MASTER

Visualization of potato genome sequencing data

van Brakel, R.B.J.

Award date:
2009

Visualization of potato genome sequencing data

Remko B. J. van Brakel

Juny 2009

Thesis submitted in partial fulfilment
of the requirements for the degree of
Master of Computer Science and Engineering
at the Visualization Group
of Eindhoven University of Technology

Thesis committee:
dr.ir. H.M.M. van de Wetering
prof.dr.ir. J.J. van Wijk
dr. E.P. de Vink

Eindhoven University of Technology
Department of Mathematics and Computer Science
Eindhoven, The Netherlands

Abstract

Improvements in DNA sequencing techniques confront researchers with ever growing datasets, which have to be processed and annotated. Advances in computer power and graphical hardware have made it possible to visualize and explore genome size datasets on inexpensive personal computers. But the visualization of datasets of these sizes is no trivial task for a biologist or even bioinformatician.

This Master project tries to create a responsive visual interface that allows for the exploration of genome size datasets. This report describes the development and research of two contrary approaches of visualizing of these kinds of datasets. The first approach uses the high level DNA structure to provide an interface to the low level sequence and annotation data. The second approach focuses on the visualization of the low level annotation data, and tries to provide an interface to visualize the complete set of annotations within the dataset. Both visualization approaches are implemented as extensions to the DNAVis2 sequence browser.

Preface

This thesis is the result of my research done at the Faculty of Mathematics and Computer Science of the Eindhoven University of Technology. It serves to partially fulfill the requirements for the degree of Master of Science in Computer Science and Engineering (CSE). The subject of research is the visualization of potato genome sequencing data; can the existing DNAVis2 toolset be extended to visualize a genome size datasets? A number of extensions to the original DNAVis2 toolset will address this question; allow the visualization and exploration of genome size datasets.

The problem statement for this thesis is: "Extend the DNAVis2 tool-set with genome size datasets support, while preserving its near real-time interactive properties." The Plant Research International (PRI) at Wageningen University and Research Center (WUR) have provided a genome size dataset to validate the developed DNAVis2 extensions, and will provide additional support with their biological knowhow.

Many people supported and helped me during the process and there are a few I would like to address especially. First of all, I want to thank my daily advisor, dr.ir. Huub van de Wetering. His contribution to my understanding of the subjects and his input regarding this thesis has been of great benefit. We have had many discussions which led to considerable improvements of my visual concepts and final implementations. I also like to thank Jack van Wijk and Erik de Vink for their participation in the thesis committee.

Furthermore, I am grateful for my parents, sister and especially my girlfriend for their ongoing support and patience. Finally, I want to express my thanks to friends for their support, help and coffee sessions.

Contents

Chapter 1

Introduction

Fifty-six years ago, a one-page report published in the British science journal "Nature revolutionized science" described the three-dimensional structure of Deoxyribonucleic acid (DNA) [42]. This discovery solved one of biologies greatest mysteries: how genetic instructions are passed on from one generation to the next, and presented scientists with a consistent model to explain the observations that Darwin and Mendel made more than a century earlier.

This three-dimensional DNA structure has been a hot and well funded research subject for more than half a century, and will probably stay this way for years to come. The field still captures the imagination of many adjacent research fields and large companies, like the pharmaceutics industry.

The improvements in DNA sequencing techniques confront researchers with ever growing amounts of data. Projects like the human genome project, completed in 2003, determined the sequences of the 3 billion chemical base pairs that make up human DNA. A large part of the analyzing is still done by manual labor. But software tools can play an important role, by providing the biologist with fast and user-friendly interfaces to explore these large datasets. The advances in computer power and graphical hardware have made it possible to visualize and explore genome size datasets on inexpensive personal computers.

The visualization of these large datasets is not a trivial task for a biologist or even a bioinformatician. Therefore Plant Research International (PRI) at Wageningen University and Research Center (WUR) and the Visualisation expertise group of the Mathematics and Computer Science department of the Eindhoven University of Technology have established a cooperation in 2003. This multidisciplinary cooperation resulted in a visualization tool called DNAVis[1], which was capable of visualizing annotated DNA sequences. This tool was later rewritten in Java which resulted in DNAVis2, and later extended with a comparative genomic visualization[2].

The goal of the participation of PRI in the Potato Genome Sequencing Consortium (PGSC)[3] is to sequence the complete genome of a potato (consisting of approximately 840,000,000 chemical base pairs), and will require the aid of visualization software that allows them to browse and explore this genome size dataset. The current DNAVis2 implementation is designed to visualize single sequence datasets and has no knowledge about the higer level DNA structures. This master project continues the development of DNAVis2 by adding support for these higher level DNA structures, and will enable the visualization and exploration of a genome size dataset.

[1]A Master project by Peeters [32]

[2]A Master project by Willems [43]

[3]The PGSC is an international group of academic and industrial organizations that are committed to sequence the complete potato genome to meet the world's food needs in the future [11]

This report will describe three new DNAVis2 extensions and their underlying theory. The inception part consisting of chapter 2 will introduce the reader to some of the concepts of biology and bioinformatics with their associated terminologies, and describes a number of existing software tool sets. Chapter 3 defines the problem, research questions, requirements, and a number of use-cases. The chapters 4 and 5 describe two DNAVis2 extensions that focus on the structural information within a dataset. Chapter 6 is dedicated to the third DNAVis2 extension, which depicts the content within a dataset. In chapter 7 requirements and use-cases are verified and a conclusion is drawn in chapter 8.

Chapter 2

Domain analysis

This chapter will introduce the reader to the field of DNA research and bio-informatics. Starting with a brief introduction into the structure of DNA, sequencing techniques, and related terminologies. This knowledge will come in handy when the potato genome dataset and the associated file formats are discussed in the second part of this chapter. The third part will focus on the status of the DNAVis2 tool set at the beginning of this project, and the final part will compare DNAVis2 with a number of other existing tool sets.

2.1 Terminology

DNA which stands for "Deoxyribonucleic acid" is a nucleic acid that contains the genetic instructions used in the development and functioning of all known living organisms (and some viruses) [7, 5]. The world of DNA is quite an alien environment for most people with a computer science background, it has its own language with a lot of terminologies. This section will not try to explain the biological details, but focus on DNA's structures and the associated terminologies.

2.1.1 DNA terminology, the top-down-approach

This section provides a top-down-approach to explain the structure behind a number of DNA related terminologies, from the complete *genome* down-to the singular *A*, *C*, *G* and *T* *nucleotides*. These terminologies are based on the biological composition of the DNA structure.

- The *genome* refers to a full set of chromosomes and includes both the coding and the non-coding sequences of the DNA used to describe the genetic makeup of an organism. This genetic blue-print is stored in every DNA molecule of an organism.
- A *DNA molecule* is used for the long-term storage of the blue-prints (read source code) that is needed to construct an organism. The information within the DNA molecule is distributed over a small number of organized structures that are called chromosomes. These chromosomes are kept together by a membrane enclosure called the nucleus.
- A *chromosome* is an organized structure of DNA. Every chromosome contains a sub-part of the total generic blue-print of the organism. The number of chromosomes and their size varies extensively between different organisms. Every chromosome contains smaller DNA segments called genes.
- A *gene* is a segment of genetic information that, taken as a whole, specifies a trait of an organism.
- A *base pair* consists of two nucleotides on opposite complementary DNA strands. Each type of base on one strand forms a bond with just one type of base on the other strand, with *A* bonding only to *T*, and *C* bonding only to *G*.

- A *nucleotide* or base is composed of organic compounds that play a central role in the metabolism. DNA consists of four different compounds that are depicted by the characters A, C, G and T. Which are the abbreviations for: adenine (A), cytosine (C), guanine (G) and thymine (T).

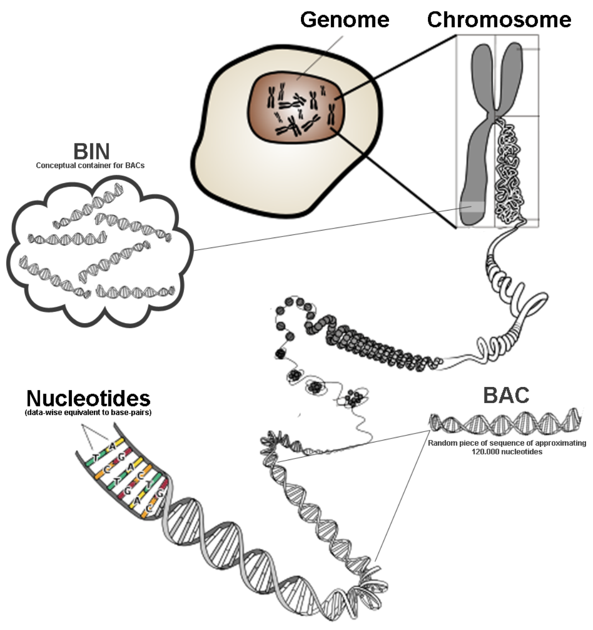

Genome
Chromosome
BIN
Conceptual container for BACs
Nucleotides
(data-wise equivalent to base-pairs)
BAC
Random piece of sequence of approximating 120.000 nucleotides

Figure 2.1: Overview of the DNA structures

2.1.2 DNA sequencing terminology, the bottom-up-approach

The term DNA sequencing encompasses biochemical methods for determining the order of the nucleotide bases of a segment of DNA. The DNA sequencing methods have evolved during the years, from manual gel-based procedures to modern automated processes. The dataset that is used during this project is sequenced on a so called "BAC by BAC"[1] basis, this is the same sequencing technique that has been used during the Human Genome Project [11, 10]. The terminology below is related to this sequencing technique.

- A *BAC* stands for Bacterial Artificial Chromosome, and is a small manageable part of a chromosome consisting of (on average) 120.000 nucleotides. Multiple BACs can be grouped together in a container called a BIN. It's not always clear to which BIN a given BAC belongs, often a possible BIN-domain is defined, while more extreme cases have no BIN information at all.

- A *BIN* represents approximately 0.8 centi Morgan (cM) [11], which is a measure to indicate a distance along a chromosome. The BINs are represented as containers that can contain a multitude of BACs. There is no sense of ordering for the BACs within a BIN, even the BINs have no real position information on a chromosome. BINs have a numbered position from 1 to n, where the numbers give some indication of the ordering.

- A *BIN-domain* is a group of one or more consecutive BINs. A BIN-domain is used when the exact BIN number of a BAC is uncertain.

[1]BAC by BAC bases sequencing is a technique that samples a DNA sequence in small subparts of (on average) 120,000 nucleotides at a time.

- A *chromosome* functions as a container for the BINs and implicitly as a container for the BACs of a subset of the dataset. The chromosomes are the highest level of data dividers within the dataset (read genome).
- The *genome* represents the total dataset of a single organism. The genome will be referred to as the dataset within this thesis: this reference will also be used although the sequencing of the potato genome is not fully completed yet.

2.1.3 DNA bioinformatics terminology

The BACs resulting from the DNA sequencing process can contain two types of lower level data, one describing the represented DNA sequence and the other the related properties.

- A *sequence*, is a digital representation of the natural DNA sequence. Sequences are usually represented in a plain text format, using the A, C, G and T characters to represent known nucleotides, and another character to indicate unknown nucleotides (depending on the used file format).
- An *annotation*, is a known or likely property represented by a part of a sequence. Most annotations are automatically generated by comparing the new sequences to a database of sequences with known properties (see figure 2.2). When a part of the new sequence resembles a known sequence-part above a certain threshold, its assumed that both represent the same genetic property. Only a small number of annotations are the work of direct DNA research, because this is extremely labor intensive [8, 9, 35].

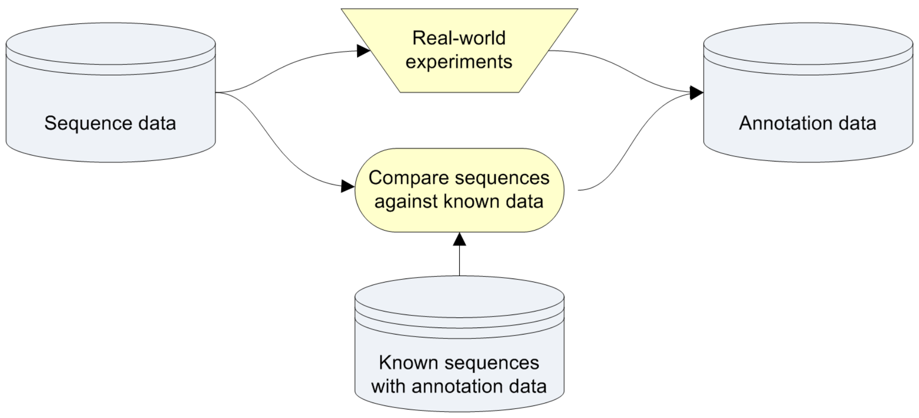

Sequence data
Real-world experiments
Annotation data
Compare sequences against known data
Known sequences with annotation data

Figure 2.2: Schematic overview of the annotations generation process

2.2 DNA datasets

This section describes the potato genome dataset that is used during this project, and will explain the structural composition of this kind of datasets.

2.2.1 DNA related datasets in general

DNA researchers are confronted by datasets consisting of thousands if not more sequences. The improvements in DNA sequencing technologies have accelerated the speed at which sequencing data is created, and future developments will only increase these data rates. This makes it inevitable that tool developers need to find better and faster ways to handle and visualize these dataset. Projects like the Human Genome Project [10, 8, 9] confront biologists and bioinformaticians with DNA sequences of about 3 billion base pairs, that have to be analyzed and annotated. These sort of datasets are useless without efficient tools to support the researchers quest for knowledge.

2.2.2 The composition of the dataset

The datasets that are created on a "BAC by BAC" basis are composed of five different data levels (see section 2.1.2): *genome*, *chromosome*, *BIN/BIN-domain*, *BAC*, and *sequence/annotation*. The data levels provide an easy means of abstracting the data at various levels, this can prove useful when visualizing genome size datasets.

The dataset will be split-up in two conceptual subparts: high level structural data, and low level content based data.

- The term *high level (structural) data* is used to indicate the *genome*, *chromosome*, *BIN*, *BIN-domain* and *BAC* data levels. These levels contain no content related data and are therefor treated as purely structural entities that can be used to navigate through the dataset as a whole.
- The term *low level (content based) data* is used to indicate the sequence and annotation data. This data level contains content based data which is linked to a singular BAC, this level contains no information about the higher level structures.

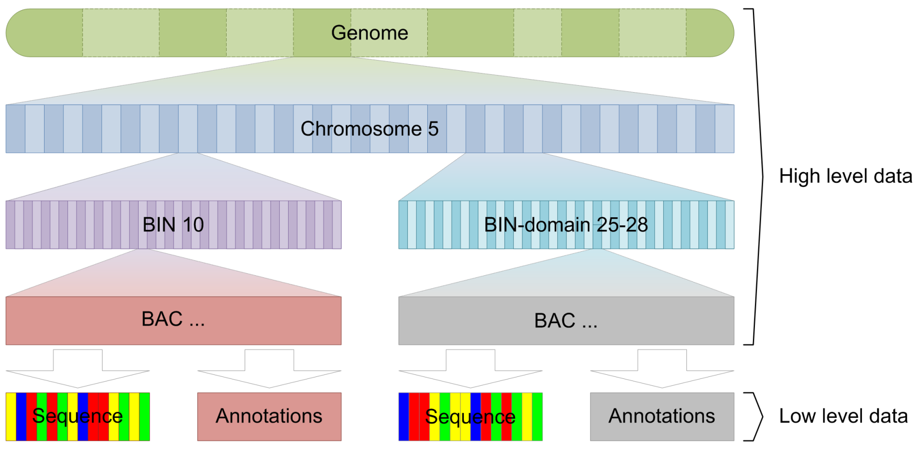


Figure 2.3: Overview of the dataset composition

The genome will be referred to as being the highest (abstraction) level within the dataset, as is represents the complete dataset, while the sequences and corresponding annotations will be referred to as lowest (abstraction) level.

2.3 DNA file formats

The PGSC dataset uses three different file formats to store the potato genome information.

- The low level sequence data is stored using the standardized FASTA file format.
- The low level annotation data is stored using the standardized GFF file format.
- The high level structural data is stored using a custom CBB file format.

The dataset consists of a large number of FASTA and GFF files to store the low level sequence and annotation data, while a single CBB file is used to describe the high level structural data. The individual file formats are briefly discussed in the following subsections.

2.3.1 The FASTA file format

The FASTA file format [33] is a standard used to store one or more[2] DNA sequences. The format is well documented and is widely supported by third party tools and software libraries. The FASTA files can be recognized by the following file extensions: `.fa`, `.mpfa`, `.fna`, `.faa`, `.fsa`, `.fas` or `.fasta`.

Listing 2.1: FASTA example

```
>RH056G12
gagagggtataagactttaagaaatctctgtcttcttcttctttagtgctatgacttgat
tccaattagtatttggtgatcccaataggtatctttcctacttcatttttcatgggtact
gaatctgaagattcttatgattttttttattgattgtcatgagctgcttcataagatgga
taaggtagagttatttggtgttgagtttatgatataccagttcaagggagacgccaaaat
gtggtggtggtctcatgttgagtgtcgactagtaaatgtactactcatgacttgggaaat
attttatagctttctatatggagaagtatataccctgaactttgaaaaataggagaagag
atccctaaatagatttgttgcaccctataaggccaaatttcatgttttattccaaatatg
gtacaccaccttgcttcactctaaaagagcaaatttgtttctttttgaagggattaaggc
```

Listing 2.1 shows a small part of a FASTA files from the PGSC dataset. The first row is the header of the file that provides the BAC name that belongs to the sequence, the rest of the listing lists the sequence data itself.

2.3.2 The GFF file extract

The GFF file format [19, 1] is a standard used to store the annotations (genes and other features) associated with a DNA sequences. GFF stands for "Gene-Finding Format" or "General Feature Format" and was proposed as a protocol for the transfer of feature information. A single GFF file stores a large number of annotations.

Listing 2.2: GFF example

```
6730472_protein BLASTX   similarity 1548 1692 . - . Target "gi|47824950|gb|AAT38724.1"; E_value 9e-56; Descriptio
6726706_protein IPRSCAN  similarity   49   61 . . . Target "IprscanSegDomain: seg"
6726706_protein IPRSCAN  similarity  179  200 . . . Target "IprscanCoilDomain: coiled-coil"
6726706_protein IPRSCAN  similarity  195  229 . . . Target "IprscanSuperFamilyDomain: Retrovirus zinc finger-like
6726706_protein IPRSCAN  similarity  205  214 . . . Target "IprscanPrintDomain: C2HCZNFINGER"; E_value 49.0
6726706_protein IPRSCAN  similarity  205  222 . . . Target "IprscanPfamDomain: zf-CCHC"; E_value 9.8e-05
6726706_protein IPRSCAN  similarity  206  222 . . . Target "IprscanSmartDomain: no description"; E_value 0.0097
6726706_protein IPRSCAN  similarity  207  220 . . . Target "IprscanProfileDomain: ZF_CCHC"; E_value 9.191
6726706_protein IPRSCAN  similarity  214  222 . . . Target "IprscanPrintDomain: C2HCZNFINGER"; E_value 49.0
```

Listing 2.2 shows a small part of a GFF files from the PGSC dataset. Every line in the listing provides one or more annotations that are associated to the sequence position described in the fourth and fifth column. The used filename of the GFF file is the only link to the associated sequence (read BAC name), there is no such information within the file data itself.

2.3.3 The CBB file format

The CBB file format (see appendix B.1) is a custom format used to describe the high level structure of the PGSC dataset. The file provides the chromosome name and BIN-domain that are associated with a BAC with a known location (hence the self conceived name "CBB" (Chromosome, BIN, BAC)). The format is only used in-house by the PGSC, and has no official documentation or tool support.

Listing 2.3: CBB example

```
"RH001P23";"NONE";"UNKNOWN";"#0000789";"0";"0";"NONE";"0";"0"
"RH001P24";"NONE";"UNKNOWN";"Singletons";"0";"0";"NONE";"0";"0"
"RH002A01";"NONE";"2";"#0006868";"60";"0";"NONE";"0";"0"
"RH002A02";"NONE";"9";"#0000873";"29";"5";"NONE";"0";"0"
"RH002A03";"NONE";"UNKNOWN";"#0004260";"0";"0";"NONE";"0";"0"
"RH002A04";"NONE";"11";"#0001969";"55";"3";"NONE";"0";"0"
"RH002A05";"NONE";"9";"#0000146";"31";"0";"NONE";"0";"0"
"RH002A06";"NONE";"UNKNOWN";"Chloroplast";"0";"0";"NONE";"0";"0"
```

[2]FASTA files with more than one sequence are called MultiFASTA files.

Listing 2.3 shows a small part of the CBB file from the PGSC dataset. The first column provides the BAC name, the third column provides the chromosome name (if known), and column five and six provide the BIN-domain (for more information see appendix B.1).

2.4 The Potato Genome Sequencing Consortium

The Potato Genome Sequencing Consortium (PGSC) [11] is an international group of academic and industrial organizations who are committed to the job of sequencing a complete potato genome to meet the world's food needs in the future. The genome is sequenced on a "BAC by BAC" basis, where a BAC is a sequence with an average length of 120,000 nucleotides, this results in about 7,000 individual BACs. The need for an overlap of 20% increases this total amount to 8,400 BACs within the final dataset. Given the fact that there are about 8,400 BACs and 12 Chromosomes, results in on average 700 BACs per chromosome. But in reality these numbers can fluctuate between 400-900 BACs per chromosome.

The PGSC potato genome is the "benchmark dataset" that has been used during this project. The PGSC dataset is not complete and still contains a lot of unnecessary data[3]. The current "work in progress" dataset consists of 78,854 known BAC names, of which 13,319 BACs have a known location based on a chromosome name and a BIN-domain. These are the BACs that are processed and visualized by this project, all BACs without a valid location are ignored. The DNA sequence and annotation data is only available for 179 BACs of these 13,319 BACs.

The 179 FASTA files in the PGSC dataset have a total size of 23MB, which results in about 400,000 lines (60 characters per line) of sequence data. This only represents about 2.3% of the total potato genome sequencing data. The 179 GFF files have a total size of 1960KB, which consists of 6420 unique attribute values divided over four attribute tag types (`Desciption`, `Blast_database`, `E_value`, `Target`).The CBB file contains 4767KB of plain text data, which is equivalent to 78,854 lines of data (one line per BAC) in the form as shown in listing 2.3 of section 2.3.3.

2.5 DNAVis2

The current version of DNAVis2 is designed to open and browse a small number of sequences at the same time [32, 43]. The environment enables users to explore a single sequences in detail using a view called a Linear View (see section 2.5.2), and provides a way to compare two sequences with each other within a Matrix View (see section 2.5.3). Both views will be described in more detail within the upcoming paragraphs. The supported file formats and internal data structures are also based on single sequence exploration (see section 2.5.1).

2.5.1 DNAVis2 data structures and file support

DNAVis2 is designed to view and browse single sequence FASTA file with one or more GFF files. The MultiFASTA[4] file can be opened, but only the first sequence is recognized. The relation between the FASTA file and one or more GFF files are defined in a custom XML based file format. These XML file points to the location where the FASTA and GFF files reside. An open-source library called BioJava [4] is used to read, store and interpret the FASTA and GFF data. This library is widely used by bioinformatics and well documented.

2.5.2 Linear View visualization

The Linear View is the most prominent visualization within DNAVis2, and enables users to visually explore the annotation data of a single sequence. This combination of a sequence with corresponding

[3]A large number of the BACs within the current dataset are theoretical and will be removed in the final dataset.
[4]A FASTA file with more than one sequence is called a MultiFASTA file.

annotation data will be referred to as a BAC, because the used PGSC dataset only contains sequences and annotations with the size of a BAC.

Overview

The visualization consists of a number of horizontal bars which are displayed onto a perspective wall [36]. The horizontal bars represent the following information, from top to bottom (see figure 2.4):

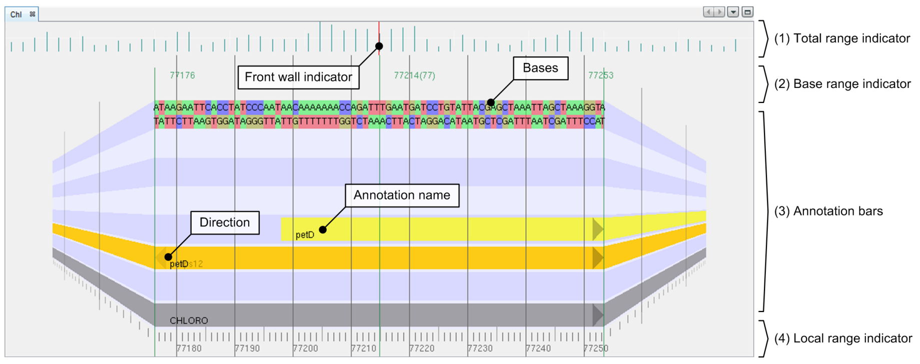


Figure 2.4: Single linear view and its various information bars

1. The top ruler displays the total range of the bases within the DNA sequence, while the red area indicates the part that is projected onto the front wall of the Linear View.

2. The tree numerical values indicate the start base position, the range, and the end base position of the DNA sequence that is currently displayed on the front wall of the view.

3. A number of annotation bars, of which the first is generally used to display[5] the base pairs of the DNA sequence. But the main purpose of these bars is to display the annotations that are associated with the displayed sequence positions.

4. The last ruler displays the base locations across the view, and gives some indirect indication of the zoom level of the current view.

Additional features

The user can explore the annotations by scrolling through the data at a desired zoom level. Both the scrolling and zooming actions respond fluently in near real-time. It's possible to link multiple Linear Views together to syncronise the zoom and scroll action. This can be used to view a single BAC file at multiple detail levels across multiple views (see figure 2.5), or to visually compare multiple BACs with each other.

Customization

The user can change the colors of the various annotations types, show or hide annotation types, switch between a small number annotation renders (flat bars, shaded bars, etc.), change the number of horizontal bars, and place the various annotations on the desired horizontal bar. These are all small alterations that enable the user to customize the visualization to his/her desired intentions.

[5]The base pairs are only visible at a sufficient detail level.

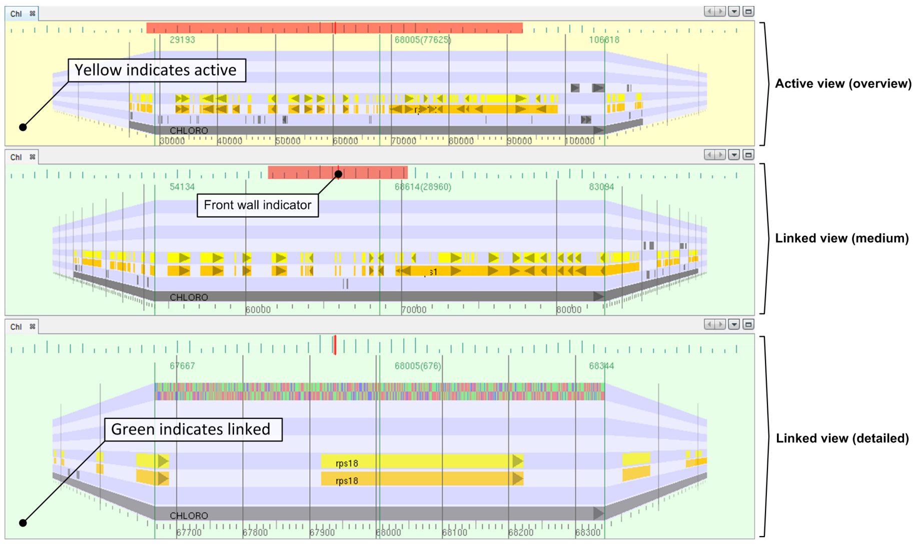


Figure 2.5: Multiple linked linear views displaying sequence and annotation data at different zoom levels

2.5.3 Matrix View visualization

The Matrix View is designed to compare two sequences and there corresponding annotations within a singular view. The view enables users to visually explore the similarities and differences between two BACs.

Overview

The visualization consists of two Linear Views with a dot-plot [34] in between. The dot-plot is created by placing the two BACs at adjacent sides of the table, and comparing the bases for every cell within the table. See figure 2.6 for a example of a small dot-plot based on the bases along the sides. Diagonal lines within the dot-plot indicate that there is a possible interesting relation between the two sequences. A top-left to bottom-right diagonal patron indicates that there is a subsequence that is equivalent in both sequences, such a equivalence is called a match. A bottom-left to top-right diagonal patron indicates a subsequence that is equivalent when one of the two subsequences is inverted, this phenomenon is called an inversion. Both the matches and the inversions are important for the biologists. Unlike the matches these inversions would be near imposable to detect by only using the standard Linear View side by side comparison technique.

For the small example (see figure 2.6) it's apparent that a dot-plot can provide the user with an easy to interpret visual guidance for finding interesting relations between two BACs. But the picture gets a bit harder to read when both sides consist of 120,000 bases, which is the case with real BACs. This results in a table with cells that occupy less than 0.01×0.01 pixels on the screen. So it's clear that the Matrix View has to have a few tricks up its sleeve in order to be useful for real-world data visualizations.

The Matrix View doesn't only compare the bases of the BACs, it also visualizes the equivalence between the annotations. These equivalences are indicated by red squares that are drawn on top of the dot-pot (see figure 2.7). The sizes of these squares are determined by the length of the matching subsequences belonging to the annotations.

