EN 600.439/639: Final Project

Fall 2013
Prof. Ben Langmead

1 Project description

You will propose and produce a final research project worth 30% of your class grade. You may work in teams of up to 4 members. You may also work individually. Larger groups will be held to a higher standard for grading purposes. We will meet to discuss your proposal by the end of Fall break week. In the last week of class you / your group will give a 12 minute presentation describing your work. All project deliverables are due 11:59pm Friday, December 6, the last day of classes. Email deliverables to `cs439@cs.jhu.edu` with "EN 600.439/639 Final project submission" in the subject line.

1.1 How to approach the project

Pick a topic that interests you (e.g. indexing, dynamic programming alignment, HMMs, assembly) and *read papers* about what others have done in that area. Some specific ideas are listed in section 2 and some papers you might read are listed in section 3. You should get an idea for what methods have been developed in that area, what they're used for, and how popular they are. We encourage you to make an annotated bibliography as you go, which you can later plug into your write-up.

Make a goal: In broad terms, this could be: adapting method A to be useful for application B. Or extending method A to additionally handle cases B and C. Or making method A run faster or use less memory using an idea from area B. Pick a goal that plays to your skills.

You should *not* focus primarily on data analysis, though you are encouraged to use real data when evaluating your method. For instance, your *goal* shouldn't be to use an existing computational method to analyze a particular dataset. If you are having trouble finding real data of the needed type, email `cs439@cs.jhu.edu`.

1.2 Project proposal

After you've formed a team and decided on a goal, meet with me so we can discuss it and I can make suggestions. I encourage you / your team to meet with me on or before October 18. Email me (`langmea@cs.jhu.edu`) to make an appointment.

1.3 Deliverables

Your submission should include:

- Source code for all software produced, as well as a detailed README that helps us navigate your code. *We should be able to easily reproduce the results you describe in your report. Tell us how to do this in your README.*
- A write-up of $N + 5$ pages, where N is the number of people in your group, following the format description in section 1.6 below.

- Your presentation slides in PowerPoint, Keynote or PDF format.
- If you are on a team, a detailed (at least 1 paragraph) description of who did what. You will be graded according to your contribution. You will be penalized if you don't include this description.

1.4 Late policy

The late policy for the final project is separate from the late policy for homework. I.e. you can't use homework late days for the final project. You may continue to submit materials up to 24 hours after the deadline, but you will be penalized. We will determine the penalty based on how late you are and how much of the submission is late. Materials submitted more than 24 hours past the deadline will not be considered.

1.5 Source code

Consider using GitHub (`https://github.com`) or a similar site for collaborating and for storing your code. But you must submit your final code to us along with your other materials. We will grade you based on the submitted code, not code in GitHub.

If you used code from other people, *be very careful to credit them* and provide a link to exactly where you obtained the code in your README.

We must be able to run your software, preferably on the undergrad or graduate cluster at JHU Computer Science. For this reason, we discourage you from submitting a project that can only be run on hardware we can't use, like a particular GPU, FPGA, etc. Check with us if you are in doubt.

1.6 Write-up

Your write-up should have the following sections:

1. Abstract (1 paragraph): Briefly describe essence of your project
2. Introduction: Why work on this? Why your approach?
3. Prior work: What did you read? What did others accomplish before you?
4. Methods and software: What did you implement? Why? How?
5. Results: How well did your method work compared to others?
6. Conclusions: What did you learn? What should we come away with?
7. Literature cited: What papers are related to your project?

Unlike a typical published research paper, your write-up should describe everything you did in the project. If you spent a non-trivial amount of time thinking about something, researching something or implementing something, even if it didn't end up being part of your main contribution, find a way to mention it in the write-up. For example, if there was a method you considered, or started to implement but abandoned, mention this in "Methods and software" and describe why you chose not to go with it. But don't spend so much time on this that it crowds out your main contributions.

In the Results section, present a thoughtful evaluation of your method. Usually, you should compare yours to other methods. If comparison is only really possible using synthetic ("made up") data, that's fine. Ideally, your comparisons should involve both synthetic and real data. One place to find real sequencing data is the Sequence Read Archive: `http://www.ncbi.nlm.nih.gov/sra`. If you need help finding an appropriate dataset, ask us.

1.7 Presentation

In the last week of class, you / your group will give a 12 minute presentation describing your work. Your presentation is a valuable opportunity to use visual aids such as plots or diagrams to bolster your arguments. There will be a brief question-and-answer period after.

