progress log with sources and articles sorted by date

Max Jonkman

July 16, 2014

week 1: Introducerend leeswerk

gelezen over:

- co-assembly
- de bruijn graphs
- string graphs en graph theory
- assemblers
- output files/fastq/bam
- MSA visualization
- VIZBIO presentaties
- javascript/Jbrowse, other tools
- assembly in cortex
- UNIX

werken aan proposal, logframe gemaakt.

week 2: Cortex

Played around with Linux and Cortex. Also downloaded and tried a lot of different research tools that are used in the field. The data for Cortex came from Illumina reads of ecoli. In the end, an enormous amount of contigs were created from these reads. It seems that, even though the amount of data is pretty huge, the quality is pretty low.

Week 3: Proposal writing proposal taking shape. Log frame is done. Looked at js libraries for visualisation and graphing. looked at in- and output files. Looked at synteny browsers. I found a lot of tools that do some sort of visualisation. They do not do what I want my tool to do, so there is a niche. However, these tools do help me in the way of customer experience: how does a good tool look and feel? Also, what functionality in regards to research is sought-after? Tools I found are SyMap, CoGE, Tablet and Strudel. I also read about different networking file formats. Most of these are proprietary for a specific tool, but most XML-based formats, as well as .dot are usefull for general purposes. Collecting network information in a JSON file is also a possibility.

29/1: proposal writing, first things tried with js. Just getting re-acquainted with the language and its quirks.

30/1: tried CoGE and cytoscape.js. Had a meeting with Sandra and Saulo about his mummer browser and what it could help me. Its browsing capacity is helpful for the lowest zoom-scale of my tool. He points to d3.js (Data Driven Documents) and node.js (network scaling tool). Cytoscape seems useful as a graph-displaying tool.

3/2: installed and tried newbler. Is an overlap assembler and doesn't make graphs but .ace files. I looked at those files with Tablet, Symap, Strudel, to get

a feel of those programs as well as to look at the data. Read manual of cortex var. Cortex var is useful, .ctx is graph. There exist different tools to rewrite a .ctx binary file to a readable .dot file.

4/2: cortex var tried extensively. Different settings, different readsets. Multicolour binary produced for testing purposes. The plan is to use this data to have something to visualise in earlier stages of my tool.

–twee weken weg voor tentamen en vakantie–

18/2-21/2: proposal writing.

24/2: Cortex var tried, it has a lot of settings. Tried more things, to produce a better testset.

25/2: starting with rapid prototyping. 1 week to produce an assembly graph on a web screen. I chose to use the easiest graphing software, which was draculagraph. Very easy syntax, although it is not very powerful.

done in this week:

- take a 100kb ecoli sequence. simulate reads digitally (long and short)
- use cortex var to produce binaries
- use .dot file for js.
- produced a 95MB .dot file

This was too big to analyse. retried with 2kb file. made a 2MB dotfile. 30k edges+nodes. apparently .ctx files are not simplified string graphs... Again this was too big.
pipeline to reproduce results:

- 100kb/2kb fasta file
- pbsim –data-type CLR input.fasta –model-qc data/model-qc-clr
  (output: reads.fastq)
- cortex-var-31-c1 –mem-height 15 –kmer-size 31 –dump-binary bin.ctx –selist reads.fastq
  output: bin.ctx
- cortex-binary-reader bin.ctx¿output.dot
- dot -Tpng output.dot done.png
  (output: picture of graph)

general outline for tool: javascript+canvas with existing graph library. Has an import/conversion tool for input file formats (for now, just .dot)

using the pipeline, a dotfile was made from a 100bp sequence. 645 nodes, with presumably a lot of overlap. However, when using the conversion script, information about the precise overlap is lost. However, these 600 nodes could eventually be written to a HTML canvas element and shown in the browser. No other functionality other than displaying my data. The nodes overlap so that

not even distinct links can be discerned, but at least it is on screen.

Proposal sent out. I made a Logical Framework to map out my objectives, and made a report that containts some background information, my delineation and approach of the problems, and the tools I will use to do so.

5/3: new week, new 1week assignment: to do: simplified string graph on screen (so more sequence, less nodes)

tried more assemblers:cortex var looking for supernode dumping, SGA, AByss, Velvet.

cortex var supernode dumping was not succesfull, because it does not produce a file that can be converted into something sensible. It just dumps some FASTA sequences. SGA installed and all its dependencies (PITA). SGA produces asqg files, which are graphs. Very promising program, but the installation cost a lot of time.

6/3: SGA attempt pipeline:

- index
- correct
- index
- filter
- overlap
- assemble
- align

2k ecoli produces 60 nodes. Somebody (outside of this lab or SGA) wrote an asqg-¿dot converter, which is handy, because my program can readily read .dot syntax.

