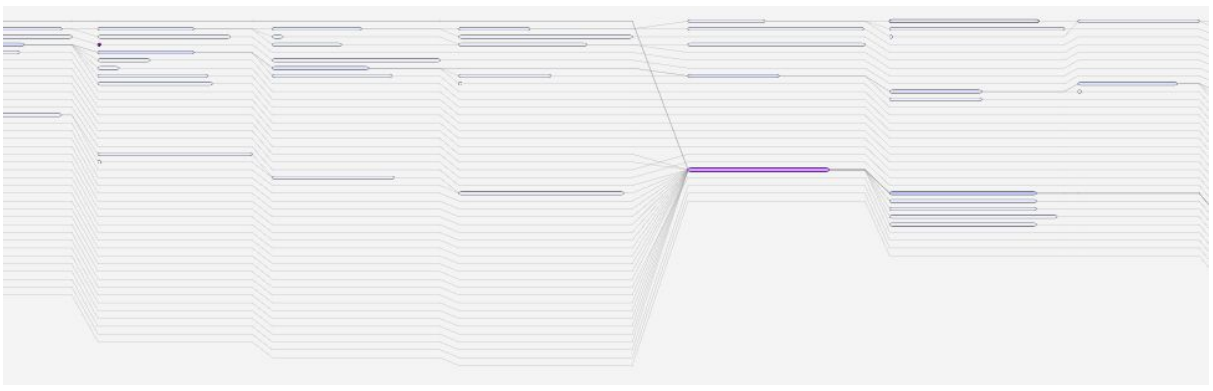
# Final Report

Desoxyribonucleïnezuur

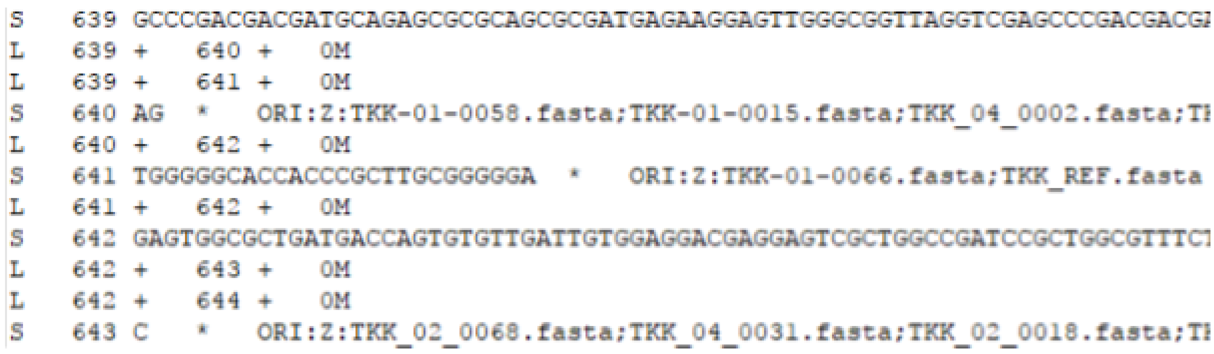


Toine Hartman - tjbhartman - 4305655
Martijn van Meerten - mvanmeerten - 4387902
Iwan Hoogenboom - ihoogenboom - 4396634
Yannick Haveman - yhaveman - 4299078
Ivo Wilms - iwilms - 4488466

# Introduction

## Problem description

As time passes, knowledge about different forms of life increases. Researchers are eager to investigate different species' DNA, but this is difficult because of the scale of details. Over time a method to 'read' sequences of DNA was developed. If sequences of multiple strains/variants of a species are digitized and merged, one can compare them. This data can be encoded as an acyclic directed graph, which models a set of genomes, where genomes consist of specified segments (i.e. nodes) and segments are connected by links (i.e. edges). This graph is presented in a file which uses the *Graphical Fragment Assembly* format. An example of a particular file, in this case containing two human genomes, is below.



```
S   639 GCCCGACGACGATGCAGAGCGCGCAGCGCGATGAGAAGGAGTTGGGCGGTTAGGTCGAGCCCGACGACG
L   639 +   640 +   0M
L   639 +   641 +   0M
S   640 AG  *   ORI:Z:TKK-01-0058.fasta;TKK-01-0015.fasta;TKK_04_0002.fasta;T
L   640 +   642 +   0M
S   641 TGGGGGCACCACCCGCTTGCGGGGGA  *   ORI:Z:TKK-01-0066.fasta;TKK_REF.fasta
L   641 +   642 +   0M
S   642 GAGTGGCGCTGATGACCAGTGTGTTGATTGTGGAGGACGAGGAGTCGCTGGCCGATCCGCTGGCGTTTC
L   642 +   643 +   0M
L   642 +   644 +   0M
S   643 C   *   ORI:Z:TKK_02_0068.fasta;TKK_04_0031.fasta;TKK_02_0018.fasta;T
```