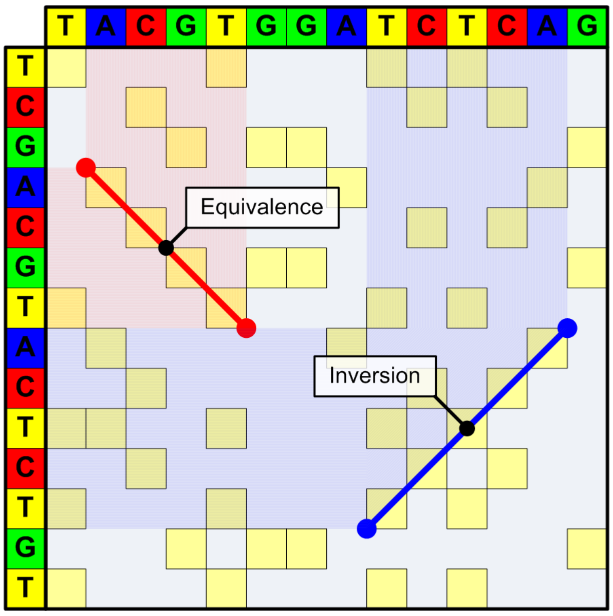


Figure 2.6: Conceptual drawing of a small dot-plot

Additional features

One way to overcome the abundance of data is to zoom in on a small subset. The Matrix View allows for seamless zooming and scrolling. Just like the Linear View, the dot-plot visualizes the relation between the currently visualized parts (read front wall) only. The zoom levels of both Linear Views are link together to keep the table in proportions, but the user is free to scroll to different positions within both views. This feature enables the user to view and compare the BACs in full detail. The Linear Views on both sides of the dot-plot can also be linked to external Linear Views providing a wider perspective.

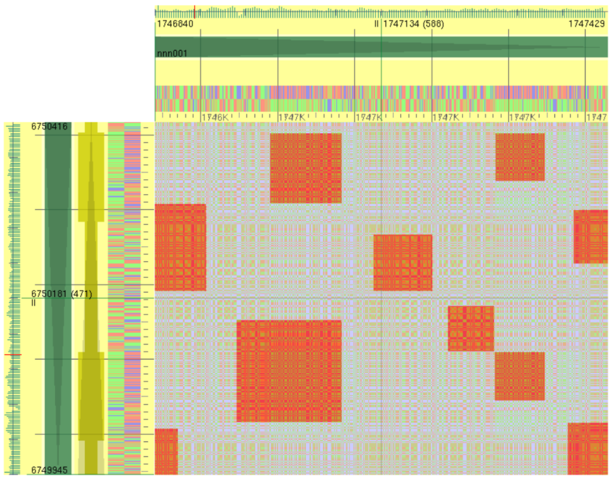


Figure 2.7: The matrix view with its two embedded linear views

Customization

The embedded Linear Views of the Matrix View suports the same customization features as described in section 2.5.2.

2.5.4 Additional views

Both the Linear View and the Matrix View visualization have a number of smaller mostly text based information windows that depict more detailed information about the annotations. Details like: annotation source, the quality score, notations in pain text, etc. Other windows provide the tools for searching and/or filtering specific properties. But all these tools are created for the exploration of single BACs.

2.6 Existing tool sets

This section presents an overview of a number of existing tools that enable the visualization and/or exploration of genome size dataset. Most genome browsers are web based and provide no real-time visual interaction, but a number of them provide quite impressive visualizations.

2.6.1 Circos

The Circos toolset [23] is a web based visualization tool, its creation was motivated by the need to visualize intra- and inter-chromosomal relationships within one or more genomes, or between any two or more sets of objects with a corresponding distance scale. The Circos web-server provides example datasets and an interface to generate publication-quality circularly composited renditions of genomic data and related annotations (see figure 2.8).

The tool has special features to improve the visualizations of the genome data. A genome is a large structure with localized regions of interest, frequently separated by large oceans of uninteresting sequence. The Circos tool improves the visualized data with the help of variable axis scaling, permitting the local magnification of genomic regions without cropping. Scale smoothing ensures that the magnification level changes smoothly. In combination with axis breaks and custom ideogram order, the final image can be easily tuned to offer the clearest illustration of your data.

The tool produces very artistic visualizations, but this is all what the tool has to offer. It provides no interactive interface to browse and explore the dataset.

2.6.2 Ensembl

The Ensembl project [13] produces a genome databases for vertebrates and other eukaryotic species, but it also offers an impressive web interface to browse and explore genomic data and related annotations. The web interface is freely available and is used by a number of other projects, the example screenshots as seen in figure 2.9 are taken from the Vega project [20] which has developed a web front-end derived from Ensembl web code.

This web tool provides at least the same functionality as the DNAVis2 toolset, and adds high level data support on top of that. But the tool has also a number of disadvantages, first of all is a web based tool requiring expensive server hardware to run, second it provides a fast interface but its still limited to web based technologies meaning no seamingless zooming or scrolling etc.

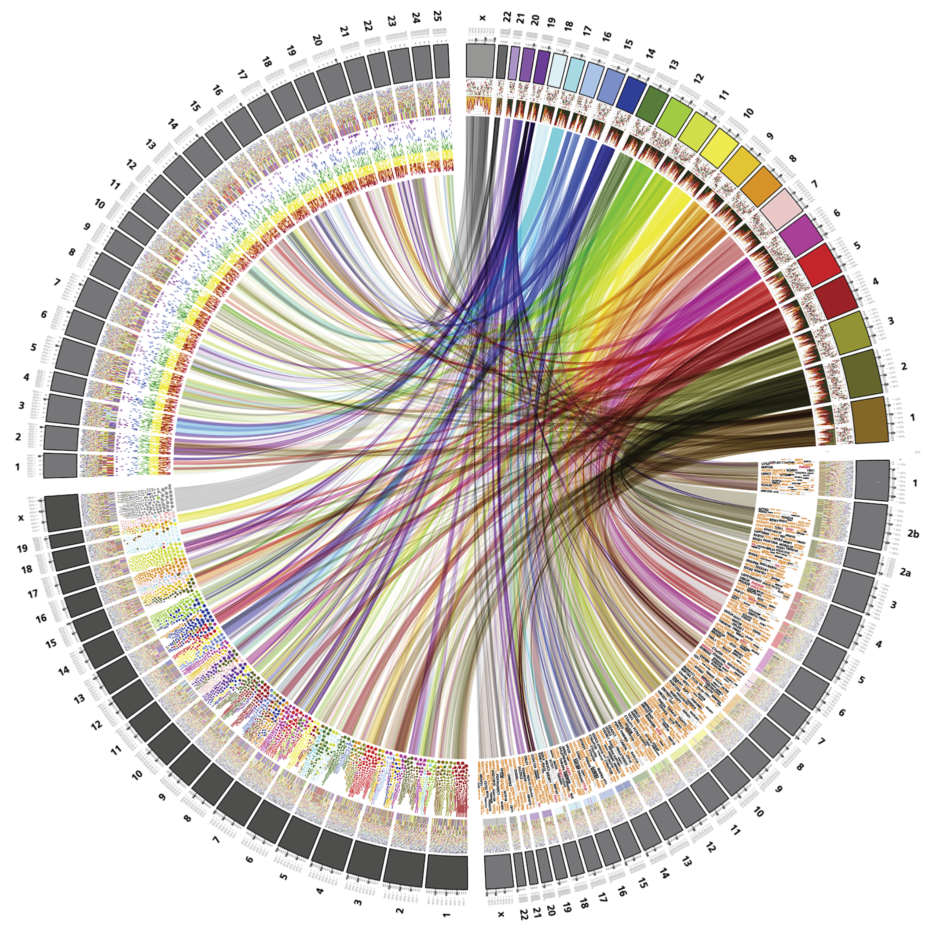


Figure 2.8: A Circos visualization displaying the annotation relations between four genomes

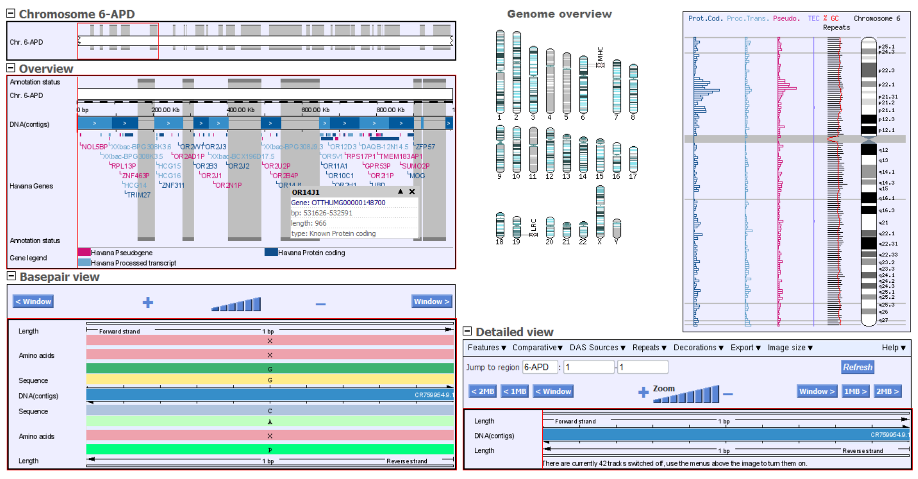


Figure 2.9: Vega displaying the sequence and annotation data of a human chromosome

2.6.3 VISTA

VISTA [24] is a comprehensive suite of programs and databases for the comparative analysis of genomic sequences. The VISTA databases offer a number of complete genome alignments that can be visualized using both the web-browser viewer as the newer Java applet viewer (see figure 2.10).

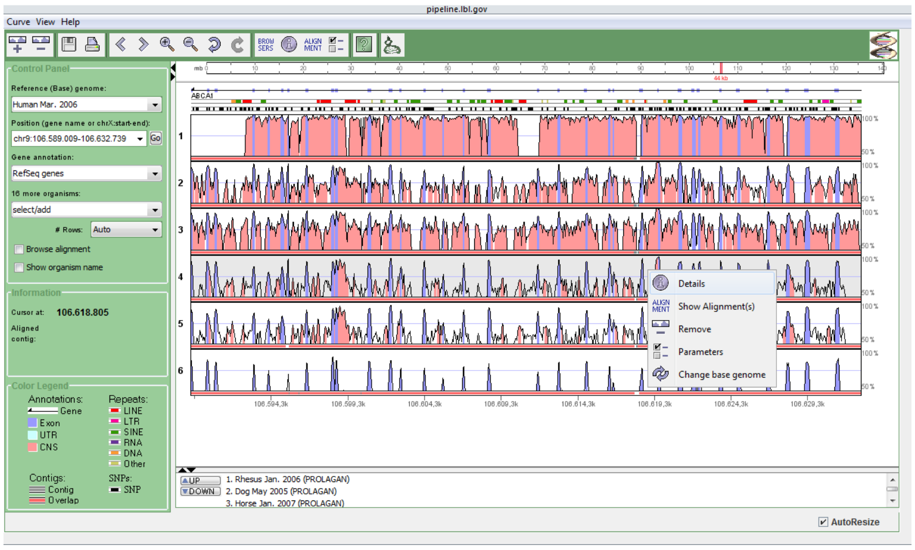

Reference (Base) genome: Human Mar. 2006
Position (gene name or chrX:start-end): chr9:106.589.009-106.632.739
Gene annotation: RefSeq genes
16 more organisms: select/add
Rows: Auto
Cursor at: 106.618.805
1. Rhesus Jan. 2006 (PROLAGAN)
2. Dog May 2005 (PROLAGAN)
3. Horse Jan. 2007 (PROLAGAN)

Figure 2.10: VISTA displaying the sequence alignments of multiple human chromosomes

But both viewers are purely designed to research alignments, and are not suited for the browsing and exploration of genome size datasets. The Java applet version of the viewer is still very browser depended, and opens all additional information in separate browser windows.

2.6.4 Argo

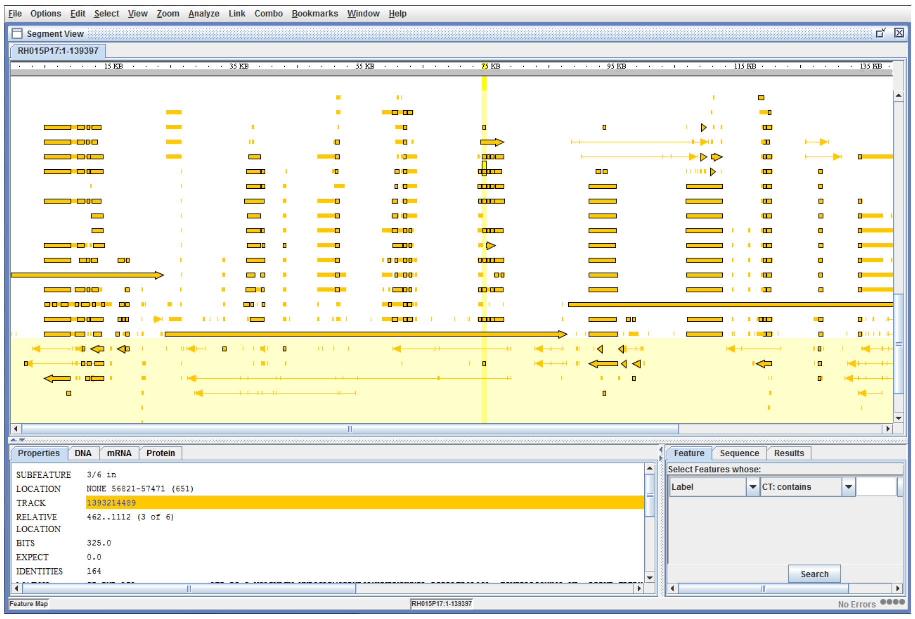
The Argo genome browser [18] is not entirely what the name suggests, is a basic annotated sequence browser (see figure 2.11) like the basic DNAVis and DNAVis2 implantations. The tool is Java based and sports a wide range of file formats.

The visualization of the tool resembles the linear view of DNAVis2, but looks far less impressive. The interface is also less intuitive because simple navigation actions like zooming and scrolling have to be done using the menu, keyboard or scroll bars. The mouse is only used for data selection, within the visualization window.

2.6.5 Overview

Non of the existing tools where able to provide a near real-time interface to explore the data properties of a genome size dataset. The Vega implementation of the Ensembl web based genome browser (see section 2.6.2) provides the best interface of all the tested tool sets. It is responsive, able to provide a overview of the genome, and provides support for low level sequence and annotation exploration. But the web interface limits it's ease of uses. Table 2.1 will provides a short overview of the tested tool sets with respect to the following properties:

1. Supports high level data structures
2. Supports sequence data



SUBFEATURE	3/6 in
LOCATION	NONE 56821-57471 (651)
TRACK	1393214489
RELATIVE LOCATION	462..1112 (3 of 6)
BITS	325.0
EXPECT	0.0
IDENTITIES	164

Figure 2.11: Argo displaying the RH015P17 sequence and annotation example data of DNAVis2

3. Supports annotation data
4. Visual representation of high level data
5. Visual representation of low level data
6. Browsing of high level data
7. Browsing of low level data
8. Near real-time interface

Table 2.1[6] provides a short overview of the tested tool sets against the requested properties discribed in the enumeration above:

	1	2	3	4	5	6	7	8
Circos	+	+	+	+	+	-	-	-
Ensembl	+	+	+	+	+	+	+	-
VISTA	-	+	+	+	-	-	+	-
Argo	-	+	+	-	+	-	+	+

Table 2.1: Overview of the use-case validations per visualization

[6]Table symbol legend: + positive validation, - negative validation.

Chapter 3

Problem description

The first part of this chapter provides the problem statement with the corresponding research questions, and gives some inside into the challenges that arise when handling and visualizing genome size datasets. The second part defines the requirements and provides the use-cases to evaluate the final solution(s).

3.1 Problem definition

This section defines the problem statement and proposes a number of research questions.

3.1.1 Problem statement

Modern sequencing techniques like "BAC by BAC" sequencing confronts biologist with a complex mixture of high level structural properties and low level sequencing and annotation data. The DNAVis2 tool-set is currently only equipped to handle the low level sequencing and annotation data, but provides no support for the associated structural properties. The goal is to extend DNAVis2 to provide the user with a detailed insight into the structural properties and annotation data of these large datasets, while preserving the real-time nature of the current DNAVis2 visualizations. This leads to the following problem statement:

> *Extend the DNAVis2 tool-set with genome size datasets support while preserving its near real-time interactive properties.*

3.1.2 Research questions

The following research questions have to be answered to fulfill the proposed problem statement:

> *RQ1: Is it possible to store and analyze a genome size dataset within the constrained environment of a modern personal computer?*

This first research question will determine if a genome size dataset can be opened and processed within a Java environment on a modern personal computer. It's the first critical step towards visualizing a genome related dataset.

> *RQ2: How does one visualize the structural properties of a genome size dataset?*

This second research question focuses on the visualization of the high level properties of the dataset. The current DNAVis2 implementation has no notion of higher level structures, so this has to be addressed to provide an interface to the low level data.

> *RQ3: How does one visualize the annotation data of a genome size dataset?*

This third research question has the goal to find a visualization that enables the exploration of a large number of annotations. The goal is to enable the user to explore the annotation properties on a global scale, without having to study the low level data elements individually.

> *RQ4: Is it possible to preserve the responsive and interactive feeling of the current DNAVis2 when visualizing genome size datasets?*

The fourth and final research question is focused on the overall performance of the new visualizations. The new visualizations will try to provide the same level of interaction and responsiveness as the original DNAVis2 linear view and matrix view.

3.2 The dataset challenges

The DNAVis2 data structures are not able to store and analyze high level structural information. The PGSC dataset consists of two types of data: the high level purely structural data, and the low level sequence and annotation data. The only cohesion between the high level and low level data are the BAC names. The following challenges have to be met, before research question *RQ1* and *RQ4* (see section 3.1.2) can be answered, and a genome size dataset can processed.

- The first challenge is the merging of both data types in such a way that the high level data becomes aware of its associated low level data, and the low level data becomes aware of its location in the bigger scheme. This makes it possible to explore the distribution of annotations across the various data level (for example the distribution of a certain annotation within the genome at chromosome level), or to request the location of a specific annotation.
- The second challenge is to find an efficient way of creating, storing, and presenting the vast amount of information that is the result from this merging. The speed of the internal data representation will play an important factor in the interactive visualizations.

3.3 The visualization challenges

The existing visualizations of DNAVis2 are not capable of displaying high level structural information, or vast amounts of low level annotation data from multiple sources. The new visualizations need to address the following challenges; answer *RQ2*, *RQ3* and *RQ4* (see section 3.1.2) and enable the visual exploration of genome size datasets while providing the same level of interactivity and responsiveness as the original DNAVis2 visualizations:

- The first challenge is to embrace the high level structural data to provide the user with a visual interface to the low level annotation data.
- The second challenge is to visualize the low level annotation data from a large number of BACs in a single visualization.
- The third challenge is to produce an interface that stays responsive while handling these very large datasets within the constrained environment of a modern personal computer.

All these challenges have to be met in order to provide the user with a user-friendly and responsive interface to explore a genome size dataset within the DNAVis2 toolset.

3.4 Requirements

This section contains a number of requirements that have to be met by the new DNAVis2 extensions. Some of these requirements are clear goals that have to be met at the end of this project. Others are still a bit vague but will get clearer in a later stage of this project. These requirements are not sorted in order of importance.

3.4.1 The data set requirements

The following section describes the dataset related requirements.

Requirement I (Data size) *The user should be able to browse a data set of arbitrary size.*
The goal is to enable datasets as large as a whole genome within the DNAVis2 tool set, while remaining responsive and user friendly.

Requirement II (Standardized data files) *The user can load sequences and annotations described in the standardized FASTA and GFF file format.*
The DNAVis2 will use the FASTA and GFF file standard to represent the DNA sequence and corresponding annotations. The individual FASTA and GFF files correspond to a single BAC within the dataset. These file formats are already implemented in the existing DNAVis2 implementation but possibly need some optimizations to allow genome size datasets to be loaded.

Requirement III (Custom data files) *The user can load the mutual relations between the chromosomes, BINs and BACs as described in the custom CBB file format.*
The standardized FASTA and GFF file format lack any form of high level structural data. Therefore the custom CBB file format has to be introduced to the DNAVis2 tool set, to describe the high level structural properties of the BACs. *(BACs with unknown positions are ignored.)*

3.4.2 The detail levels requirements

The following section describes the requirements of the various data abstraction levels.

Requirement IV (Genome level) *The user can view and browse the individual chromosomes in a genome.*
The user will be presented with a visual representation of all the known chromosomes within the data set. This visualization is the highest level representation of the data set, and can be used as a starting point for the exploration of lower data levels.

Requirement V (Chromosome level) *The user can view and browse the individual BINs and BIN-domains within a chromosome.*
The user will be presented with a visual representation of all the known BINs and BIN-domains within a chromosome. This visualization is the second highest level of data representation. The user is free to move back to the previous (higher) level, or to the next (lower) level of data representation.

Requirement VI (BIN/BIN-domain level) *The user can view and browse the individual BACs within a BIN or BIN-domain.*
The user will be presented with a visual representation of all the BACs within a BIN or BIN-domain, and is free to move back to previous (higher) level, or to the next (lower) level of data representation.

Requirement VII (BAC level) *The user can view and browse the sequence and annotation data belonging to a BAC.*
The user will be presented with a visual representation of all the information that is known about the BAC in question. This visualization can simply reuse the already implemented Linear View, which enables the user to view and explore the sequence and annotations as described in section 2.5.2.

3.4.3 The user interaction requirements

The following section describes the user interaction related requirements that have to hold for the DNAVis2 extensions.

Requirement VIII (Navigation) *The user should be able to navigate through the complete dataset in an interactive and continuous way.*
The user will be able to move freely through the various abstraction levels, enabling the user to browse the complete dataset.

Requirement IX (Overview) *The user needs to be kept location aware and context aware when browsing the dataset.*
The user will always be presented with a visual or textual clue that describes the location in the hierarchy and the type of data that is visualized.

Requirement X (Responsiveness) *The user should keep a sense of direct feedback after a interaction.*
The DNAVis2 environment will present the user with direct feedback after a user action. This can be by providing the requested information within a very short time span, or by a visual indication that the tool set is computing the requested information. Requests that take some time to compute will provide the user with intermediate or partial results when feasible.

3.4.4 The data manipulation requirements

The following section describes the data manipulation requirements that have to hold for the DNAVis2 extensions.

Requirement XI (Selection) *The user can select and deselect subparts of the dataset on all data abstraction levels.*
The user can select and deselect sub-elements (chromosome, BIN/BIN-domain and BAC) of the dataset on all the available data (see section 3.4.2) abstraction levels.

Requirement XII (Filtering) *The user can filter the data on a large number of criteria.*
The user is provided with a wide range of filter options to emphasis or suppress certain properties, or to partly automate the selection or deselection of sub-selections.

Requirement XIII (Emphasizing) *The results of a selection and/or a filter action are visually emphasized on all detail levels.*
The selection (manual or by filtering) of parts of the dataset will be visualized on all the data abstraction levels, to support the user when browsing through the levels in order to create subsets.

(Example: Select all BACs in a chromosome minus the BACs in the first BIN, by selecting the chromosome in the highest level, and subsequently go down one level to deselect the first BIN. This enables the user to make a sub-selection of the dataset in easy and visual intuitive manner.)

Requirement XIV (Data distribution) *The distribution of the various data elements can be visualized on all detail levels.*
The user can visualize the distribution of high level or low level data elements across the dataset as a whole, or on one of the numerous abstraction levels.*(Example: visualize the BAC distribution across the BINs in a specific chromosome.)*

Requirement XV (Automatic data extraction) *The tool will automatically extract data properties and presents them to the user.*
The DNAVis2 environment will automatically extract a number of possibly interesting data properties. Like the frequency of occurrences of the various data properties or the listing of a special kind of annotation data.

3.5 Use-cases

This section provides a number of use cases, used to verify the overall functionality of the extended DNAVis2 tool set. *(See appendix C section C.1 and C.2 for the corresponding UML use-case diagrams.)*

Use-case I (BAC distribution) *Visualizing the BAC distribution*
The biologist opens the CBB file that describes the high level structural properties of the PGSC dataset within the DNAVis2 tool set, and visualizes the BAC distribution across the highest level elements within the dataset.

Use-case II (Exploring high level) *Exploring the high level structure*
The biologist opens the CBB file that describes the high level structural properties of the PGSC dataset, and navigates through the various abstraction levels using a singular visual environment of the DNAVis2 tool set.

Use-case III (Selection) *Creating a sub-selection*
The biologist opens the CBB file that describes the high level structural properties, and creates a sub-selection by selecting and deselecting data elements at the various abstraction levels using a singular visual environment within the DNAVis2 tool set.

Use-case IV (Common annotation) *Find the most common annotations*
The biologist opens all the available GFF files of the BACs with a known chromosome and BIN-domain, visualizing the number of times a annotations occurs, and finding the maximum values within the dataset.

Use-case V (Specific annotation) *Find all BACs with a specific annotation*
The biologist opens all the available GFF files of the BACs with a known chromosome and BIN-domain, and visualizes the BACs that contain a specific annotation.

Use-case VI (Specific text) *Visualize annotations containing specific text*
The biologist opens all the available GFF files of the BACs with a known chromosome and BIN-domain, and visualizes the annotations that contain a specific text or symbol.

Chapter 4

Chromosome, BIN and BAC visualization

DNA related information is highly structured, but the size can be overwhelming at the lower abstraction levels.The structured nature makes it possible to view the data at various abstraction levels. The chromosome, BIN and BAC (CBB) visualization tries to embrace the highly structured characteristic to create an intuitive and browsable graphical model, by incorporating various abstraction levels into a single visualization. Providing the user with a picture that enables a better understanding of the dataset. The visualization has no knowledge of the existence of low level sequences or annotation data. The main purpose is to describe the distribution of the high level elements within the dataset, especially the distribution of the 13.319 BACs.

4.1 The CBB visualization

The genome being the complete dataset is not explicitly depicted by this visualization. The chromosome, BIN, BIN-domain and BAC levels are translated into a visualization, starting with the chromosome as the highest level.

4.1.1 Chromosome visualization

The chromosomes are the biggest sub-elements within the dataset, and are depicted using large horizontal bars that divide the view into a number of vertically stacked segments. The number of segments is equal to the number of chromosomes within the dataset. Each horizontal bar is labeled with the name of the chromosome that it represents (see figure 4.1). This layout provides the maximum amount of horizontal drawing space. To ensure an adequate amount of vertical space, a technique called X- and Y-distortion [36] is used, whereby the bar with the users focus is extended horizontally and vertically, while all other bars are slightly compressed (see figure 4.2). This makes it possible to keep all information in view, while providing enough space to depict the user's point of interest in sufficient detail.

4.1.2 BIN visualization

The notion of BIN numbers are depicted by drawing a ruler like texture across the whole surface of the chromosome bars (see figure 4.3). Every horizontal color change depicts a single BIN, every tenth BIN number is indicated by a subtle deviation in color scheme, and is labeled with the corresponding bin number.

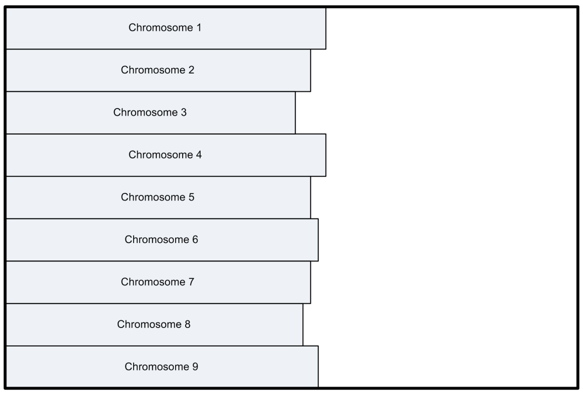


Figure 4.1: Concept drawing of nine chromosomes without user focus

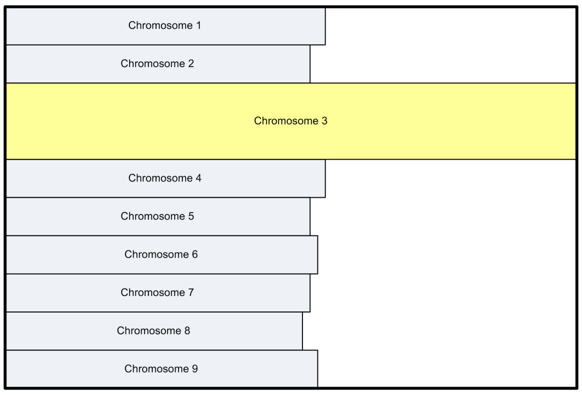


Figure 4.2: Concept drawing of user focus on chromosome 3

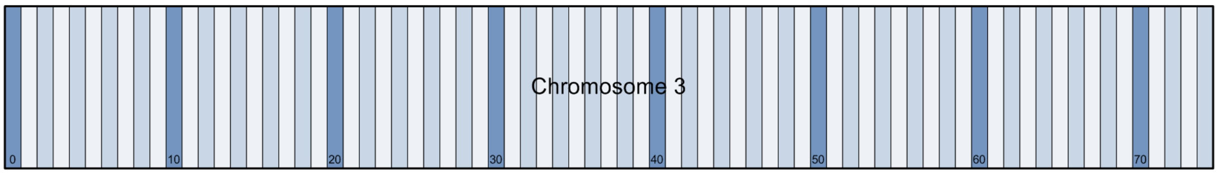


Figure 4.3: Concept drawing of the ruler like texture, indicating the BIN numbers on the chromosome bar

4.1.3 BIN-domain visualization

The BIN-domains are translated into horizontal bars that indicate the size of the BIN-domain (see figure 4.4). These horizontal bars are drawn on top of the chromosome bars, the horizontal position indicates the BIN numbers that correspond with BIN-domain. The BIN-domain bars are sorted by the X-position of their first BIN number from the lowest to the highest number. The BIN-domains with equal first BIN-numbers are sorted by their BIN-domain size, from large to small. Whenever a BIN-domain bar is added, it is placed at the lowest non-overlapping vertical position onto the chromosome bar, resulting in stacks of BIN-domain bars with a smooth left side and possible irregular right size.

