08 2013.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition, 2009.

[7] M. Dietzfelbinger, T. Hagerup, J. Katajainen, and M. Penttonen. A reliable randomized algorithm for the closest-pair problem. *Journal of Algorithms*, 25(1):19 – 51, 1997.

[8] R. C. Edgar. Search and clustering order of magnitude faster than blast. *Bioinformatics*, 2010.

[9] M. Farrar. Striped smith–waterman speeds database searches six times over other simd implementations. *Bioinformatics*, 23(2):156–161, 2007.

[10] L. Fu, B. Niu, Z. Zhu, S. Wu, and W. Li. Cd-hit: accelerated for clustering the next-generation sequencing data. *Bioinformatics*, 28(23):3150–3152, 2012.

[11] M. Ghodsi, B. Liu, and M. Pop. Dnaclust: accurate and efficient clustering of phylogenetic marker genes. *BMC Bioinformatics*, 12(1):271, 2011.

[12] J. Ji, J. Li, S. Yan, Q. Tian, and B. Zhang. Min-max hash for jaccard similarity. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pages 301–309, Dec 2013.

[13] W. Kaisers, H. Schwender, and H. Schaal. Hierarchical clustering of DNA k-mer counts in RNA-seq fastq files reveals batch effects. *ArXiv e-prints*, May 2014.

[1] The text is in german

[14] M. Kazimianec and A. Mazeika. Clustering of short strings in large databases. In *Database and Expert Systems Application, 2009. DEXA '09. 20th International Workshop on*, pages 368–372, Aug 2009.

[15] W. Li, L. Fu, B. Niu, S. Wu, and J. Wooley. Ultrafast clustering algorithms for metagenomic sequence analysis. *Briefings in Bioinformatics*, 2012.

[16] Z. Rasheed and H. Rangwala. A map-reduce framework for clustering metagenomes. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 549–558, May 2013.

[17] D. Russell, S. Way, A. Benson, and K. Sayood. A grammar-based distance metric enables fast and accurate clustering of large sets of 16s sequences. *BMC Bioinformatics*, 11(1):601, 2010.

[18] Y. Si, P. Liu, P. Li, and T. P. Brutnell. Model-based clustering for rna-seq data. *Bioinformatics*, 2013.

[19] M. Thorup. High speed hashing for integers and strings, 2014.

[20] J. F. Trevor Hastie, Robert Tibshirani. *The Elements of Statistical Learning: Datamining, Inference, and Prediction.* Springer Series in Statistics, second edition, 2008.

[21] P. Wölfel. *Über die Komplexität der Multiplikation in eingeschränkten Branchingprogrammmodellen.* PhD thesis, Dortmund, Univ, 2003.

[22] X. Yang, J. Zola, and S. Aluru. Large-scale metagenomic sequence clustering on map-reduce clusters. *Journal of Bioinformatics and Computational Biology*, 11(01):1340001, 2013. PMID: 23427983.