A *GFA* file containing 10 tuberculosis genomes

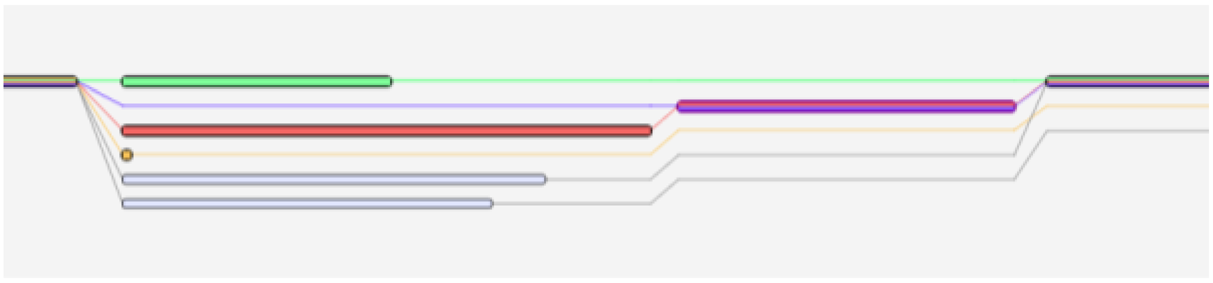
As researchers are looking for ways to compare genomes, it was asked to develop a software product to visualize and query genome graphs fluently, annotate them (several 'descriptions' are added by *General Feature Format*-files) and save interesting structures to review later.

## Structure

This document is the final report of the Desoxyribonucleïnezuur team, for the Programming Life context. The report presents the main functionalities of the final product and to what extent these functionalities satisfy the need of target users. The product has been evaluated using user testing and where it lacks, a failure analysis is performed. Furthermore, this document contains a section on interaction design. In this section, a user test is described, where a user interacts with the program using the think out loud method. The results of the experiment, and measures taken to improve on any problems encountered are also described. Finally, possible future improvements will be presented.

## Overview

The purpose of the product was to build a ‘multiple genome visualizer’. One of the main use cases is to get a better understanding of how multiple genomes of a certain species compare to each other. To see where they are similar and where they are different, but also to be able to highlight genomes (partially) based on specific criteria.



10 strains of the tuberculosis bacteria visualized

The product was designed for GenomeViz Inc. This corporation is doing research on differences in DNA sequences. They hope to further improve their knowledge on DNA with the created visualizer. They required an application that would visualize the differences in genomes in an intuitive way. This is done by using a directed acyclic graph. In this graph nodes represent a sequence of base pairs. Edges in this graph mean that two nodes with an edge between would in reality be connected.

## Reflection

This section describes our view on the product and process from a software engineering perspective.
Although we sometimes struggled to implement essential features, we are positive about our process. We tried to adhere to the essentials of the Scrum principle, where we specifically stuck to “Working software over comprehensive documentation” and “Responding to change over following a plan”. Sometimes we tried to implement things too generic and with too many edge cases is mind. This cost us a lot of time and did not always work out in the end.
Some software engineering methods were very successful for us during the course of the project. Pair programming for example, proved itself to be a very effective way to solve complex problems, or to think about special cases on simple problems. Also, codewise, we maintained strict rules about code quality and structure and we used the Github pull-request review functionality extensively to ensure quality and mutual understanding of the code base.

## Description

This section will go into the technical details of the implemented functionalities. One of the first functionalities implemented, is the ability to load and parse GFA files. The main challenge with this functionality is the size of these files. The data contained in these files cannot be loaded into memory, but instead has to be cached on a hard disc. This means loading files for the first time ('cold' load) writes a cache file to a hard disc, containing all the necessary data for visualizing the genomes. The cold loading time depends on several properties of the genome graph (mainly its size, the amount of genomes and the lengths of the sequences). In all subsequent loads of the same file, only a 'warm' load (reading the cached data) is required, which is near instant because the data was stored in an efficiently usable format.

