CSE 584A Algorithms for Biosequence Comparison Spring Semester 2018

Final Project Instructions

Assigned: 3/26/2016 *Due Date: 5/2/2016*

1 Overview

Your final project for CSE 584 should be centered on a software tool that integrates methods that we've discussed in class, as well as (possibly) some of your own ideas or techniques from the literature, to solve a nontrivial computational problem. The course bias is naturally towards bioinformatics problems, but I'm willing to entertain projects that solve problems in other application domains. The main requirements are as follows:

1. Your project should address a practically interesting problem that entails searching, indexing, or clustering sequential data. This data could be biological sequences, documents, discrete time series, or some other appropriate data type.
2. Your project should be designed to scale to very large data sets (e.g. entire mammalian genomes, all of GenBank, all of Wikipedia, etc.) or otherwise deal with problems that would be intractable, or at least very computationally intensive, if solved naively. Your actual implementation might not quite reach this goal due to time constraints, but it should be written so as not to introduce avoidable algorithmic or implementation bottlenecks to scalability. Try to make it so you can (and do) test your actual implementation on at least a gigabyte of data.
3. You should implement the core stringology algorithms required for your project yourself. Exceptions to this rule may be made by prior consultation with me (e.g., it's not reasonable to make someone implement their own SVD unless they really, really want to).

Your project is due by Wednesday, May 2nd at 5 PM so that I have time to review all projects before final grades are due.

Projects may be done individually or in a group of two. However, a two-person project must be strictly more than one person's worth of work.

2 Detailed Requirements

You will turn in all project materials via a GitHub repository. I will create a final project repository for you.

The main thing you will turn in is a write-up describing your project. This write-up should be 6-10 pages single-spaced (it can be longer if you want, but do *not* exceed 15 pages.) The write-up should include

1. An abstract with a brief summary of the project goals and your major results.
2. A description of the problem you're trying to solve, including both a justification of why it's interesting and a formal statement of the problem in stringology terms (if appropriate).
3. A review of existing methods and tools for solving this and related problems. I expect at least some investigation of the literature to find out what the state-of-the-art methods are for your problem.

4. A description of your implementation, including the algorithms you used, important design choices you made, and issues arising with the implementation and scaling to large sequences. You need not repeat the descriptions of algorithms that are already "common knowledge" from class, but you *must* justify why you used one method and not another!

5. An evaluation of your implementation, showing that it **works correctly and scales to large data sets** (as time and code permit). You should quantitatively evaluate the performance of your implementation. If you are familiar with other easily accessible tools that solve the same problem as yours, you might want to do a direct head-to-head comparison. Your results may include synthetic test cases but should also include at least one "real-world" test of significant size.

6. A bibliography (this doesn't count toward the page limit).

You should also check your final code into the repository. However, I should be able to judge the quality, scope, and utility of your project from the write-up alone, without looking at your code. If you are so inclined and have the background, you might also consider directing your efforts to answer a particular biological question.

Please do *not* check in multi-gigabyte data or output files.

3 Proposing your Project

No later than class time on **Monday, April 2nd**, please email me a proposal for your project, which should be about a page. You should specify the practical problem you plan to solve, briefly describe why it is important, and state some specific goals for how big an input you will handle, what your output will look like, and what interesting features your application will have. If there are multiple algorithmic approaches that you could take to implement your application, please specify which one you plan to use. Please also address how you plan to obtain test cases for your project.

If you are doing a project in a group of two, please specifically address why the scope of the project is appropriate for two people, how the project partners plan to divide up the work, and how they will coordinate their efforts. In particular, *please indicate whose repository will be used to turn in the project*, so I can make sure both partners have access to it.

I will review your project proposal and contact you if I have any suggestions or concerns.

4 Examples of Project Ideas

4.1 PCR Primer Design Tool

A common problem in the design of high-throughput genomics experiments is the following: given a long DNA sequence and a list of regions to be amplified from that sequence, design a pair of PCR primers specific to each region. From a string-matching point of view, the most important property of your primers should be that the primers for a given region uniquely amplify it. More aggressively, no primer (or long substring of a primer) should appear in the sequence anywhere other than where it is intended to hybridize. Besides this uniqueness constraint, there are other important constraints on PCR primer design, particularly choosing primers whose base compositions (fractions of A/T vs G/C) allow them all to work at about the same temperature.

This project will require you to read up on the basics of PCR and DNA melting (the standard composition-based temperature estimate of $T_m = 2n_{GC} + n_{AT}$ is fine). Looking at some of the existing tools for primer and probe design should also give you some ideas.

You should use some of the scalable inexact matching methods we will discuss in class during April to match potential primers against the genome. These are the methods used in BWA and Bowtie (though you should implement them yourself!).

For inspiration, see, e.g.:

- H-H. Chou, A-P. Hsia, D.L. Mooney, and P.S. Schnable. "Picky: oligo microarray design for large genomes." *Bioinformatics* 20:2893-2902, 2004.
- A. Untergasser, I. Cutcutache, T. Koressaar, J. Ye, et al. "Primer3 – new capabilities and interfaces." *Nucleic Acids Research* 40:e115, 2012.

4.2 Clustering Tool for de novo mRNA Assembly

RNASeq is a powerful technology for directly measuring the frequency of different messenger RNAs among all those being expressed in a population of cells. The most common approach to RNASeq is to align each read back to the genome from which it came. However, if the genome's sequence is not completely known, it may be useful to *assemble* mRNA sequences directly by piecing together overlapping reads. This is particularly helpful if the goal is to discover novel splice variants of known mRNAs or to find genes that, for whatever reason, may not be represented in your genomic sequence.