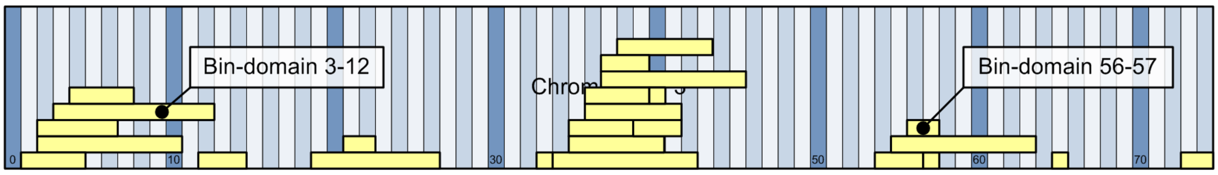


Figure 4.4: Concept drawing of the BIN-domains depicted on-top of the chromosome bar

The limited vertical space provided within the chromosome bars, makes it impossible to depict every single BAC by drawing its BIN-domain. The clustering behavior combined with the small BIN-domains, results in hundreds of BACs that share the same BIN-domain. So drawing every single BAC will result in very high stacks of BIN-domains. At the same time, this sharing of the same BIN-domain provides a convenient way to reduce the number of elements, by combining all the BACs with the same BIN-domain into a single visual element. Resulting in a far smaller number of BIN-domains that have to be mapped onto the chromosome bar (see figure 4.5, step (A) to (B)).

To prevent data loss as a result of combining multiple BACs into a singular element, some additional

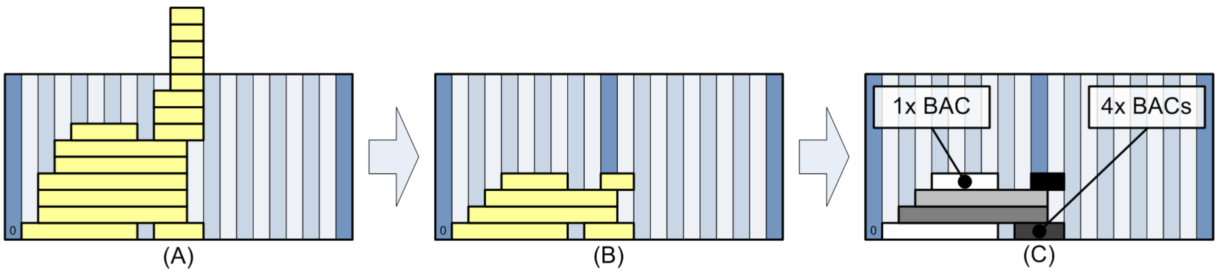


Figure 4.5: Concept drawing of the element reduction and the color-mapping

information about the data densities of each element has to be added. In this case the color of the BIN-domain bar is used to indicate the number of BACs represented by a particular bar (see figure 4.5, step (B) to (C)). The merging of BACs breaks the one to one mapping of the visualization, but the resulting visualization is still able to provide an intuitive representation of the overall data structure and data distribution (see figure 4.6). While it's not possible to directly view or interact with individual BACs, it's no real impediment to the objective of the view (showing the structure and distribution of data within the dataset). The use of gray scale color-mapping leaves some room to use color to indicate selections or the presence of lower level properties.

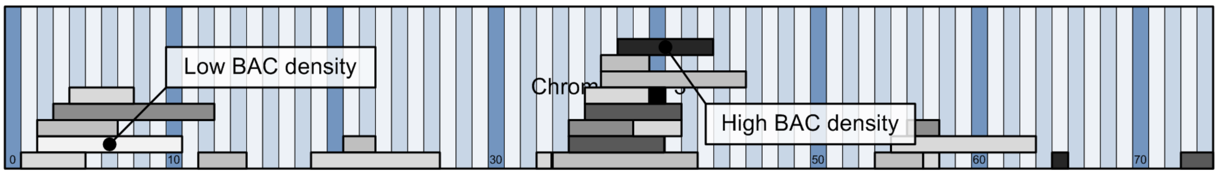


Figure 4.6: Concept drawing of the BIN-domains with gray scale color-mapping

4.1.4 Interaction

The user can interact with the visualization by selecting or deselecting data elements at the various levels, providing the user with an easy and fast way of creating sub-selections within the dataset. These sub-selections are visualized by (for example) coloring the borderlines of the selected element (see figure 4.7). The sub-selections can be used in views that are aimed at lower level data visualization.

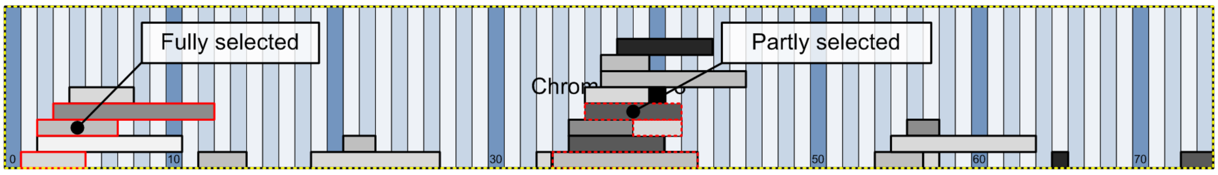


Figure 4.7: Concept drawing of BIN-domain and chromosome selection indication using colored borders *(A dotted border indicates partly selected and a full color border indicates fully selected. A red color is used for BIN domains and yellow for chromosomes.)*

4.2 Results

This section provides an overview of the implemented CBB visualization. The CBB visualization is implemented as a extension of the DNAVis2 tool set. The implementation details of the CBB visualization extension are described in appendix D.

The current implementation supports the CBB file format as described in section 2.3.3, which is assumed to use the `*.cbb` extension. The CBB visualization does not scan or read the GFF and FASTA files that are associated with the CBB data. The default DNAVis2 file browser (see figure 4.8) is used to open the CBB files, these can be recognized by their blue CBB icon (see figure 4.8).

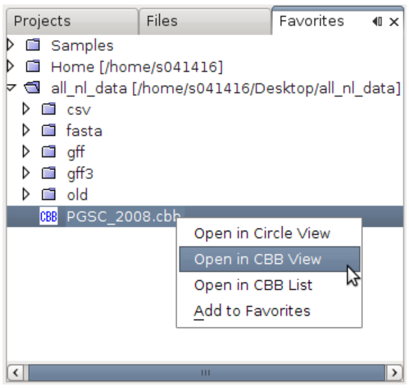


Figure 4.8: Screenshot of a CBB file in the DNAVis2 file browser

The default CBB visualization implementation uses grey scale mapping to indicate the number of BACs within a BIN-domain. Selected and partly selected chromosomes and BIN-domains are indicated by colored edges, red for fully selected elements and yellow for partly selected elements.

Figure 4.9 provides a screenshot showing the PGSC dataset within the CBB visualization. Chromosome 4 is in focus and is therefore depicted larger than the other chromosomes. Chromosome 3 is fully selected and chromosome 5 is partly selected, all other chromosomes contain no selected data elements.

The control window provides a number of selection options to ease the selection process:

- *Select all*: Selects all the data elements within the CBB visualization.
- *Unselect all*: Clears the selection within the CBB visualization.
- *Invert all*: Inverts the selection within the CBB visualization.

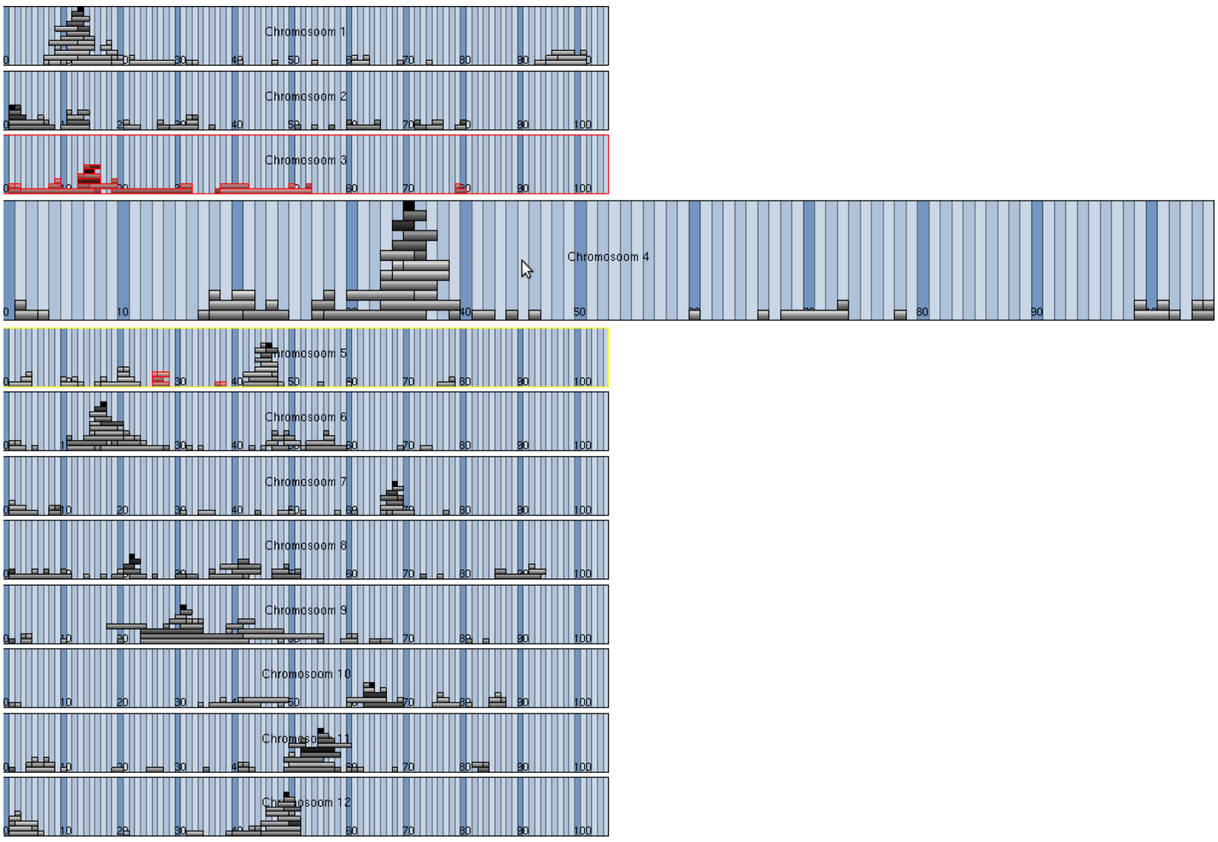


Figure 4.9: Screenshot of the CBB visualization implemented within DNAVis2

Chapter 5

Circle visualization

The circle visualization is designed to depict hierarchical structures (like tree structures), and provides the user with an intuitive and easily browsable interface that enables the exploration of the dataset. The visualization provides the user with a detailed overview of the current abstraction level, while providing a direct connection to the previous and next abstraction level. The circular visualization uses the same structural CBB data as the CBB visualization (see chapter 4), but adds additional knowledge to the BAC elements in the form of the location and size (in bytes) of corresponding FASTA and GFF files. The goal is to provide the user with an uncluttered and unambiguous interface[1], that enables the exploration of a genome size dataset (like the PGSC dataset). The visualiztion provides a unified data-representation of the various data levels (*genome*, *chromosome*, *BIN*, *BIN-domain*, and *BAC*) and acts as a high level interface for the low level FASTA and GFF data.

5.1 The basic circle visualization

This section explains the overall concept behind the circle visualization and the basic navigation mechanism. DNA related examples are used to illustrate various properties of the visualization, but the illustrated concepts can also be used to represent other hierarchical structures. The visualization displays the data as a planetary structure, where the active node is placed in the center with all the direct children grouped around it in a circle (see figure 5.1). The direct children can be expanded, enabling the user to preview the (simplified) next data level without having to navigate to the next level (see figure 5.2). The expanded children can also be collapsed back into their original state. The previewing is not limited to the direct children nodes only, but will become less useful when the nodes are getting smaller and smaller.

5.1.1 Basic Navigation

Navigating through one of the child nodes will lead to the next data level (level 2), making the child node the new active node. Hence, placing it in the center, with its children (the roots grandchildren) placed around it (see figures 5.3 and 5.4). Navigating through the center (active) node leads back to the previous data level (level 1). Turning the parent of the current center node into the new active node, and the center node itself into one of the direct children.

This navigation scheme enables the user to navigate through the whole dataset by browsing up and down the various data levels (see figure 5.5[2]). The circle of the previous level (if available) is drawn

[1]Unambiguous interface: a well defined visual style and a clear navigation mechanism

[2]In contrast to what the concept drawing shows, its also possible to navigate to a next data level without previewing it first.

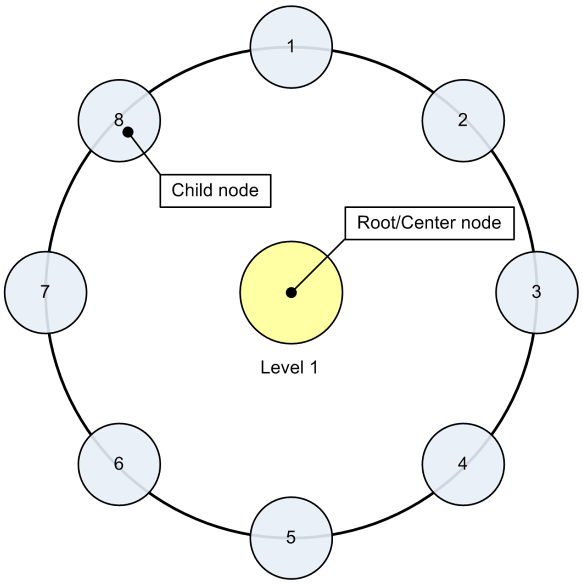


Figure 5.1: Concept drawing of the root node and its direct children

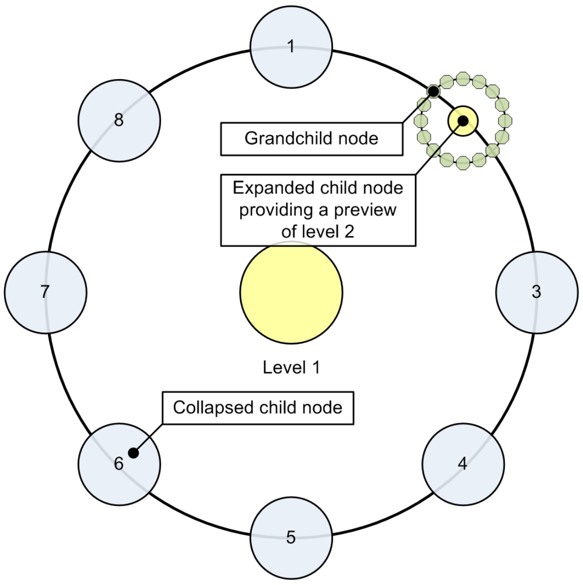


Figure 5.2: Concept drawing of the root node with its second child node expanded

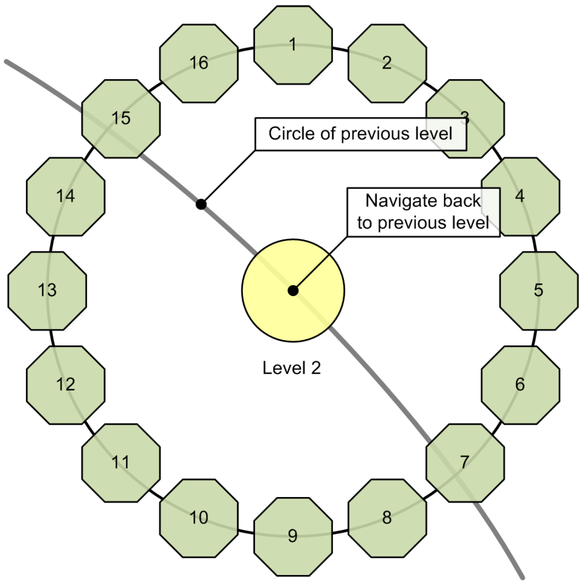


Figure 5.3: Concept drawing of the child node and its direct children (the roots grandchildren)

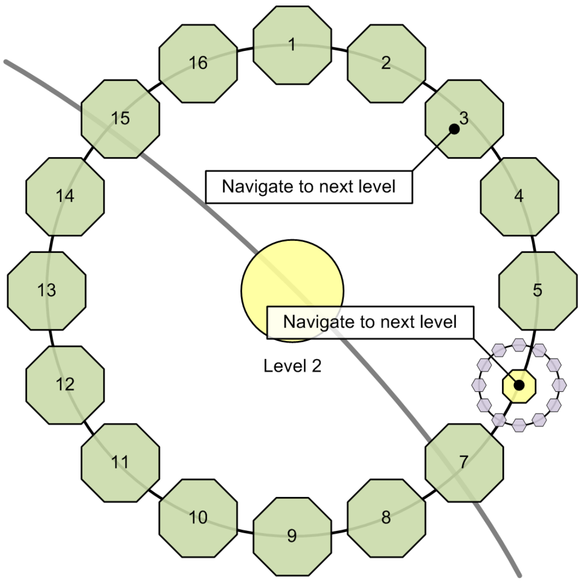


Figure 5.4: Concept drawing of the child node with its sixth child (the roots grandchild) node expanded (previewing the roots over grandchildren)

behind the active node, to visualize its position with respect to its parent node in the previous level. This is a little memory aid to help the user to keep track of his current location within the dataset.

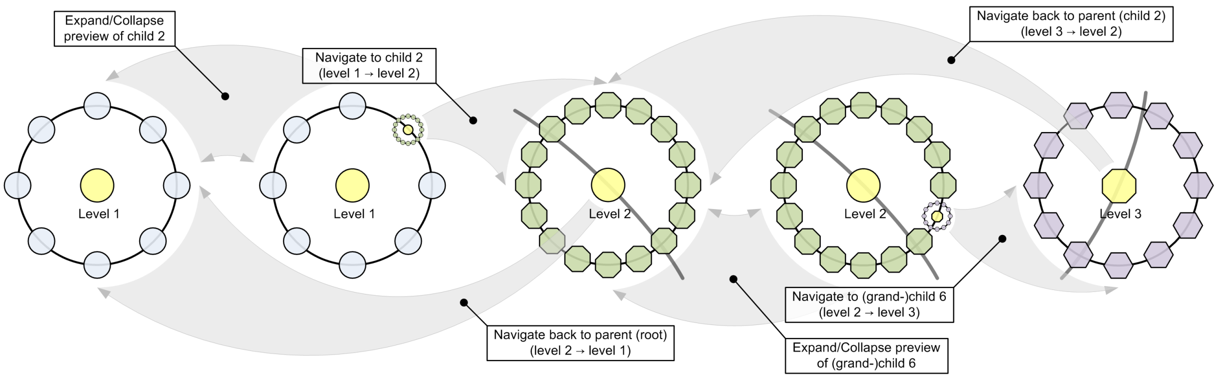

Expand/Collapse preview of child 2
Navigate to child 2 (level 1 → level 2)
Navigate back to parent (child 2) (level 3 → level 2)
Level 1
Level 1
Level 2
Level 2
Level 3
Navigate back to parent (root) (level 2 → level 1)
Navigate to (grand-)child 6 (level 2 → level 3)
Expand/Collapse preview of (grand-)child 6

Figure 5.5: Concept drawing of the previewing of, and navigation through multiple data levels.

5.1.2 Node placement and scaling

The size of the child nodes can be used to indicate quantitative properties, providing the user with an intuitive way of visualizing the distribution of certain data properties within the dataset. This sizing of the child nodes can be used on both structural *(number of children, number of leafs)* and non-structural *(bytes of information)* data properties. The distribution of the child nodes across the circle is also affected by their size, where the equal sized nodes can be distributed evenly across the circle, the scaled nodes need to be distributed according to their size (see figure 5.7).

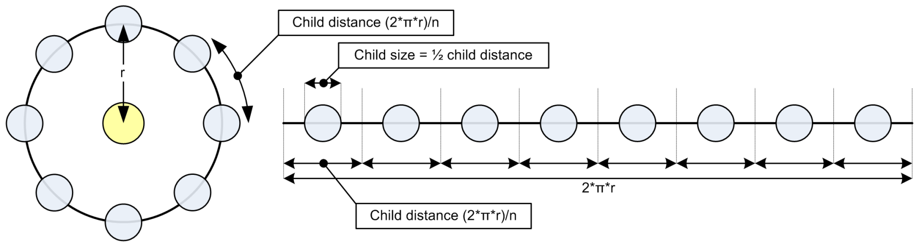


Figure 5.6: One dimensional child node placement and scaling for equivalent sized nodes

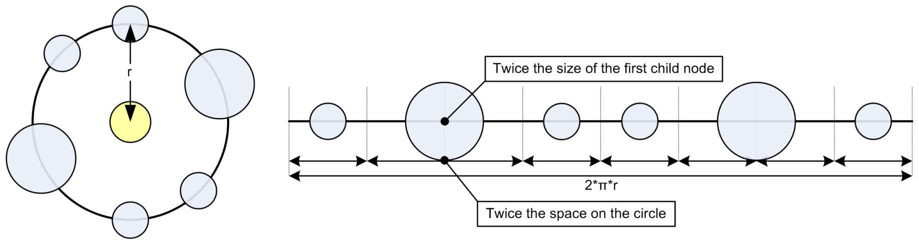

r
Twice the size of the first child node
2*π*r
Twice the space on the circle

Figure 5.7: One dimensional child node placement and scaling for different sized nodes

The location and size of the child nodes is determined by their weight factor. Every child node is assigned a weight factor, which indicates some selected data property. These weight factors are used to allocate space on the circle outline. The circle outline is seen as a one dimensional area (a line) with the size $2\pi r$, and represents the combined weight of all the child nodes. Every child node is placed in the center of its assigned (one dimensional) area, and is scaled according to the size of the area combined with a fixed factor to prevent mutual intersection (a factor of 0.5 is used in the examples).

5.1.3 Custom icons

Intuitive icons can be used to represent the nodes at the various data levels, making the visualization far easier to interpret and to keep track of the current location. The icons can also be used to add extra information to the visualization without the need of additional screen space. The following example displays some DNA related data using two custom icons: the chromosome icon, and the BIN icon (see figures 5.8 and 5.9). Extra information is added by either coloring the icon, or drawing it transparent. The colored icons represent nodes that contain next level data (hence, have children), while the transparent icons represent nodes that do not contain next level data.

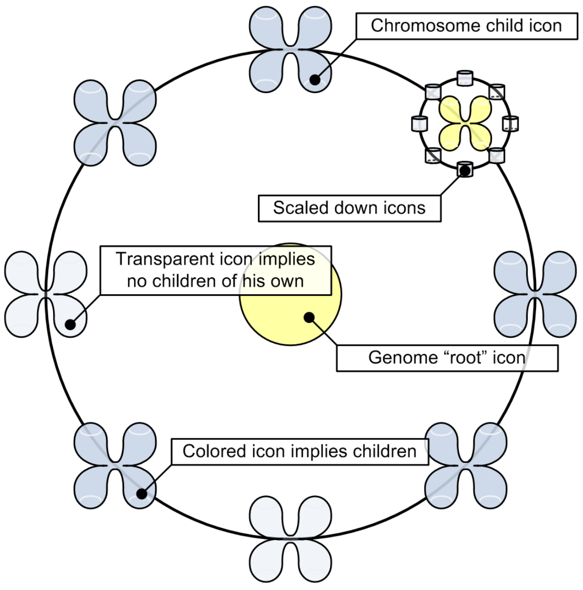

Chromosome child icon
Scaled down icons
Transparent icon implies no children of his own
Genome “root” icon
Colored icon implies children

Figure 5.8: Concept drawing of custom icons at genome level

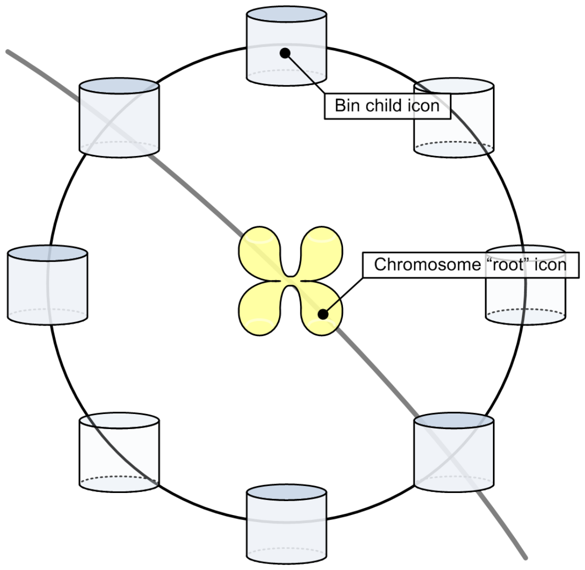


Figure 5.9: Concept drawing of custom icons at chromosome level

5.1.4 Inner circle

The unused space between the center node and the child nodes is called the inner circle, and can be used to display additional information about the current data level. For example, the selection of the direct child nodes by drawing partitioned rings that contain a segment for every child node, where colored segments indicate selected child nodes (see figure 5.10). Multiple ring levels can be used to indicate multiple selections at once.

5.1.5 Outer circle

The space at the outer side of the child nodes is called the outer circle, and can also be used to display additional information about individual child nodes. For example a label to display the name of the child node, or some statistical data using graphs, or a combination of both (see figure 5.11(a), 5.11(b) and 5.11(c)).

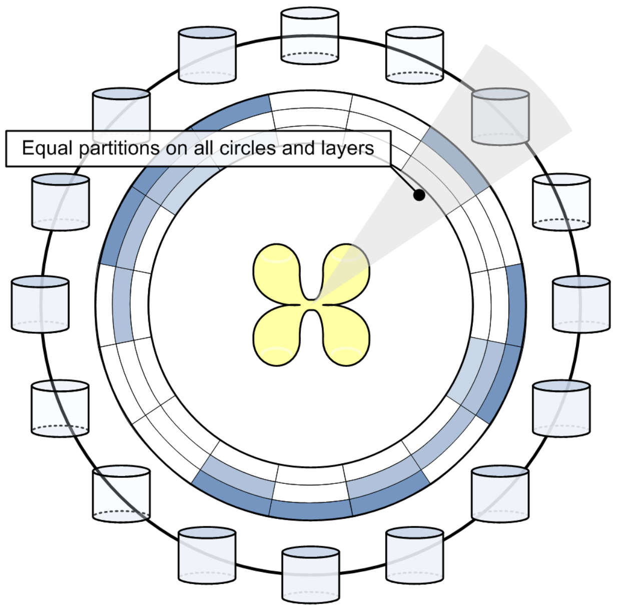


Figure 5.10: Concept drawing of a selection inner circle with three levels

Because the outer circle contains information about individual child nodes only, it is not that susceptible to icon scaling related problems. It can freely follow the outer contours and shift along with the icons.

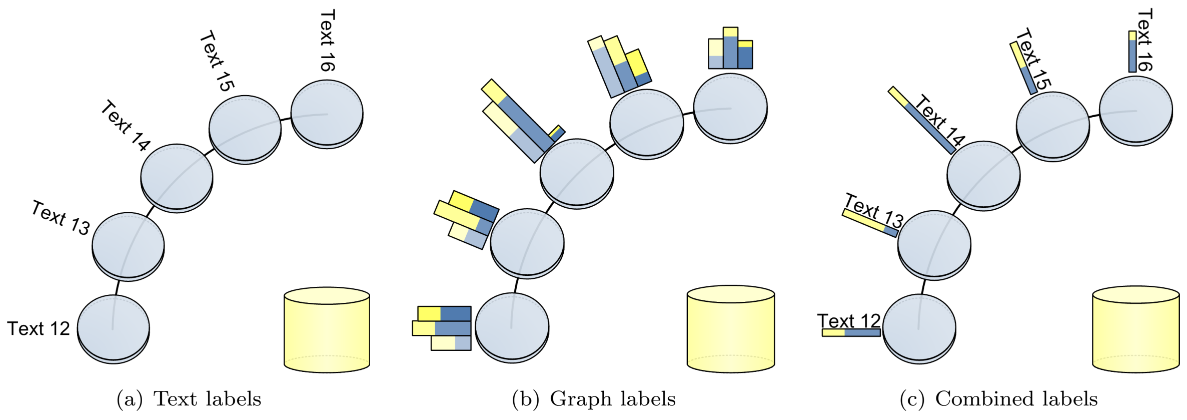


Figure 5.11: Concept drawings of possible outer circle labels

5.1.6 Direction and order

The start and direction indicator is represented by an arrowhead and provides the means to display ordered data within the circular visualization (see figure 5.12). The arrow shape separates the first child node from the last child node and indicates the direction in which the data is represented. The size of the start and direction indicator scales along with the first and last child node, to prevent the obstruction of other visual elements.