The particular content of your presentation is up to you. You should cover the highlights of your write-up, using plenty of pictures. Come prepared with an outline of what you want to say and rehearse it beforehand to ensure you don't go over 12 minutes.

1.8 Grading

Your grade will be calculated by adding up the scores we assign in each of the following categories:

- Is the submitted code clean, easily navigable and well documented? (7 pts)
- Can we easily reproduce results in the writeup? (7 pts)
- Write-up:
  - Abstract and introduction (4 pts)
  - Related work and literature cited (5 pts)
  - Methods and software, results, conclusions (12 pts)
- In-class presentation (10 pts)

Grades for team members who clearly contributed less effort will be scaled accordingly.

2 Project topic ideas

Below are some potential starting points for project topics. You do not have to propose one of these exact topics. You may propose a topic not on this list, or a variation on a topic in the list. If you use one of these ideas, use it as a *starting point*. You still need to look at literature and develop some of your own ideas.

1. Solving a dynamic-programming alignment problem [13] is $O(mn)$ time. But there might be a much faster way of showing that there is no alignment of X to Y that rises above a certain score. Or there might be a much faster way of getting useful bounds on the number of mismatches and gaps between X and Y. This information could, in turn, greatly reduce the number of dynamic programming cells we have to fill in. Investigate this in a realistic setting and see what you find.

2. *Streaming algorithms* [59] and *sketch data structures* (e.g. the CountMin sketch [6]) can be very appropriate tools for some problems in genomics. Think about the nature of these methods, what they're useful for, and propose a novel application.

3. Here's a specific idea involving streaming algorithms and sketch data structures: Design a streaming algorithm that takes a stream of sequence reads (e.g. as we're aligning them), builds a sketch data structure summarizing them in some way (e.g. counting number of times each length-k subsrting occurs) and use this information to detect which read nucleotides are likely to be sequencing errors. You could extend the method to additionally build a sketch of the reference genome and use this to detect which read nucleotides are likely to be variants (e.g. SNPs).

4. The reference genome is never exactly the same as the genome being sequenced. This can lead to biases, since it's "harder" to align reads that come from parts of the genome that have many differences. As reads align, though, we get more information about how the sequenced genome differs from the reference. We can use this information to "update" the reference; i.e. to insert mismatches and gaps to make the reference genome match the genome being sequenced more closely. Write software that does this. Think about how to efficiently incrementally update the indexes discussed in class. See: [48], [47]. Also see this: [12].

5. We discussed co-traversal in class, the idea of traversing a trie of the k-neighborhood of P while simultaneously traversing a suffix trie of T. This leaves the question of how to *order* the traversal. E.g. depth-first, breadth-first, best-first? This type of problem is sometimes called "combinatorial search", or just "search," and there is much literature describing various good ways of ordering the search. Read up on this and thoroughly compare a few strategies, such as Beam search [57], BLFS [45], or ILDS [20]. Implement them alongside a suffix index. You might also think about how having upper and lower score bounds might help.

6. Implement an efficient suffix-sorting algorithm. Suffix sorting could be a prelude to building a suffix-array index, an FM Index, or both. Implement one or more algorithms for this and show how they compare to the implementations in in Bowtie [23] and BWA [25]. Consider speed and memory usage.

7. Implement a suffix sorting algorithm in parallel, perhaps using Hadoop (http://hadoop.apache.org). See: [21].

8. Consider ways that the FM Index or another space-efficient index can be used for assembly. For instance, you might place all the the sequence reads in an index and then use it somehow to find exact and/or approximate overlaps between reads. See: [51], [52]. (Careful: this is a big topic.)

9. Consider ways that a suffix index or a sketch data structure such as a CountMin sketch [18] can be used for error correction. Suggest novel a novel method for error correction. See: [18], [18].

10. Consider how the suffix index approaches we studied in class could be extended to the Multiple Sequence Alignment (MSA) problem. See: [14], [8], [24]. (Careful: this is a big topic.)

11. For people comfortable with C/C++ and low-level programming: Implement a few SIMD-accelerated dynamic programming schemes and suggest situations in which one might be preferred to others. For instance, is one better for DNA alignment, as opposed to, say, protein alignment? Is one better for longer queries? See: [44], [9], [43].

12. For people comfortable with C/C++ and low-level programming: Implement a dynamic-programming "server" that takes dynamic programming problems as input and asynchronously returns a result, say, in the form of a score and a traceback. This gives the server the freedom to, for example, buffer up DP problems and solve them in "blocks." Propose some appropriate algorithms to use for this setting (look at [43]).

