A project in "Projektmanagement im Softwarebereich": Implementation of LAGAN using the SeqAn library

Morouj, Bashar

Freie Universität

June 1, 2018

0.1 Chapters

1. Chapter: Introduction
2. Chapter: Project goals
3. Chapter: Analysis of risks
4. Chapter: Project phases
5. Chapter: Project structure
6. Chapter: Procedure and schedule planning
7. Chapter: Realization
8. Chapter: Project final statements and results

Chapter 1

Introduction

In modern times it has become increasingly important to analyze Genomes for many different reasons. Be it for medical reasons or just to understand evolutionary processes even more. One major tool for Genome Analysis is called the sequence alignment. A sequence alignment compares homologous parts of two or more genomes based on a scoring scheme of positive and negative values for certain alignments of single characters.

In our case we talk about a so called global sequence alignment(Figure 1.1A), which compares whole genomes. The most prominent and simple algorithm is Needleman-Wunsch[NBWD70]. Albeit prominent and simple, it is certainly not the most efficient, since every calculation needs a whole matrix of Values, while the size of the matrix corresponds to the size of both genomes, that are compared. Obviously a global sequence alignment, using Needleman-Wunsch, on two large genomes (e.g. human genome) would consume too much space and time to efficiently work with it.

Within there lies the main point of implementing LAGAN [BDC+03], an algorithm to efficiently align sequences without the downsides of Needleman-Wunsch. To counter those downsides one needs an limited area to run the alignment on, so one does not need to calculate the whole matrix. LAGAN can generate a so called "*banded global alignment*", where those bands are the mentioned limited areas. To find the areas I can limit our alignment to, without influencing the score of the alignment in a negative way, I first need to find short matches of length q in my sequences (Figure1.1B). I will call those short matches seeds from now on. Those seeds will then be used as anchors for the banded alignment by creating limited areas between those anchors and around them(Figure1.1D). Those small windows are being used as the limited areas to run Needleman-Wunsch on to calculate the global alignment. LAGAN was published 2013 by Michael Brudo, Chuong B. Do, Gregory M. Cooper, et al. in 2003.

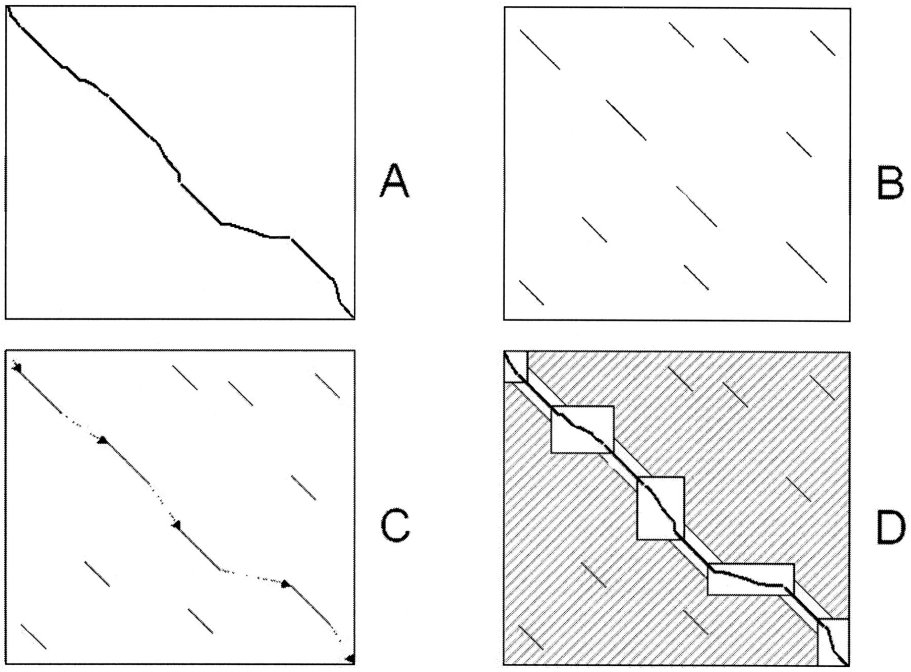

Figure 1.1: Rough sketch of how LAGAN works.

Chapter 2

Project goals

The goals of the project can be classified into one obligatory and two optional ones. The mandatory goal is implementing LAGAN that chains its seeds simply by their distance to each other. The user should be able to select two FASTA files that contain a genomic sequence and select the length q of the seeds in the command line. This simple version of LAGAN will produce a global banded sequence alignment faster than the ordinary Needleman-Wunsch.

