Visualization of co-assembly string graphs

A proposed MSc thesis by Max Jonkman (reg. no. 890714412120), 21-02-2014

Summary
After obtaining a reference genome, the next step would be to create a pan-genome that encompasses all the genetic variation available within a species. Among others, co-assembly can be used to create a pan-genome. Co-assembly is a method to assemble the sequence of multiple (related) samples at the same time. The necessity of a reference genome is gone with this method, and it creates new possibilities in regard to the comparison of the assembled sequences (Nijkamp, 2012).

Computationally, pan-genome analysis can be done using variant calling software. However, this produces an enormous list of variants. To help computers interpret the results, human pattern recognition via a visual tool is a helpful addition.

To help the analysis of a pan-genome, a browser tool will be developed that will visualize the data and helps pattern recognition. This tool, called Stringit (string graph imaging tool) will be built in javascript, and be made available online. The overall goal will be to make co-assembly analysis easier, something that synteny browsers already achieve in the case of 'normal' assembly analyses.

Delineation of the problem
The research in this group is mainly focussed on development and application of methods to use –omics data more efficient in agricultural and food science. Comparison between new and known genome sequences is done a lot, and with more and more species sequenced, these sequences continually have a great deal in common. Using co-assembly to create a pan-genome is a good way to find the similarities and variation between multiple similar strains.

Main goal: The production of a pan-genome visualisation tool that helps the analysis of co-assembly data.

Main research question: What is the best way to visualize the data produced by a co-assembly, so that it is most easily analysed?

Subquestion: how does a co-assembler work, and what sort of data is produced? What are the steps the underlying algorithm takes in regard to de bruijn graphs?

Subquestion: What is the best way to visualize, manipulate, and present this sort of data?

Subquestion: What are the limitations of similar, already existing, tools?

Subquestion: What are, next to the visualization of co-assembly data, other functionalities that are requisited for a broad adaptation to the use of Stringit?

Scientific background
In short, a genome is sequenced in the following way. A sequencing machine takes DNA as input, fragmented in small pieces. It produces a text file of the sequence of each of those pieces. Dedicated assembler programs take all these files (sometimes several millions), and stitch them together via overlap to create a genomic sequence. This is a long and memory-intensive process that takes a lot of processing power. To reduce the resources these calculations take and the time this process costs, modern assemblers use an algorithm that makes use of graph theory.

Most current assemblers use de bruijn string graphs to process the reads while in the process of assembling. It does this by first breaking up the reads in even smaller overlapping, k-mers. These usually have a length of somewhere around 25-31. Then it looks through all these k-mers and start noticing overlaps. Not only will it find overlaps between k-mers that originate from the same read, but also from different reads. These will also be grouped, and the combined read will be stored as a graph (Compeau, 2011).

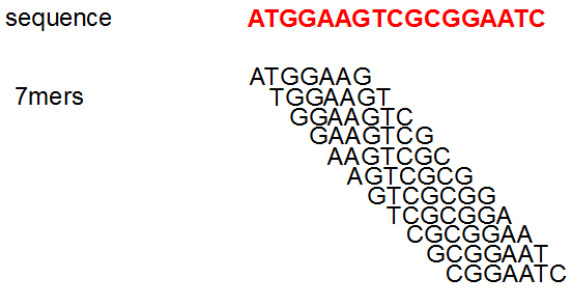

Figure 1: k-mers, with k=7. Picture adapted from http://www.homolog.us/blogs/blog/2011/07/28/de-bruijn-graphs-i/

In such a graph, the nodes represent overlaps between k-mers, and the edges represent the differences between them. In an ideal case, only a single node is formed, averting the need for such an approach. However, biological imperfections of the sample, as well as technological artifacts, will introduce small errors. Luckily, each bit of sequence is covered a multitude of times (general coverages go between 30 and 100). This makes sure that when there is an error that introduces an edge between two nodes, the error sequence is covered less than the correct sequence. The assembler program chooses a consensus from the paths with higher coverages when it travels through the graph.

There are a lot of different assemblers currently available. While most of them work as outlined above, they may differ on the exact execution of it. Only a few are capable of co-assembly, and some are better with smaller genomes than with larger. Via competitions such as GAGE(-B) and Assemblathon, there have been efforts to objectively compare a variety of different assemblers. The general consensus of these comparisons was that there is a broad range of metric that assemblers need to be good at in order to be considered good. As such, there is not a single 'best' assembler, and a researcher must do his best to pick a suitable one to his specific samples (Bradnam et al., 2013). A few of the metrics that are usually used in the judgment of assemblers are the size of the contigs that are produced, and paired with that, the N50. N50 is the size of a contig at which all contigs of that size and larger contain at least 50% of the assembled sequence. The larger that number, and the lower the number of contigs, the better the data.

It is virtually impossible that only a single contig is produced as output. DNA contains a lot of repetitive elements, which make it hard for the assembler to decide which edge is the correct one to take. To alleviate this problem, the already assembled sequence of a related species can be used alongside the current one, if one is available. The reference sequence can then be used as a guide, with which the assembler can then again pick the right path.

