Titel

Summary (1)

lots of data produced, visualization and analysis lacking

when there is no reference available, how is quality judged?

“There is increasing recognition in the field that improvements in visualization tools will be essential for understanding our growing wealth of data.”

but what makes for a good visualization?

Needs to be multiscale

easy to navigate despite scale of (pan)genome

simplify data or leave out where applicable

multiple input formats

new technologies makes research easier. We can now generate a lot of data-translating

however

analysis can't keep up with data

visualizing the finished work is essential for efficient analysis

properties of data:

prerequisites of visualization because of that

Index

Table of Contents

[Summary (1) 2](#__RefHeading__834_983993461)

[Index 3](#__RefHeading__836_983993461)

[Table of Contents 3](#__RefHeading__359_2135839423)

[Introduction (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 7](#__RefHeading__848_983993461)

[Results & discussion (4) 8](#__RefHeading__850_983993461)

[Results 8](#__RefHeading__852_983993461)

[Discussion 9](#__RefHeading__854_983993461)

[Conclusion (2) 9](#__RefHeading__856_983993461)

[Future work (3) 10](#__RefHeading__858_983993461)

[References (1) 11](#__RefHeading__860_983993461)

[Appendix A 13](#__RefHeading__862_983993461)

Introduction (4)

Assembly

To generate a genome sequence, a biological sample is fed into a sequencing machine. These machines can function in different specific ways, but regardless of that, it will not provide a single data file with the full sequence. Instead, a long list of small files is created, each containing a small part of the sequence. The quality of each of these reads is not perfect, and they can contain random errors. To counter these errors, redundancy is built in. The end result is a big set of reads with overlapping sequences.

^meer

De bruijn graph

Assembler programs use sophisticated algorithms to turn these reads into a larger sequence. An assembler such as SGA <bron> uses de bruijn string graphs to connect overlapping reads into a bigger sequence. The assembler first breaks up the reads in even smaller overlapping parts, called k-mers. These usually have a length of somewhere around 25-31 basepairs (bp). It iterates through all these k-mers and starts noticing overlap. Not only will it find overlap 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). This way, overlaps between different reads can be easily found, taken into consideration the size of the dataset.

^meer

<plaatje met overlapping k-mers [www.homolog.us/blogs/blog/2011/07/28/debruijn-graphs-i/](http://www.homolog.us/blogs/blog/2011/07/28/debruijn-graphs-i/)>

In the ideal case, this graph consists of only a single node, comprising of all overlapping k-mers, representing the full genomic sequence. However, biological imperfections of the sample, as well as technological artifacts will introduce small errors, which create new nodes and edges between them. Each edge represents a diverging path between two possible sequences. The redundancy in the reads is used to determine which of the possibilities is the correct one, as random errors will be represented less. The algorithm traverses the graph and removes the path that it considers an error.

After this process, overlapping nodes are condensed into a contiguous sequence (a ‘contig’). Then, a reference sequence of a related species is used as a scaffold, to place each contig roughly in the right place of the genome. This removes even more edges, because the known reference sequence provides additional information on the placement of neighbouring contigs, giving more certainty to the resulting sequence.

Current developments in technology provide researchers with a broader set of tools than before. These can be used to deal with the problems that arise when dealing with newly sequenced organisms.Whena new species is sequenced, a suitably close related reference sequence might not be available.

Pan-genome

Another newer branch of research is the creation of a pan-genome. Where a reference sequence consists of a single static sequence, a pan-genome of a specific (group of) species can account for all the variation that is available within that set. Comparing an assembled contig with a reference sequence can filter out some unusual or rare biological variation, so a pan-genome refrains from using a reference sequence, thereby avoiding skewing the results. A lot of effort is put into optimizing *de novo* assembly, to be able to deliver high-quality sequences without the use of a reference sequence.

^meer

Co-assembly

Creating a pan-genome is usually done with a specific type of *de novo* assembly, called 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. In the de bruijn graph representing this dataset, each sample is assigned a 'colour'. Then, when traversing the graph, the colour is used as a route. With this, differences in sequence coming from different samples can be easily identified (Nijkamp, 2012). Analysis of a pan-genome lets us discover more different genotype variants, which would be useful in cancer research (Lawrence, 2013), and it can show us which genes are carried on throughout evolution. (Lefébure, 2007)

The field of co-assembly is very young <bron over eerste vermelding/publicatie/tool>, and not a lot of tools have yet been developed specifically for it. There are no widely adopted best practices, as is evident from the variety of available data formats to store large amounts of sequencing data, such as SAM, BAM, vcf, and ace <meerdere bronnen>. There are several tools in existence that are capable of analysing pan-genomes *<PanGP tool, PanFunPro, PGAT>*. However, these tools are created and used to review statistics about the data, and do not care about visualization thereof.

^meer over tools

requirements

To make a functional visualisation of a co-assembly, a few requirements need to be met. As long as the community doesn't settle on a standard for data storage, the tool needs to work with a variety of input formats. The tool needs to be able to show the full extend of the data and be able to show small details. On the other hand, it needs to prevent 'getting lost' and losing the general overview of the data. Follow-up analysis needs to be possible, so data should be able to be exported to a broad set of analysis tools, such as BLAST or comparative genome analysis tools <bronnenX>. Here, suggestions are done to create a tool that satisfies these requirements.

