***Methods***

*1. Overview.* Stringit is a tool that visualises the data inside assembly graph files. The file is loaded in the webapplication, and is processed to a visual graph, which is displayed on screen. There, several tools can be used to analyse the data. The graph can be exported for use in other tools (such as alignment mappers).

*2. Stringit*. Stringit is a browser-based tool, where users can upload their assembly graphs, after which they can analyse the graph network in the browser window.

It does this by parsing the input file to a javascript object containing all the information on nodes and edges. Using information contained in the graph file, it groups the nodes based on the origin of the sample, creating differently coloured groups and making co-assembly analysis possible. Zooming into different tiers is possible, each consecutive one showing less overall sequence, and more detail. Based on the zooming tier, nodes get aggregated into larger nodes. This way, only information relevant for that level is displayed.

The nodes give information (via a mouse-over HUD) about the sequence(s), and edges denote overlap between nodes. When the sequence diverges because of variation, this is visually represented by two different nodes that converge behind the variation (such as a SNP or indel), which forms a ‘bubble’. The user can analyse any overlap and divergence between species or strains with this combination of zooming tiers, node aggregation, and bubble recognition.

*2.1 Input data*. As input for Stringit, a graph file is needed. Several assemblers are capable of producing such a file. So far, the graph formats of SGA and 454 (“Newbler”) are implemented. However, any graph file is usable, in principle. The formats are recognized internally and processed into a JSON object, containing all the information of the original nodes and edges.

*2.2 D3.* Stringit uses a powerful javascript library to handle the graph data called d3. D3 is short for Data Driven Documents. It is used to bind data to DOM elements. Stringit uses a force-directed graph layout, one of several ways D3 can be used to display graph data. Nodes representing sequences are connected by edges representing overlap, and the internal d3.js mechanism lay this out on screen. Additional functionality is added with a mouse-over display that shows information on the current node, as well as zooming in and out to regions of interest.

***3. Features:***

*3.1. Zoom tiers:* to be implemented. The highest zoom level will contain sequence information, while the lowest zoom level contains the full sample. The further zoomed in, the finer the resolution of nodes and variance that is displayed.

*3.2 HUD:* to be implemented. Will contain information on current selection of nodes and/or edges

*3.3 Exporting*. Because of the broad acceptance <source> of the format, .dot is chosen as the file format for exporting the user-created network graph. The .dot file format is a generic file format, readable by text editors. Its syntax is suited for describing (network) graphs. Various tools exist for conversion of .asqg and other file formats to the .dot format. However, it has no native way of storing information of overlap in nodes. Included in Stringit is a function for conversion of the data to .dot, maintaining all the input information.

***Supplemental Information***:

*SGA*. SGA, short for String Graph Assembler, uses a suffix array-based method to order the sequencing data into a graph that describes overlaps between sequencing reads. From readset to assembled genome, SGA uses several steps.

SGA starts its assembly with several cleanup steps, to get rid of low-quality and duplicated reads, as well as remove errors in them. After that, an FM-index of the reads is created, which is then used to find the overlap between reads. With this, the string graph is constructed. This last step produces the .asqg file used by Stringit.

However, the SGA assembler then continues, and uses the created graph to make contigs. In later steps of the program, read pairing can be taken into account to collect contigs into scaffolds.

To create a usable test case, at first the GAGE recipe for SGA was used. To simulate a co-assembly, the same recipe was used on simulated readsets of the combined genomes of *e. coli* and *e. albertii*.

SGA source: <http://bioinformatics.oxfordjournals.org/content/26/12/i367.abstract>

GAGE recipe:

<http://gage.cbcb.umd.edu/recipes/sga.html>

*ASQG.* The origin of the acronym is unknown, but considering its function, it is probably related to Assembly SeQuence Graph. This file format is a text file with a specific format. It can be read with any text editor.

|  |
| --- |
| Example.asqg |
| HT VN:i:1 ER:f:0 OL:i:45 IN:Z:reads.fa CN:i:1 TE:i:0  VT read1 GATCGATCTAGCTAGCTAGCTAGCTAGTTAGATGCATGCATGCTAGCTGG  VT read2 CGATCTAGCTAGCTAGCTAGCTAGTTAGATGCATGCATGCTAGCTGGATA  VT read3 ATCTAGCTAGCTAGCTAGCTAGTTAGATGCATGCATGCTAGCTGGATATT  ED read2 read1 0 46 50 3 49 50 0 0  ED read3 read2 0 47 50 2 49 50 0 0 |

Each line of the file starts with either HT, VT, or ED.

The first line of the file is the Header Tag, and starts with HT. It contains information on the file version (VN), and parameters associated with the creation of the graph (such as minimum overlap length, and origin fasta files).

Lines starting with VT describe a VerTice, or a node of the graph. This line contains the name of the (contig) node, and its sequence.

Finally, lines starting with ED describe the EDges. In order, this line describes the following features:

1. sequence 1 name
2. sequence 2 name
3. sequence 1 overlap start (0 based)
4. sequence 1 overlap end (inclusive)
5. sequence 1 length
6. sequence 2 overlap start (0 based)
7. sequence 2 overlap end (inclusive)
8. sequence 2 length
9. sequence 2 orientation (1 for reversed with respect to sequence 1)
10. number of differences in overlap (0 for perfect overlaps, which is the default).