7/3: This new success leads me to try a full genome. However, I find that an assembly of the full e. coli genome produces 3 million nodes. I scale back again, and use a 2k sample. I process it in cytoscape.js. I still try to find useful data samples. There are not a lot of publicly available datasets that produce a nice contig graph, and even if there are some, I will eventually need a co-assembler to try with, because that is the data that I need to visualize. I have no success in this regard, so far.

When trying to handle a lot of nodes/edges, cytoscape quickly hangs. It is a known issue, and the internet advises to use sigma.js, because it is better at handling lots of nodes.
weekend: run SGA on server with new found data

11/3: use higher coverage when creating artificial reads. try different genome region for 2k part.

13/3: clusterfuck all around. decide to use GAGE recipes and data for production of data.

14/3: sigma.js is not strong enough as well. start using d3.js. is useful for all sorts of data and visualisation, not just graphs. A very powerful library that has an enormous amount of development and documentation. Full genome of s.aureus from GAGE produces 6000 nodes. This, together with the power of d3, finally produces something worth showing.

17/3: need a webserve to handle json, made a localhost. d3 seems to be able to handle 6701 nodes just fine. After some rendering, I can show a circular genome of S.Aureus on my browser screen. Although hidden from view, the nodes still contain information about their sequence, and the links still specify if they connect the contigs in order, or in a reverse-complement order.

18/3: wrote all of the tex file up to today from notes and logbook.

19/3: After a new discussion, it is decided I can trick SGA in doing a sort of co-assembly. I give it data from 2 related genes. This is the Human and Baboon version of the ADH1A gene. After simulating reads, both files are merged, and a single assembly is done.

20/3: the assembly produces an asqg file. Scrolling through the file, I can see it is mainly two distinct networks, with only a bit of overlap, instead of a single network with only a bit of divergence like I wanted.

21/3: A lot of research in the article discussion. After some searching I find an article called C̈hromosome-scale scaffolding of de novo genome assemblies based on chromatin interactions: It is very interesting, and that is the reason it was already discussed in November. I pick a new article, and research it. The final article is N̈onhybrid, finished microbial genome assemblies from long-read SMRT sequencing data: It is about using only PacBio long reads for the assembly of genomes.

24/3: More research on the article. Also some research on JSON, to use as an internal file format for storing the data.

25/3: A new review, with a new 1 week deadline. The full pipeline needs to be automated, from reads to visualisation with as little human intervention on the data as possible. Parsing of the data from reads to .asqg file was already done, so that only leaves the file format conversion and display of that data.

26/3: Worked on file conversion, from .asqg to JSON storage. It works, but still needs properly escaping of characters, so malicious individuals can't crash my code or server.

27/3: Updating the graph with data, after the page already is loaded, is more tricky than originally thought. Updating the graph with a few functions gives all sorts of interesting errors, and it still doesn't work.

28/3: Frustrated with my (lack of) progress, I decide to read on other subjects. I made a github account, and set everything up for full online access to all of my work.

31/3: more research and coding regarding dynamic loading of data into d3 graphs. I am still not convinced if the 1 week deadline is reachable, because there are still unexplained errors in updating the graph.

1/4: Got it working

2/4: new deadline: write-up of methods section so far. Also implement some new functionality. Wrote a list of things to-do.

week of 8/4: congress. Poster of Jurgen Nijkamp contains some tricks for variant splitting in assembly graphs. Read about splitMEM use notes to expand

14/4: combined fastq files of simulated reads of e.coli and e.albertii, and put them through Newbler and SGA. Used the final graph files as de-facto co-assembly cases.

Newbler output is noticably different from SGA. The main difference is the way the nodes are built. Because SGA produces a graph in an intermediate state, it contains overlapping sequences between connected nodes. Adjacent nodes from Newbler do not overlap. Rather, edges notify a split in sequence linearity. This has the added bonus of giving directionality. Edges can come from either the 3'- or 5'- end, which is documented in the output file. This makes graphing more visually appealing.

15/4 - 17/4: Finalized producing Newbler assembly. Made Stringit compatible with its files.

22/4: Newbler creates an .ACE file that contains all the mapped reads. This file can be used by Stringit to assign groups/colors to the different nodes. A script was written to analyse this (large) file. For every contig, it counts which of the reads mapped to that contig originate from which sample. It is found that from the used testcase (259 nodes), many nodes contain reads mainly from one source. However, a few (17) have mapped reads from both samples.