The next functionality is drawing the actual graph. This is done with a near perfect layout, which means minimal edge crossings, and node clips (edges crossing nodes). Before drawing the graph, the necessary data needs to be loaded from the cache and into memory. After this the required segments need to be found. Which is done by finding children and parents of a given center node. To be able to layout these nodes, they are separated into layers. This creates a better structure in the graph and allows the algorithm to sort the segments, so that edge crossings are minimized. Another feature to minimize edge crossings is to create dummy nodes where genomes skip layers. These dummy nodes behave the same as normal nodes, but have the appearance of edges, and ensure that edges are cleanly directed around other drawn nodes and edges.

Another key functionality of the product is being able to navigate through the graph by panning, zooming, or direct center-node queries. Panning through the graph loads and removes nodes dynamically from memory to keep the application responsive and fast. Zooming in and out of the graph loads and removes nodes in a similarly dynamic way. There is a bound on the zoom level to keep the layout useful. To allow for more zooming and more intuitive layout, there is an option to toggle SNP. This will collapse simple bubbles in a semantic way, whilst retaining most of the same information. this will keep the graph from cluttering and confusing the user.

# Human Computer Interaction

Before finalizing the product, it was tested with a potential end user. A biologist from the USA interacted with the application and its features. This was done to find places where the usability of the product for the end users can be improved. From this test the existing features could be improved and essential quality of life features that the user missed could be implemented.

## Method

The participant for the test was a biologist who does research in the field the application was developed for. The participant has a need for this kind of application and is a potential user of the finished product. For the test, the think-out-loud method was used. The user would talk about what he was thinking and trying to do while performing certain tasks. The test was performed over skype and both visually and aurally recorded. The audio of the recording was transcribed to be fully anonymous, and the video extensively studied by all member of the team, after which the video and all other identifying data of the test was removed.

## Results

The user encountered several quality of life features that were missing. For example, after typing in the node to navigate to, the user is expected to click the draw button. Instead, the user tried to push the enter button to navigate. Another lacking feature was being able to highlight all genomes that go through a particular node. Performing this task would take the user about 10-15 clicks, while a drop-down menu with several options would be much more intuitive. When the user tried to identify which color represents which highlighted genome, the user first had to click on all edges and nodes with a particular color to find out what color represented what genome. The user gave as a suggestion, to display the genomes in the list in their respective colors.

The main feature where the product lacked severely was not having annotations. They stated that this is the most important feature for the application if it would really be used for research.

## Discussion

From the user test, multiple conclusions were drawn, the main one being the fact that the lack of annotation functionality makes the application virtually useless. Furthermore, a lot of suggestions for quality of life features were made. Several of these ended up in the final product (legend of genome colors, <enter> instead of clicking buttons, etc).
All in all, the user testing offered some insights on features end-users are expecting. It would have been even more useful if multiple users tested the application. There was also too little time between the user test and the deadline for the final product to implement all the suggested functionality.

# Evaluation

The final product consist of several parts and was developed step by step. Every step (milestone) will be evaluated separately.

## Parsing

The first step was to parse the provided GFA files into a directed graph structure. This parser is not the fastest, but it does cache all parsed data. This means that subsequent loads of the same file are instant and theoretically the GFA is not even needed anymore.

## Layout

Visualizing the graph is an important functionality of the final product. It is important the graph is laid out with minimal overlapping nodes and/or edges. The software uses an advanced algorithm to achieve this, which results in no overlapping nodes and no edges overlapping nodes. Sometimes edges do overlap nodes, but this is inevitable. All in all, the layout is near perfect.

## Dynamic loading

Due to the size of these files it is not possible to store all data in memory. Therefore, the application has algorithms for dynamically loading data from the cache and into memory and removing unused data. This make sure that while navigating through the graph, the application remains responsive and fast.

## Visual encodings

Visual encodings are the information that can be gained by just looking at visual clues. In the project these are the features: edge width, node gradient, and genome highlights. The gradient color and edge thickness make it easy to see which genome paths are frequent. On graphs with a large amount of genomes it is hard to find enough differentiable colors. In typical use, the user will never want to highlight more than ten genomes simultaneously. Therefore, the color for a genome is assigned on selecting the genome. This allows us to add colors that are as far away in terms of differentiability as the previously assigned colours. When multiple highlighted genomes are highlighted that go through a single node or edge, a rainbow structure applied to encode this.