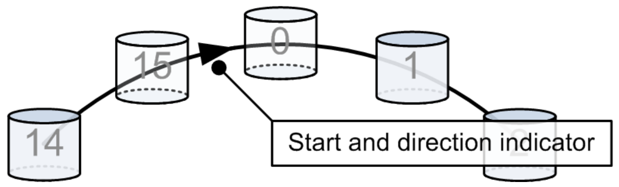


Figure 5.12: Concept drawing of the start and direction indicator

5.2 The extended circle visualizations

The default circular visualization enables lots of additional information to be displayed using the inner and outer circle, but it's not the most screen space efficient solution to display the child nodes. This can become a problem when the data set requires a very large amount of children to be displayed at once. The following section will describe an additional visualization used to overcome the space limitations, while attempting to minimize the deviation from the original circular visualization scheme.

5.2.1 The phi-ball based visualization

The default circular visualization places all the children on a circle around the active (center) node, resulting in a configuration where all children have an equal distance to the center. This provides a one dimensional area of $2\pi r$ to draw the child nodes without mutual overlap, resulting in linear downscaling of the child nodes when there count increases.

One way to reduce the shrinking is to make use of a two dimensional area. One of the most efficient ways of using a two dimensional area are treemaps [39, 38, 3, 40], but this visualization style deviates greatly from the default circular visualization.

The concept of a circular placement has to be incorporated to achieve a feeling of unity when mixing both the default and the new high child count visualization. This calls for a compromise between optimal screen usage and conserving the overall feeling of the circular visualization.

An elegant compromise is found in the form of the so called phi-ball algorithm [25, 22], which is designed to place circular shapes on top of a sphere. An adapted version of the algorithm can be used to place circular shapes around the active (center) at various distances without mutual overlaps (see figure (see figure 5.13).

The algorithm clearly proves beneficial for the size of the child nodes (see figure 5.15 and 5.16), but it also has some shortcomings:

- The direction and order of the child elements is no longer displayed.
- The inner and outer[3] circle can not be used. (Tasks as labeling are no longer possible using this approach.)
- The phi-circle algorithm looses its area efficiency when the number of elements reduce (see figure 5.14).

The phi-ball visualization has to be seen as a compromise to accommodate the extremes in a data set, and is not a full replacement for the default circular visualization. Additional effects like customized icons and coloring can be used to compensate for some of functionality that is lost, but all these techniques can also be used within default circular visualization.

[3]The space around the circle is still available, but the one on one mapping with the circle elements is lost.

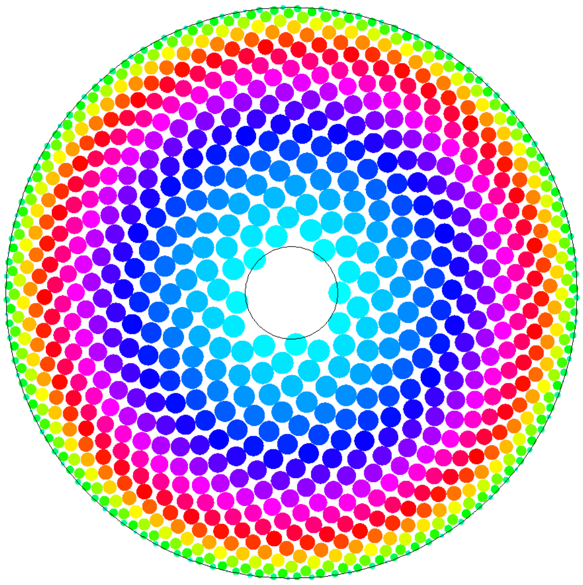


Figure 5.13: Example of a phi-circle with 900 child elements [37]

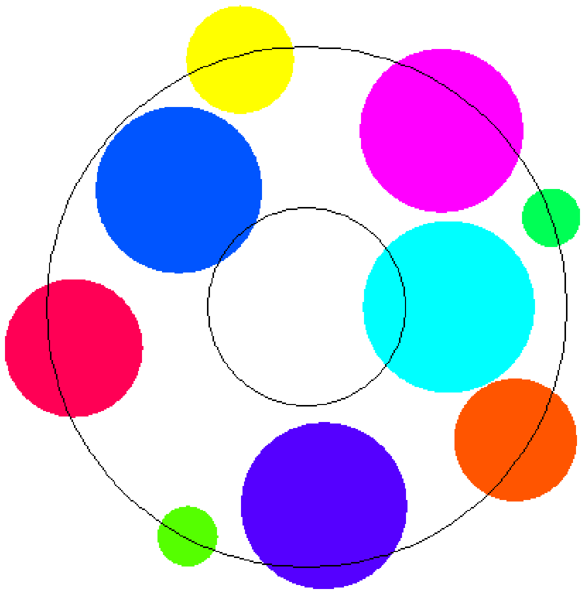


Figure 5.14: Example of a phi-circle with 9 child elements [37]

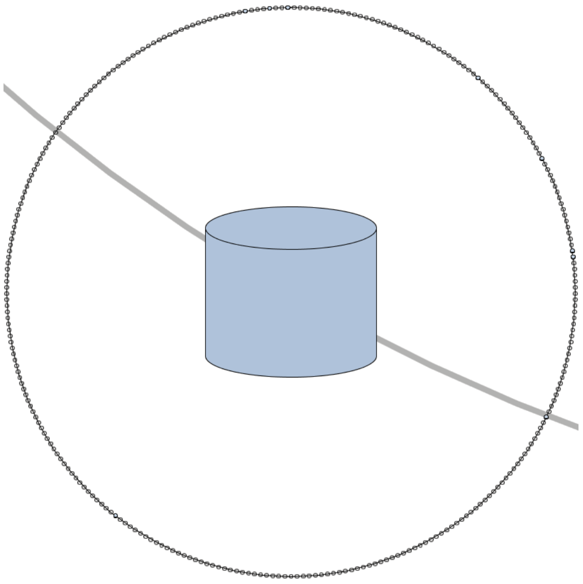


Figure 5.15: Screenshot of a circle visualization with 293 child nodes

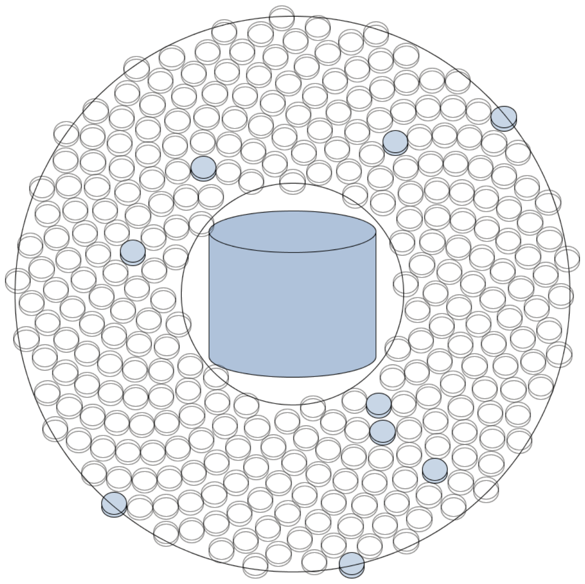


Figure 5.16: Screenshot of a phi-ball visualization with 293 child nodes

5.3 The DNA specific circle visualizations

The DNA specific circle visualizations depicts the *genome level*, *chromosome level*, *BIN/BIN-domain level*, and *BAC level* using the genome as root node. Every abstraction level is indicated with it's own custom icon.

All elements except the BIN-domains are visualized as circle nodes. The BIN nodes indicated at chromosome level are grouped together using the inner circle. This inner circle is a circular implementation of the technique used in the CBB visualization, that maps BIN-domain bars on top of the BIN number ruler of the chromosome bars (see section 4.1.3). But instead of stacking the BIN-domains bottom upwards, the grouping bars are mapped on the inner circle from the most outer layer to the most inner layer (see figure 5.17). The BIN-domains itself can be opened like a normal child node to display the individual BACs associated to the BIN-domain.

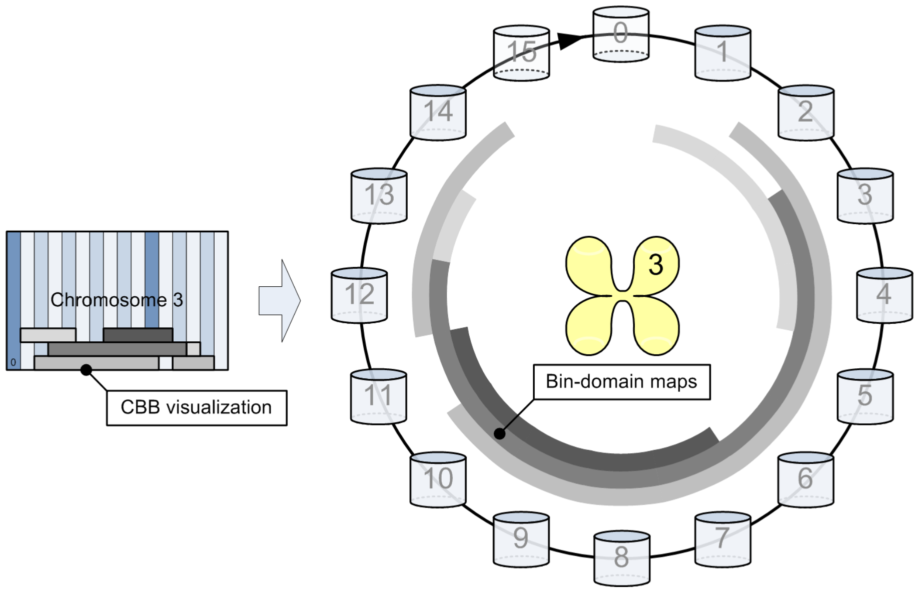


Figure 5.17: BIN-domain mapping

The user interface of the DNA specific circle visualization is extended to provide the following additional features:

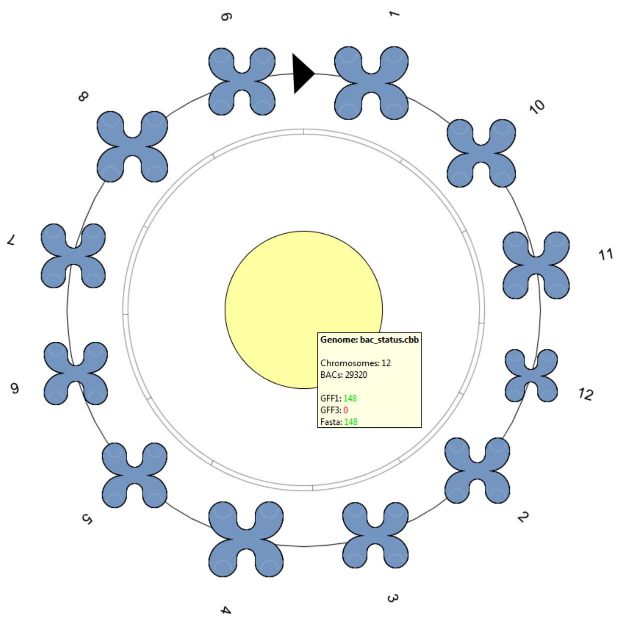
- Icon scaling options is extended with: scale by number of bytes of FASTA data, scale by number of bytes of GFF, scale by number of bytes of both FASTA and GFF data.
- Tool tips are added to provide additional information about the nodes, for example: the name of the node, the number of children, the number of leafs and the number and total file size of the associated FASTA and GFF files.
- Pop-up menu to open selection and individual elements in other visualizations.

5.4 Results

This section provides an overview of the implemented circle visualization. The circle visualization is implemented as an extension of the DNAVis2 tool set. The implementation details of the circle visualization extension are described in appendix E.

The circle visualization is the default view for the CBB file format (see section 2.3.3), meaning that the DNAVis2 file browser will open a CBB file in this visualization when the user does not specify a specific visualization. It's assumed that CBB files use the `*.cbb` extension, and that the corresponding FASTA and GFF files are placed in the same location or a subdirectory of the CBB file location. The visualization will check the availability and file size of the associated GFF and FASTA files.

Figure 5.18 shows the default startup configuration of the circle visualization, the dataset is displayed at genome level using the selection inner circle and the label outer circle.



Genome: bac_status.cbb

Chromosomes: 12
BACs: 29320

GFF1: 148
GFF3: 0
Fasta: 148

Figure 5.18: Screenshot of the circle visualization default startup configuration displaying the PGSC dataset

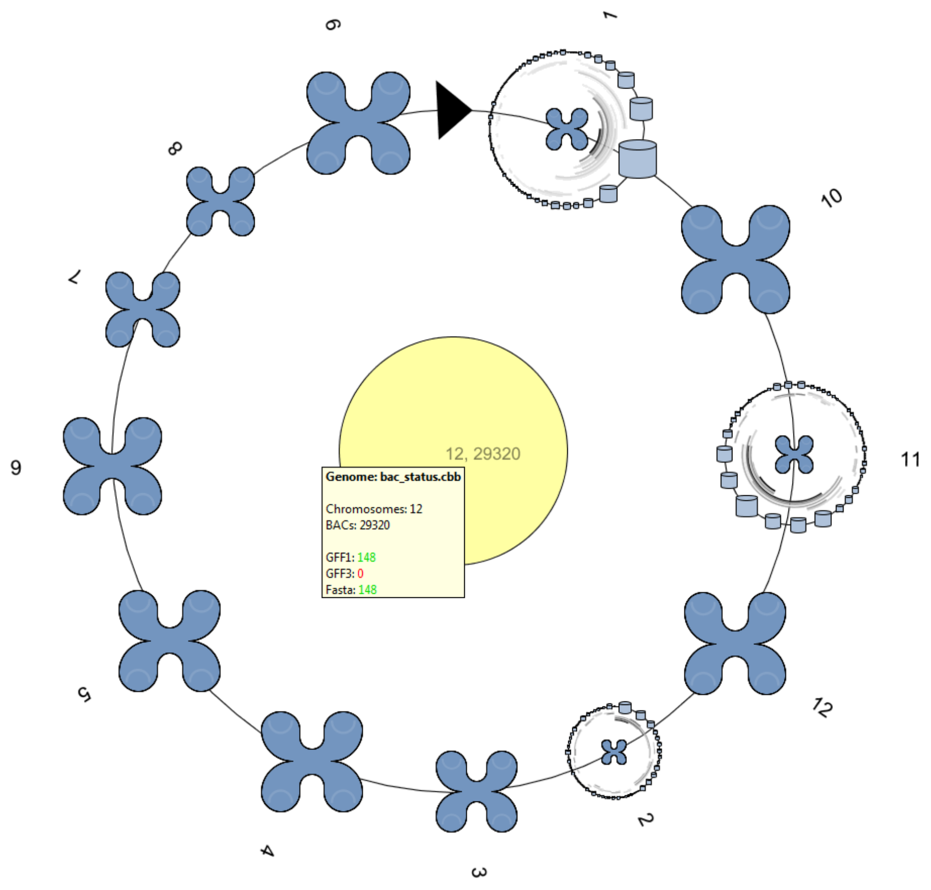
5.4.1 The abstraction levels

This part provides an overview of the abstraction levels: *genome level*, *chromosome level*, *BIN level* and *BIN-domain* level depicted by the circle visualization. The low level sequence and annotation data is visualized using the existing linear view (see section 2.5.2).

Genome level

The genome level (see figure 5.19) provides the user with the highest abstraction level of the circle visualization, the center node represents the complete dataset, and the surrounding nodes represent the chromosomes. This abstraction level has no additional inner or outer circles outside the default selection and label circles. The inner selection circle can be used to select child nodes (in this case chromosomes). Fully selected child nodes are indicated by a red selection bar and partly selected child nodes are indicated by a pinkish selection bar.

The center node can be used to expand or collapse all the chromosome nodes around it, individual chromosome nodes can be expanded or collapsed by interaction with the child nodes. Every node has its own tooltip and popup menu, providing additional information and interaction options.



Genome: bac_status.cbb

Chromosomes: 12
BACs: 29320

GFF1: 148
GFF3: 0
Fasta: 148

Figure 5.19: Screenshot of the circle visualization at genome level, displaying the PGSC dataset

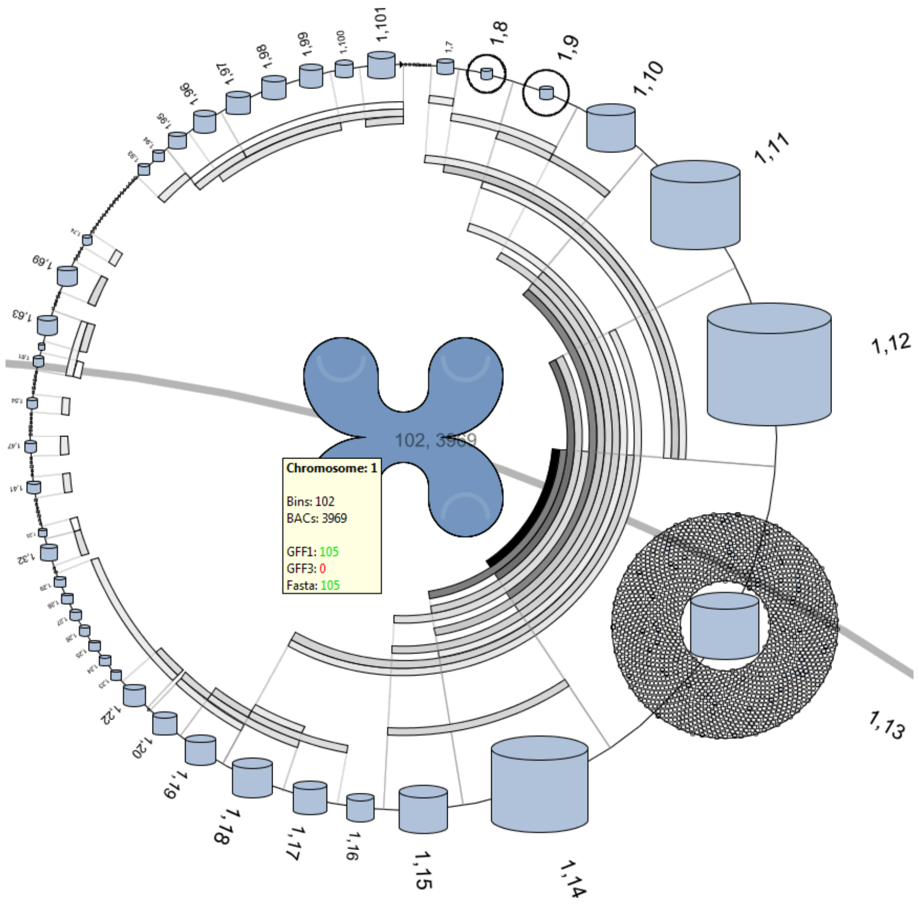
The center nodes can be used to navigate to a higher abstraction level, while the child nodes can be used to navigate to a lower abstraction level. Navigation to a higher abstraction level is not possible at genome level because it's the highest level available, but the chromosome child nodes can be used to navigate to the chromosome level.

Chromosome level

The chromosome level (see figure 5.20) represents the data of a single chromosome. The center node represents the chromosome and the surrounding child nodes represent the associated BINs. The chromosome render has an additional BIN-domain inner circle (see section 5.3) that visualizes the BIN-domains of the chromosome.

The chromosome level supports the same interactions as the genome level (described above), but at this level it is also possible to travel back to a higher abstraction level (genome level) using the center node.

The child nodes without lower level data are displayed transparent, so only color filled child nodes can be used to navigate to a BIN level. The BIN-domain inner circle adds a second lower abstraction, this BIN-domain level can be accessed using the BIN-domain bars of the inner circle. The BIN-domain level cannot be previewed, but can be opened like the BIN level.



Chromosome: 1

Bins: 102
BACs: 3969

GFF1: 105
GFF3: 0
Fasta: 105

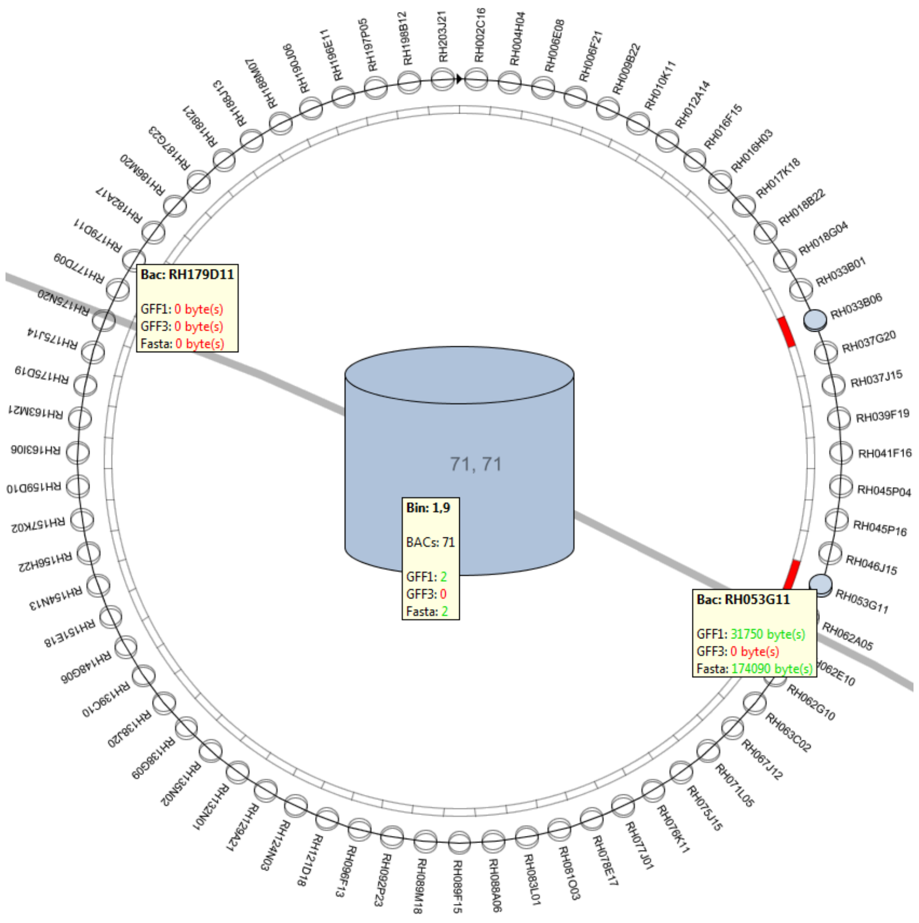
Figure 5.20: Screenshot of the circle visualization at genome level, displaying chromosome 1 of the PGSC dataset

BIN level

The BIN level (see figure 5.21) represents the data associated to a single BIN. The center node represents the BIN and the surrounding child nodes represent the associated BACs. A single BAC can exist in multiple BINs, hence the BIN-domains. This abstraction level has no additional inner or outer circles outside the default selection and label circle. But for the BIN level it can be useful to use the phi-ball visualization (see section 5.2.1) instead of the basic circle visualization, because of the large number of BAC nodes.

The current implementation can switch automatically between the basic circle visualization and the phi-ball visualization depending on the available screen space (in pixels) for an average child node. The BIN and BIN-domain levels are the only abstraction levels with enough child nodes to need the phi-ball visualization, but the phi-ball visualization can also be used by the other abstraction levels

The BIN level provides the same basic interactions as all the other abstraction levels, but the BAC child nodes have no circle representation for their low level sequence and annotation data. Navigating to a BAC node will open the sequence and annotation data in a linear view.



Bac: RH179D11

GFF1: 0 byte(s)
GFF3: 0 byte(s)
Fasta: 0 byte(s)

Bin: 1,9

BACs: 71

GFF1: 2
GFF3: 0
Fasta: 2

Bac: RH053G11

GFF1: 31750 byte(s)
GFF3: 0 byte(s)
Fasta: 174090 byte(s)

Figure 5.21: Screenshot of the circle visualization at BIN level, displaying BIN 9 of chromosome 1, containing 71 BACs

BIN-domain level

The BIN-domain level (see figure 5.22) represents the data associated to a BIN-domain. The center node represents the BIN-domain and the surrounding child nodes represent the associated BACs. The BIN-domain level visualization is completely equivalent to the BIN level visualization, except for the center node which represents a BIN-domain instead of a BIN.

5.4.2 The customizations

The circle visualization requires almost no adjustments during normal use. But the visualization offers a small number of settings that let the user customize the visualization.

Circle or phi-ball visualization

The preferred circle visualization type:

- *Circle visualization:* Resulting in the basic circle visualization as described in section 5.1, enabling the inner and outer circle renderer.
- *Phi-ball visualization:* Resulting in the phi-ball visualization as described in section 5.2.1, disabling the inner and outer renderer.

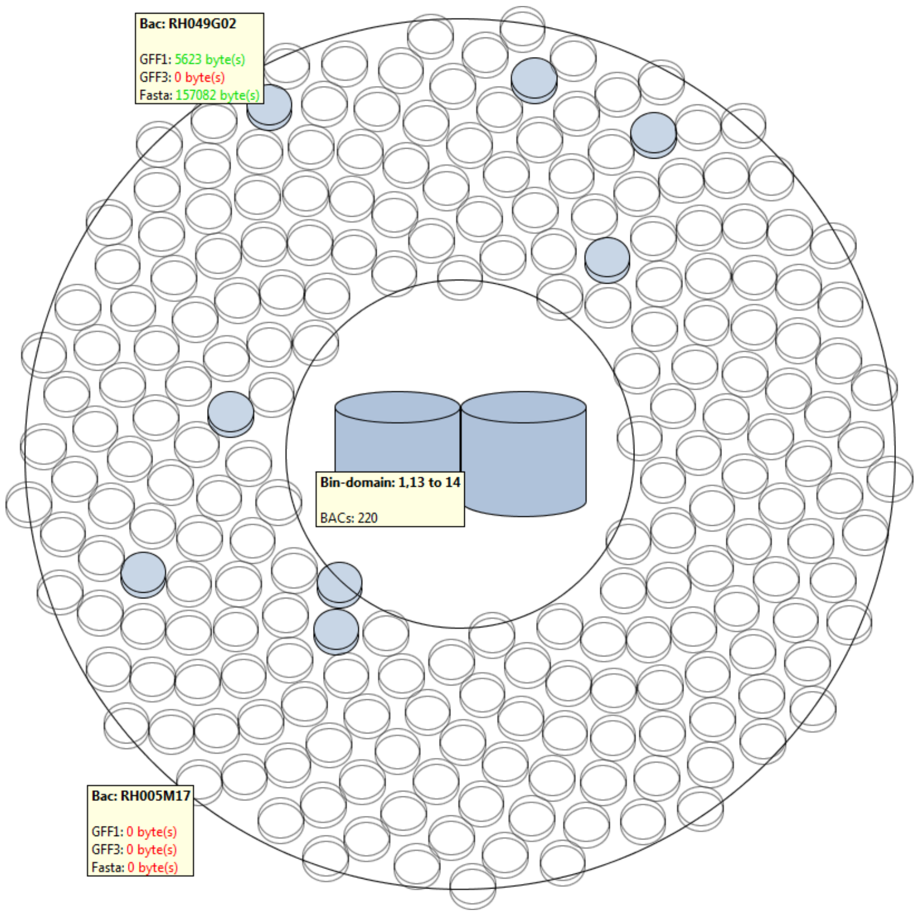


Figure 5.22: Screenshot of the phi-ball visualization at BIN-domain level, displaying BIN-domain 13 to 14 of chromosome 1, containing 220 BACs

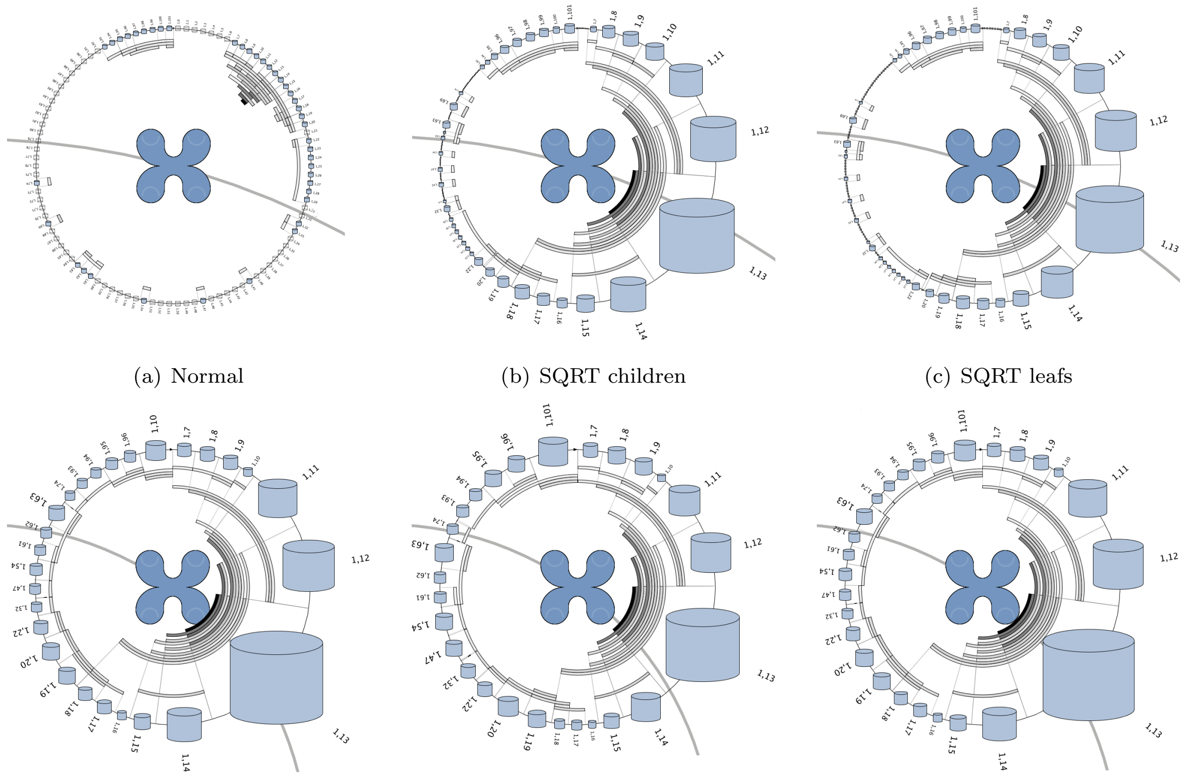
- *Auto visualization:* Letting the visualization decide which visualization provides the best result, depending on the number of child elements and the available screen space in pixels.