23/4: To visualize the nodes that contain reads from both samples, a pie chart representation is chosen.

may: The article discussed by Dick mentions the analysis of Highly Polymorphic

polyploid organisms. In the article, a tool is developed to make this easier. This can also be a niche target of Stringit: load the different chromosomes as different samples, and let Stringit point out their differences and overlap. The point of the article was that regular assemblers (they tested SPAdes and Velvet) are not very good at picking them apart, and their new assembler (dipSPAdes) is. They use data synthesized from s.commune genomes as sample data. This is a bacteria that is very heterozygous for different samples. I might need to test the data they generated with their assembler, and compare our analyses.

The structure of the result section is based on different tests. Each part is meant to answer a single research sub-question, and tests a different part of Stringit.

SplitMEM (from the article) is a tool to efficiently analyze overlap in de bruijn graph nodes (contigs) between different genomes. It is therefore a pan-genome analysis tool, written very efficiently. It utilizes suffix trees and the FM index to search for repeat regions, so that they can be distinguished between co-assembled organisms. This also ensures that differences locked between differentiating repeats can be seperated. Smart stuff.

To incorporate the code for the directionality, Stringit needs to touch the second and fourth column of the 'edges' list in the Newbler ACE file. These columns give the direction of the left and right nodes (5'-3' or 3'-5'). Stringit will always make the edges point to the 5' end of the non-reverted node, so that it will follow proper read order. It needs to be done this way because there is currently no way to have a node have separate sides for the edges to attach to.

To obtain the readmappings from yet another file, an external python script is developed. This script is faster in analysing the (big) file that contains all the mapped reads. In the future, aside from Newbler, this script will also support SAM and AMOS databanks.

AMOS bank files contain not just the contigs and scaffolds associated with an assembly, but additionally contain the mapped reads, and a graph representation of the assembled genome. The datasets from the Marygold paper (3-ecoli and 7-ecoli) contain CTG, CTE, and CTL banks. These can respectively be converted to contig files, edge files, and mapped reads. The contig files can then be used as nodes in Stringit. An external Python file is developed to extract this information from the CTG,CTE,CTL bank report created with AMOS (text dump from binary).

Herevaluatie wetenschappelijk nut Stringit

Terug naar de basis: we kijken naar co-assemblies. Hoe kan mijn tool het beste gebruik maken van die kennis tov de data? Hoe kan het dat op zo'n manier doen dat het iets toevoegt aan het huidige assortiment aan tools, en niet alleen maar een mooi plaatje is?

Als we weten dat we naar een co-assembly kijken terwijl we een contig graph visualiseren, dan is het interessant om te kijken naar de regios in de graph waar de soorten samen voorkomen. Dit uit zich in een contig graph als nodes en edges die een combinatie zijn van meerdere soorten/origins/samples.De beste manier om hier de nadruk op te leggen is het verschillend behandelen van deze meervoudig samengestelde regio's (waar de nodes zijn samengesteld uit reads van verschillende bronnen) vergeleken met enkelvoudige regio's (nodes waarvan de reads overwegend 1 bron hebben).

Dit kan door de regios respectievelijk:

1. in het midden en aan de rand van het scherm weer te geven
2. fellere en valere kleuren toe te kennen
3. groter en kleiner weer te geven
4. meer en minder ruimte op het scherm te geven

De verdeling tussen deze regios is echter niet zwart-wit. In meervoudige regios is er een overeenkomst tussen beide sequences. Afhankelijk van hoe groot we een regio definieren, ontstaat er een verschil in wat een meervoudige regio precies is. Ook binnen een regio van een vaste grootte zijn er nodes en paden die meer of minder interesse wekken. De combinatie van deze twee variabelen leidt er toe dat er een selectie gemaakt moet worden. Als er naar een grotere regio gekeken wordt, moeten de minder interessante paden wegvallen om meer overzicht te scheppen in het restant.

Het wegvallen van de paden wordt gedaan door het samenvoegen van nodes. Hierbij worden de 'minder interessante' paden geaggregeerd tot een enkele node, wat meer ruimte laat voor de andere nodes. De schaal van de regio bepaald wat er als 'minder interessant' wordt gedefinieerd: hoe groter de regio, hoe groter de drempelwaarde van diversiteit/overlap/verschil in sequence wordt. Onder deze drempelwaarde wordt er samengevoegd, erboven niet.

Om al deze functies toe te voegen aan Stringit, moet het programma in staat zijn deze eigenschappen aan de graph data af te leiden. Dit betekent dat het correct de verschillende nodes moet kunnen splitsen in groepen, gebaseerd op de verschillende samples. Ook moet het de nabijheid van andere nodes kunnen zien om deze samen te kunnen voegen. De data die hiervoor nodig is moet van de gebruiker komen, en moeten worden gestandaardiseerd. Deze abstrahering kan alleen gebeuren als er een conceptueel verschil wordt gemaakt tussen wat

er gevisualiseerd wordt en de onderliggende data. Op het moment wordt de letterlijke assembly graph op het scherm gebracht door Stringit. Dit zal moeten veranderen, en er zal onderscheid moeten worden gemaakt tussen de assembly graph en de gevisualiseerde graph. In de laatste is een node niet altijd meer een enkele contig, maar een samenvatting van alle contigs en links die hierin zijn geaggregeerd.