Computationally, the first step in mRNA assembly is to divide the reads into clusters according to overlap. If reads S and T overlap by at least a certain number of bases (perhaps allowing for a few differences due to sequencing error), then they should be clustered together. Clustering may be done transitively (i.e. if S overlaps T and T overlaps U, put them all in one cluster), or more stringent criteria may be used. In any case, the goal is to divide a set of perhaps millions of short reads into a number of clusters, each of which can then be assembled using special-purpose assembly algorithms.

Your goal for this project is to read up on existing *de novo* mRNA assembly tools, choose and implement an algorithm for the required overlap detection, and finally perform clustering on large read sets. If you have time, you can use an existing algorithm or piece of software to actually do the final assembly step.

You can either try to find overlaps directly, or use an index-based procedure in which you enumerate all the long k-mers in the read set and try to find potentially overlapping reads based on shared k-mers. In either case, you should use a space-efficient method for matching. For example, you could use the k-mer enumeration technique from Homework 3 to find all distinct long k-mers, then recover for each the set of reads that contain it, and finally cluster together reads that share one or more long k-mers (and then check the clusters for overlaps).

For inspiration, see, e.g.:

- J.T. Simpson, K. Wong, S.D. Jackman, J.E. Schein, S.J. Jones, and I. Birol. "ABySS: a parallel assembler for short read sequence data." *Genome Research* 19:1117-1123, 2009.
- M.G. Grabherr, B.J. Haas, M. Yassour, J.Z. Levin, et al. "Full-length transcriptome assembly from RNA-Seq data without a reference genome." *Nature Biotechnology* 29:644-652, 2011.

4.3 k-mer Kernels for Genetic Distance

Consider the problem of measuring the evolutionary distance (i.e. amount of mutation) separating two organisms' genomes. For two small sequences, we've seen that distance is measured by aligning the sequences and then counting the number of differences in the alignment. For very large genomes, alignment can be quite challenging, especially when the sequences are not collinear because one or both have undergone large-scale rearrangements.

An interesting recent approach to estimating evolutionary distance uses the following idea. If two organisms' genomes are not very different, they will match at a large fraction of homologous base positions. Hence, there is a good chance that any stretch of k contiguous positions will consist entirely of matches. Conversely, if we can measure the fraction of all unique k-mers appearing in either genome that are common to both, then we can use this fraction to compute an estimate of genetic distance. This approach has the added advantage that *we don't even have to assemble the two genomes* – we can measure the set of overlapping k-mers using the unassembled read sets. You can find details of this approach in Fan et al.'s recent paper describing their AAF program.

Your job in this project is to implement a phylogenetic distance measurement based on k-mer counting. However, instead of using Fan et al.'s memory-intensive hashing approach to counting shared k-mers, use a space-efficient method based on the 2BWT. The method is a simple extension of the k-mer counting algorithm described in Homework 3; you can find it in a recent paper by Belazzougui and Cunial.

Validate your method by applying a distance-based phylogeny algorithm to a collection of genomes. The AAF paper has some suitable data sets, and distance-based phylogeny tools are readily available online. For this project, I don't mind if you use an existing software package to build the BWTs, though you should write the actual k-mer counting code yourself.

For inspiration, see, e.g.:

- Fan, Ives, Surget-Groba, and Cannon, "An assembly and alignment-free method of phylogeny reconstruction from next-generation sequencing data," *BMC Genomics* 16:522, 2015.
- Belazzougui and Cunial, "A framework for space-efficient string kernels," *Proc. 2015 Conf. on Combinatorial Pattern Matching* 26-39, 2015.

4.4 Scalable BWT Construction

The "obvious" algorithm to construct a BWT from a string is first, to construct its suffix array, and second, to extract the BWT from the previous character of each suffix. This approach works great for smallish sequences (you worked an example of a few hundred megabytes in class), but it's still too space-intensive for many-gigabyte sequences or collections of reads. For this reason, researchers have sought space-efficient methods to build a BWT directly from a sequence without constructing its full suffix array.

Your goal is to find and implement a method to build the BWT that has two properties: (1) it is space-efficient enough to use on multi-gigabyte sequences, and (2) it is parallelizable, using either multiple CPU cores or a GPU. The classic scalable BWT algorithm is by Hon et al., though there are many successor methods. Two notable successors are by Kärkkäinen (used by Bowtie 2) and Lam et al. (used by BWA), but there are even newer approaches that scale to collections of short reads totaling *hundreds of gigabytes* in size. You should survey the available methods as part of your project report and justify your choice of what to implement.

You should demonstrate and benchmark your implementation on a collection of short reads totaling at least 10 gigabases. You can find such read sets at the NIH Sequence Read Archive.

For inspiration, see, e.g.:

- Hon, Lam, Sadakane, and Sung, "Constructing Compressed Suffix Arrays with Large Alphabets," in *Proc. ISAAC 2003*, LNCS 2906, 240-249, 2003.
- Kärkkäinen, "Fast BWT in smalls pace by blockwise suffix sorting," *Theoretical Computer Science* 387:249-257, 2007.
- Lam, Sung, Tam, Wong, and Yiu, "Compressed indexing and local alginment of DNA," *Bioinformatics* 24:791-797, 2008.

- Sirén, "Compressed full-text indices for highly repetitive collections," University of Helsinki CS Dept. Tech Report A-2012-5, 2012. Linked from `http://jltsiren.kapsi.fi/rlcsa`.
- Liu, Hankeln, and Schmidt, "Parallel and Space-efficient Construction of Burrows-Wheeler Transform and Suffix Array for Big Genome Data," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2015. accepted. `http://parabwt.sourceforge.net/homepage.htm#latest`
- Liu, Lou, and Lam, "GPU-accelerated BWT construction for large collection of short reads." Preprint 1401.7457 on arXiv.org, 2015.