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 <bron?><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>

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.