De huidige staat van Stringit

Stringit is in staat de databank van een Newbler assembly te lezen. Het gaat hierbij om de bestanden die de contig graph bevatten en de aligned reads. De eerste wordt gebruikt voor het maken van de graph, en de tweede wordt gebruikt voor de groepering van nodes. Op dit moment lijkt het er op dat deze data genoeg is om zowel de huidige functies als de hierboven genoemde nieuwe functies te faciliteren. Newbler is niet de enige die deze data aanbiedt, er zijn een aantal andere programma's die dit ook bundelen in een databank. Voorbeelden hiervan zijn AMOS en SAM. Het is mogelijk dat Stringit deze bronnen later ook zal ondersteunen, en kan gebruiken als input data.

Het verdelen van de groepen in nodes gaat als volgt. De read map wordt geanalyseerd, en van elke node wordt bepaald hoeveel reads van elk sample erop mappen. De data daarvan wordt gebruikt om de nodes in groepen toe te wijzen. Een node met alleen maar reads van sample 1 wordt toegewezen aan groep 1. Een node met een mix van reads verdeeld over groep 1 (70%) en groep 2 (30%) wordt aan beide groepen toegewezen, met een relatief gewicht van 0,7 en 0,3 voor de respectievelijke groepen.

Het genereren van deze verdeling is (nog) niet dynamisch, maar er bestaat een werkende testset. De aanname die moet worden gedaan voordat deze input wel dynamisch kan worden verkregen, is dat de gebruiker kennis heeft welke reads afstammen van welk sample. Dit is alleen te bereiken als samples individueel gesequenced worden, en daarna samen worden gevoegd voor de assembly. In het geval dat een enkele sample meerdere organismen bevat, heeft Stringit geen manier deze uit elkaar te halen. Er zijn op het moment geen plannen hier verandering in te brengen. Wel zijn er plannen om Stringit dynamisch de read mapping door te lezen, en deze informatie toe te voegen aan de nodes.

Aannames over toekomstige functies

Om regios te creeeren aan de hand van naast elkaar gelegen nodes, is het nodig om door de nodes en edges te lezen en daarmee de de edges van een specifieke node te volgen om zijn buren te vinden. Hiermee kan worden bepaald waar de regios zijn waar nodes kunnen worden samengevoegd naar aanleiding van het gekozen zoomniveau. Op het moment lijkt het er wel op dat het huidige framework (d3.js) in staat is om dynamisch de nodes samen te voegen, maar het is nog onduidelijk hoe dit precies moet werken.

Gekeken moet worden naar welke data precies wordt gebruikt om zoomniveaus en drempelwaardes mee te bepalen. Een van de opties hiervoor is een combinatie van het verschil in groepering van naastgelegen nodes, en de lengte van de sequence tussen een bepaalde node en een node een paar stappen verder. Hiermee kan een schaal worden gecreeerd waarop de drempelwaardes van de verschillende niveaus op worden geplaatst.

Met behulp van de al geimplementeerde groepering van nodes, en de nog te implementeren bepaling van interessante regios kunnen de visuele aspecten van Stringit worden gemaakt. Stringit zal de meer en minder interessante regios respectievelijk meer en minder naar de aandacht van de gebruiker brengen, waarna deze met behulp van een Head-Up Display en de verschillende zoomniveaus in detail naar de weergegeven informatie kan kijken.

Stringit kan dit bijna geheel doen met de informatie die over de nodes. Het scherm bevat informatie over de sequence, over de geaggregeerde nodes, over de oorsprong van de reads, en kan zelfs toegang bieden tot verdere tools om de sequence mee te analyseren (BLAST, annotation, etc). Op het moment is er in Stringit nog maar weinig aan deze functies gedaan. Een individuele grootte van elke verschillende node, gebaseerd op de onderliggende sequence, is al wel geimplementeerd, en nodes zijn gekleurd aan de hand van hun toegewezen groep. Verdere visuele aanwijzingen voor interessante regios, en tevens de HUD, moeten echter nog toegevoegd worden.

Uiteindelijk zal Stringit gebruikt kunnen worden om snel de interessante, overlappende gedeeltes sequence uit een co-assembly te halen. Ook kan er worden gekeken naar toepassingen omtrent polyploidy, metagenomics, of het vinden van synteny. Dit valt echter buiten het bereik van dit project.

