MVP for Biohackathon

1. Rewrite SequenceTubeMap Backend? (exchange NodeJS with Django).
2. Get whole graph (50 - 1000 base pairs):
    a. Input:
        i. vg?
        ii. xg?
        iii. json? (derived from vg view)
        iv. Gfa?
    b. Output
        i. JSON format defined in vg view
3. Give it to SequenceTubeMap (STM)
    a. STM is only able to process the json output of vg view of a given graph in vg format.
    b. (We will extend in JSON v2.0 etc. for annotation and position metadata)
4. Place SVG elements on the page
    a. Inspect what STM produces

STM Investigation

https://vgteam.github.io/sequenceTubeMap/

- Does IVG use STM as an isolated component?
  → In principle, the tubemap.js can be used as an isolated component in order to create the SVG. Required are an SVG identifier, nodes, tracks and optionally reads. See React component TubeMap.js.
- Does MOMI-G use STM as an isolated component?
  → MoMI-G uses a different and modified version of STM, see Differences between STM and MoMI-G tubemap. It is part of MoMI-Gs front-end implementation.
- Do we want to have STM as an isolated component?
  → I think this might be better in order to test and develop more independently our front-end.
  But what we aim for now is to have the STM front-end and the Django back-end. So our future front-end already brings all the features, IVG has until now.
- Is there a formal definition of the STM JSON input? In documentation?
  {"node": [{"sequence":"ACTG", "id":"1"}, ...],
  "edge": [{"from":"1", "to":"2"}, ...],
  "path":[{"name":"_alt_0be379decfe598b898d0b1c4ccc7dedd039538fd_0","mapping":[{"position":{"node_id":"193"},"edit":[{"from_length":1,"to_length":1}],"rank":"266"}, ...]}
  }
- What does STM produce? SVG objects?
  → From an vg view JSON it manipulates a given SVG.
- What are the responsibilities of D3 library?
  → The D3.js library is used in order to
    - manipulate the SVG object
    - add user interaction with SVG object
    - draw the objects attached to the SVG object

  Example
  It basically simplifies the handling of the SVG object including the drawing process.

- If there's 0 D3 functionality, what does STM already do?
  → There is a huge amount of D3 functionality. What can be done is to separate the layout code from the actual manipulation of the SVG code → tubemaplayout.js + tubemapd3.js
- How extensible is the JSON? Are there version numbers?

- Is there a place for node ordering or position information?
  →The JSON output from vg gives the nodes and paths already in the right ordering.

→ There are no version numbers for JSON.

Undocumented Behavoiurs in STM

See: src/util/demo-data.js

```
export const inputTracks1 = [
  { id: 0, name: 'Track A', indexOfFirstBase: 1, sequence: ['A', 'B', 'D', 'E', 'G', 'H', 'J', 'K', 'M', 'N', 'P', 'Q', 'S', 'U', 'W', 'X', 'Z', 'AA', 'AE',
'AG'], freq: 3000 },
  { id: 1, name: 'Track B', sequence: ['A', 'B', 'D', 'E', 'G', 'H', 'J', 'K', 'M', 'N', 'Q', 'S', 'U', 'AA', 'AE', 'AG'], freq: 15 },
  { id: 2, name: 'Track C', sequence: ['A', 'B', 'D', 'E', 'G', 'H', 'J', 'K', 'M', 'N', 'P', 'Q', 'S', 'V', 'W', 'X', 'Z', 'AB', 'AE', 'AG'], freq: 300 },
  { id: 3, name: 'Track D', sequence: ['B', 'C', 'D', 'E', 'G', 'H', 'J', 'L', 'M', 'N', 'P', 'R', 'S', 'U', 'W', 'Y', 'Z', 'AC', 'AF', 'AG'], freq: 4 },
  { id: 4, name: 'Track E', sequence: ['B', 'D', 'F', 'G', 'I', 'J', 'L', 'M', 'O', 'P', 'Q', 'S', 'T', 'V', 'W', 'Y', 'Z', 'AD', 'AF', 'AG'], freq: 2 },
];
```

In this example, there are two keys “indexOfFirstBase” and “freq” on input tracks.

- indexOfFirstBase: If indexOfFirstBase is specified, then a ruler is displayed above the TubeMap.
- Freq: If freq is specified, then the thickness of the path can be modified.

Sorry, I mistakenly understood. “indexOfFirstBase” and “freq” are not required as input of GFA json, but is optional in an internal data structure of STM. MoMI-G uses these features to modify the appearance of TubeMap.

Proposition: Communication Architecture modified from Simon’s MSc Thesis

After the web socket (ws) handshake took place, the whole communication regarding the visualization is tunneled through the ws. Every information exchange should be compressed. The event-based ws architecture reacts to incoming messages (IM). The type property of an IM determines the nature of the graph viz answer (GVA), the server has to formulate. Every answer is held in a JSON object.

GVA

Depending on the IM the client requested either a subgraph of a data set (at a certain position and path) to start browsing with (initial GVA) or an extension of the current graph view (given a node id to start at and an extension direction). In both cases the server collects the requested subgraph and two flags indicating if the left end or the right end of the whole variant graph was reached. In the initial GVA, other metainformation is added: a) The data set name, b) the type, c) the starting sequence position, d) all feature names of the whole variant graph, e) the sequence length of the reference path, ~~f) the 100 diverse RGB colors~~, and g) the minimum node id of the subgraph. The to ‘string’ converted (initial) GVA is sent via the ws to the client.