Child node scaling algorithm

The selection of a child node scaling algorithm. This is probably the most useful customizable setting of the visualization, and allows for the visualization of distribution of high level and low level data elements within the dataset.

- *Normal scaling:* No additional scaling, child nodes are drawn at their fixed size (see figure 5.23(a)).
- *Number of children:* The child nodes scale linear with the number of direct children of a node.
- *Square root of number of children:* The child nodes scale with the square root of the number of direct children of a node (see figure 5.23(b)).
- *Number of leafs:* The child nodes scale linear with the number of leaves in a node.
- *Square root of the number of leafs:* The child nodes scale with the square root of the number of leaf in a node (see figure 5.23(c)).
- *Square root of bytes of FASTA data:* The child nodes scale with the square root of the number of bytes of FASTA data related to a node (see figure 5.23(d)).

- *Square root of bytes of GFF data:* The child nodes scale with the square root of the number of bytes of GFF data related to a node (see figure 5.23(e)).
- *Square root of bytes of FASTA and GFF data:* The child nodes scale with the square root of the number of bytes of FASTA and GFF data related to a node (see figure 5.23(f)).



(a) Normal (b) SQRT children (c) SQRT leafs

(d) SQRT FASTA (e) SQRT GFF (f) SQRT GFF+FASTA

Figure 5.23: Screenshots of child node scaling algorithms

The inner and outer circle

The user can choose between these inner circles at all abstraction levels, but the BIN-domain inner circle will only be visualized at chromosome level, or in the preview of the chromosome levels (see figure 5.19 and 5.20).

The use of child node scaling poses some trouble for the inner circle, through the irregular contours created by their outlines. The current implementation scales the selection inner circle down so it is small enough to fit between the largest child node and the center node.

But the BIN-domain inner circle needs all the room it can get to place its stacks of BIN-domain bars. Following the contours of the child nodes is one solution, but this creates a rather distorted, unpleasing and hard to interpreted visualization (see figure 5.24). A better solution is shifting the BIN-domain bars by a full ring level at a time, placing a bar on the first level that is not obstructed by an icon or another bar. This result in a visualization that is far easier on the eyes (see figure 5.25).

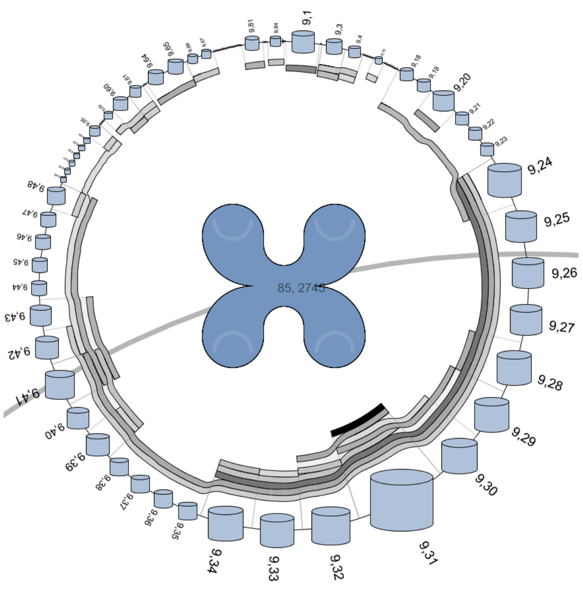


Figure 5.24: Screen shot of the inner circle following the icons contours

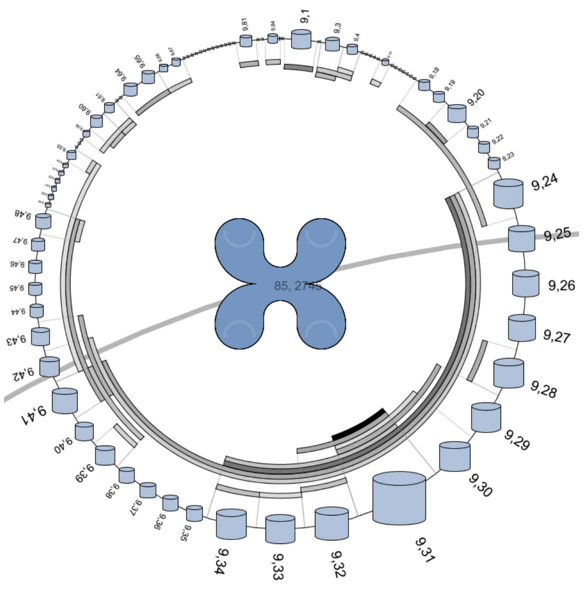


Figure 5.25: Screen shot of the inner circle shifting a full level at a time

The outer circle doesn't need any special measures because it only provides information about individual child nodes, and can therefore move along with these individual nodes.

5.5 The performance enhancements

The circle visualization uses a number of tricks to reduce the computational complexity of its visualizations.

5.5.1 Multiple renderers

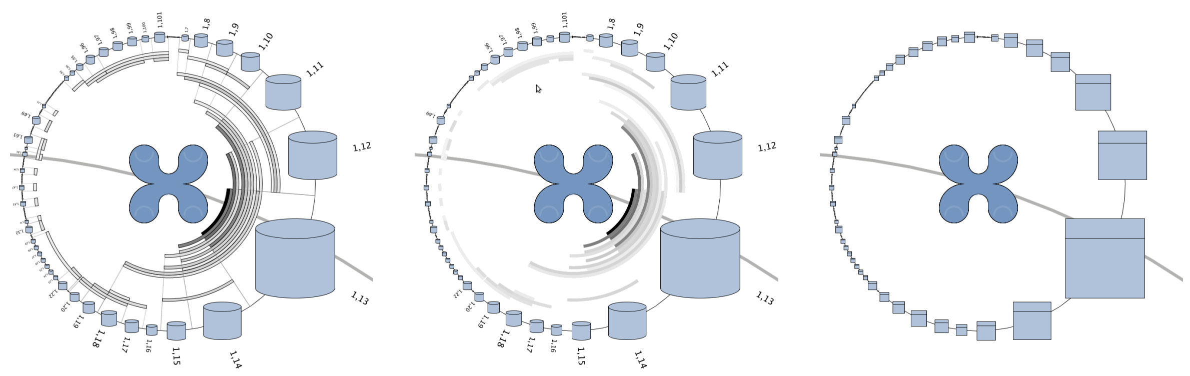
The distribution of the visualization across multiple renderers makes it possible to update part of the visualization without having to redraw the complete visualization. When for example the inner or outer circle renderer's are changed, only that render has to be redrawn, all other renderer's can reuse there previously created data. Another advantage of multiple renderers is that the code can easily be optimized to make use of multiple processor cores.

5.5.2 Semantic zooming

Every renderer knows the size of the pixels on the screen, this value can be used to calculate the screen space occupied by a shapes or objects (drawn by the renderer). The renderer can decide to reduce a shapes complexity or simply skip the shape altogether, depending on the used screen space. The use of semantic zooming [30, 41] eliminates over complexity when drawing to a small part of the screen. Figure 5.26(a), 5.26(b) and 5.26(c) show an example of the semantic zooming steps used by the BIN-domain inner circle renderer.

5.5.3 Sporadic update

While semantic zooming can provide a performance increase by reducing the complexity of the visualization, it also adds additional computations because the renderers have to recalculate their internal data. So it's not smart to recalculate the internal renderer data every time the user zooms by a small amount.



(a) Full detail (b) Low detail (c) No detail

Figure 5.26: Example of semantic zooming at chromosome level, with BIN-domain indication

The sporadic update enhancements will only recalculate the internal renderer data when the zoom factor is changed beyond a certain threshold. The visualization will still be scaled and redrawn, but the internal data of the renderer's will only be recalculated every time the threshold is exceeded. This has the side effect that the transition between two semantic zooming states can occur at different zooming levels, which is potentially disruptive to the overall user experience.

5.5.4 Optimized transformations

This optimization makes use of the fact that most widely used libraries are highly optimized. The early versions of the circle visualization were aimed at easy portability between render-pipelines, and therefore used its own transformation library to rotate, scale and translate the shapes and text. While the implementation proved to be very portable, it lacked in overall performance when dealing with complex visualizations. Therefore the choice was made to replace the storage and transformation functions with versions that where less portable but used optimized library functions, in this case the Java2D library[29]. The current implementation uses Java2D shape objects to store the rendered data, and uses Java2D transformations to rotate, scale and translate the shapes. Thereby effectively increasing the performance of complex visualizations and even adding GPU support to the transformation calculations.

The Java2D approach was chosen over a possible OpenGL [16, 31, 2] implementation for a number of factors:

- The Java2D library provided a clean and easy to implement (object oriented) way to store and transform two dimensional shapes. Creating the same implementation for OpenGL would require quite some additional coding.
- The Java2D library provided almost all the shapes needed by the circle visualization, and the missing shapes could be easly constructed by combining multiple default shapes.
- The Java2D library is a standard library, unlike the JOGL library used by the other views within DANVis2.
- It seemed a good test case to check whether the Java2D library is a possible replacement for the currently (by other visualizations) used JOGL OpenGL library.

Chapter 6

Property matrix visualization

The property matrix is designed to explore the relations between the data elements of a genome size dataset. The property matrix gives the user the freedom to cross-reference low level data with low level data, high level data with high level data, or low level data with high level data, providing a versatile environment to explore a wide range of data properties. Additional filter options allow the user to focus on specific data attributes, while grouping algorithms provide the means to find similarities between data elements.

The property matrix supports the following high level data elements: *genome*, *chromosomes*, *BINs* and *BACs*, and low level data elements: *annotation types* and *annotation attributes* (these will be referred to as *annotations*). The property matrix provides a framework to cross-reference any combination of high and low level data elements, and assures an uniform visualization style for all the supported data combinations.

The cross referencing of high level data with high level data will provide information about the shared low level information. The cross-referencing of low level data with low level data will provide information about the common high level data. The cross-referencing of high level data with low level data will provide the user with an overview of the occurrences of the low level data element within the high level data element.

6.1 The visualization

The property matrix visualization is composed of two bars with a dot-plot [34] in between. The two bars depict the row and column data that is cross-referenced within the dot-plot (see figure 6.1). This composition enables a wide range of data to be visualized, mapping the data types onto the bars and displaying the resulting information inside the dot-plot elements. The dot-plot elements use color to depict numerical values, like frequency of occurrence or the size of a list.

6.1.1 The side bars

The vertical and horizontal bar are called row bar and column bar, and are used to visualize the data type elements like: BAC name, bin-domain, chromosome name, annotation type and annotation attribute. Their values are represented in the following textual form:

1. *BAC name*, BAC name as defined in the CBB file: `<BAC name>`
2. *BIN-domain*, chromosome name (usually a number), followed by a BIN number domain indicated by a start and stop number: `<chromosome name>','<start no.>'-'<stop no.>`

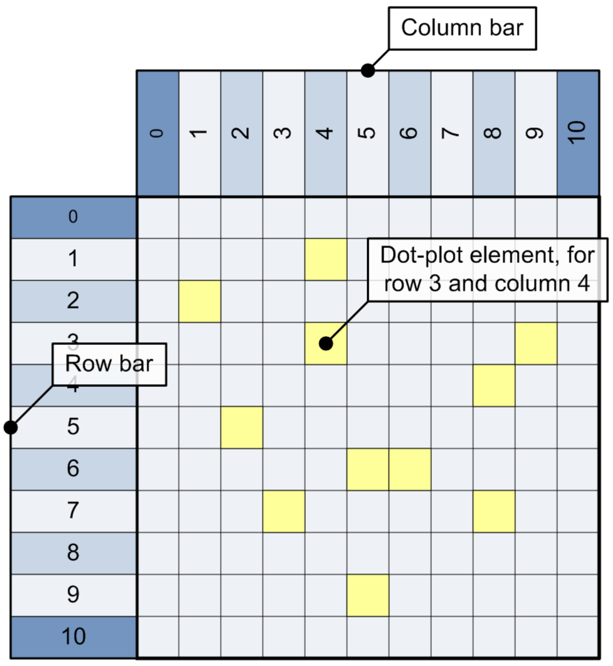


Figure 6.1: Concept drawing of a small property matrix

3. *Chromosome name*, chromosome name (usually a number) as defined in the CBB file: `<chromosome name>`
4. *Annotation type*, annotation type as defined in the GFF file(s) or artificially created: `<attribute type>`
5. *Annotation attribute*, raw annotation attribute string as defined in the GFF file(s), or artificially created: `<raw attribute string>`

Every row or column element will display a single line of text. The column elements text is rotated −90 degrees, this orientation uses less screen space and provides better readability than a downward cascade of letters [6]. The texts will be clipped at the left or top side of the bar elements (see figure 6.2). Tool tips are used to display the whole unclipped text to the user, with the addition of a row or column number (see figure 6.3).

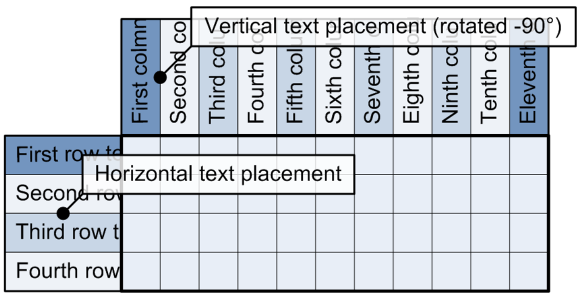

Vertical text placement (rotated -90°)
Horizontal text placement
First row te
Second row
Third row t
Fourth row

Figure 6.2: Concept drawing of text placement & clipping

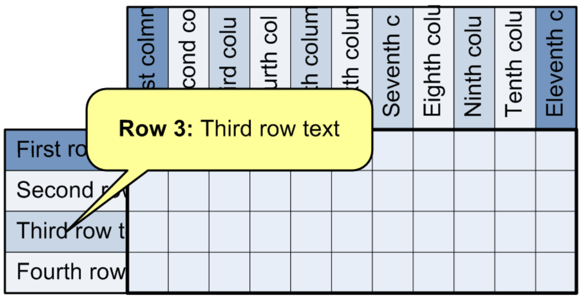


Figure 6.3: Concept drawing of a row element tool tip

6.1.2 The dot-plot elements

Every combination of a row and a column element has a corresponding dot-plot element within the dot-plot matrix. The color of this dot-plot element is used to represent a numerical value (see figure 6.4), this numerical value is based on the number of occurrences of a single element or the size of a list of elements. The displayed color coding will depend on the applied color map and the numerical domain. Additional textual information is provided by a tool tip that lists the *numerical value*, *BACs* or *annotations* that are represented by the dot-plot element (see figure 6.4[1]).

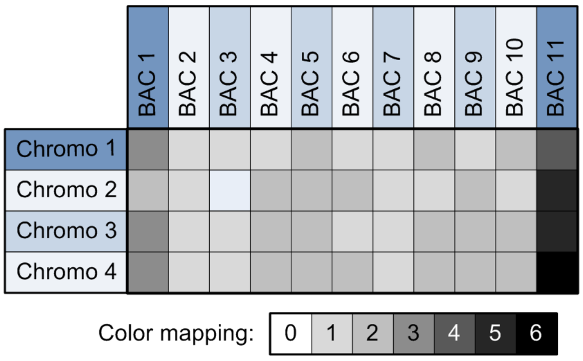


Figure 6.4: Concept drawing of a dot-plot using color mapping from 0→6 to white→black

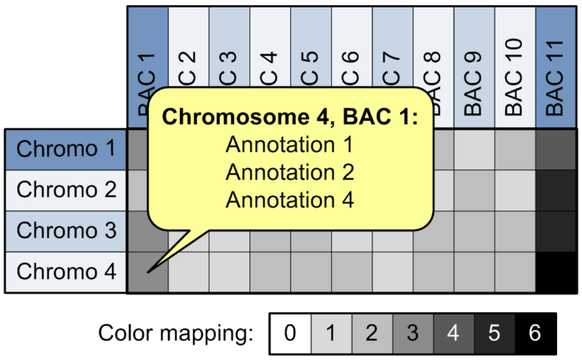


Figure 6.5: Concept drawing of a dot-plot element tool tip, listing the annotations in both chromosome 4 and BAC 1

6.1.3 Zooming and scrolling

When the dataset gets to big too depict within the available screen space, zooming and scrolling can be applied to overcome the space limitations. The property matrix allows for zooming and scrolling in both directions independently, but it's also possible to synchronize these actions. The top-left corner (between the two bars) is used to provide an overview that indicates which part of the dataset is currently visible within the visualization. The surface of the top-left square area is used to represent the dataset as a whole, while a (in the examples case yellow) rectangular area indicates the size and location of the dataset that is currently visible (see figure 6.6).

6.1.4 Highlighting

The highlighting of the row and column elements that correspond to the dot-plot element with the users focus makes the exploration of a large dataset easier. Because it's quite hard to see the corresponding row and column element without any visual aid (see figure 6.7 and 6.8).

6.2 Artificial annotations

The property matrix uses two artificially created annotation types: SPECIES and KEYWORDS. The SPECIES annotation type extracts the species declarations from the normal GFF data and presents them as a separate type. The KEYWORDS annotation type is also created using the normal GFF data.

[1]A grayscale mapping from white to black is chosen because it provides the highest contrast on the white paper.

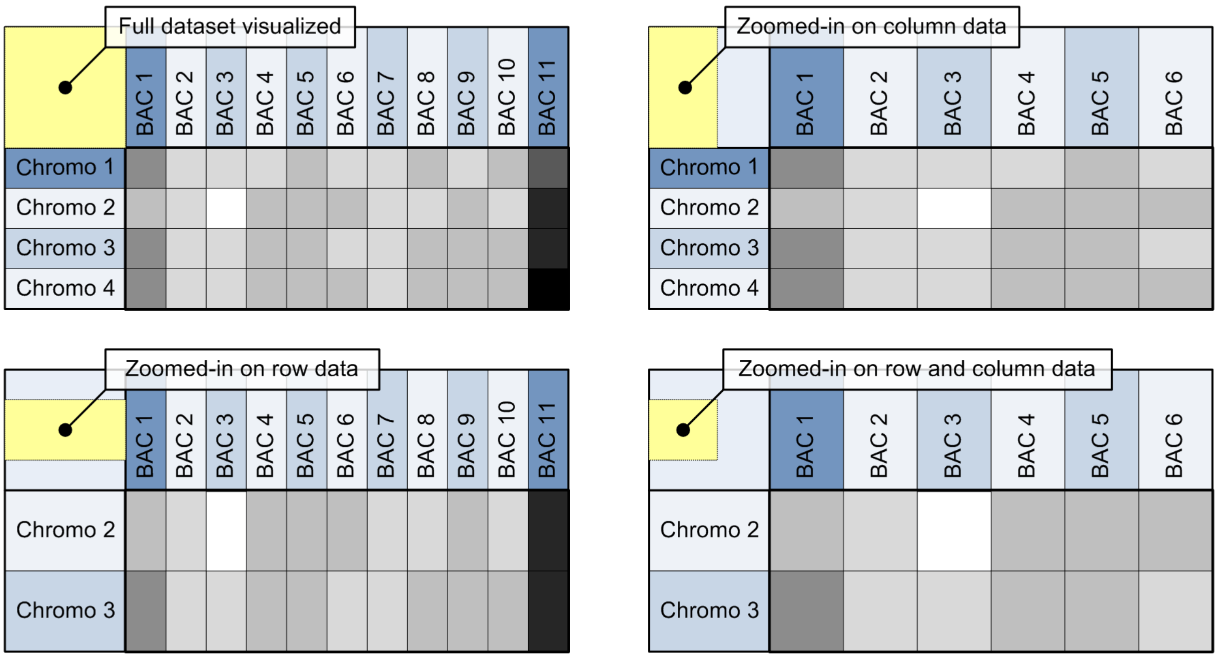


Figure 6.6: Concept drawings of the overview in the top-left corner that indicates the visible data part

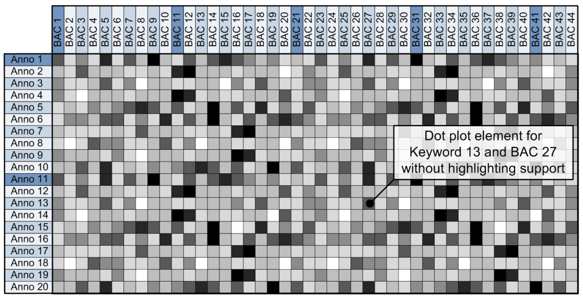

Dot plot element for Keyword 13 and BAC 27 without highlighting support

Figure 6.7: Concept drawing of a larger property matrix without cursor position highlighting

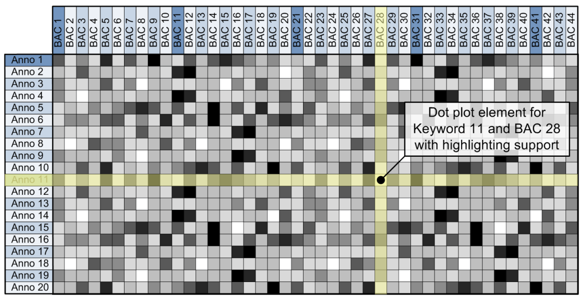

Dot plot element for Keyword 11 and BAC 28 with highlighting support

Figure 6.8: Concept drawing of a larger property matrix with cursor position highlighting

This type divides all the GFF notation data into singular words of three or more consecutive characters.

The artificial annotation types are automatically generated when the GFF data is loaded into the property matrix visualization. They can be easily recognized because there annotation type names are written in capital letters.

6.3 Sorting the matrix

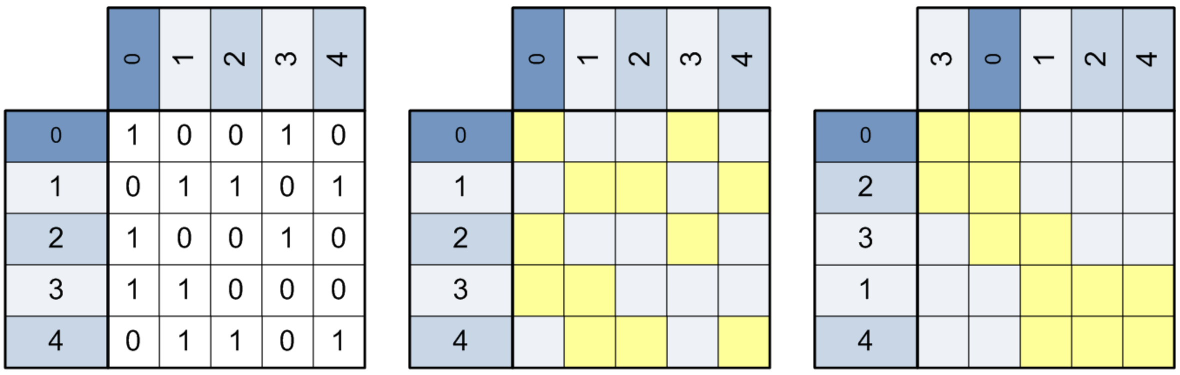
The rows and columns of the dot-plot in the property matrix visualization can be shifted without affecting the data represented by the visualization. This allows for a number of sorting schemes to be applied to the dataset, in order to make the dot-plot more readable, and to show structure in the dataset that was otherwise hidden.

6.3.1 Sorting a single row or column

The sorting of a single row or column on the numerical value of the corresponding data element can be useful when investigating a single property within the property matrix. It also enables the user to visualize the distribution of a single property. This way of sorting the data is very simple to implement and has a negligible computational complexity of $O(n \log(n))$, where n represents the number of elements in the row or column.

6.3.2 Sorting on minimal difference

The sorting on minimal difference is inspired by the Gray code, also known as the reflected binary code [15, 14]. The idea is to minimize the mutual differences between the rows and/or columns (see figures 6.9(a), 6.9(b) and 6.9(c)). This leads to the clustering of rows and/or columns that have many similarities in their data properties. (The outcome of the sorting algorithm is order dependent and has therefore no unique outcome without additional sorting rules.)



	0	1	2	3	4
0	1	0	0	1	0
1	0	1	1	0	1
2	1	0	0	1	0
3	1	1	0	0	0
4	0	1	1	0	1

(a) Unsorted data (b) Unsorted dot-plot (c) Sorted dot-plot

Figure 6.9: Example to illustrate sorting on minimal difference

The sorting algorithm treats the data elements as if it are binary elements (data or no data). The Hamming distance [17] is used to indicated the distance between a row/column. Sorting a single direction has a computational complexity of $O(n^2m)$, where n is the number of rows that have to be sorted, and m is the number of column elements, or visa versa. Sorting in both directions has a complexity of $O(n^2m + m^2n)$.

6.4 Filtering

The property matrix visualization can be used with a wide range of data type configurations. This versatility makes it a useful tool to browse and explore various aspects of a dataset. Additional data filter options provide the means to manipulate the data shown within the visualization framework. These filters enable the user to reduce the dataset, by including or excluding data properties.

The first filter method is based on the annotation types. The user can include or exclude annotation types in the visualization. The second filter is a raw (case-insensitive) text filter, which can be configured to include or exclude row/column/dot-plot elements that contain a particular text. This filter is useful to greatly reduce the displayed row and column elements, or to search for specific properties within the dot-plot.

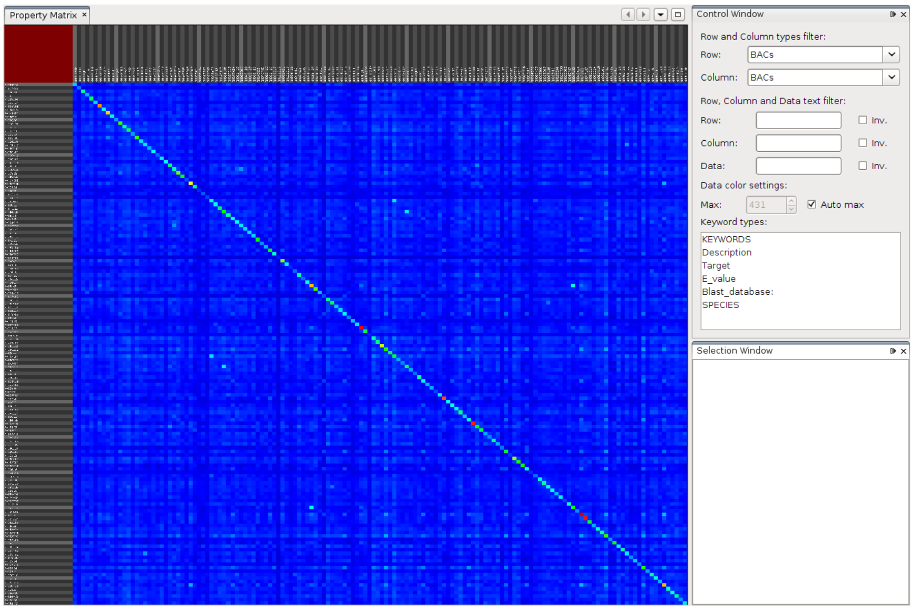
A less powerful but useful way of manipulating the visualization, is by changing the numeric domain of the used color map. This enables the user to see a more detailed color definition within a smaller numerical domain.

6.5 Results

This section provides an overview of the implemented property matrix visualization, which is implemented as an extension of the DNAVis2 tool set. The implementation details of the property matrix visualization extension are described in appendix F.

The property matrix visualization uses both the high level CBB file data and the corresponding GFF files. The visualization can be opened directly from the DNAVis2 file browser by selecting the property matrix view to open de CBB file, or indirectly by the circle visualization creating a (sub-)selection in the circle visualization.

Figure 6.10 shows the default startup configuration of the property matrix visualization. The cross-reference types are both set to *BACs* without additional filtering, resulting in the visualization of the number of similar *annotations* between the BACs. The clear top-left to right-bottom line is the result of comparing every BAC with itself, providing the highest possible number of similar annotations. A rainbow color mapping is used to depict the dot-plot values, the range is automatically set to the highest value within the dot-plot.



Row and Column types filter:
Row: BACs
Column: BACs
Row, Column and Data text filter:
Row: ☐ Inv.
Column: ☐ Inv.
Data: ☐ Inv.
Data color settings:
Max: 431 ☑ Auto max
Keyword types:
KEYWORDS
Description
Target
E_value
Blast_database:
SPECIES

Figure 6.10: Screenshot of the property matrix visualization default startup configuration displaying the PGSC dataset

6.5.1 Multiple cross-referencing settings

Table 6.1 provides a overview of the supported cross-reference settings and the resulting dot-plot data. The dot-plot will display the data as color mapped elements, representing the occurrence of the elements or the size of the returned list of elements.