Werkplan voor de komende tijd

Om aan de node-aggregatie te kunnen werken, is er een testcase nodig waar verschillende niveaus van verschil/overlap te merken zijn in de nodes. Het lijkt erop dat de 3-ecoli of 7-ecoli testcase van Marygold deze verschillen bevat. Allereerst zullen deze datasets dus leesbaar moeten worden voor Stringit. Dit betekent dat de AMOS bank geanalyseerd moet kunnen worden, en daaruit alle data gehaald moet kunnen worden: contigs, links, read mappings, sample origin. Hiermee kan de graph worden gevisualiseerd, en groepen kunnen worden toegekend (beiden automatisch door Stringit). Hierna kan een manuele inspectie gebieden opleveren die nog niet optimaal worden gevisualiseerd. Dit kan dan worden aangepast in Stringit, wat de basis legt voor de node-aggregatie.

Zonder precies te zien hoe een interessante regio zich zal uiten in Stringit, is het moeilijk te voorspellen wat de beste manier van visualiseren is. Er is er een groot verschil tussen het aggregeren van enkelvoudige regio's en van meervoudige regio's. Tenslotte, waar de grens moet worden getrokken voor wat een bepaalde regio begrenst, is voor een enkelvoudige regio vrij duidelijk: totdat deze niet meer enkelvoudig is. Voor een meervoudige regio is deze grens niet zo duidelijk, omdat meervoudige regio's ook enkelvoudige nodes kunnen bevatten (zoals bijvoorbeeld in het geval van een bubble/bulge).

Verschillen in de graph kunnen zich op een schuivende schaal anders uiten. Het is op dit moment onbekend of de criteria van een 'interessant gebied' lineair meebewegen met de criteria van een bepaald zoomniveau. Voor elk zoomniveau moet hier apart naar worden gekeken, wat niet kan zonder testcase. Pas als duidelijk is naar welke data in de (assembly) graph er daarvoor gekeken moet worden, kunnen er criteria worden gemaakt voor het begrip 'interessante regio. Aan de hand van die criteria kunnen dan zowel de aggregatie als de eerder genoemde visuele aspecten gekoppeld worden.

Het werken met een dataset met meer dan 2 samples zorgt ook voor een beter beeld als het gaat om de layout-aanpassingen. Wat de beste manier om het zwaartepunt van elk sample te plaatsen is, hoe de groepering zich precies uit (geen groepen die tov elkaar in de knoop raken), en hoe de piechart gevormd wordt, is op dit moment nog allemaal alleen duidelijk voor een visualisatie met 2 samples.

Ook moet worden gekeken wat een efficiente manier is om langs de nodes en edges te reizen en regio's te bepalen. Hiervoor bestaan al verschillende methodes (Dijkstra's algoritme, A*, etc), maar het is niet duidelijk hoe die precies geimplementeerd kunnen worden in Stringit. Ook kan er voor worden gekozen om brute-force door de lijst te gaan.

Exactere uitwerking node-aggregatie

Een node stelt bij elk zoomniveau het volgende voor, van meest ingezoomd naar minst:
1. De contig
2. De contig en zijn buren, als dat er maar maximaal 2 zijn. Dit gebeurd recursief om een hele reeks in 1 groep te kunnen zetten.
3. De groep van hierboven, en zijn directe buren, gebaseerd op het overheersen van 1 bepaalde sample. Het doet dit door een node te selecteren met zo'n overheersende verhouding, en daar alle buren van te zoeken die een vergelijkbare verhouding hebben.
4. In dit niveau worden niet alleen de nodes van 1 sample bij elkaar gevoegd, maar ook de nodes waarvan de verhouding heel erg op elkaar lijkt, ook al komt die van meerdere samples.
5. De complete (sub)graaf. Op dit niveau bevinden zich geen edges.

Doel:

De graaf wordt voorbereid uit de data. Voor elke node (1) wordt de gemiddelde variatie bepaald. Ook het globale gemiddelde wordt uitgerekend (4). Naast de huidige gegevens over een node, komt er een nieuwe genaamd 'zoom'. Dit is een lijst met 5 variabelen, 1 voor elk zoomniveau. De variabelen geven de node-id aan voor dat zoomniveau. Voor zoomniveau 2 bijvoorbeeld, kunnen er 3 nodes zijn met waarde 1. Dit betekent dat als we naar zoomniveau 2 kijken, deze 3 nodes worden weergegeven als 1 node.

Een node selecteren laat gegevens zien over de node en omvattende contig(s). Totale lengte aan sequence, totaal aantal contigs, gemiddelde variatie (dus de reden waarom deze node zo is samengevat). De eerste klik selecteert een node, de tweede laat hem los. Meerdere nodes kunnen tegelijk worden weergegeven, en als deze nodes onderdeel zijn van dezelfde groep (onderdeel van dezelfde node in een hoger niveau), dan wordt hun data wordt samengevat/bij elkaar opgeteld. Zo niet, dan wordt het naast elkaar weergegeven (tot een maximale breedte), om vergeleken te kunnen worden. Ook worden er links naar tools in beeld gebracht, waarmee de sequence van de contig(s) verder onderzocht/geannoteerd kan worden.