The first optional goal is implementing LAGAN with the CHAOS[BCG+03] algorithm to chain the seeds. CHAOS is a sensitive algorithm that detects local alignments and finds short inexact words, instead of long exact ones.

The last optional goal would complete LAGAN to be the originally intended LAGAN algorithm where the CHAOS algorithm is recursively applied to the areas between found seeds with relaxed parameters like shorter seeds. The idea behind it is to find long, well conserved words and then shorter inexact ones.

Every accomplished goal will enable the user to align large genomes in a way shorter time than the ordinary global sequence alignment.
For this project I will use the SeqAn[KRW17] library. SeqAn is an open source C++ library of various algorithms for analyzing sequences with an emphasis on biological data.

The deadline of this project is the 31.05.2018 and has to be presented on the first of June 2018.

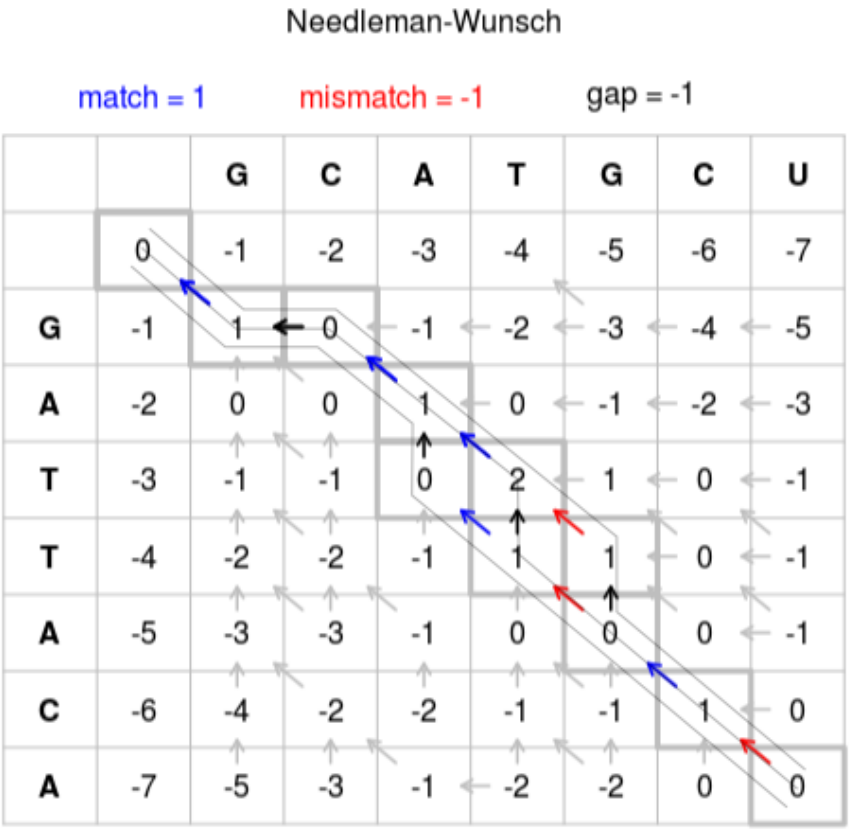

Figure 2.1: An example of backtracing after Needleman-Wunsch was calculated on two example sequences. The backtracing allows to generate the alignment based on the matrix values.

Chapter 3

Analysis of risks

3.1 Data loss

A risk was the loss of data, through loss or damage of my computer. To keep this risk at minimum I used the Github repository, www.github.com/snmember, to store updated versions of my code. In fact, the case of data loss didn't occur.

3.2 Problems with Seqan

Another risk posed problems with SeqAn since I already experienced a problem because of a conflicting Python package in my SeqAn path. To keep this risk at minimum damage I will ask the SeqAn experts, if a problem should appear.

3.3 Bad time management

Bad time management could threaten the course of the project. As countermeasures I planned the weekends and the entire last week before the deadline of the project as buffer time.

Chapter 4

Project phases

Phase	Description	Timespan	Milestone
1	Research	26.04. - 31.05.	constant research on the subject for enabling better understanding of LAGAN, the SeqAn library and C++ itself.
2	SeqAn Argument Parser	01.05. - 04.05.	Enabling the user of LAGAN to choose sequences and seed length for the alignment. If the user does not choose a seed length then a default value will be generated
3	q-gram-index building	01.05. - 04.05.	At this point the program calculates hash values for every q-mer in both sequences and store them in tupels of hash value and starting position of the q-mer.
4	Generating seeds	07.05. - 11.05.	LAGAN can find short matches through our q-gram tupel list and creates seeds accordingly.
5	Banded alignment	14.05. - 22.05.	The final stage of programming is completed, when LAGAN calculates a global sequence alignment based on the seeds and returns the alignment and score.