3 Literature Suggestions

Here are several references that should get you started in your literature survey once you have some idea of what direction you want to go in. If there's a relevant computational topic you want to learn more about that isn't represented here, email us and we'll augment the list.

- Indexing: [13], [35], [1], [17], [10], [23], [25].
- Read alignment: Bowtie: [23], Bowtie 2: [22], BWA: [25], BWA-SW: [26], GEM [36], SHRiMP: [46], GASSST [42].
- Assembly: Velvet: [61], SGA: [52], ABySS [53], (unnamed succinct approach) [5]. Assembly review articles: [49], [38], [4].
- Error correction: Quake: [18]. Musket: [33].
- Read compression: CRAM: [11], Quip: [16], NGC: [41].
- Basic dynamic programming: [13].
- Accelerating dynamic programming with special-purpose hardware. With SIMD: [44], [9], [43]. With GPUs: [50], [28], [31], [32], [29], [30], [27], [19], [34]. With FPGAs: [3]. (Before considering using a CPU or FPGA, recall that we have to be able to easily run your software! Ask us if you are in doubt.)
- Multiple sequence alignment: CLUSTAL: [24], MUSCLE: [8], MGA: [14].
- RNA sequencing and spliced alignment: TopHat: [54], Cufflinks: [55], MapSplice: [56], Spaln2: [15].
- Metagenomics: Phymm / PhymmBL [2].
- Search strategies for combinatorial search / tree search problems such as the co-traversal we discussed in class: Beam search [57], BLFS [45], or ILDS [20]. General article on tree search: [60].
- Indexing and querying collections of sequence reads: [7], [39].
- Incrementally updating suffix indexes: [48], [47].

- Streaming algorithms [59], sketch data structures (e.g. CountMin sketch [6], which is related to the Bloom Filter [58]). Applications of sketch data structures to k-mer counting [37], metagenomics assembly [40], and compression [16].
- Use of FM Index for assembly: [51], [52].

References

[1] Mohamed Ibrahim Abouelhoda, Stefan Kurtz, and Enno Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms*, 2(1):53–86, 2004.

[2] Arthur Brady and Steven L Salzberg. Phymm and phymmbl: metagenomic phylogenetic classification with interpolated markov models. *Nature methods*, 6(9):673–676, 2009.

[3] Y. Chen, B. Schmidt, and D. L. Maskell. A hybrid short read mapping accelerator. *BMC Bioinformatics*, 14(1):67, Feb 2013.

[4] Phillip EC Compeau, Pavel A Pevzner, and Glenn Tesler. How to apply de bruijn graphs to genome assembly. *Nature biotechnology*, 29(11):987–991, 2011.

[5] Thomas C Conway and Andrew J Bromage. Succinct data structures for assembling large genomes. *Bioinformatics*, 27(4):479–486, 2011.

[6] Graham Cormode and S Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[7] Anthony J Cox, Markus J Bauer, Tobias Jakobi, and Giovanna Rosone. Large-scale compression of genomic sequence databases with the burrows–wheeler transform. *Bioinformatics*, 28(11):1415–1419, 2012.

[8] Robert C Edgar. Muscle: multiple sequence alignment with high accuracy and high throughput. *Nucleic acids research*, 32(5):1792–1797, 2004.

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

[10] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 390–398. IEEE, 2000.

[11] Markus Hsi-Yang Fritz, Rasko Leinonen, Guy Cochrane, and Ewan Birney. Efficient storage of high throughput dna sequencing data using reference-based compression. *Genome research*, 21(5):734–740, 2011.

[12] Alaa Ghanayim and Dan Geiger. Iterative referencing for improving the interpretation of dna sequence data. 2013.

[13] Dan Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology.* Cambridge University Press, 1997.

[14] Michael Höhl, Stefan Kurtz, and Enno Ohlebusch. Efficient multiple genome alignment. *Bioinformatics*, 18(suppl 1):S312–S320, 2002.

[15] Hiroaki Iwata and Osamu Gotoh. Benchmarking spliced alignment programs including spaln2, an extended version of spaln that incorporates additional species-specific features. *Nucleic Acids Research*, 2012.

[16] Daniel C Jones, Walter L Ruzzo, Xinxia Peng, and Michael G Katze. Compression of next-generation sequencing reads aided by highly efficient de novo assembly. *Nucleic Acids Research*, 2012.

[17] Juha Kärkkäinen and Peter Sanders. Simple linear work suffix array construction. *Automata, Languages and Programming*, pages 187–187, 2003.