Omhoog scrollen laat alle geselecteerde nodes 1 niveau naar beneden gaan (inzoomen). Voor alle contigs in die node wordt het zoomniveau 1 verlaagd, en aan de hand van de groepering die dan genoemd wordt, worden ze weer bij elkaar geplaatst. Inzoomen deselecteert alle nodes, omdat de geselecteerde node niet meer bestaat.

Omlaag scrollen zoomt de geselecteerde node uit, waardoor alle nodes die in diezelfde groep horen (in het hogere niveau) worden samengevouwd tot 1.

Voor elk weergaveniveau worden de nodes die in het niveau erboven bij elkaar gegroepeerd worden, dichter bij elkaar gezet dan nodes die dat hogere niveau in een andere groep zitten. Hierdoor blijft de algemene structuur zelfs bij het inzoomen duidelijker, en ook maakt het de verschillen meer opvallend. Wat is hier voor nodig?

Een algorithme dat een node pakt, hem aan een groep toekent, door alle links leest, alle buren van de node bekijkt, deze beoordeelt op hun gelijkendheid met de gekozen node. Als ze genoeg op elkaar lijken, dan word en ze bij de node in dezelfde groep geplaatst. Nodes die er buiten vallen, maar alleen maar buren binnen 1 groep hebben, worden ook toegevoegd aan de groep, maar er wordt wel bijgehouden dat dit alleen maar indirect is (interessant!).

Ook moet er worden bepaald wanneer een node 'voornamelijk' uit hetzelfde sample bestaat. Bij een mix van 7 samples, kan 25% al een grote fractie zijn. Dit kan eventueel worden gebaseerd op (standaard)deviaties van het lokale of globale gemiddelde, maar dat is op dit moment nog niet duidelijk.

Re-evaluation of assumptions and future plans

Since the previous pages have been written, I learned a lot about the functioning of Stringit and assemblies in general, I have seen a lot more data, and some of the assumptions that I had before can be resolved. The following text tries to reconsolidate the previous pages with the facts as they stand now, and the current vision for Stringit.

One of the previous unknowns was the selecting criteria for the different zoom tiers. Currently, 5 different tiers are defined and can be calculated from the mapped reads for each contig. However, they are no longer based on read length, and they do not use any form of threshold values. Instead, the tiers are based on the differences and similarity between neighbouring nodes. The 5 chosen tiers make sure that the data can go from 1 chromosome per node to 1 contig per node in a few easy steps. One other aspect of Stringit that has disappeared is the notion of an 'interesting region'. Stringit will no longer define such a thing, instead using its visual tools to display the different nodes and tiers more clearly.

Stringit is currently capable of using AMOS bank files. The completion of a script that gathers all necessary information in an easy-to-read file for use in Stringit is evidence that a large data bank can be consolidated in a relatively small file. Hopefully, there will be enough time in this project to write similar scripts that read for example SAM/BAM files, or Newble ACE files. Without all the information about the contigs, their links, and the reads that map to them, Stringit can not do a proper job, and the availability of a wide range of data formats makes Stringit available for more users.

It is interesting to note that when the zoom tiers are finished, Stringit should finally be capable of handling all the nodes that Cortex produces. Cortex has a k-mer graph as output, not yet concatenating the overlapping sequences in a single node. Tier 2 takes care of that in Stringit anyway, which drastically reduces the number of visible nodes. Since Cortex doesn't provide any read mapping, this still wouldn't produce a nicely researchable network, but the possibility is at least there.

In the beginning of April, a to-do list was made with functions that still needed inclusion in Stringit. As it stands now, a few of these have not been finished yet.

Zooming/grouping of nodes: functions have been written, error-correction needs to start. This function will be included. escaping characters in loaded files: is not necessary, as no file is uploaded to a server. All calculations will be done client-side, so at worst a wrong file will crash the browser. Hierarchical clustering with node size as edge weight: node sizes or overlap sizes no longer are considered useful information for Stringit, so this function wil not be included.