Materials & Methods (3)

Overview

Stringit is a web-based tool tool for visualizing co-assemblies. It is 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. Some sources provide more information than is necessary for Stringit. To consolidate these differences, several Python scripts have been developed to read this data and present it in a uniform way.

The data necessary for Stringit consist of all the calculated contigs, their connections to other contigs, and a mapping of all the different reads to the contigs. Additionally, the number (and name) of the sample sources need to be known.

The scripts process all the data in their various formats, and store the information on all the contigs and links in a text file. From this text file, a basic network can already be built. Each line either represents a node or an edge. A node consists of all the information contained in the contig: its sequence, its name, and a count of how many reads from each origin map to this contig. An edge represents both the source and the target, and the direction of the edge, which provides information on the orientation of both the contigs.

Currently, Stringit supports both AMOS and Newbler <data format sources>. For the AMOS databank to be readable in a text editor, first a bankreport needs to be generated on the contained contigs, edges, and reads. Newbler presents its output in a readable folder and can be analysed immediatly.

Functions

After conversion, the file is loaded in Stringit. The contigs and links are stored in an array, creating the basis of the graph. For each contig, it uses the read mappings to determine an origin sample if the readmap show a clear majority from a single sample. Otherwise, its origin will not be defined. Nodes with a single sample as origin represent a genomic sequence that is not shared between different samples, and instead is only present in a single sample. The focus of a co-assembly analysis is on the differences and similarities between samples, which makes differentiating between these two possibilities necessary <bronnenX><plaatje van pie chart>.

Before the graph can be displayed, the contents of each tier need to be determined. The tiering of the graph is based on scale. On the lowest tier, each node represents a single contig. Thus, this tier consists only of the earlier stored array. Higher tiers aggregate these nodes together based on similarity. The second tier collects all the nodes that only have two neighbours. This consolidates strings of nodes into one, decluttering the screen. The third tier groups all nodes that are next to each other and share the same origin sample. The fourth tier takes this a step further. It not only groups nodes that share a likeness in origin, but groups any set of nodes that share a similarity in read mapping *<plaatje van vergelijkbare regio met veel vergelijkbare piecharts>*. Biologically, this should point to any region where two samples have similar sequences. The fifth tier simply groups all the nodes that connect to each other. This makes it easier to identify orphan nodes that could either signify an assembly error or a biological phenomenon. Additionally, it separates nodes on different chromosomes.

In principle, nodes that were grouped together in a specific tier, should also be grouped together in a higher tier. However, this cannot be done automatically, because there are feasible scenarios were this should not happen because of biological variation between the samples *<voorbeeldplaatje van nodes>.*

Each tier now consists of its own set of nodes and edges, where each node represents a different set of contigs, but through these contigs and the original network that they form, the tiers are connected vertically.

With the help of the d3.js library, all the different elements described above can be visualised. For the lowest tier, each node is represented by a pie chart, which displays the percentage of reads from each sample that map to that contig. In higher tiers, the pie chart is replaced for a single color to signify the node as a group of contigs rather than just one. The size of the node represents the average sequence length of the contained contigs.

To separate nodes from each other, they are under the influence of gravity. Nodes get pulled together based on sample origin *<gravity points plaatje>*, separating nodes with different origins. Nodes without an assigned origin aren't pulled towards a specific point, but will get suspended between their connections that do have one, to naturally sort them based on similarity *<zoom-in van vorige plaatje>*

When the visualization starts, only the nodes representing the highest tier get displayed. Hovering over a node and scrolling down will expand that node into the group that it represents, going down a single tier. Scrolling up will collapse it again. This way, all tiers of the region of interest can be viewed consequently, without cluttering the screen with nodes representing other regions or groups. *<plaatje van gehackte testcase voor dit>*

Every node is also clickable, to bring up a panel with additional information. It shows the sequence, or the aggregated sequences in the case of a higher-tier node. It also gives the possibility of further research, providing links to BLAST or annotation tools, among others *<meer concrete dingen>*.

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.

During development, 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 (4)

Results

design

compared to other tools

what does the data look like, and for what is it used.

niche of use

showcases for different areas-of-interest

limitations of other tools that Stringit lifts

Discussion

input/output

different formats, how do other tools handle that

user tests

what are actual requirements, and how can they be accomplished by other tools

Stringit is capable of handling several different input formats. Besides supporting Newbler and .bam files, it supports AMOS databanks as well, opening access to every assembler compatible with AMOS. This custom input approach is different from for example PGAT, which only allows analysis of known sequences already in their database. Other tools (PanGP, PanFunPro) have that same restriction. Stringit is unique in that way, as it allows for research on not-yet annotated genomes.

Stringit is best for use in just assembled datasets. Before any annotation, or a search for similarity has taken place, Stringit can be used to search for regions of interest, focussing the work on smaller parts of the genome, thereby improving efficiency. It allows for

Why would you use Stringit

* identify shortcomings in assembly
* quickly identify overlapping regions between species
* find overlap or SNPs inside your region of interest.
* Export region of interest to synteny or BLAST tools
* Get statistics on spacial division of reads from each organism

*PanGP tool, PanFunPro, PGAT*

* “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/AMOS/454)
  + 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?

Conclusion (2)

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

Future work (3)

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)

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>

Pangenome

<http://genomebiology.com/2007/8/5/r71>

<http://www.nature.com/nature/journal/v499/n7457/full/nature12213.html%3FWT.ec_id%3DNATURE-20130711?message-global=remove&WT.ec_id=NATURE-20130711>

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