Data type	**Data type**	**Resulting information**
Genome	Genome	List of all annotations in the dataset
Genome	Chromosome/BIN/BAC	List of all annotations in the chromosome/BIN/BAC
Genome	Annotation types	List of all annotations of annotation type in the dataset
Genome	Annotations	Number of appearances of the annotation in the dataset
Chromosomes	Chromosomes	List of all annotations in both chromosomes
Chromosomes	BINs	List of all annotations in both the chromosome and BIN
Chromosomes	BACs	List of all annotations in both the chromosome and BAC
Chromosomes	Annotation types	List of all annotations of annotation type in the chromosome
Chromosomes	Annotations	Number of appearances of the annotation in the chromosome
BINs	BINs	List of all annotations in both BINs
BINs	BACs	List of all annotations in both the BIN and BAC
BINs	Annotation types	List of all annotations of annotation type in the BIN
BINs	Annotations	Number of appearances of annotation in the BIN
BACs	BACs	List of all annotations in both BACs
BACs	Annotation types	List of all annotations of annotation type in the BAC
BACs	Annotations	Number of appearances of annotation in the BAC
Annotation types	Annotation types	List of all BACs containing data of both annotation types
Annotation types	Annotations	List of all BACs containing the annotation and annotation type
Annotations	Annotations	List of all BACs containing both annotations

Table 6.1: Possible data combinations and the resulting information

Figure 6.15 displays the cross-reference of *BACs* with the *Annotations* of the type *SPECIES*. The user can include or exclude one or more annotation types for every cross-reference configuration. The vertical lines in the dot-plot indicate that a small number of the annotations occur in almost every BAC within the dataset. The most prominent column (read vertical line) belongs to the species Solanum demissum (nightshade).

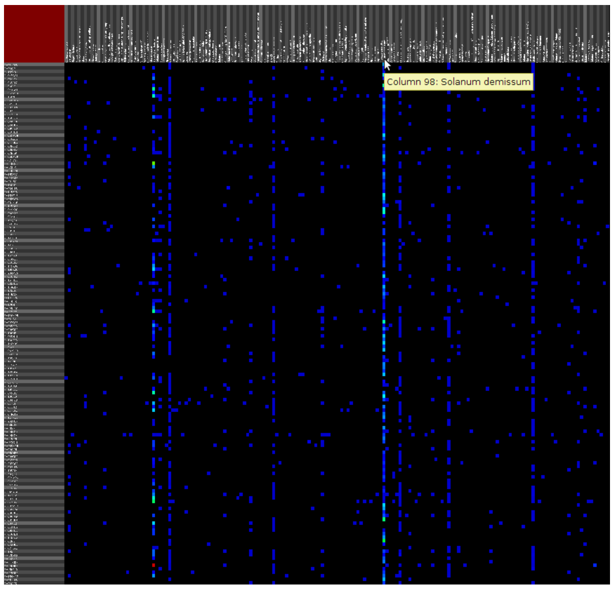
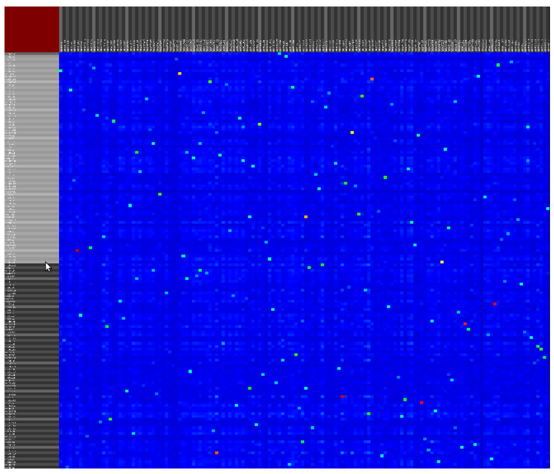


Figure 6.11: Screenshot of the property matrix visualization set to cross-reference *BACs* with *Annotations*, using the *SPECIES* type filter

6.5.2 Zooming and scrolling

The property matrix has support for zooming and scrolling, in both the row and column direction. The user can zoom in by selecting a part of the row or column (see figure 6.12(a)), the selection will subsequently become the visible part of the dot-plot (see figure 6.12(b)).


(a) Selecting the row area

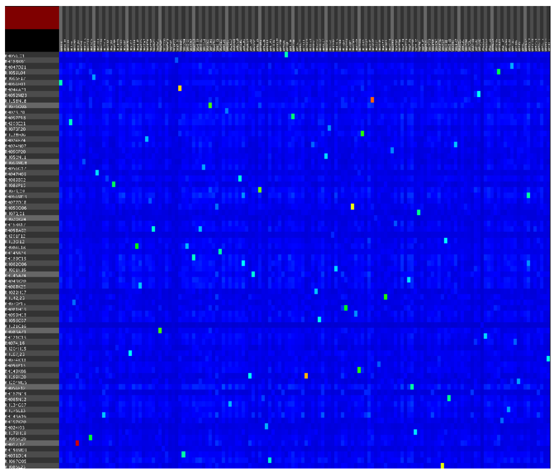
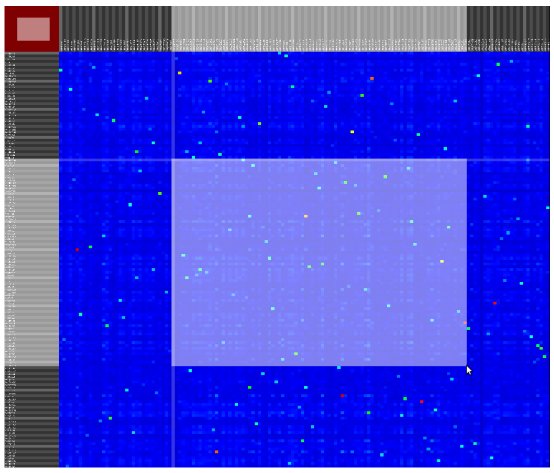

(b) Displaying the selected row area

Figure 6.12: Screenshot of the property matrix visualization and row area selection

The same zoom by selection technique can also be applied on the dot-plot itself (see figure 6.13(a)), a selected two dimensional area will become the visible part of the dot-plot (see figure 6.13(b)).


(a) Selecting the dot-plot area

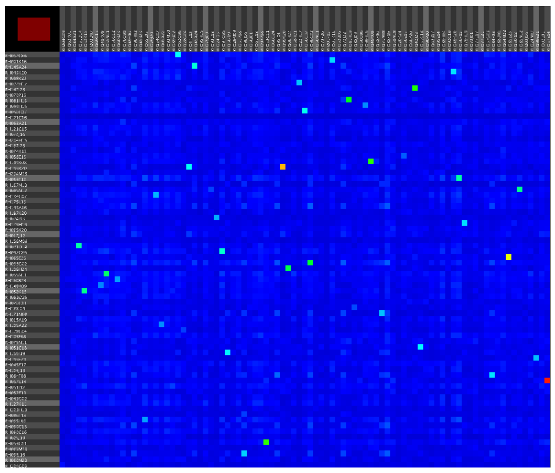

(b) Displaying the selected dot-plot area

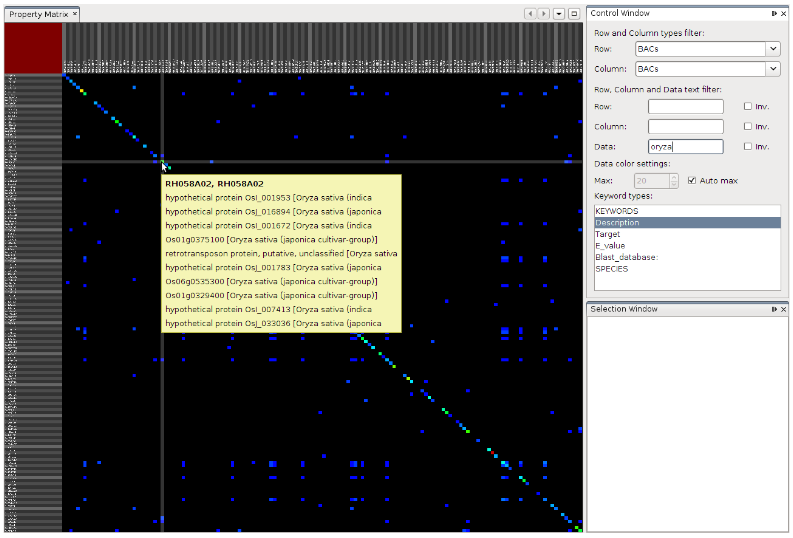
Figure 6.13: Screenshot of the property matrix visualization and dot-plot area selection

6.5.3 Filtering

The property matrix enables the user to include or exclude one or more annotation types. This filter can be used within all cross-reference configurations. Figure 6.15 has already used this filter type to visualize only the SPECIES annotations within the BACs of the PGSC dataset. The available annotation types are listed in the control window of the property matrix, the selection of none or all of the annotation types will both result in the visualization of all the annotation types in the dataset.

Beside the annotation type filter it is also possible to filter on raw text. There is a separate text filter for the row, column and data elements. The text filtering can also be inverted to remove data containing a specific text.

Figure 6.14 shows the result of using the text filter on a *BACs* by *BACs* cross-reference, with a *Description* annotation type filter. The resulting dot-plot will only display the data elements that contain the requested ”oryza” text.

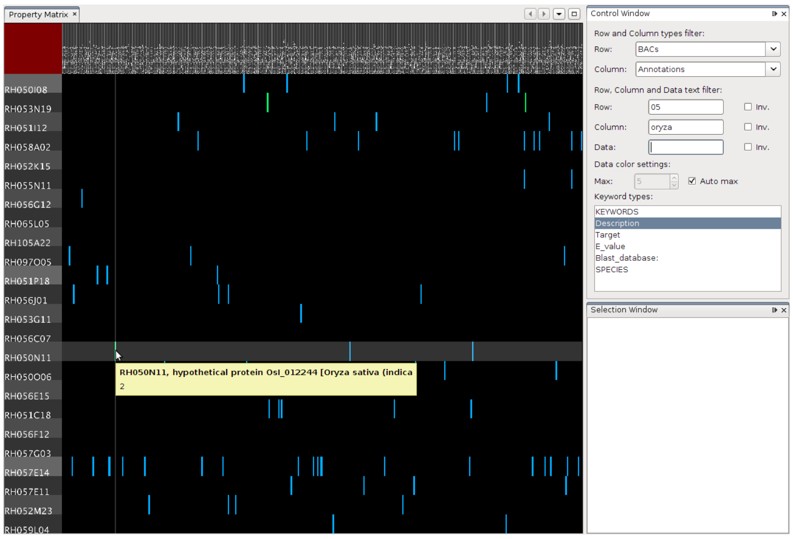


RH058A02, RH058A02
hypothetical protein OsI_001953 [Oryza sativa (indica
hypothetical protein OsJ_016894 [Oryza sativa (japonica
hypothetical protein OsI_001672 [Oryza sativa (indica
Os01g0375100 [Oryza sativa (japonica cultivar-group)]
retrotransposon protein, putative, unclassified [Oryza sativa
hypothetical protein OsJ_001783 [Oryza sativa (japonica
Os06g0535300 [Oryza sativa (japonica cultivar-group)]
Os01g0329400 [Oryza sativa (japonica cultivar-group)]
hypothetical protein OsI_007413 [Oryza sativa (indica
hypothetical protein OsJ_033036 [Oryza sativa (japonica

Row and Column types filter:
Row: BACs
Column: BACs
Row, Column and Data text filter:
Row: ☐ Inv.
Column: ☐ Inv.
Data: oryza ☐ Inv.
Data color settings:
Max: 20 ☑ Auto max
Keyword types:
KEYWORDS
Description
Target
E_value
Blast_database:
SPECIES

Figure 6.14: Screenshot of the property matrix visualization set to cross-reference *BACs* with *BACs*, using the *Description* type filter and the data text filter

Figure 6.15 shows the property matrix cross-referencing *BACs* with *Annotations*, using the *Description* annotation type. Both the row and column text filter are used to reduce the size of the dot-plot. The row displays only BACs containing the text *”05”*, and the column displays only the annotations that contain the text *”oryza”*.



RH050N11, hypothetical protein OsI_012244 [Oryza sativa (indica
2

Row and Column types filter:
Row: BACs
Column: Annotations
Row, Column and Data text filter:
Row: 05 ☐ Inv.
Column: oryza ☐ Inv.
Data: ☐ Inv.
Data color settings:
Max: 5 ☑ Auto max
Keyword types:
KEYWORDS
Description
Target
E_value
Blast_database:
SPECIES

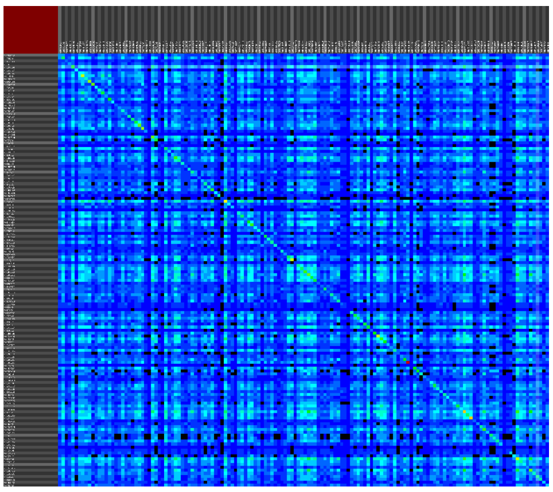
Figure 6.15: Screenshot of the property matrix visualization set to cross-reference *BACs* with *Annotations*, using the *Description* type filter and the row and column text filter

6.5.4 Sorting

The dot-plot can be sorted in a single direction (row or column), or in both directions (row and column). The combination of a row and column element is used as a key to request a dot-plot data element, so shifting a row or column element will automatically shift the complete dot-plot line associated to that row or column.

The property matrix provides two different sorting algorithms, the fist and simplest algorithm sorts the data element values of a single dot-plot line in descending order (from left to right). The row or column elements will be rearranged to accommodate the sorting, thereby rearranging the complete dot-pot. Figure 6.16(a) shows the property matrix cross-referencing *BACs* with *BACs*, using the *SPECIES* annotation type filter, figure 6.16(b) show the same information but with the first BAC sorted in both directions.

The sorting on minimal differences between the neboring (horizontal or vertical) dot-plot lines is computation intensive and can take quite some time when sorting a big dataset. This sorting technique can be used in both directions, resulting in the clustering of row and column elements that share similar data properties. Figure 6.17(a) shows the property matrix cross-referencing *Annotations* with *Annotations*, using the *KEYWORDS* annotation type filter, and figure 6.17(b) show the same information with the dot-plot sorted on minimal difference.


(a) Unsorted

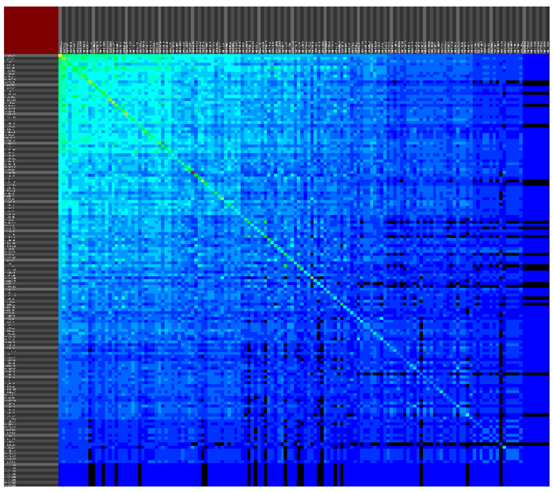
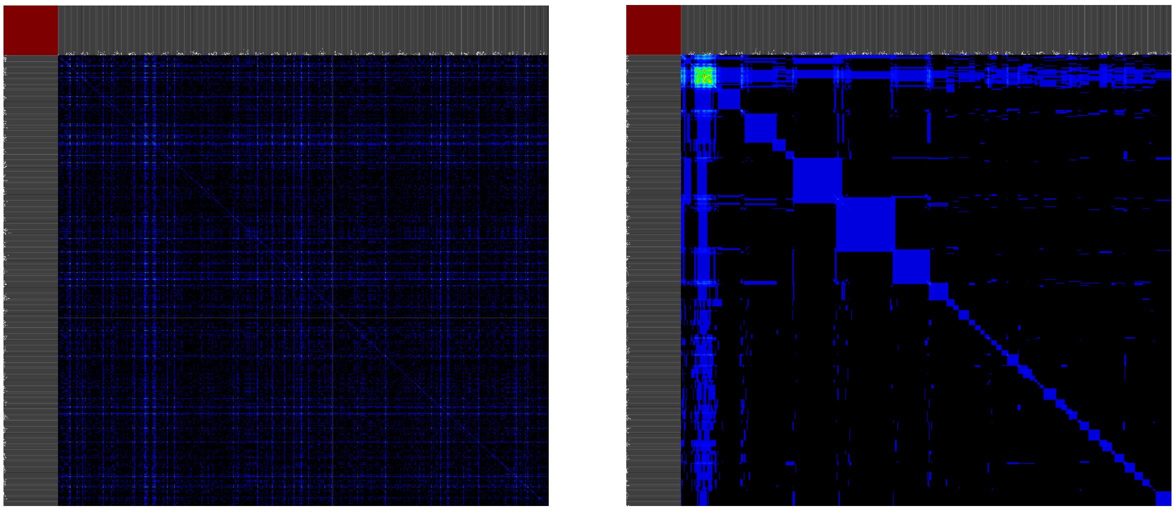

(b) Sorted on the data values of the first row and column

Figure 6.16: Screenshot of the property matrix visualization cross-referencing BACs with BACs, using the SPECIES type filter



(a) Unsorted (b) Sorted on minimal difference

Figure 6.17: Screenshot of the property matrix visualization cross-referencing Annotations with Annotations, using the KEYWORDS type filter

Chapter 7

Evaluation

This chapter evaluates the CBB visualization, circle visualization and the property matrix visualisation as described in the previous chapters (4, 5, and 6). First the requirements of section 3.4 are verified in section 7.1. Experimental results using the use-cases of section 3.5 as guideline are described in section 7.2.

7.1 Verification of requirements

This section verifies the requirements as described in section 3.4 by checking them against the new functionalities provided by the DNAVis2 extensions.

Requirement I (Data size) The current version of the PGSC dataset can be visualized and browsed within all tree visualizations. The high level CBB and circle visualizations provide a responsive near real time interface. The low level property matrix visualization is able to visualize all the combinations of high level and low level data elements, but can become sluggish and memory hungry when all the annotation data within the dataset are visualized at once. This visualization will need some additional optimizations, will it ever be able to visualize the final (complete) PGSC dataset.

Requirement II (Standardized data files) All the GFF and FASTA data of the PGSC dataset can be loaded using the existing DNAVis2 implementation. But the visualizations use a custom data structure to store the loaded annotations, because the existing (BioJava[1]) structures use too much memory.

Requirement III (Custom data files) The CBB file format describing the mutual relations between the chromosomes, BINs and BACs within the PGSC dataset can be loaded using a custom parser. This parser translates the CBB data into a tree like data structure that describes the high level structures of the dataset. This structure can be depicted using the CBB and circle visualization.

Requirement IV (Genome level) The individual chromosomes in a genome can be visualized using the CBB and circle visualization.

Requirement V (Chromosome level) The individual BINs and BIN-domains within a chromosome can be visualized using the CBB and circle visualization.

Requirement VI (BIN/BIN-domain level) The individual BACs within a BIN or BIN-domain can be visualized using the circle visualization.

[1]BioJava is an open-source project dedicated to providing a Java framework for processing biological data.

Requirement VII (BAC level) The existing linear view can be used to visualize the sequence and annotation data within a BAC. (The integration of a linear view like structure within the circle visualization was considered, but was dropped because of time constrains.)

Requirement VIII (Navigation) The circle visualization allows the user to move freely through all the high level abstraction levels ($genome \rightarrow chromosome \rightarrow BIN \wedge BIN - domain \rightarrow BAC$). The BACs that contain low level ($sequence \wedge annotation$) data can be opened within the existing linear view from the circle visualization.

Requirement IX (Overview) The property matrix provides an overview of the visualized data and the current location, using the small matrix overview in the top left corner. The CBB visualization keeps the entire visual representation of the high level data structure (except the individual BACs) in view, providing a natural global overview. The circle visualization uses another approach to keep the user location and context aware:

- The various abstraction levels have their own custom icon representation.
- The next data level and previous location (when available) are visualized within the visualization itself.
- The center element and the child element are labeled with textual clues.
- Tool tips provide detailed information about each element within the visualization.

Requirement X (Responsiveness) The high level CBB and circle visualizations provide a responsive near real time interface. The property matrix provides intermediate partial updates when processing large date visualizations, and divides the workload over multiple threads when a multi core processor is available.

Requirement XI (Selection) The high level CBB and circle visualizations provide an easy interface to select and deselect elements at multiple abstraction levels (see section 4.1.4 and 5.1.4).

Requirement XII (Filtering) The property matrix visualization provides an extensive collection of filters that enable the user to emphasize or suppress a large number of properties. The high level visualizations provide the user with a small number of automated selection macros like: *select all BACs*, *invert selection*, *select all BACs with GFF data* and *select all BACs with FASTA data* to speed up the selection process.

Requirement XIII (Emphasizing) The high level CBB and circle visualizations emphasize the fully and partly selected data elements, providing an intuitive interface to create sub-selections.

Requirement XIV (Data distribution) All three visualization provide some kind of data distribution visualization. The CBB visualization visualizes the BAC distribution across the BIN-domains (using coloring). The circle visualization (using icon scaling) and property matrix (using coloring) are distribution visualizations.

Requirement XV (Automatic data extraction) The high level circle visualization provides the user with a number of statistics like: the number of direct children in a element, the number of leafs (read BACs) in a element, the amount of FASTA and GFF data in a element (in bytes). The property matrix visualization extends the default annotations with a number of synthetically created annotation types that extract possibly interesting features from the default annotation data (a list of individual words, and a list of species declarations).

	I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XII	XIII	XIV	XV
CBB	+	+	+	+	+	-	-	-	+	+	+	-	+	+	-
Circle	+	-	+	+	+	+	±	+	+	+	+	+	+	+	+
PM	-	+	+	+	+	+	-	-	-	-	-	+	-	+	+

Table 7.1: Overview of the requirement validations per visualization

7.2 Verification using use-cases

This section verifies the proposed use-case scenarios described in section 3.5 using the DNAVis2 extensions.

Use-case I (BAC distribution)
The biologist can visualize the BAC distribution across the BIN-domains by opening the CBB file within the CBB visualization. This visualization presents the biologist with a visualization of all the chromosomes with the BIN-domains mapped on top. The color coding of the BIN-domains indicates the BAC distribution (see figure 7.1).

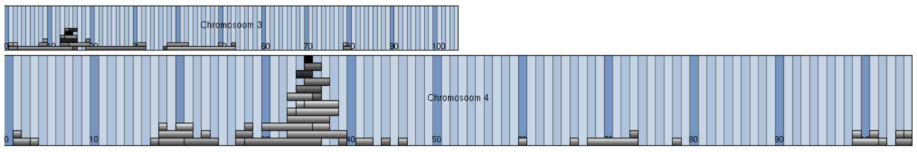


Figure 7.1: Screenshot of the CBB visualization displaying the BAC distribution across chromosome 3 and 4 using white to black color mapping.

The biologist can visualize the BAC distribution across the higher levels, by opening the CBB file in the circle visualization and setting the icon scaling to number of leafs (see figure 7.2 and 7.3).

Use-case II (Exploring high level)
The biologist can browse the abstraction levels, by opening the CBB file in the circle visualization and using the navigation mechanism described in section 5.1.1 to browse through the various levels (see figure 7.4(a), 7.4(b) and 7.4(c)).

Use-case III (Selection)
The biologist can create a sub-selection by opening the CBB file within the CBB visualization, the (de-)selection mechanism described in section 4.1.4 can be used to (de-)select chromosomes and BIN-domains (see figure 7.5).

The biologist can create a sub-selection by opening the CBB file in the circle visualization and using the navigation mechanism of use-case II (above) to browse through the various levels. The (de-)selection mechanism described in section 5.1.4 can be used to (de-)select the genome, chromosomes, BINs, BIN-domains and BACs.

Use-case IV (Common annotation)
The biologist can find the most common annotation by opening the CBB file within the property matrix visualization, and sets the cross reference types to annotation data and genome. The property matrix will then visualize the number of times annotations occur within the genome (see figure 7.7).

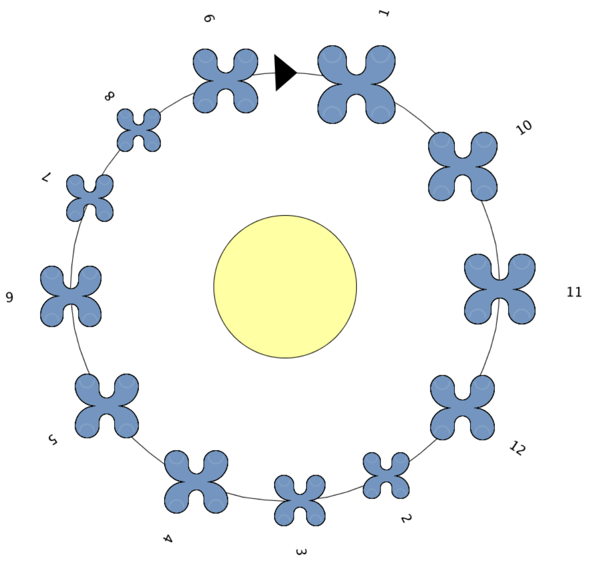


Figure 7.2: Screenshot of the circle visualization displaying the BAC distribution across the chromosomes using square root scaling of the number of BACs.

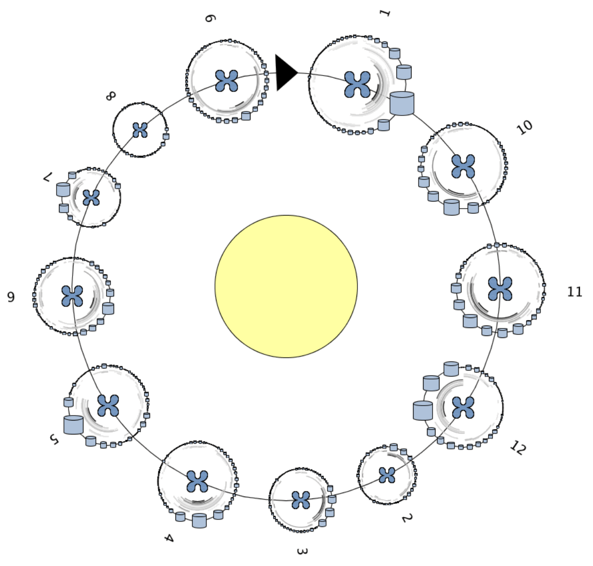
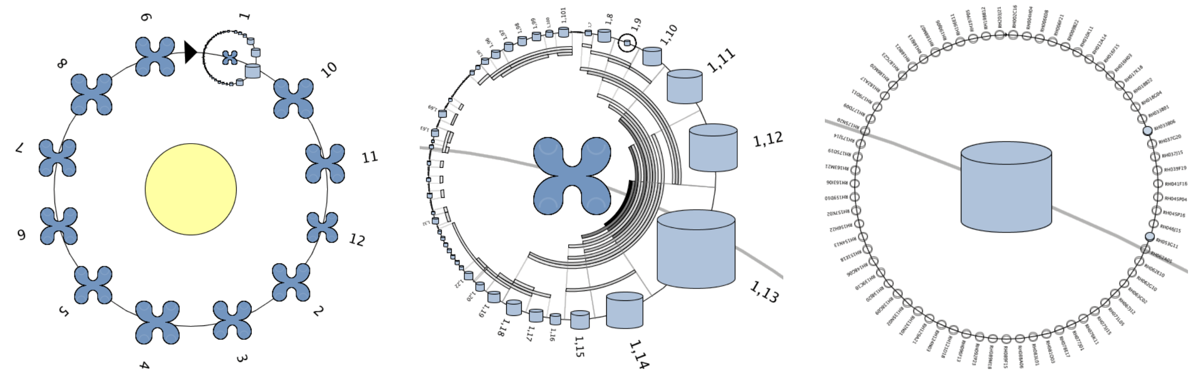


Figure 7.3: Screenshot of the circle visualization displaying the BAC distribution across the chromosomes and their BINs using square root scaling of the number of BACs.