Hierarchical clustering might still be used (see below), but not based on these criteria. The use of node direction when drawing edges is also already commented on. This will not be included. Filetype recognition: Stringit can recognize the file coming from either Newbler or AMOS, and hopefully SAM in the future as well. Graph exporting: there is currently a small function that will write the graph as a .dot graph to a file, but this has not been touched since the inclusion of several newer functions. As such, graph exporting in formats that can be used by other programs (cytoscape, graphviz) is not yet finished. Depending on the ease of the task and the time at hand, the function will get updated to include more data from the current state of Stringit. PNG export is not yet included at all. Exporting the graph as a vector/image file cannot be done yet. Users can take a printscreen though. HUD/mouseover for information. This is another larger project that still needs to be done. It is one of the more useful parts, providing information about selected nodes/sequence, and giving access to further research tools. Most of the things have either been finished or have been rendered obsolete. A few projects still need work, but generally, Stringit came a long way since then, either removing the need for a function, or including it.

Both Newbler and AMOS include the direction of the nodes for each edge. However, in its current form Stringit doesn't need these directions. For scaffolding reasons it is understandable why they are included, but currently Stringit draws the edges out of nodes in such a way that not only is it not necessary to include them, but nontrivial as well.

In the previous pages, several visual clues were listed to help the user distinguish regions within the graph. Currently, not all of these are implemented. The lengthening and shortening of edges between nodes can be used to distinguish between groups even if they are expanded, and the node size will still be used to relay sequence size. However, a visual clue to signalize the tier of the different nodes has not been planned yet, while that would be good to show clearly. Using node opacity doesn't achieve the necessary effect for this case. Another possibility would be the thickness of the border around every node, or the color/hue of the connecting edges.

One assumption that still stands is the assumption that a single sample in the co-assembly consists of a single organism. Stringit can not handle a single-sample mix of organisms (for example a metagenome). This is a physical limitation, and not something that can be solved with just Stringit. It all depends on the output of the used read mapper how the samples can be distinguished (usually by name).

A few cases of node layouts have been made, to see how Stringit will react on these while tiering the nodes. In the best case scenario, scrolling through the tiers will easily show the different regions in the mini-networks, making manual search for finishing or other tasks a lot easier.

The use of graph-travelling algorithms has been mentioned before. Currently, it is not deemed necessary for Stringit. Most decision regarding nodes belonging to a certain group can be made based on a single edge. However, the selection criteria for tier 4 currently is not very efficient. It is based on read map averages over the different nodes in a group in regard to the global average and the surrounding average. If differently calculated, a single-linking clustering approach could be used to speed up this process.

Use of algorithms like Dijkstra or A* will not be necessary, because no function of Stringit will be used to calculate an efficient path between two nodes. Biologically, this information would not be of use.

Although the work on the group assignment within tiers is out of the production stage, it still needs to be tested and refined. Only after that can we look into further function of the d3.js library, for collapsing and expansion of the displayed graph.

Lastly, briefly reading through the Log Frame provided with the proposal shows the progress so far. All activities are fulfilled (except for a final presentation), all indicators and evidence of those activities is available on either github or the server, and there were no additional assumptions found to hamper that work. Moving one row up to output, the only indicator not yet fulfilled is the usability of Stringit by any other person than its maker. There are plans to do this in the near future, as soon as development of the node aggregation is finished.

determining tiers

Terminology: the words 'node' and 'group' can often reference the same thing here. Nodes in the graph represent a single group, and a group consists of nodes from a lower tier. Therefore, when nodes get grouped, they form a single node in a tier above. The double use of the word 'node' in the previous sentence can be cause of some confusion. The below text tries to be as clear as possible.

After the file is processed and a graph is created, a new function is called, that determines which nodes fall together for every tier.

This function iterates through all nodes for every tier, and when it finds a node that has not yet got assigned to a group, it makes a new group. For every node in the group, all the neighbours are checked for their similarity. 'Similarity' is a definition that changes for every tier. For tier 5, it means all neighbours. For tier 1, every node gets its own group. Tier 2, 3, and 4 gradually fill the space in between these two extremes. Neighbours that match the chosen node, get placed in the same group. As long as the group keeps growing, all the neighbours of these newly added nodes get checked.

Eventually, all nodes get assigned to a group for every tier. From this, a new graph is constructed for every tier. Every group is considered 1 node in that tier. The edges in this graph are recalculated based on the links between the individual nodes in the group. For example, if node 1 has links with node 2, 3, and 4, and node 2 and 3 are considered to be in the same group as node 1, then that new group still has a link with node 4. The node based on the group of these 3 will have a link to either node 4 itself, or the group that it is incorporated in, depending on the zoom level of that group.

Other information is loaded into the aggregated nodes as well. Based on the read distribution and assigned origins of the underlying nodes, the same properties are calculated for the group as a whole.

After this is done, it is possible to display not the raw contig graph (tier 1), but the collapsed version of it (tier 4). Scrolling into selected nodes expands them individually into lower tiers, all the way to the individual contigs. This way is chosen both for performance reasons (redrawing the whole graph takes a lot of time), as well as clarity reasons. Expanding the whole graph doesn't serve a purpose if the majority isn't being looked at at that timepoint. Expanding the graph selectively provides a better look at the overarching structure, while still allowing for an in-depth analysis of the contigs of interest.