For the creation of a pan-genome, the use of a reference genome is not possible. A pan-genome is a genome sequence that accounts for all the possible genetic variation within (part of) the species. A reference genome is merely a chosen single sequence, which does not account for this variation. Using a reference genome when creating a pan-genome is therefore not possible, because the reference will rule out any aberrant data as technical variation, instead of as biological variation. This is especially bad when biological variants is specifically what is being researched (Nijkamp, 2012). Therefore, a lot of effort is put into optimizing *de novo* assembly, which does not require a reference sequence.

For the process of creating and analysing a pan-genome, the assembly process needs to be slightly altered compared to described above. And as already described, a reference genome cannot be used. An efficient way to combine both these limitations is the use of a co-assembly. In a co-assembly, two or more samples are assembled concurrently. In this way, they can be used as a guide or control for each other, and more biological variation is more easily spotted. The way this is done with the de bruijn graphs, is that each sample is assigned a 'colour'. Then, when traversing the graph, the colour is used as a route. With this, differences in sequence get directly mapped to different samples (Nijkamp, 2012).

Different assembly programs produce different file formats in which this mapping is exported. Additionally, graph files already have a lot of different file extensions associated with them. In regard to the tool that will be developed, several different options for input and output file formats are viable.

Ideally, multiple input formats are accepted, and output can be produced in a file format that is usable for further research. There are other tools available for research on assembled sequence data. Synteny browsers such as Strudel or Symap serve a different purpose, and will most likely be used Stringit. To make the transition between tools as easy as possible, using the same file formats whenever possible is preferable. Other tools such as pan-genome profile analysis tool PanGP are also used in this field, and also need to be considered.

In short, Stringit needs to be able to convert the (binary) graph files produced by assemblers into something universally readable, after which the visualisation can take place. Additionally, exportation into other useful file formats need to be considered.

Approach
Together with this document, a Logical Framework is attached (Appendix 1). It documents the here proposed activities in a concise form.

Output
A web-based tool will be developed in javascript, for visualisation of co-assembly data. Specifically, the visualisation of colored de bruijn graphs will be made. All its functions will be documented in an accompanying report. This report will also contain a study on related tools for the field, what their limitations are, and how they differ from this visualisation tool. While no final name has been chosen, the tool shall be called Stringit, short for "string graph imaging tool".

Activities
Literature will be read on a variety of related topics: (co-)assembly, pan-genome analysis, javascript tool development, data visualisation, and any article that will be discussed during group meetings.

Meetings with the supervisor (Dick de Ridder) will be conducted first weekly, and later bi-weekly. These meetings will be used as a check on progress, and to remove any hurdles the project might experience. During group meetings, the progress on Stringit will be regularly reported on. These updates consist both of a quick recap of done work, as well as a bigger presentation on the progress of this project.

Javascript will be used to develop Stringit, and a website will be launched where it will be hosted. In this stage, the following parts have been identified as being necessary for Stringit:

1. Conversion between different file formats; for easy import and export of data, so that integration with other tools is not a limiting factor for analysis.
2. An intuitive interface; pattern recognition is easiest if the string graph data is presented in a natural form.
3. Different zoom levels; the research possible with Stringit ranges from genome-wide to sequence-specific.

Other parts than these might be added to Stringit. In this category fall functionalities like annotation tools and extended export functions. However, as of yet it is unclear whether these functions are necessary, or if there is enough time to implement them.

All these activities will be digitally logged, so that progress can be easily checked.

Necessities
For the completion of this project, a few things need to be available:

1. Access to the server for the creation of co-assembly data.
2. Lectures on data handling and pipelines from the course Advanced Bioinformatics (BIF-30806).
3. Webhosting for the online implementation of the tool.

Summary
The following parts will be delivered in the timeframe of this project:

1. A functional website with a tool for browsing co-assembly string graphs (Stringit)
2. An in-between progress update in a group meeting
3. A final presentation on the project
4. A report detailing all the work that is done on Stringit, as well as a review of relevant literature.
5. A log diary with notes that maps the process of the development of the project

Final report
The final documents of the project will be delivered in the form of a thesis report. It will include sections on the following subjects:

1. Documentation on the workings of Stringit
2. A review of other relevant and related pan-genome and co-assembly analysis tools
3. Scientific background on the subject

External contacts
The use of external contants is largely unnecessary. There have been talks with other people from the group about the technical workings of similar MUMmer-based genome browser, but it is unlikely that that tool will be used in the development of Stringit. Its creator is possibly a valuable source of information.

Literature
How to apply de Bruijn graphs to genome assembly. Compeau, Pevzner, Tesler. 2011
http://www.nature.com/nbt/journal/v29/n11/full/nbt.2023.html

Strudel Comparative Map Viewer, Bayer et al.
http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3077070/

SyMAP: A system for discovering and viewing syntenic regions of FPC maps. Soderlund et al., 2006
http://genome.cshlp.org/content/16/9/1159.full.pdf

De novo detection of copy number variation by co-assembly. Nijkamp et al., 2012
http://bioinformatics.oxfordjournals.org/content/28/24/3195.long

Assemblathon 2: evaluating *de novo* methods of genome assembly in three vertebrate species. Bradnam et al., 2013
http://www.gigasciencejournal.com/content/pdf/2047-217X-2-10.pdf