Client

The actual visualization of a variation graph is performed in the client’s browser whose machine provides the necessary calculation power for that task. The user’s starting point is the selection of a data set triggering the requests of

a) a starting subgraph element, b) a subgraph extension to the left (if possible) and
c) a subgraph extension to the right (if possible) sending outgoing messages (OM) in JSON format over the ws. Such a message states a) the current data set name, b) the start sequence position (only necessary to send, if a starting element is ordered), c) the type, d) if a starting segment is requested, e) the minimum node id of the current view, f) the maximum node id of the current view, g) the extension direction (if a starting element is ordered, this is undefined), and h) the name of the currently selected path. A JSON example of such a message can be seen in the following:

```
{
  "dataSetName": "human_mitochondrium"
  "startPos": 1,
  "type": 0,
  "startSegment": false,
  "maxNodeId": 0,
  "minNodeId": 0,
  "direction": undefined,
  "pathName": "NC_012920.1"
}
```

The server answers with messages of types GVA.

Graph Drawing Process

Generally, the drawing process of a variation graph is organized into an element by element approach: In the graph visualization the starting element (a received initial GVA) is drawn as the middle element and extensions to the left or right are drawn beside it. When processing such a single graph element, the layout of the given nodes, paths and features are calculated first. Then, the actual drawing is performed by usage of the D3.js library. An extension graph element is processed in the same way. Additionally, the extending graph element receives relevant information from neighbouring elements so that the visual connection of the element itself with its neighbours can be assured.

Memory Management

When browsing huge variation graphs, the visualization approach described so far would extend the current graph view with new subgraphs element by element. This would result in an extensive memory consumption. This issue is being dealt with as follows: If the SVG's width reaches 4000 pixel or more, the most left element (if the last element was added on the right) or the most right element (if the last element was added on the left) is discarded and not visualized any more. This ensures that only a limited amount of subgraph elements are hold in RAM. Elements not visualized or not reachable in a reasonable amount of time are therefore discarded. This minimizes the memory usage of the client side software.

Compressed Communication by Josiah

Some factors to consider in what gets communicated to the client:

- For performance, use the ws architecture to avoid resending redundant header information like dataSetName
- Alias all accessions into ribbons once, then only send ribbons, not all 10K+ accessions
- Nodes should have a suggested ordering, which could be defined by the "dominant ribbon" path listed first
- New nodes can be added to either end by simply listing their ribbon membership and the client can calculate the difference and pathing

Toshiyuki on parallelism of backend

I assume that even UCSC Genome Browser or Ensembl doesn't support for such simultaneous accesses for the server. If you need to boost the performance of the server (for example, in a tutorial session that more than thirty people access the backend server), just add hardware resources, i.e. scale out. So, I think we do not have to pay so much attention to parallelization in MVP. Parallelization comes along with troublesome lock or mutex, that makes codes unnecessary complicated in general.

Client-Backend API by Toshiyuki and Josiah

Backend Graph Specification #4
What requests can a client send to the backend-server? We will start with a Flask Websocket API.

How do we want the user to browse through the graph? If we display one subgraph and he browses e.g. to the right, there will be an end to that subgraph, but not to the whole graph. Suggestion:

1. We start with a central subgraph (CS) which is initially displayed to the user.
2. We make two further subgraph requests: One subgraph is appended left of the CS and one subgraph is appended right of the CS. These calculations shall be happening in a background thread so the fluent browsing is not interrupted.

Challenges: Align the tips of the graphs.

- Subgraph(accession, begin, end, zoom=Optional) returns a subgraph
    - Nucleotide position on a chromosome (given an accession path name)
    - All Accession names must be unique
    - Sanity checks for zoom level in place. Can be auto-calculated based on begin and end.
    - Return: a single JSON of graph layout
        - JSON format follows current TubeMap inputs
        - **T** Question: Should backend layout a small subgraph; or is it enough to ; because I am interested in rendering overall view like minimap
        - **T:** Currently, subgraph retrieving from vg backend is a bottleneck for the large subgraph in MoMI-G; first I will try XG library written by Erik to know whether
            - The subgraph returned by vg does not contain the coordinates of all path, that makes difficult to know the coordinates of all paths written in
            - I will check the current implementation
- gene(gene_name, accession=Optional) returns begin / end coordinates for gene name in a list of accessions
- GET '/overview' an overview for the whole graph (e.g. minimap or bandage) Second Phase Dev Milestone
    - Precalculate force directed graph layout OR render pixel image in zooming canvas
    - Return: a single JSON of graph layout (It is impossible to render in frontend.)
- download(accession, begin, end) download a PDF/SVG/PNG/ file of the subgraph
    - Return: a single file
- sources(accession, begin, end) GFA/gGFF/GAM(read alignments)

<WIP: We will rethink later.>

Backend Graph Specification #4

Content Migrated to Github Issue #4.

Backend Layers

- XG - Nucleotide level graph
- Precomputed Summary Levels
- API for serving user requests

Development Options

Issue #5.

Effect to the backend:
<Move it into github.>

Idea: **Starting point**
We can start with what we can do for now and what we have. Using two or more indices of XG, we can have several layers as a starting point of this prototyping. Also, vg can compute bubbles of the given graph; that is a kind of graph summarization.

- 1st layer: “full” graphs
- Following layers: summarized graphs at varying levels of detail by HaploBlocker

We can define a mapping between 1st layer and 2nd layer; that might be a “graph translation” in VG.
We can add another layers between 1st and 2nd layer, or “bird eyes view”.
Also, we need “wiser” subgraph retrieving system that can retrieve around the subgraph for caching, those ideas are already described above.

A simple backend service will do

1. Given “full” graphs, compute bubbles (by vg) and additional summarized graph layers (Josiah’s algorithm), and store them into multiple xg binary.
2. Precompute layouts of summarized graph layers on initializing.
3. Run as a daemon and respond JSON for the client’s request.

Progress Log