Working tiering found by somebody else.

Source: http://bl.ocks.org/GerHobbelt/3104394

Very interesting showcase of grouping of nodes. Only contains 2 tiers and uses clicking to toggle, so it is different from the use of stringit. However, it can be used as a guide for implementing these features in stringit.

Normally, the graph with data gets loaded once. Here, the graph with data changes regularly, and every time it does, it gets passed again to the force layout functions. This means that from the original data not only a network is generated, but also a list is created that contains all the nodes currently being displayed. Each time a node is clicked to expand, this list gets updated (the grouping node gets removed and the underlying nodes get added), and it is newly passed to the force layout.

Another feature is the placing of a 'hull' around a group of expanded nodes. Although redundant with the node color in this showcase, it helps identifying with nodes that belong to the same group. Because node color does not indicate groups in stringit, these hulls can be used to more clearly indicate grouped nodes.

Nodes and hulls react to clicking. Clicking on a node or a hull toggles its state, going from collapsed to expanded and vice versa. With not too much work, this can be altered to trigger on a combination of mouseover and scrolling up or down, not only replacing toggling between two states with scrolling through more than 2 states (5 in the case of stringit), but reserving the clicking motion for other functions that have more to do with specific nodes (such as bringing up a HUD).

Some functions that are included in this showcase is the possibility of displaying multiple links between nodes. In the original showcase, clicking on a grouped node once only displays the links of all the nodes included in that group, to the nodes that they would normally link to. Only after it is clicked again does it go away in favor of the expanded nodes to be displayed. Normally, d3 would render multiple links between 2 nodes on top of each other, making them look all like 1. With the use of helper nodes as a sort of intermediates (like in the rendering of a Bezier curve), and a separate force layout function, multiple links can be displayed separately from each other. This is not a function that is necessary for the use of stringit. In this showcase however, it contributes to a large portion of the code. This makes it harder to take out and use the remainder as a clear test-case for stringit.

Another less-important function of this showcase is the reaction of nodes on each other and the gravity. Normally, nodes can overlap (partially). Here however, they repulse each other, for a more clear visualization. It is hard to predict if stringit would benefit from this function. On the one hand, this function would

provide more clarity between different groups. On the other hand, it increases the need for a bigger screen/viewbox to fully display the whole graph clearly. Although it might be relatively easy to implement, I don't think it would add enough benefit to offset this insecurity of it fitting on the screen or not.

Lastly, this program uses a 'debug mode', where more 'under the hood' parts gets exposed if it is on. For example, the helper nodes get displayed in this mode, and information about nodes and groups is logged in the console on mouseover. The use of these features is pretty smart and something I must remember for future projects.

With this out of the way, what needs to be added to stringit for this to work?

Between parsing the data from file and displaying it as a graph, a new list needs to be created. This list stores all the nodes that currently get displayed. When first starting the simulation, this includes all the nodes from the starting group (level 5 is preferred). When a node gets scrolled on, both this list and the simulation get updated to show the new nodes, remove the node representing that group, and redraw all the links. Using this list also makes it pretty easy to hide nodes (for example separate parts of the graph that are not of interest), because they would just get removed from the list.

The creation of a list containing all the nodes is trivial. Nontrivial however, is the reaction on mouse movements. Using a combination of mouseover (to detect which node is being hovered over) and scroll, as well as detecting whether it gets scrolled up or down is not an easy task, although I am sure it can be done.

I really need to clean and update my sources...

sources

- graphs and string graph theory
  - colored de bruijn graphs (2011)
  - succinct de bruijn graphs (2013)
  - detecting superbubbles (2013)
  - de bruijn graphs blog
  - string graph
  - fm index and string graph assembly
- assembler bronnen
  - assembler review summary
  - newbler blog
  - GAGE
  - GAGE-B
  - Assemblathon 2
  - cortex var user manual
- Assemblers
  - SGA
  - ABySS
  - Cortex
  - Velvet
  - Velvet wiki
  - Velvet and explanation
- general sources
  - co-assembly
  - marygold
  - limitations of next gen assembly
  - Synteny overview
  - start of human pan-genome
  - pan-genome definition with lactobacillus
- other tools
  - WG multiple alignment format (MAF)
  - Mugsy (works without reference
  - SAMtools
  - SyMAP
  - Strudel Comparative Map Viewer
  - Pangenome Stats
- visualisation
  - visualisation tools list
  - visual design principles (VIZBIO keynote)
  - big list of d3.js examples
  - network file formats review
  - Githb page with latest version
  - big list of code and examples in d3 data visualisation