Chapter 5

Project structure

1. Research
   (a) understanding LAGAN
   (b) learning q-gram function in SeqAn
   (c) learning the SeqAn Argument Parser
   (d) understanding the SeqAn Iterator class
   (e) learning the Seed class and Seed extension algorithm in SeqAn
   (f) learning how to perform a banded sequence alignment in SeqAn
   (g) finding FASTA files to test the program on
2. SeqAn Argument Parser
   (a) parsing paths from the command line
   (b) parsing seed length from command line
   (c) turning seed length into an option and generating default value
3. q-gram-index building
   (a) implementing tuples of hash value and starting position
   (b) implementing list of said tuples
   (c) building a SeqAn Iterator
   (d) iterate over sequences to calculate the q-gram-index and fill the list of tuples.
4. Generating seeds
   (a) sorting list of tuples based on hash value
   (b) implementing SeqAn Iterators for both lists
   (c) implementing third SeqAn Iterator as support to calculate cross products in both lists.
   (d) finding matches, generating seeds, extending them and add them into a seed set

5. Banded alignment

   (a) chain the found seeds

   (b) creating a banded alignment

   (c) writing the alignment into a FASTA file.

   (d) testing

5.1 Research

The research phase has the most work packages since it is the longest running phase and essential for every work package in the other phases. Many of them involve the understanding of the SeqAn library, since I will implement LAGAN strictly by using SeqAn. The research phase also involves understanding LAGAN and reading its paper to know which steps to take for the program.

The final part of the research phase is searching for genomes in the Internet to test LAGAN on and eventually modify the program.

5.2 SeqAn Argument Parser

The work packages in this phase are rather straight forward and consist of implementing LAGAN, so it needs two arguments(the paths to the sequences) and an option, the seed length.

5.3 Building q-gram-index

To find common q-mers as fast as possible, and thus are able to generate seeds, a q-gram-index is supposed to be build on the two sequences. A q-gram index is a hash-value calculated based on the length of the chosen seed length (namely the q) and the given alphabet. So by calculating those, I can easily find short matches without comparing each character. Instead I just compare hash values.
To calculate such hash values on every q-mer in the two sequences I need to create two SeqAn Iterators to run the calculation in a loop on these sequences.

5.4 Generation of seeds

To generate the seeds as fast as possible, I will sort the list of tuples ascendantly based on their hash values. I will then build two SeqAn Iterators and point them to one of these lists respectively. The Iterators will then compare hash values and move the Iterator, which points to a smaller value, to the next element. In case of a cross product, a third Iterator will save the position of the first match in the second list and moves the second Iterator back to this particular position, while the first moves on with each match until they unmatch. In case of a match I will write a function to generate the seed, extending it in both directions by searching for matching characters and then adding it to a set of seeds.

5.5 Chaining seeds and aligning the sequence

SEQAN offers algorithms to chain seeds in a seed set and calculate a banded alignment based on them.
The other two work packages consist of writing a function to output an alignment into a FASTA file and repeatedly using LAGAN on sequences I found in my researches.

5.6 Report and presentation

The final step will be the analyzing of my results, writing of my report and preparation for the presentation on June the first.

Chapter 6

Procedure and schedule planning

Figure 6.1: The research phase is the only phase that is ongoing in every other phase of the project. phases that are parallel to each other can be realized without the other one but every phase that is scheduled after another one cannot be started before the previous phase finished.
The critical path are the red bars that are exactly set one day before the start of a new phase to ensure the schedule is being followed. The last red bar ends at 29.05. and is the deadline for programming so I can solely focus on my report and presentation.

Chapter 7

Realization

During my work on the SeqAn tutorials I already finished the function for building a q-gram-index on the two sequences. The program had no SEQAN Argument parser, but it worked on two sequences that were read by the program.

On Monday, the 01.05., I met up with Jörg Winkler to discuss the start of my project and what I already built into the program. He showed me the use of SEQAN Iterators and how to sort my String of q-gram-indices. That week I built two Iterators into my program that would point to the hash values in both strings and compare them. I still did not build the SEQAN Argument Parser and I noticed my Iterators do not take cross products as various matches into account. That could be resolved in the next meeting with Jörg Winkler on Monday, the 07.05., when he gave me advice on how to approach the Iterator problem.