(a) Genome level (b) Chromosome level (c) BIN level

Figure 7.4: Screenshots of three consecutive data levels in the circle visualization

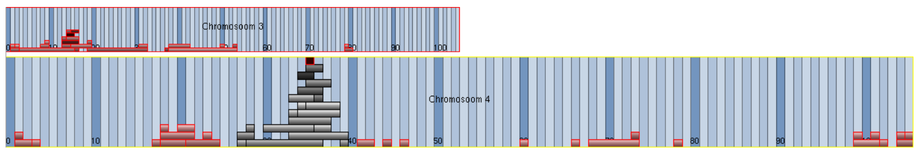
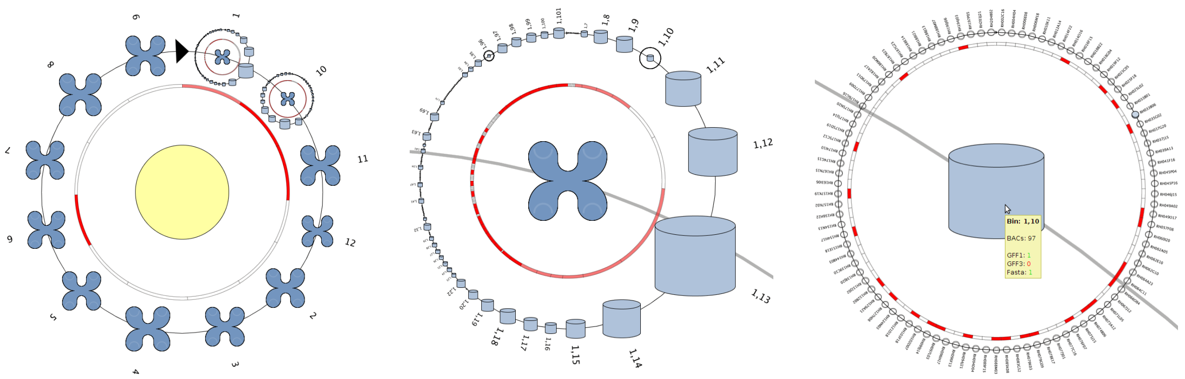
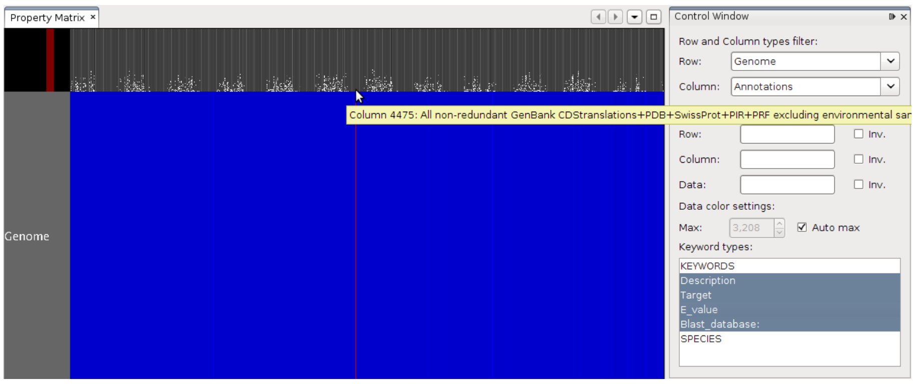


Figure 7.5: Screenshot of the CBB visualization displaying the BAC distribution using black to white color mapping and selection.



(a) Genome level (b) Chromosome level (c) BIN level

Figure 7.6: Screenshots of three consecutive data levels in the circle visualization



Column 4475: All non-redundant GenBank CDStranslations+PDB+SwissProt+PIR+PRF excluding environmental sar

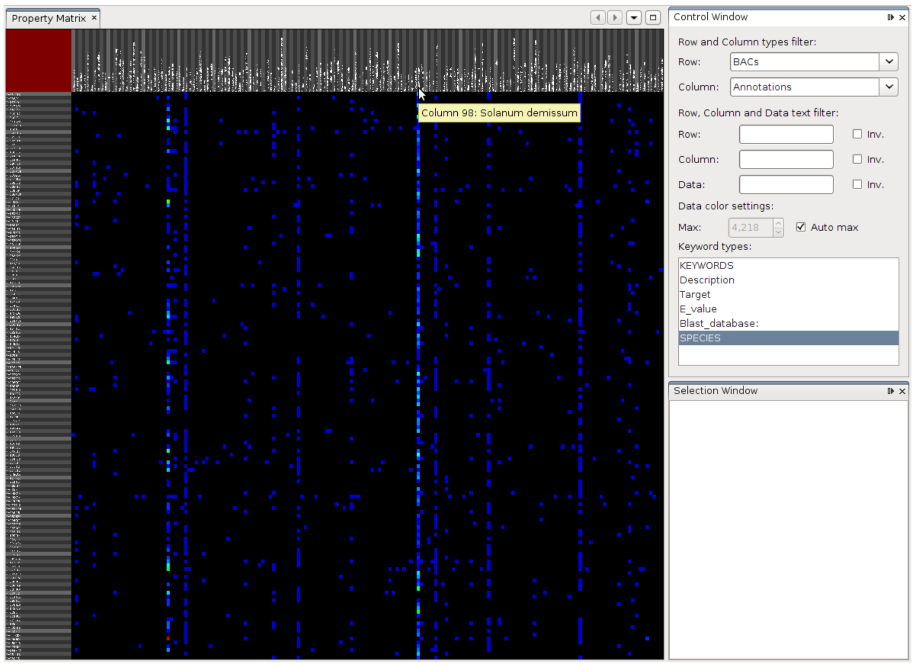
Row and Column types filter:
Row: Genome
Column: Annotations
Row: ☐ Inv.
Column: ☐ Inv.
Data: ☐ Inv.
Data color settings:
Max: 3.208 ☑ Auto max
Keyword types:
KEYWORDS
Description
Target
E_value
Blast_database:
SPECIES

Figure 7.7: Screenshot of the property matrix visualization set to cross reference genome with annotations, using rainbow color mapping

The biologist can also load the data indirectly into the property matrix visualization, by opening the sub-selection (all the BACs with annotation data) for instance in a property matrix visualization.

Use-case V (Specific annotation)
The biologist can find all BACs with a specific annotation by opening the CBB file within the property matrix visualization, and set the cross reference types to annotation data and BAC. The property matrix will then visualize the number of times an annotation occurs within the BACs (see figure 7.8). The biologist can also load the data indirectly into the property matrix visualization, see use-case IV (above).

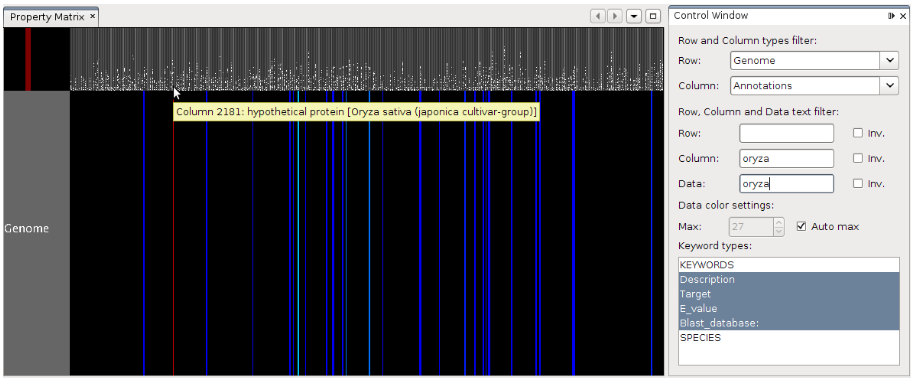
Use-case VI (Specific text)
The biologist follows all the steps of use-case IV (above) to visualize all the annotation data within the genome, and uses the annotation data text filter to filter for the specific text. The property matrix will then visualize only the annotations that contain this specific text (see figures 7.9 and 7.10).



Column 98: Solanum demissum

Row and Column types filter:
Row: BACs
Column: Annotations
Row, Column and Data text filter:
Row: ☐ Inv.
Column: ☐ Inv.
Data: ☐ Inv.
Data color settings:
Max: 4,218 ☑ Auto max
Keyword types:
KEYWORDS
Description
Target
E_value
Blast_database:
SPECIES

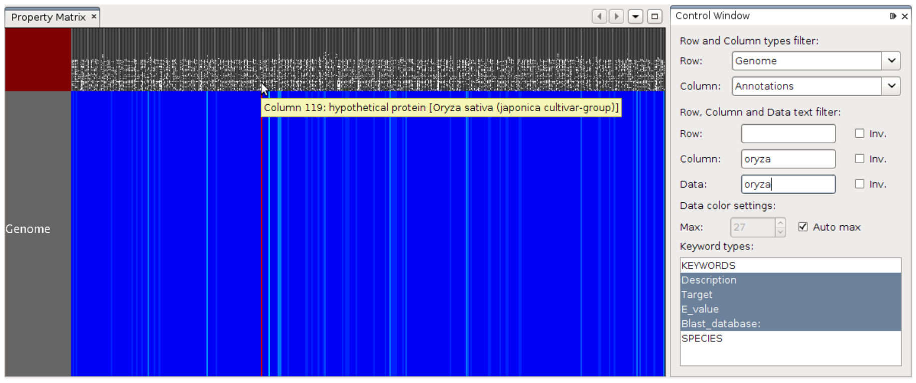
Figure 7.8: Screenshot of the property matrix visualization set to cross reference BACs with annotations, using rainbow color mapping



Column 2181: hypothetical protein [Oryza sativa (japonica cultivar-group)]

Row and Column types filter:
Row: Genome
Column: Annotations
Row, Column and Data text filter:
Row: ☐ Inv.
Column: oryza ☐ Inv.
Data: oryza ☐ Inv.
Data color settings:
Max: 27 ☑ Auto max
Keyword types:
KEYWORDS
Description
Target
E_value
Blast_database:
SPECIES

Figure 7.9: Screenshot of the property matrix visualization set to cross reference genome with annotations, using rainbow color mapping and raw text filtering on the dot-plot elements



Column 119: hypothetical protein [Oryza sativa (japonica cultivar-group)]

Control Window

Row and Column types filter:

Row: Genome

Column: Annotations

Row, Column and Data text filter:

Row: ☐ Inv.

Column: oryza ☐ Inv.

Data: oryza ☐ Inv.

Data color settings:

Max: 27 ☑ Auto max

Keyword types:

KEYWORDS
Description
Target
E_value
Blast_database:
SPECIES

Figure 7.10: Screenshot of the property matrix visualization set to cross reference genome with annotations, using rainbow color mapping and raw text filtering on the dot-plot and column elements

	I	II	III	IV	V	VI
CBB	+	±	+	-	-	-
Circle	+	+	+	-	-	-
PM	-	-	-	+	+	+

Table 7.2: Overview of the use-case validations per visualization

7.3 Results

The combination of the circle and property matrix visualization are able to validate all the requirements described in section 3.4 (see table 7.1), with the observation that the circle visualization needs the existing linear view to open the low level sequence and annotation data due to the lack of support for the visualization low level data. The CBB visualization is more limited than the circle visualization, but is still able to satisfy most of the high level data related requirements.

The use-case evaluation shows a similar result in table 7.2, where the combination of circle and property matrix visualization are able to validate al the use-case scenarios.

Chapter 8

Conclusion

8.1 Answering the research questions

This section will describe the measure in which the research questions and the problem statement are satisfied. Starting with the three research questions as defined in section 3.1.2.

> *RQ1: Is it possible to store and analyze an genome size dataset within the constrained environment of a modern personal computer?*

Both the CBB and circle visualization are able to open, store and visualize the high level structure of the current PGSC dataset. The current version of the PGSC dataset contains more high level (13,319 BACs) data than the final version will have (8,400 BACs) [11]. Therefore it's safe to conclude that both visualization will have no problem displaying the high level structure of the complete genome.

The property matrix visualization is able to open, store and visualize the annotation data of the current PGSC dataset. The current version of the PGSC dataset contains only 2.3% (179 BACs) of the sequencing and annotation data of the final version (8,400 BACs). The implementation pushes the memory limits of the Java virtual machine when setting the visualization to cross-reference annotation data with annotation data. This results in a visualization of about $10,000 \times 10,000 = 100,000,000$ dot-plot elements. So it's unlikely (but not impossible) that the visualization will be able to display the annotation data of a complete genome. But it can be concluded that the current implementation can visualize the current version of the PGSC dataset.

For both the high and low level it's safe to say that loading the data in itself is not the problem, but it can be hard to keep the visualizations of the data within the memory constrains of the Java environment.

> *RQ2: How does one visualize the structural properties of a genome size dataset?*

The CBB and circle visualization are used to visualize the high level data as described in section 2.2.2. Both visualizations have their own strong points and weaknesses (see section 7.1 and 7.2), but overall the circle visualization will be the preferred visualization to visualize the high level structural properties of the dataset, because the CBB visualization does not visualize the individual BACs within the dataset.

> *RQ3: How does one visualize the annotation data of a genome size dataset?*

The property matrix visualization is used to visualize the annotation data of the PGSC dataset. The visualization uses a dot-plot matrix with color coded elements, and can depict all annotation data within a single visualization.

> *RQ4: Is it possible to preserve the responsive and interactive feeling of the current DNAVis2 when visualizing genome size datasets?*

Both CBB and circle visualization provide a near real-time interface while browsing and exploring the PGSC dataset, which is in line with the existing DNAVis2 visualizations.

The circular visualization can become a bit sluggish when visualizing all annotation data within the PGSC dataset at once, but is very responsive when visualizing smaller subsets. The visualization tries to reduce its lag providing partial updates and by utilizing multiple processor cores when available. But it's unlikely that the current implementation will stay responsive during the visualization of the final PGSC dataset.

8.2 Overview of the achievements

The following listing provides a overview of the achievements reached during this master project:

- Added a user-friendly interface to browse and explore the high level data structure of the PGSC dataset, in the form of the circle visualization (and CBB visualization).
- Added a flexible interface to examine the low level annotation data of the PGSC dataset, in the form of the property matrix visualization.
- Provided a high level interface, that enables the user to create sub-selection of the dataset and to open the low level data within other visualizations like the linear view and property matrix.

8.3 Overview of the limitations

The following listing provides an overview if the limitations of the new DNAVis2 extensions:

- The high level structure extensions only support the custom (in-house format of the PRI) CBB file format, making them useless when visualizing data from other sources.
- The CBB visualization cannot visualize individual BACs, only the number of BACs in a BIN-domain.
- The circle visualization needs to use additional visualizations to display the low level sequence and annotation data.
- The high level visualizations have no knowledge about the low level information besides knowing its existence and the data size in bytes.
- The annotation data within the property matrix visualization in not linked to other visualizations.
- The lack of direct feedback of real biologists, has led to a visualization style that is purely based on the vision and perception of people outside the field of DNA research. A singular session with a biologist of the PRI led us to believe that the visualizations provide different but useful perspective regarding the PGSC dataset. The biologist was overall very positive about the explorative interface of the circular visualizations, and the versatility of the property matrix visualization. But a single enthusiastic biologist is no guarantee that biologists in general will appreciate the produced visualizations.

8.4 Verdict

The problem statement of section 3.1.1 requests for extension of the DNAVi2 tool set that adds support for genome size datasets, while preserving the near real-time properties of the original DNAVis2.

This problem statement is largely fulfilled by the three visualizations that where added during this master project. The current extended DNAVis2 is able to provide the user with a high level and low level overview of the "benchmark" (PGSC) dataset. Both the CBB and circle visualization are able to provide the user with a responsive interface to browse and explore the high level structural data, while providing access to the low level sequence and annotation data.

The visualization of all the annotation data within a dataset of this size proved to be more difficult to do in near real-time. The current property matrix visualization implementation can visualize all the annotation data within the "benchmark" (PGSC) dataset, but can become sluggish when the dot-plot matrix gets too big. This and the memory requirements of this visualization need to be optimized in order to fully fulfill the problem statement.

8.5 Future work

The DNAVis2 extensions described in this thesis are the first attempt to add support for high level data structures to the DNAVis2 tool set. Like most first attempts, it is not perfect. This section will describe a number of suggestions for improving the new extensions.

8.5.1 CBB visualization

This listing describes a number of suggestions for improvements of the CBB visualization:

- *Replace by circle visualization:* The CBB visualization can probably be fully replaced by the circle visualization, despite the fact that the circle visualization doesn't keep all high level data element in view while exploring the various data levels.
- *Individual BACs:* The CBB visualization cannot visualize individual BACs and availability of low level data.
- *Java2D for compatibility:* The usage of the Java2D library instead of the JOGL bindings will improve compatibility. The visual complexity of the CBB visualization is quite low, so performance is not an issue.
- *Redraw optimization:* Optimize the redrawing of the visualization. The current implementation uses multiple threads (one per chromosome) to speed up the BIN-domain mapping calculations. But the redrawing of the visualization after selecting or deselecting a data element is not optimized, and will result in a redraw of the whole visualization.

8.5.2 Circle visualization

This listing describes a number of suggestions for improvements of the circle visualization:

- *Low level data:* The circle visualization can be extended to display the low level data, to enable the user to view the complete dataset within a singular visualization.
- *Low level scaling:* The icon scaling can be extended to show the distribution of low level sequence and annotation data within the dataset. Think of scaling on the number of specific sub-sequences or annotations.
- *Animation:* Animation of the level transitions can help a user to better remember its current location.
- *Path tracer:* The addition of internet browser like previous and next buttons to trace back the steps taken through the data. (A nice combination with the animation of the levels transitions described above.)

- *Overview:* An overview mode that temporary zooms out to show the path from the current location to the root.
- *Multithreading:* Multithreading and partial updating, to improve the performance for big datasets. The recursive nature of the circle visualization (recursion per child node) makes it quite easy to add multiprocessor support to the circle renderer and to do partial redraws.
- *Improve screen usage:* There are possibly better ways to visualize circle nodes with a large number of children, than the proposed phi-ball solution (see section 5.2.1). For example a extension of the basic circle visualization with a movable magnified area, that scales up a part of the circle while compressing the rest (see figure 8.1).

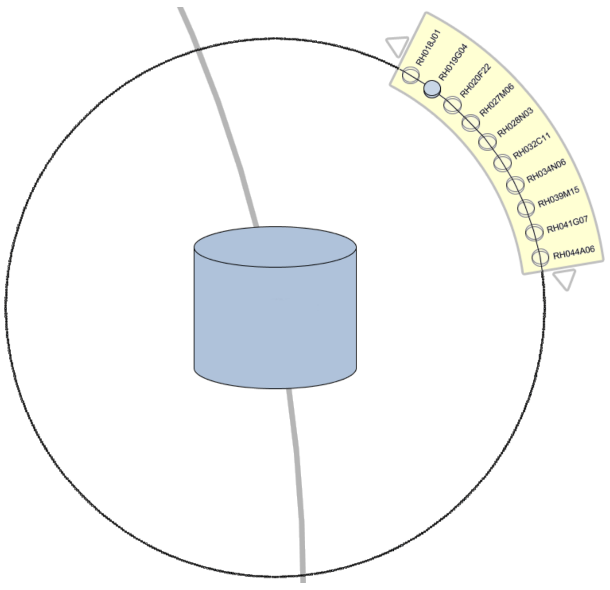


Figure 8.1: Concept drawing of the circle visualization depicting 641 children with a magnified area.

8.5.3 Property matrix visualization

This listing describes a number of suggestions for improvements of the property matrix visualization:

- *Linking:* The linking of the property matrix and other visualizations like the circle visualization and linear view can possibly prove useful. For example a link between the property matrix and a linear view could provide addition options like: the highlighting of annotation in the property matrix that are selected in the linear view, using the annotations selected in a linear view as filter in the property matrix, highlight or filter data in the linear view using information from the property matrix.
- *Manual move or remove:* The option to move or remove rows or columns can be useful to create a custom data overview.
- *Sorting:* Better sorting algorithms, which possibly use biological knowledge to sort the rows and columns of the dot-plot.
- *Guarantee visibility:* Guarantee the visibility of the highest data values, let a pixel represent the highest data value when multiple dot-plot elements share the area of a single pixel.

- *Optimize low detail:* The current property matrix relies on zooming, scrolling and sorting in clusters to visualize large data visualizations (5000 $\times$ 5000 dot-plot, that has multiple data elements per pixel). There may be better ways of presenting these large cross-references.
- *Java2D for compatibility:* The usage of the Jave2D library instead of the JOGL bindings will improve compatibility. The property matrix visualization consists of simple box shaped elements, but the sheer number of elements may pose a performance problem when Java2D is used. (Especially under Linux/Unix based operating systems, because the hardware GPU acceleration of Java2D is currently less mature on these platforms.)
- *Redraw optimization:* Optimize the redrawing of the visualization. The current implementation uses multiple threads to speed up the cross-referencing calculations. But the redrawing of the visualization is not optimized, and will result in a redraw of the whole visualization (even for small actions like the highlighting of the mouse pointer location (see section 6.1.4))

8.6 DNAVis2 improvement suggestions

This section suggests a number of possible improvements and suggestion that come to mind after having spent much time with the DNAVis2 tool set:

- The advantages of rewriting DNAVis2 in Java to improve better portability between different operating systems are largely undone by using the JOGL library. So it may be wise to investigate the option to replace the JOGL visualization by a default library like Java2D.
- The current user-interface lacks uniformity, and the added extensions are certainly no exception to this. The user experience will certainly improve when the tool gets a more uniform navigation and visualization style.
- The memory limit (about 2GB) of the current Java virtual machine can become a problem when the tools focus changes to supporting larger (genome size) datasets. But future improvements to the virtual machine and the prevailing of 64-bit operating systems will most likely fix this problem.

Bibliography

[1] The Sequence Ontology Project and L. Stein. Generic feature format version 3. `http://www.sequenceontology.org/gff3.shtml`, 2008.

[2] E. Angel. *OpenGL : A Primer.* Pearson Education, Inc., 3rd edition edition, 2008.

[3] M. Balzer and O. Deussen. Voronoi treemaps. *Information Visualization, IEEE Symposium on*, 0:7, 2005.

[4] BioJava. The biojava project. `http://biojava.org`, 2009.

[5] G.W. Burns and P.J. Bottino. *The science of genetics.* Macmillan Publishing Company, 6th edition edition, 1989.

[6] M.D. Byrne. Reading vertical text: Rotated vs. marquee. *Human Factors and Ergonomics Society Annual Meeting Proceedings*, 46:pp. 1633–1635, 2002.

[7] D.P. Clark. *Molecular biology: Understanding the genetic revolution.* Elsevier Academic Press, 2005.

[8] International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature*, 409:860–921, 2001.

[9] International Human Genome Sequencing Consortium. Finishing the euchromatic sequence of the human genome. *Nature*, 431:931–945, 2004.

[10] International Human Genome Sequencing Consortium. The human genome project. `http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml`, 2009.

[11] Potato Genome Sequencing Consortium. The potato genome sequencing consortium (pgsc). `http://www.potatogenome.net`, 2009.

[12] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press and McGraw-Hill, 2001.

[13] EMBL EBI and the Wellcome Trust Sanger Institute. Ensembl. `http://www.ensembl.org/index.html`, 2009.

[14] C. Faloutsos. Gray codes for partial match and range queries. *IEEE Transactions on Software Engineering*, 14(10):1381–1393, 1988.

[15] F. Gray. Pulse code communication. U.S. Patent 2 632 058, 1953.

[16] Khronos Group. Opengl. `http://www.opengl.org`, 2009.

[17] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 26(2):147–160, 1950.

[18] Broad Institute. Argo genome browser. `http://www.broadinstitute.org/annotation/argo`, 2009.

[19] Wellcome Trust Sanger Institute. Gff: an exchange format for feature description. `http://www.sanger.ac.uk/Software/formats/GFF`, 2009.

[20] Wellcome Trust Sanger Institute. The vertebrate genome annotation (vega). `http://vega.sanger.ac.uk/index.html`, 2009.

[21] JOGL. Java bindings for opengl. `https://jogl.dev.java.net`, 2009.

[22] Ernst Kleiberg, Huub van de Wetering, and Jarke J. Van Wijk. Botanical visualization of huge hierarchies. *Information Visualization, IEEE Symposium on*, 0:87, 2001.

[23] M. Krzywinski. Circos - visualizing the genome, among other things, `http://mkweb.bcgsc.ca/circos/?home` 2009.

[24] Lawrence Berkeley Mational Laboratory. Vista. `http://genome.lbl.gov/vista`, 2009.

[25] Bernd Lintermann and Oliver Deussen. Interactive modeling of plants. *IEEE Computer Graphics and Applications*, 19(1):56–65, 1999.

[26] Natalia Lpez, Manuel Nez, Ismael Rodrguez, and Fernando Rubio. Introducing the golden section to computer science. *Cognitive Informatics, IEEE International Conference on*, 0:203, 2002.

[27] Sun Microsystems. Class collections. `http://java.sun.com/j2se/1.4.2/docs/api/java/util/Collections.html`, 2009.

[28] Sun Microsystems. Class defaultmutabletreenode. `http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/tree/DefaultMutableTreeNode.html`, 2009.

[29] Sun Microsystems. Java 2d api. `http://java.sun.com/products/java-media/2D/index.jsp`, 2009.

[30] University of Michigan. Glossary for the nsf/darpa/nasa digital library project. `http://www.si.umich.edu/UMDL/glossary.html`, 2009.

[31] OpenGL(R), D. Shreiner, M. Woo, J. Neider, and T. Davis. *OpenGL(R) Programming Guide : The Official Guide to Learning OpenGL(R), Version 2 (5th Edition).* Addison-Wesley Professional, August 2005.

[32] T.H.J.M. Peeters. Interactive visualization of annotated dna sequences. Master's thesis, Eindhoven University of Technology, 2004.

[33] ProteomeCommons.org. Hupo standardized fasta format. `https://proteomecommons.org/tranche/examples/proteomecommons-fasta/fasta.jsp`, 2009.

[34] N.B. Robbins. *Creating More Effective Graphs.* Wiley-Interscience, 2005.

[35] N. Semova, R. Storms, T. John, P. Gaudet, P. Ulycznyj, X.J. Min, J Sun, G. Butler, and A Tsang. Generation, annotation, and analysis of an extensive aspergillus niger est collection. *BMC Microbiology*, 6:7, 2006.

[36] R. Spence. *Information Visualization.* ACM Press Books, Pearson education ltd., Edinburgh Gate, Harlow, Essex CM20 2JE, England, first edition, 2001. ISBN 0-201-59626-1.

[37] H.M.M. van de Wetering. 2d phi-ball demo application, 2009.

[38] Frank van Ham and Jarke J. van Wijk. Beamtrees : Compact visualization of large hierarchies. *Information Visualization, IEEE Symposium on*, 0:93, 2002.

[39] Jarke J. van Wijk and Huub van de Wetering. Cushion treemaps: Visualization of hierarchical information. *Information Visualization, IEEE Symposium on*, 0:73–78, 1999.

[40] Roel Vliegen, Jarke J. van Wijk, and Erik-Jan van der Linden. Visualizing business data with generalized treemaps. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):789–796, 2006.

[41] G. Watson. Lecture lecture 15 - visualisation of abstract information (edinburgh virtual environment centre), 2004.

[42] J. D. Watson and F. H. C. Crick. Genetical implications of the structure of deoxyribonucleic acid. *Nature*, 171(4361):964–967, May 1953.

[43] C.M.E. Willems. Interactive visualization in large scale comparative genomics. Master's thesis, Eindhoven University of Technology, 2007.

Appendix A

Development environment

A.1 Development environment

The DNAVis2 extensions are written in Java using the Netbeans IDE 6.1 development environments and Java Development Kit 6. DNAVis2 makes heavy use of the Netbeans rich-client application framework, for its graphical user interface and file handling. JOGL is used to provides the Java bindings for OpenGL, enabling GPU support. The BioJava libray provides FASTA and GFF file support, data structures, and parsing routines for the sequence and annotation data.

The Java code base of DNAVis2 should guarantee flawless operation on a large number of operating systems. But the JOGL library can prove to be problematic on more exotic operating systems, because it uses operating system specific bindings. The BioJava libray is written in Java, and will work on all Jave supporting platforms.

The DNAVis2 extensions are developed on two differed machines, with the following hardware specifications:

Development system at TU/e	
CPU	Intel Core 2 Quad Q6600 2.4GHz
RAM	3GB DDR2-800 (single channel mode)
GPU	Nvidia GeForce 8800GT 512MB
OS	Ubuntu 8.04, 8.10 and 9.04
Development system at home	
CPU	Intel Core 2 Duo E4400 3.0GHz
RAM	4GB DDR2-800 (dual channel mode)
GPU	ATI Radeon HD4870 1GB
OS	Microsoft Vista x64 and Windows 7 RC x64

Appendix B

Additional domain knowledge

B.1 CBB file format

The CBB file format is custom format used to describe the positions of the BACs, by there chromosome and BIN-domain (hence the self contrived name "CBB" (Chromosome, BIN, BAC)). The format is only used in-house by the PGSC, and has no documentation or tool support. The position data is described using a semi-colon (`;`) separated column structure, and all values are quoted with double quotes (`"`). Each line represents a single BAC position.

The lack of documentation makes is difficult interpret the data represented by the nine columns, its not even sure that all columns represent sensible data. But the following four columns are used to determine the BACs location:

- `<column 1>` The BAC name
- `<column 3>` The chromosome name
- `<column 5>` The BIN number, or the center of the BIN-domain
- `<column 6>` The uncertainty factor of the BIN location

The BAC name `<column 1>` is always known. But this is not always the case for the chromosome name and BIN number/BIN-domain. Unknown chromosome names `<column 3>` are indicated by a capital `"UNKNOWN"`, instead of a name or number value. The BIN position has to be calculated using both the BIN number `<column 5>` and the uncertainty factor `<column 6>`. A precisely known BIN location has a uncertainty factor of `"0"`, the BIN number in `<column 5>` indicates the position. But when the position is less certain, a positive uncertainty factor is provides, in steps of 0.5. For example a BIN number of `"5.5"` and a uncertainty factor of `"1.5"` indicates the BIN-domain 4 to 7.

The CBB file contains plain text data in the form as shown in listing B.1 (one line per sequenced BAC).

