Project Description

CS 4390/5390
Fall 2019

Due: December 13, 2019, 23:59
Check-In Due: November 13, 2019.

Students can work in pairs or as individuals. Projects should be submitted either via the course blackboard or via email by the due date. **Late projects will not be accepted due to the proximity of the due date to the end of the semester.**

The goal of this project is to get hands-on experience with the algorithms we discussed in class as well as fully understanding the background and use for these algorithms. Along with the implemented material, students should submit a written summary of their project, which includes the relevant background for the algorithms used. This should be formulated in the style of piece of primary literature with sections for: (i) background and introduction with proper citations, (ii) methods, (iii) results, (iv) conclusions. Depending on the project chosen the length of each of these sections would be adjusted, but the total length should be between 6-10 pages including citations, figures, etc. If working in a team, this document should also include each member's individual contribution.

There are 3 major options to choose from and they are listed below. Students may also choose to self-design a project with the approval of the instructor. **In all cases students should discuss their project proposal with the instructor before they begin.**

A short check-in writeup (approx. 1 page) detailing the planned project and the current progress by 13 November 2019. These check-ins should be emailed to the instructor.

1 Implement algorithms from class

This project consists of implementing at least 5 algorithms (7 for teams of 2) from class. The subset of algorithms chosen will define the complexity needed for implementation. For each algorithm chosen it should be clear that what is implemented actually runs the algorithm rather than calling a library by displaying intermediate results in a well formatted manner (i.e. printing the Smith-Waterman dynamic programming table). The chosen algorithms should cover the breadth of the course, i.e. not just alignment, but also something from next-generation sequencing, genome alignment, motif finding, etc.

An example of this project might be to implement Smith-Waterman, Needleman-Wunsch, Edit Distance, The Match-Count algorithm from homework 1, and a k-mer based read clustering algorithm. Appropriate tunable parameters should be exposed (i.e. replacement scores, k-mer lengths, etc).

The submitted text would focus on the background of the algorithms implemented and what parameters are and are not exposed to the user, and any design decisions made. The results could contain evidence that the program works (i.e. sample output).

2 Create a Pipeline

Using some tools we discussed in class, create a pipeline (chain of tools) to complete some biological analysis. Some possible ideas are shown below. For this project you must use at least 3 tools from the literature and the pipeline itself should be of the students own creation (i.e. more than just running a preexisting pipeline).

- RNA-seq quantification (read alignment, transcript assembly, transcript quantification)
- long-read genome assembly (read correction, read assembly, read mapping)
- differential expression analysis (read mapping, differential expression, visualization)
- structural variation detection (genome alignment, SV detection, read mapping)
- phylogeny generation (sequence database search, multiple alignment, phylogeny)
- gene-tree/species tree reconciliation
- etc.

3 Implement and Update Algorithms From Class

Choose an algorithm that we discussed in class (or another related algorithm). Implement the algorithm described and make some improvement either to the resources necessary or to the running time.

Based on the complexity of the chosen algorithm, this may include implementing several related algorithms. For example, it is not appropriate to just re-implement banded Smith-Waterman, here Smith-Waterman would be the original algorithm and banded-ness would be the extension. It would be appropriate to implement a banded version of all of the pairwise alignment algorithms we saw in class (Smith-Waterman, Needleman-Wunsch, and the Gotoh).

If the algorithm discussed is already available, the modifications made should be significant.

Some examples would be

- adapt MashMap to use a universal hitting set
- implement BLAST using TensorFlow
- etc.

Grading Rubric

Of the total 20 points possible for the project, the break-down by part is as follows:

	Project		
Project Element	1	2	3
Check-In	5	5	5
Implementation			
Low level (compiles, runs, etc.)	2	2	2
Correctness	3	3	3
Write-Up			
Understanding of the algorithms used	5	3	5
Understanding of the contexts of the algorithms	1	3	1
Describes the implementation clearly	1	2	3
Demonstrates correctness	3	2	1