Following that I was able to build the SeqAn Argument parser and the tools for the banded alignment into my program, but was unable to use the parsed seed length into my q-gram-index or align my sequences. When I asked to Jörg Winkler for help he pointed out that I was using an ungappedShape instead of simpleShape. The problem was that ungappedShape needs the length of q at compile time and could not be changed by an argument in the command line.

The next problem with the alignment was because I was running LAGAN on debug mode and some exception was hindering the progression of the program. It could be resolved by running LAGAN without debug mode. From then on I was able to test FASTA files like the Ebola Virus or *E.Coli* and noticed that the program did not work if the second FASTA file is longer than the first one.
On Thursday the 15th Jörg found out it was a fault in my while-loop that was corrected by him. From then on LAGAN worked well with the previously mentioned files and additional ones like streptococcal genomes or Salmonella.

While testing I tried various seed lengths for the sequences and noted down the running time for performance analyzing purposes, which I will explain further in my results.

The next meeting on Thursday the 22th I was discussing with Jörg Winkler on how to write my report and he gave me advice on what kind of cases I should test with my program. We found out my LAGAN returns a segmentation fault, in case my program does not find seeds. Following that I changed my LAGAN program by adding a conventional global alignment if there are no seeds to be found.

Additionally I was able to generate the seeds using CHAOS and fulfilled one of the two optional goals along with the obligatory.

Chapter 8

Project final statements and results

8.1 Results

I tested my LAGAN program on multiple sequences from *Echerichia Coli*, *Streptococcus agalactiae*, *Streptococcus pneumoniae*, *Salmonella enterica serovar Paratyphi*, *Salmonella entrica arizonae serovar*, *Bundibugyo Ebolavirus*, *Zaire Ebolavirus*, *Drosophila simulans*, *Drosophila melanogaster*, *Drosophila miranda*, *Drosophila busckii*, *Fusarium venenatum*, *Talaromyces pinophilus* and some FASTA file sequence examples from the Internet.

Then I tested the ordinary global sequence alignment from SeqAn on the Ebola viruses and on the streptococcal genomes to compare the performances to my LAGAN project. The reason I did the ordinary alignment on these sequences is because I did the runtime analysis of LAGAN primarily on Ebola and Streptococcus Genomes.

The first downside of just Needleman-Wunsch was it's slow runtime, LAGAN in fact proved to be faster since it aligned the Ebola viruses at most seed lengths in much less time than the ordinary alignment. At $q = 10$, where LAGAN was at it's fastest (Figure 8.1), the program just needed approximately 100 milliseconds while the normal alignment needed up to 2.3 seconds. The only seed lengths where LAGAN was slower where from 7 down (at 7 LAGAN needed 8 seconds)

The other downside of space consumption became more apparent in the test on the streptococcal Genomes than the time consumption. LAGAN could align those genomes that are larger then $2 * 10^6$ (Figure8.2), while the ordinary sequence alignment returned `std::bad_alloc`. `std::bad_alloc` is an exception that is thrown by C++ if the program tries to generate storage but can not because the limit is reached. That means LAGAN could align two genomes, which would be too large to align normally.

When testing The first Chromosome of the above named Drosophila and the first Chromosome of *Fusarium venenatum* and *Talaromyces pinophilus*, even after 20 hours the program did not terminate, no matter the seed length. So unfortunately I could not test genomes larger than 10^7. The project itself can be regarded as a success since my program prove to be faster and more efficiently in terms of storage. A future user can select two sequences and a seed length and align larger genomes in little time.

