Titel

Summary (1)

visualisatie moet zo.

Index

Table of Contents

[Summary (1) 2](#__RefHeading__834_983993461)

[Index 3](#__RefHeading__836_983993461)

[Introductie (4) 4](#__RefHeading__838_983993461)

[Materials & Methods (3) 6](#__RefHeading__840_983993461)

[Overview 6](#__RefHeading__842_983993461)

[Stringit File Format 6](#__RefHeading__844_983993461)

[Functions 6](#__RefHeading__846_983993461)

[Datasets 6](#__RefHeading__848_983993461)

[Results & discussion (5) 8](#__RefHeading__850_983993461)

[Results 8](#__RefHeading__852_983993461)

[Discussion 8](#__RefHeading__854_983993461)

[Conclusion (1-2) 9](#__RefHeading__856_983993461)

[Future work (2) 10](#__RefHeading__858_983993461)

[References (1-2) 11](#__RefHeading__860_983993461)

[Appendix A 12](#__RefHeading__862_983993461)

Introduction (4)

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 www.homolog.us/blogs/blog/2011/07/28/debruijn-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 <gage paper> and Assemblathon <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.

Materials & Methods (3)

Overview

Stringit is a tool for visualizing co-assemblies. It is a web-based tool, built with javascript and the d3.js library. Its core tasks consist of the following:

* parsing (co-)assembler output
* reading the parsed data and determine underlying structures
* display the graph, accounting for the structures

The first part of these tasks is performed by external scripts, written in Python. They read the data coming from an AMOS databank or a 454 library. Then, they write the relevant data into a unified format that can then be loaded into Stringit itself.

Aside from native javascript code, the d3.js (Data Driven Documents) library is used. This library is used to bind data to DOM elements. Additionally, it provides visualization tools. In Stringit, a force-directed layout is used to display the network, and piecharts are used to provide more information in the node itself.

Stringit File Format

Before Stringit can use the provided data, it needs to be parsed. Different sources each provide a different syntax

about relevant information and syntax

Functions

readFile

handleContent

groepering

determineTiers

contains

matchcriteria

makeGraaf

radius

gravity

coordinates

Datasets

To test the use of Stringit during development, datasets from different sources were used. Two datasets that were used in <Marigold paper> were downloaded, and relevant data about contigs, reads, and edges was extracted with the AMOS bankreport function. The format of this report was used to create the Python script for conversion from AMOS to a format that could be read by Stringit.

An artificial dataset was created as well. Reads were simulated from the known e. coli and e. albertii genomic sequence, and pooled together. These pooled reads were then assembled with the 454 package (“Newbler”). After parsing with the right Python script, this data could be imported to Stringit.

In previous versions, several other datasets were used. Most notably a graph file in the asqg format, used by the SGA assembler <bron>. However, as more functions were added, support for this format was dropped, as it did not contain all the necessary information. The asqg format only contains information on the assembled reads, and not on the origin of the reads. Without an external read mapper available, Stringit no longer had all the information it needed.

Results & discussion (5)

Results

* “Stringit’s acceptance of multiple input formats, and use of a universally-accepted output format, makes it widely accessible, and unique in the field”
  + Features: different inputs and handling of them, exporting into .dot
  + Test: Compare my input/output system with that of other tools
  + In-depth: file formats, and their layout and uses (SAM/dot/asqg/454/others)
  + Answered research question: What is the best way to manipulate and present this sort of data?

* “Stringit has an intuitive visual representation for easy access to all information (that is provided by nodes)”
  + Features: node HUD, node sizes
  + Test: compare design with other tools
  + In-depth: How does the visual style of Stringit (and d3) benefit the user?
  + Answered research question: What is the best way to manipulate and present this sort of data?
* “The zooming feature of Stringit allows for both a grand overview and a detailed in-depth analysis for a full and unrivaled view of all the data”
  + Features: zooming
  + Test: user tests.
  + In-depth: Are user expectations of the functionalities of Stringit fulfilled?
  + Answered research question: What are other functionalities that are requisited for the functioning of Stringit?
* “Grouping the contig nodes based on sample read mapping readily shows regions of overlapping and/or diverging sequence”
* “Combining (co-)assembly and read map data into a single visualization is something that is not provided by other tools”
  + Features: data-translating code, coloring of nodes based on mapped reads (future: highlighting of co-mapped reads from different samples?)
  + Test: use cases. Show how easy it is to find regions of interest, compared to other tools
  + In-depth: niche of tool. When to use this, and when to use other tools.
    - (HP) diploid genomes/co-assemblies
    - Not: synteny, msa, read mapper?
  + Answered research question: What are the limitations of similar tools?

Discussion

requirements of good visualisation

how far are different tools already

Conclusion (1-2)

“How is co-assembly data best visualized, so that it is suitable for analysis?”

Future work (2)

future work (methods):

displaying single tier

changing graph based on clicks

displaying information based on selected node

how far is Stringit from the discussed goal

rest of the research questions

what needs to be done to get there

References (1-2)

graphs and string graph theory

http://www.nature.com/ng/journal/v44/n2/full/ng.1028.html#supplementary-information

http://alexbowe.com/succinct-debruijn-graphs/

http://arxiv.org/pdf/1307.7925v1.pdf

http://www.homolog.us/blogs/blog/category/de-bruijn/

http://www.cs.toronto.edu/~brudno/csc2427/myers.pdf

http://bioinformatics.oxfordjournals.org/content/26/12/i367.full.pdf

assembler bronnen

http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2874646/?tool=pubmed

http://contig.wordpress.com/2010/06/10/running-newbler-de-novo-assembly/

<http://gage.cbcb.umd.edu/>

<http://bioinformatics.oxfordjournals.org/content/early/2013/06/05>

[bioinformatics.btt273.full.pdf](http://bioinformatics.oxfordjournals.org/content/early/2013/06/05/bioinformatics.btt273.full.pdf)

http://www.gigasciencejournal.com/content/pdf/2047-217X-2-10.pdf

http://cortexassembler.sourceforge.net/cortex\_var\_user\_manual.pdf

Assemblers

http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3290790/pdf/549.pdf

http://www.cs.utoronto.ca/~brudno/csc2431w10/abyss\_explore.pdf

http://seqanswers.com/forums/showthread.php?t=16695

http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2336801/

http://seqanswers.com/wiki/Velvet

http://genome.cshlp.org/content/18/5/821.full.pdf

general sources

http://www.ncbi.nlm.nih.gov/pubmed/23047563

http://bioinformatics.oxfordjournals.org/content/29/22/2826.long

http://www.nature.com/nmeth/journal/v8/n1/pdf/nmeth.1527.pdf

<http://bioinformaticsonline.com/blog/view/4574/tools-to-detect-synteny-blocks-regions->among-multiple-genomes

<http://www.genomeweb.com/sequencing/start-building-human-pan-genome-bgi-de-novo->assembles-two-genomes-illumina-data

<http://www.plosone.org/article/info>\%3Adoi\%2F10.1371\%2Fjournal.pone.0068731\#pone- 0068731-g006

other tools

https://usegalaxy.org/u/dan/p/maf

http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3031037

http://www.bcgsc.ca/platform/bioinfo/software/sam

Appendix A

code en zo