[18] David R Kelley, Michael C Schatz, and Steven L Salzberg. Quake: quality-aware detection and correction of sequencing errors. *Genome Biol*, 11(11):R116, 2010.

[19] Petr Klus, Simon Lam, Dag Lyberg, Ming S Cheung, Graham Pullan, Ian McFarlane, Giles SH Yeo, and Brian YH Lam. Barracuda-a fast short read sequence aligner using graphics processing units. *BMC research notes*, 5(1):27, 2012.

[20] Richard E Korf et al. Improved limited discrepancy search. In *Proceedings of the National Conference on Artificial Intelligence*, pages 286–291, 1996.

[21] Fabian Kulla and Peter Sanders. Scalable parallel suffix array construction. *Parallel Computing*, 33(9):605–612, 2007.

[22] Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with bowtie 2. *Nature methods*, 9(4):357–359, 2012.

[23] Ben Langmead, Cole Trapnell, Mihai Pop, Steven L Salzberg, et al. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biol*, 10(3):R25, 2009.

[24] MA Larkin, G Blackshields, NP Brown, R Chenna, PA McGettigan, H McWilliam, F Valentin, IM Wallace, A Wilm, R Lopez, et al. Clustal w and clustal x version 2.0. *Bioinformatics*, 23(21):2947–2948, 2007.

[25] Heng Li and Richard Durbin. Fast and accurate short read alignment with burrows–wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

[26] Heng Li and Richard Durbin. Fast and accurate long-read alignment with burrows–wheeler transform. *Bioinformatics*, 26(5):589–595, 2010.

[27] Chi-Man Liu, Thomas Wong, Edward Wu, Ruibang Luo, Siu-Ming Yiu, Yingrui Li, Bingqiang Wang, Chang Yu, Xiaowen Chu, Kaiyong Zhao, et al. Soap3: ultra-fast gpu-based parallel alignment tool for short reads. *Bioinformatics*, 28(6):878–879, 2012.

[28] Yongchao Liu, Douglas L Maskell, and Bertil Schmidt. Cudasw++: optimizing smith-waterman sequence database searches for cuda-enabled graphics processing units. *BMC research notes*, 2(1):73, 2009.

[29] Yongchao Liu and Bertil Schmidt. Evaluation of gpu-based seed generation for computational genomics using burrows-wheeler transform. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 684–690. IEEE, 2012.

[30] Yongchao Liu and Bertil Schmidt. Long read alignment based on maximal exact match seeds. *Bioinformatics*, 28(18):i318–i324, 2012.

[31] Yongchao Liu, Bertil Schmidt, and Douglas L Maskell. Cudasw++ 2.0: enhanced smith-waterman protein database search on cuda-enabled gpus based on simt and virtualized simd abstractions. *BMC Research Notes*, 3(1):93, 2010.

[32] Yongchao Liu, Bertil Schmidt, and Douglas L Maskell. Cushaw: a cuda compatible short read aligner to large genomes based on the burrows–wheeler transform. *Bioinformatics*, 28(14):1830–1837, 2012.

[33] Yongchao Liu, Jan Schröder, and Bertil Schmidt. Musket: a multistage k-mer spectrum-based error corrector for illumina sequence data. *Bioinformatics*, 29(3):308–315, 2013.

[34] Ruibang Luo, Thomas Wong, Jianqiao Zhu, Chi-Man Liu, Edward Wu, Lap-Kei Lee, Haoxiang Lin, Wenjuan Zhu, David W Cheung, Hing-Fung Ting, et al. Soap3-dp: Fast, accurate and sensitive gpu-based short read aligner. *arXiv preprint arXiv:1302.5507*, 2013.

[35] Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. *siam Journal on Computing*, 22(5):935–948, 1993.

[36] Santiago Marco-Sola, Michael Sammeth, Roderic Guigó, and Paolo Ribeca. The gem mapper: fast, accurate and versatile alignment by filtration. *Nature methods*, 9(12):1185–1188, 2012.

[37] Páll Melsted and Jonathan K Pritchard. Efficient counting of k-mers in dna sequences using a bloom filter. *BMC bioinformatics*, 12(1):333, 2011.

[38] Jason R Miller, Sergey Koren, and Granger Sutton. Assembly algorithms for next-generation sequencing data. *Genomics*, 95(6):315, 2010.

[39] Philippe Nicolas, Salson Mikaël, Lecroq Thierry, Léonard Martine, Commes Thérèse, and Rivals Eric. Querying large read collections in main memory: a versatile data structure. *BMC Bioinformatics*, 12.