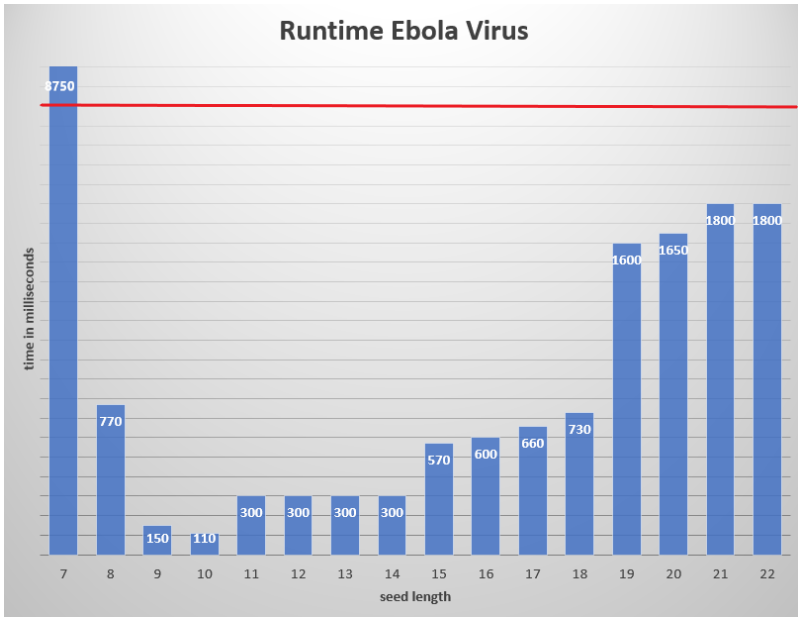

Figure 8.1: A graph of the runtime of LAGAN on *Zaire Ebolavirus* and *Bundibugyo Ebolavirus* . Both Genomes are slightly larger than $1,8 * 10^4$ The minimum is at a seed length of 10. Longer q are slower but still faster than the conventional Needleman-Wunsch (the red line) since they need less than 2 seconds to align the Genomes and still return the same score as the complete Needleman-Wunsch.
The reason why longer seeds make the algorithm slower could be that longer seeds statistically mean less seeds. So then the smaller windows between our "anchors" grow larger and there is much more calculating to do in comparison to shorter seeds. Small q like 7 and under could have influenced the running time negatively because of many seeds that are generated, extended, ran CHAOS on and then chained. The program could have been slowed down because of those operations.

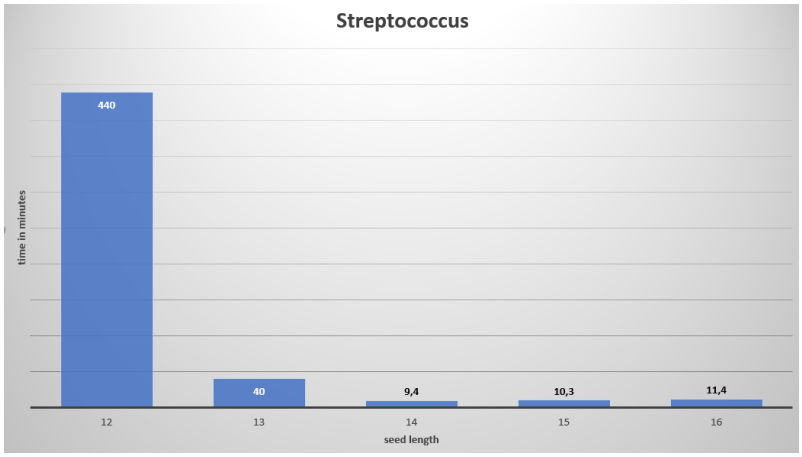

Figure 8.2: Similar to the test on the Ebola Virus short q mean a longer running time for probably the same reasons. For example $q = 13$ resulted in a runtime of 40 minutes, a seed length of 12 needed even 7 hours. Seeds larger than 17 were tested but there were no seeds of that length to be found. The minimal running time lies at a seed length of 14 with 9 minutes, which can be considered great improvement from being unable to align genomes to such a practicable time.

8.2 Project final statements

Even though there were some hurdles in my project, since I do not have much experience as a programmer and even less in project management, I really enjoyed this experience. I learned much about C++, SeqAn and myself. Every mistake and unreached goal in this project taught me a lesson on how to approach larger projects and how to plan even more in detail to ease certain work packages.

I still feel a sense of accomplishment, because I was able to fulfill one of the optional goals and could align sequences faster than the "normal" global sequence alignment and even align two sequences, that would have been too large to align ordinarily. This project furthered my interest for sequence analysis and hopefully I will find the time and motivation to modify my LAGAN program even more in the future.

Bibliography

[BCG+03] Michael Brudno, Michael Chapman, Berthold Gottgens, Serafim Batzoglou, and Burkhard Morgenstern. Fast and sensitive multiple alignment of long genomic sequences. *BMC Bioinformatics*, 4(66), 2003.

[BDC+03] Michael Brudno, Chuong Do, Gregory Cooper, Michael F. Kim, Eugene Davydov, Eric D. Green, Arend Sidow, and Serafim Batzoglou. Lagan and multi-lagan: efficient tools for large-scale multiple alignment of genomic dna. *Genome Research*, 13(4):721–31, 2003.

[KRW17] M. Ehrhardt H. Hauswedell S. Mehringer R. Rahn J. Kim C. Pockrandt J. Winkler E. Siragusa G. Urgese K. Reinert, T. H. Dadi and D. Weese. The SeqAn C++ template library for efficient sequence analysis: a resource for programmers. *Journal of biotechnology*, 261:157–168, 2017.

[NBWD70] Needleman, Saul B., Wunsch, and Christian D. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–53, 1970.