## Bookmarks

To provide the user with a better experience whilst using the program it is possible to store bookmarks of points of interest. They can be stored at any location the user deems necessary to store. These bookmarks can then be loaded from a file at a later moment in time. Bookmarks can be opened and stored at any moment in time. However you can only store a bookmark of the graph that is currently loaded. A bookmark consists of a unique name and an optional description to signify the content of a bookmark. In our opinion the bookmark functionality provided is greatly incorporated into the program and is a great addition for the user.

## Annotations

We did not implement annotation functionality. This was due to the fact that we did not realise in time that without this functionality the program would be fairly useless in an actual setting. This is a big miss on our part, but since the short time left we decided to focus on making the program more usable than to try and implement this feature. If we would have to do the project again we would put way more priority on this feature.

# Outlook

This product was created as a project for a group of computer science students. This means that after this project they will most likely not continue working on this product. This doesn’t mean that the product is finished, just that the original developers won’t continue working on it. Someone else might inherit this product and improve on it. This section lists some features that could be added or improved. There will also be estimations for development times. Due to the nature of software development, giving a good estimation is not easy, certainly not without knowing the development team. The estimations assume a single developer who is reasonably familiar with the code. Note that implementing also includes writing tests, documentation, pull requests, and other miscellaneous tasks.

## Major features

### Annotations

Annotations are a major feature which, unfortunately could not be implemented due to lack of time (see Evaluation for reasons). Annotations are encoded in an `*.gff` file, and semantically represent things such as genes. They can be used as bookmarks to mark interesting properties within the genomes. The difference between annotations and normal bookmarks is that they represent a range of nucleotides within a genome, instead of a single node. They can also be created with other software, which makes them easy to share. Annotations also have more detailed information with different fields, which opens up the possibility for complex searches. The major feature annotations can be divided into several smaller features, which themselves can be divided into even smaller features:

- Parsing annotations. Not really a usable feature on its own, but required to make the other features work.
- Show annotations on graph.
  - This involves at least marking the start and end point
  - either show the annotation underneath the graph, or somehow show them flowing through nodes end edges.
  - show the name of the annotation in the graph.

    - select annotations (by clicking within the graph): highlight the path through the graph, show information about the annotation in a different pane.
    - Show annotations on minimap.
- Search through annotations using case insensitive substrings
    - specify the field to search in
    - go to annotation within graph (make sure that the full annotation is visible within the graph).

This is a major feature. Getting just the bare bones features (parsing, show list, go to location) working could be done within two weeks (80 hours), but implementing all the features above will probably take much longer, somewhere in the direction of a month.

### Semantic zooming

While the requirements for semantic zooming was met by just replacing simple bubbles with special SNP nodes, there is more that could be achieved in this area. One possibility would be to trigger a bubble collapse on a particular zoom level. This would improve the overall layout and keep the graph from cluttering. There are some more advanced features that could visually be replaced with one or more nodes, and it might even be an idea to give users the option to choose what structures are replaced. However, doing this would require a pretty big overhaul of the way that semantic zooming works, and therefore would take at least a few days, or much more if a lot of structures need to be replaced.

## Usability

One feature that would really improve the usability of the program is a global ability to click on things and then go to them. A quick idea for a way to do this is always open information when clicked, and have a button within the information to navigate to it in the graph. Another idea is using left mouse button to open the info, and using CTRL + left to navigate within the graph. Some examples:

- Nodes
    - parents / children in the info of another node.
    - parents / children in the info of an edge
    - mutations within an SNP.
    - Start and end nodes of annotations
- Genomes:
    - in the info of a node
    - in the info of an edge
    - in the info of an annotation
    - Another useful feature would be to highlight a genome from these places, without first going to the genome or to the genome highlighting window.

Implementing all of these would take about one week.

## Performance

The performance of the product is acceptable in almost all instances, but could certainly be optimized to provide an even smoother experience. That said, there are also some things that should certainly be faster, such as highlighting genomes, and the cold load of files.
Improving these should certainly be possible, but accomplishing a speed improvement is always a tradeoff with space in memory and cache. It also not possible to say what exactly is deemed acceptable for future versions of the program, so giving an estimate on the time requirements is especially hard for this feature. A very rough estimate would be about two weeks.

## Robustness

In the last few weeks the stability of the application has been improved greatly, but unfortunately it is not perfect yet. It also would be nice if the application would show user friendly error messages, with instructions on how to resolve the problem. The time for this is obviously very dependent on how stable is considered stable enough, but this could easily take a few weeks.