[40] Jason Pell, Arend Hintze, Rosangela Canino-Koning, Adina Howe, James M Tiedje, and C Titus Brown. Scaling metagenome sequence assembly with probabilistic de bruijn graphs. *Proceedings of the National Academy of Sciences*, 109(33):13272–13277, 2012.

[41] Niko Popitsch and Arndt von Haeseler. Ngc: lossless and lossy compression of aligned high-throughput sequencing data. *Nucleic acids research*, 41(1):e27–e27, 2013.

[42] Guillaume Rizk and Dominique Lavenier. Gassst: global alignment short sequence search tool. *Bioinformatics*, 26(20):2534–2540, 2010.

[43] Torbjørn Rognes. Faster smith-waterman database searches with inter-sequence simd parallelisation. *BMC bioinformatics*, 12(1):221, 2011.

[44] Torbjørn Rognes and Erling Seeberg. Six-fold speed-up of smith–waterman sequence database searches using parallel processing on common microprocessors. *Bioinformatics*, 16(8):699–706, 2000.

[45] Kevin Rose, Ethan Burns, and Wheeler Ruml. Best-first search for bounded-depth trees. In *Fourth Annual Symposium on Combinatorial Search*, 2011.

[46] Stephen M Rumble, Phil Lacroute, Adrian V Dalca, Marc Fiume, Arend Sidow, and Michael Brudno. Shrimp: accurate mapping of short color-space reads. *PLoS computational biology*, 5(5):e1000386, 2009.

[47] Mikael Salson, Thierry Lecroq, Martine Léonard, and Laurent Mouchard. A four-stage algorithm for updating a burrows–wheeler transform. *Theoretical Computer Science*, 410(43):4350–4359, 2009.

[48] Mikaël Salson, Thierry Lecroq, Martine Léonard, and Laurent Mouchard. Dynamic extended suffix arrays. *Journal of Discrete Algorithms*, 8(2):241–257, 2010.

[49] Michael C Schatz, Arthur L Delcher, and Steven L Salzberg. Assembly of large genomes using second-generation sequencing. *Genome research*, 20(9):1165–1173, 2010.

[50] Michael C Schatz, Cole Trapnell, Arthur L Delcher, and Amitabh Varshney. High-throughput sequence alignment using graphics processing units. *BMC bioinformatics*, 8(1):474, 2007.

[51] Jared T Simpson and Richard Durbin. Efficient construction of an assembly string graph using the fm-index. *Bioinformatics*, 26(12):i367–i373, 2010.

[52] Jared T Simpson and Richard Durbin. Efficient de novo assembly of large genomes using compressed data structures. *Genome research*, 22(3):549–556, 2012.

[53] Jared T Simpson, Kim Wong, Shaun D Jackman, Jacqueline E Schein, Steven JM Jones, and İnanç Birol. Abyss: a parallel assembler for short read sequence data. *Genome research*, 19(6):1117–1123, 2009.

[54] Cole Trapnell, Lior Pachter, and Steven L Salzberg. Tophat: discovering splice junctions with rna-seq. *Bioinformatics*, 25(9):1105–1111, 2009.

[55] Cole Trapnell, Brian A Williams, Geo Pertea, Ali Mortazavi, Gordon Kwan, Marijke J Van Baren, Steven L Salzberg, Barbara J Wold, and Lior Pachter. Transcript assembly and quantification by rna-seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nature biotechnology*, 28(5):511–515, 2010.

[56] Kai Wang, Darshan Singh, Zheng Zeng, Stephen J Coleman, Yan Huang, Gleb L Savich, Xiaping He, Piotr Mieczkowski, Sara A Grimm, Charles M Perou, et al. Mapsplice: accurate mapping of rna-seq reads for splice junction discovery. *Nucleic acids research*, 38(18):e178–e178, 2010.

[57] Wikipedia. Beam search — wikipedia, the free encyclopedia, 2013. [Online; accessed 2-March-2013].

[58] Wikipedia. Bloom filter — wikipedia, the free encyclopedia, 2013. [Online; accessed 2-March-2013].

[59] Wikipedia. Streaming algorithm — wikipedia, the free encyclopedia, 2013. [Online; accessed 2-March-2013].

[60] Wikipedia. Tree traversal — wikipedia, the free encyclopedia, 2013. [Online; accessed 2-March-2013].

[61] Daniel R Zerbino and Ewan Birney. Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome research*, 18(5):821–829, 2008.