Listing B.1: CBB example data from the PGSC dataset

```
"RH001P23";"NONE";"UNKNOWN";"#0000789";"0";"0";"NONE";"0";"0"
"RH001P24";"NONE";"UNKNOWN";"Singletons";"0";"0";"NONE";"0";"0"
"RH002A01";"NONE";"2";"#0006868";"60";"0";"NONE";"0";"0"
"RH002A02";"NONE";"9";"#0000873";"29";"5";"NONE";"0";"0"
"RH002A03";"NONE";"UNKNOWN";"#0004260";"0";"0";"NONE";"0";"0"
"RH002A04";"NONE";"11";"#0001969";"55";"3";"NONE";"0";"0"
"RH002A05";"NONE";"9";"#0000146";"31";"0";"NONE";"0";"0"
"RH002A06";"NONE";"UNKNOWN";"Chloroplast";"0";"0";"NONE";"0";"0"
```

Appendix C

Additional problem description information

C.1 Sub-case scenarios and diagrams

The following use-cases will be used as sub-cases in the UML use-case diagrams of section 3.5 and C.2. They describe the functionality of commonly used data manipulations and visualizations.

Sub-case I *High level structure*
The DNAVis2 tool set reads the BAC locations for all the BACs with a known chromosome and BIN domain from the CBB file, and generates a tree like structure of the form: $genome \rightarrow chromosome \rightarrow BIN \rightarrow BAC$, and scans the base-directory (of the CBB file) and its subdirectories for FASTA and GFF files with the same name as the BACs within the tree leafs. The file location and file size information of the matching FASTA and/or GFF files is added to the corresponding tree leafs, resulting in the following data structure: $genome \rightarrow chromosome \rightarrow BIN \rightarrow BAC \rightarrow Sequence \wedge Annotation$.

Sub-case II *High level structure visualization*
The biologist visualizes one or more of the five different high level structure visualizations, and selects or deselects BACs[1] when needed.

Sub-case III *Annotation structure*
The DNAVis2 tool set reads the annotation information from the GFF file format, and generates a data structure that stores the cohesion between the BAC names, the annotation types, and the annotations itself.

Sub-case IV *Annotation visualization*
The biologist selects two data types (*structural data*, *annotation types* or *annotation data*) and explores their mutual relationships. Filtering can be uses to refine the visualization.

C.2 UML use-case diagrams

This section contains the UML use-case diagrams belonging to section 3.5. These UML use-case diagrams use the sub-cases described of section C.1 above.

[1]The selections are defined in BACs because it's the smallest subpart of the high level structural data. Higher level data elements can be selected by selecting all the BACs that belong to that element.

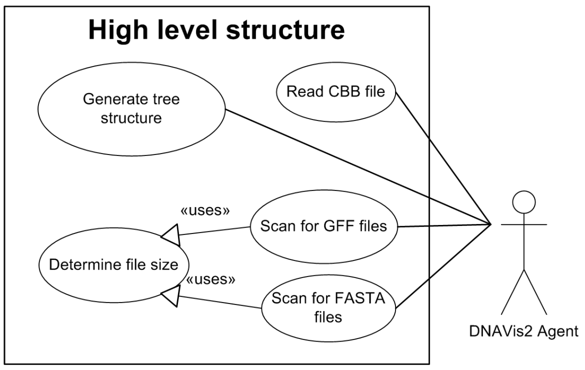


Figure C.1: Sub-case I

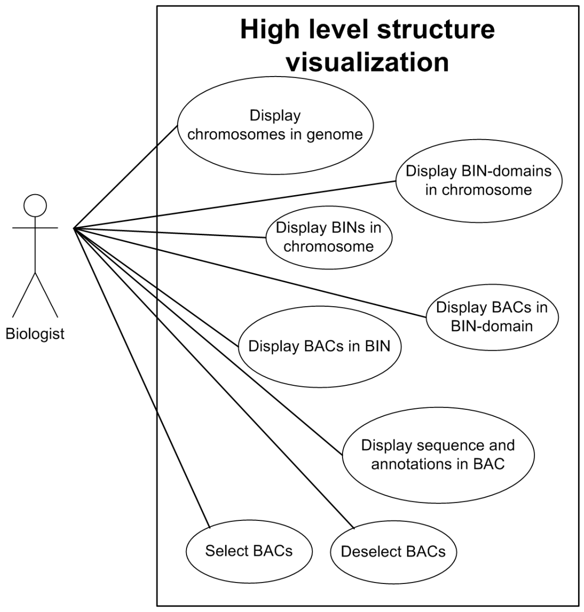

High level structure visualization
Display chromosomes in genome
Display BIN-domains in chromosome
Display BINs in chromosome
Display BACs in BIN-domain
Display BACs in BIN
Display sequence and annotations in BAC
Select BACs
Deselect BACs
Biologist

Figure C.2: Sub-case II

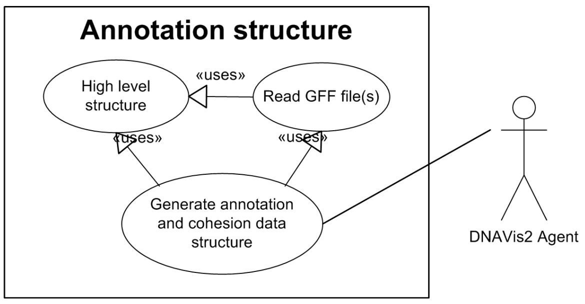


Figure C.3: Sub-case III

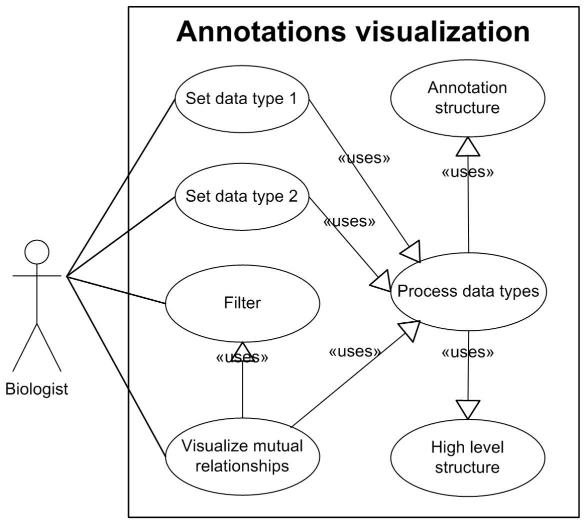


Figure C.4: Sub-case IV

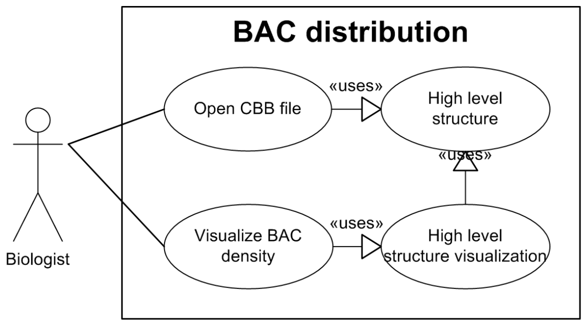


Figure C.5: Use-case I

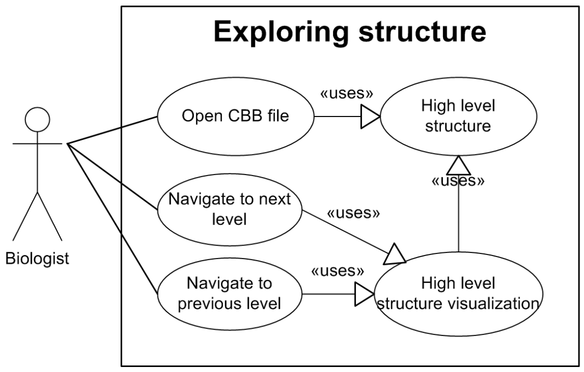


Figure C.6: Use-case II

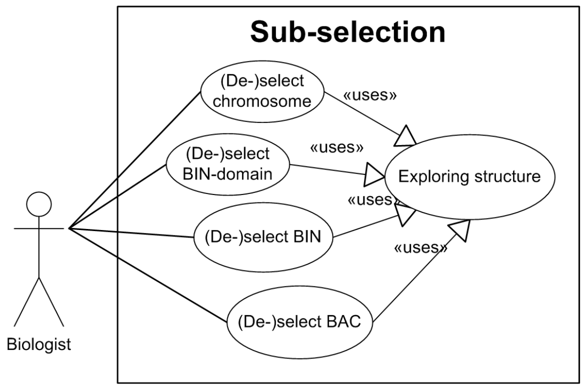


Figure C.7: Use-case III

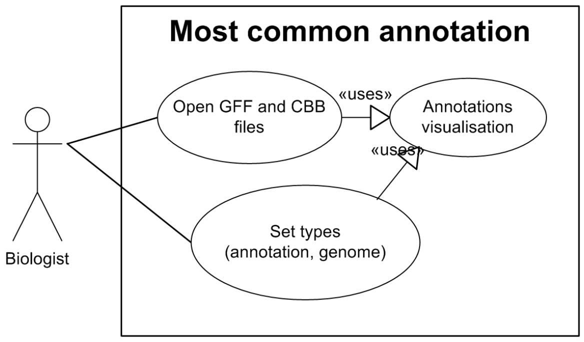


Figure C.8: Use-case IV

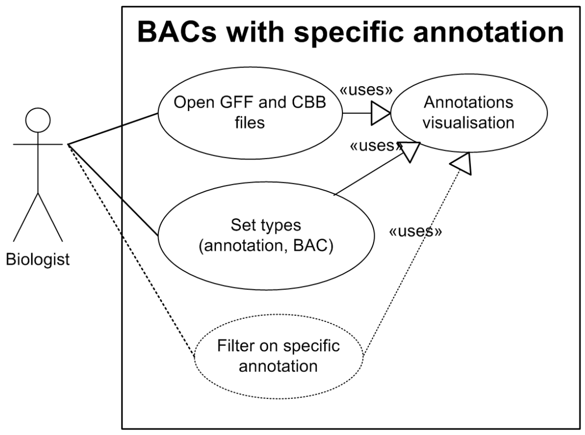

BACs with specific annotation
Biologist
Open GFF and CBB files
«uses»
Annotations visualisation
«uses»
Set types (annotation, BAC)
«uses»
Filter on specific annotation

Figure C.9: Use-case V

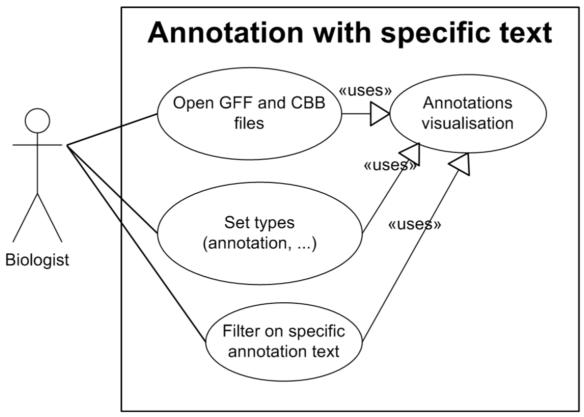

Annotation with specific text
Biologist
Open GFF and CBB files
«uses»
Annotations visualisation
«uses»
Set types (annotation, ...)
«uses»
Filter on specific annotation text

Figure C.10: Use-case VI

Appendix D

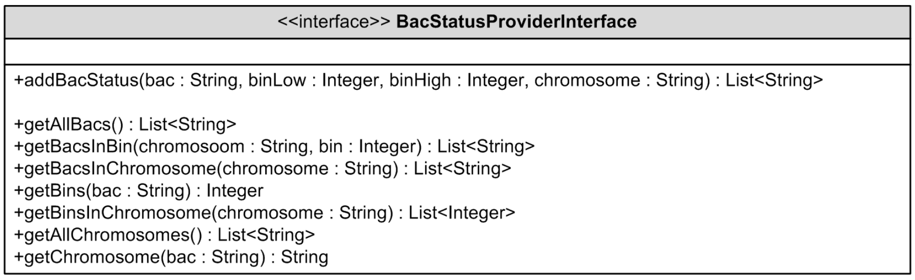
Additional CBB visualization information

D.1 The CBB implementation

This section describes the implementation details of the CBB visualization. The first part provides an overview of the implemented features and the various components. The chapter is concluded with a number of screenshots showing the final implementation within the DNAVis2 toolset.

D.1.1 The data structure

The CBB visualization uses a tree like (implemented using lists and sets) data structure to represents the high level hierarchical structure of the PGSC dataset. The data structure is equipped with a simplified interface described by the `BacStatusProviderInterface` (see figure D.1), to provide easy access to the data.



```
<<interface>> BacStatusProviderInterface

+addBacStatus(bac : String, binLow : Integer, binHigh : Integer, chromosome : String) : List<String>

+getAllBacs() : List<String>
+getBacsInBin(chromosoom : String, bin : Integer) : List<String>
+getBacsInChromosome(chromosome : String) : List<String>
+getBins(bac : String) : Integer
+getBinsInChromosome(chromosome : String) : List<Integer>
+getAllChromosomes() : List<String>
+getChromosome(bac : String) : String
```

Figure D.1: UML class diagram of the BacStatusProviderInterface

The current implementation of the `BacStatusProviderInterface` is the `BacStatusProvider` class that uses a binary search algorithm [12] combined with sorted lists to speed up the data requests. This is by no means an optimal solution, but is fast enough for the current data visualizations.

D.1.2 The visualization components

The CBB visualization consists of a number of components. The most important components will be discussed in this section.

The BIN-domain mapper

The visualization of all the BIN-domains of individual BACs would result in far too many data elements to be visualized in this visualization (see section 4.1.3). But the `BacStatusProviderInterface` is only able to provide the BIN-domain of individual BACs. Therefore some pre-computing steps are needed before the data can be visualized.
The BIN-domain mapper (see figure D.2) uses the elements reduction steps described in section 4.1.3) to combine the BACs with equivalent BIN-domains, and stacks them in an efficient manner.

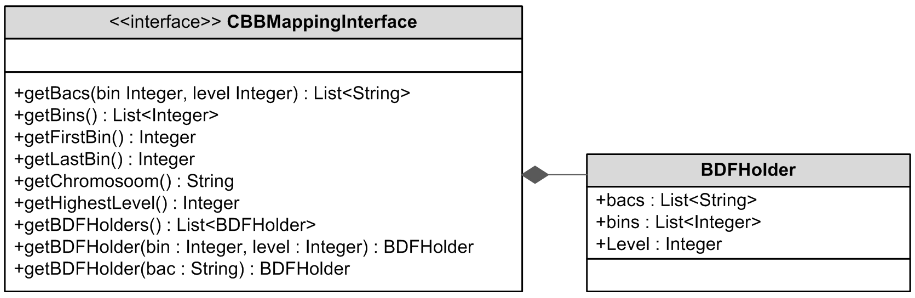

<<interface>> CBBMappingInterface
+getBacs(bin Integer, level Integer) : List<String>
+getBins() : List<Integer>
+getFirstBin() : Integer
+getLastBin() : Integer
+getChromosoom() : String
+getHighestLevel() : Integer
+getBDFHolders() : List<BDFHolder>
+getBDFHolder(bin : Integer, level : Integer) : BDFHolder
+getBDFHolder(bac : String) : BDFHolder
BDFHolder
+bacs : List<String>
+bins : List<Integer>
+Level : Integer

Figure D.2: UML class diagram of the CBBMappingInterface

The renderers

The CBB visualization uses multiple renderers, which are all based on the `ViewRenderer` class, provided by the existing DNAVis2 implementation. This ViewRenderer class provides the JOGL [21] interfaces, needed to use OpenGL [31, 2] within the DNAVis2 environment. All the renderers draw directly to the screen.

D.1.3 The CBB visualization

The CBB visualization has one main renderer that activates a chromosome renderer, per chromosome. Every chromosome renderer starts a BIN-domain mapper in a separate thread to pre-calculate the BIN-domain mapping for the chromosome. (This is a small multiprocessor optimization to reduce the initialization time of the visualization). When a chromosome is done with its pre-calculation it will update the screen, and draw its content to the screen.

Every chromosome is drawn in its own viewport, so scaling the chromosomes is simply done by changing the dimensions of the viewports. Selecting and deselecting elements will still require a real redraw of the chromosome in question.

Appendix E

Additional circle visualization information

E.1 Phi-ball algorithm

The phi-ball algorithm is bio-inspired and uses the golden section to place the nodes in a spiral motion around the center node. The golden section (defined as $\varphi = \frac{\sqrt{5}+1}{2}$) appears in nature to solve all kinds of problems, for example in the placement of tree branches around the trunk. Consecutive trunks are placed in an angle of approximately 137.5 degrees, this equals a complementary angle of $0.618034 \times (2 \times \pi)$. It turns out that $\frac{\sqrt{5}+1}{2} = 1.618034$ which, considered as a angle represents a whole turn plus the 0.618034 of a whole turn. The golden section is the real number which is the worst approximated by rational numbers. Meaning that is needs the highest number of repetitions (of all numbers), before it repeats (overlaps) itself. This makes perfect sense in the trees case, overlapping branches would be useless at catching sunlight (there main function) [26].

But this most irrational real number is not only useful in nature, its also practical for placement algorithms like the phi-ball algorithm. This outer spiraling algorithm uses the φ defined above as repeating angle (see figure E.1), the size of the element(s) determine the distance from the inner circle (not depicted in concept drawing).

E.2 The circle implementation

This section describes the implementation details of the circle visualization, by providing an overview of the implemented features and corresponding modules.

E.2.1 The data structure

The circle visualization uses a tree structure to represent the high level structural data of the dataset. This structure describes the mutual relation and distribution of the *genome*, *chromosome*, *BIN* and *BAC* elements, and provides information about the location and size of the FASTA and GFF files associated to the individual BACs.

The implemented tree structure is based on a extension of the `DefaultMutableTreeNode` class (see java documentation [28]), called the `CircleElement` class (see figure E.2). The internal `userObject` of `DefaultMutableTreeNode` is used to store the additional (`CircleDataInterface` type) information within a node. Nodes can contain the following additional information:

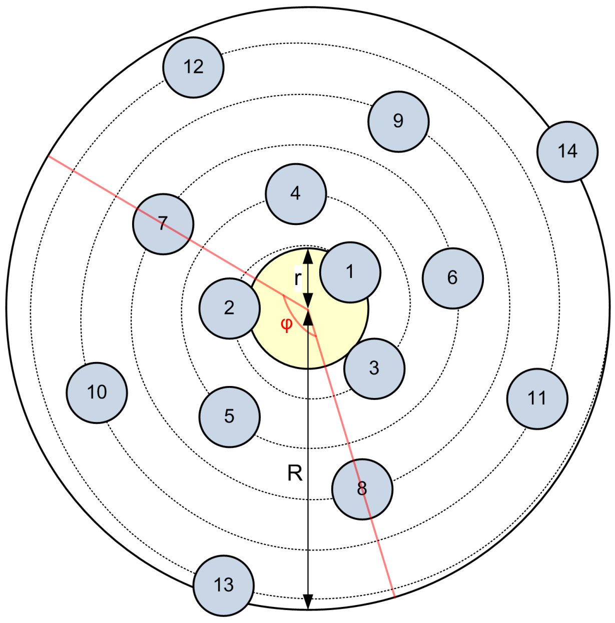


Figure E.1: Concept drawing of the phi-ball spiral concept, using a angle of φ (about 137,5 degrees)

- The name of the tree node.
- The state of the node (active or inactive), this determines whether the visualization will draw the expanded or collapsed visualization. The center node is always active, unless its a leaf node.
- The custom icons of the node.
- The location of the FASTA file, if available.
- The size (in bytes) of the FASTA file(s), if available.
- The location of the GFF file, if available.
- The size (in bytes) of the GFF file(s), if available.

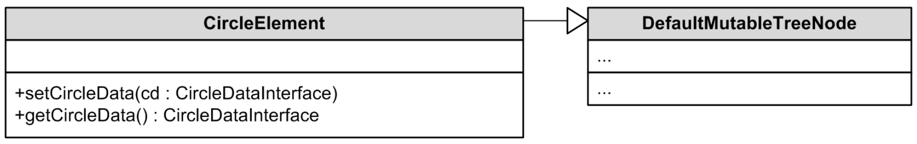

CircleElement
+setCircleData(cd : CircleDataInterface)
+getCircleData() : CircleDataInterface
DefaultMutableTreeNode
...
...

Figure E.2: UML class diagram of CircleElement

The tree structure is created by looping though various levels (chromosome, BIN and BAC) of the PGSC dataset (see section 2.2.2). This is a simple translation (see figure E.3) of the CBB data structure (see section D.1.1) into a circle visualization compatible tree structure. The transformation has a time complexity of $O(n\dot{m})$, where n equals the number of known BACs and m the size of their BIN-domains.

The `CircleElement` nodes for the BIN-domains are calculated on the fly whenever a BIN-domain is entered. The created BIN-domain `CircleElement` is named using the *chromosome* + *BIN-domain*, all the BACs within the BIN-domain are added as child nodes, and the already existing chromosome node is set as parent. This computation has a complexity of $O(n)$, where n is equal to the number of BACs in the BIN-domain.

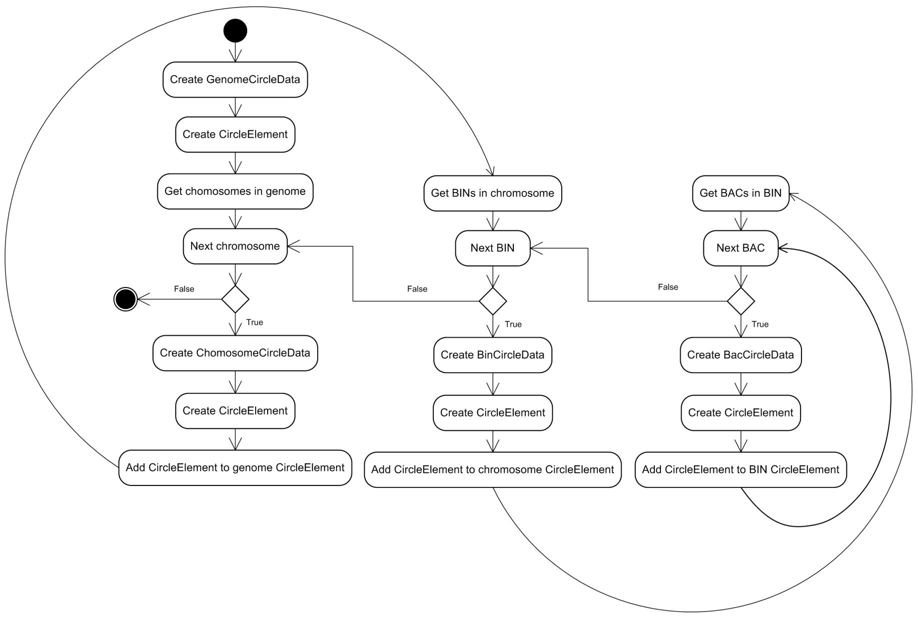


Figure E.3: UML state diagram of the tree creation process

E.2.2 The visualization components

The circle visualization consists of a large number of components. The most important components will be discussed in this section.

The shape, color and transformation routines

The `ShaperInterface` (see figure E.4) provides an interface to describe shapes, text, color and transformations. The current `ShaperJ2D` implementation uses the Java2D shape and transformation functions internally. All objects within the circle visualization are created using the basic shapes, text, color and transformations provided by the `ShaperInterface` type.

The sizing and positioning of the nodes

The `SlicerInterface` defines an interface to provide information about the size and the position of child nodes within the circle visualization. The default implementation can scale the child elements using the following default scale types:

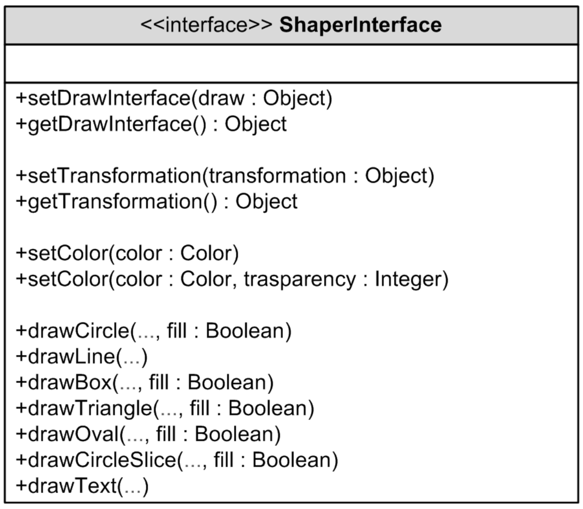
- `NORMAL` All elements have the same size.
- `CHILD_COUNT` Elements are scaled by their child count.
- `SQRT_CHILD_COUNT` Elements are scales by the square root of the child count.
- `LEAF_COUNT` Elements are scaled by their leaf count.
- `SQRT_LEAF_COUNT` Elements are scaled by the square root of the leaf count.

A number of additional DNA specific types where added to provide information about the GFF and FASTA files:

- `SQRT_GFF_BYTE_COUNT` Elements are scaled by the square root of the bytes of GFF data.
- `SQRT_FASTA_BYTE_COUNT` Elements are scaled by the square root of the bytes of FASTA data.

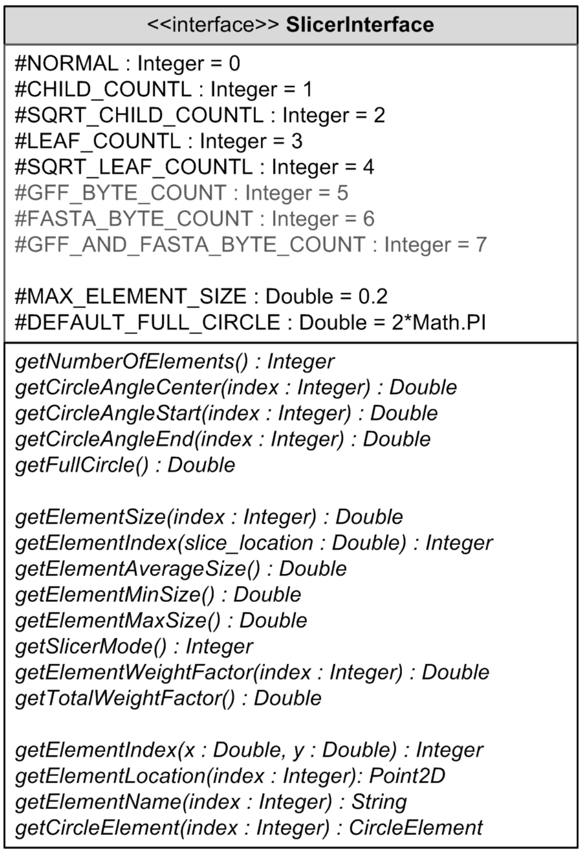
- `SQRT_GFF_AND_FASTA_BYTE_COUNT` Elements are scaled by the square root of the bytes of GFF and FASTA data.

The `SlicerInterface` provides also a number of frequently used calculations like: index to position, index to size, and position to index (see figure E.5).



<<interface>> ShaperInterface
+setDrawInterface(draw : Object)
+getDrawInterface() : Object
+setTransformation(transformation : Object)
+getTransformation() : Object
+setColor(color : Color)
+setColor(color : Color, trasparency : Integer)
+drawCircle(..., fill : Boolean)
+drawLine(...)
+drawBox(..., fill : Boolean)
+drawTriangle(..., fill : Boolean)
+drawOval(..., fill : Boolean)
+drawCircleSlice(..., fill : Boolean)
+drawText(...)

Figure E.4: UML class diagram of ShaperInterface



<<interface>> SlicerInterface
#NORMAL : Integer = 0
#CHILD_COUNTL : Integer = 1
#SQRT_CHILD_COUNTL : Integer = 2
#LEAF_COUNTL : Integer = 3
#SQRT_LEAF_COUNTL : Integer = 4
#GFF_BYTE_COUNT : Integer = 5
#FASTA_BYTE_COUNT : Integer = 6
#GFF_AND_FASTA_BYTE_COUNT : Integer = 7
#MAX_ELEMENT_SIZE : Double = 0.2
#DEFAULT_FULL_CIRCLE : Double = 2*Math.PI
getNumberOfElements() : Integer
getCircleAngleCenter(index : Integer) : Double
getCircleAngleStart(index : Integer) : Double
getCircleAngleEnd(index : Integer) : Double
getFullCircle() : Double
getElementSize(index : Integer) : Double
getElementIndex(slice_location : Double) : Integer
getElementAverageSize() : Double
getElementMinSize() : Double
getElementMaxSize() : Double
getSlicerMode() : Integer
getElementWeightFactor(index : Integer) : Double
getTotalWeightFactor() : Double
getElementIndex(x : Double, y : Double) : Integer
getElementLocation(index : Integer): Point2D
getElementName(index : Integer) : String
getCircleElement(index : Integer) : CircleElement

Figure E.5: UML class diagram of SlicerInterface

The renderers

The circle visualization uses multiple renderer's that are all based on the `BasicRenderer` class (see figure E.6). This class provides the internal storage of a large number of colored shapes (filled or transparent), thereby allowing complex drawings to be created and reused. Its functionality can roughly be compared with the *glList* function of *OpenGL* [31, 2]. A number of additional functions provide a standardized interface, used by renderers within the circle visualization.

The drawing routines

The drawing routine within the circle visualization is used to output the visual (shape, text and color) information. The drawing routine is based on the texttttDrawJ2DInterface and is not limited to real drawing actions, and can also be used to write to a storage structure like a `Collection<DataElementJ2DInterface>` implementation. The real drawing implementation of texttttDrawJ2DInterface uses the *Java2D* library.

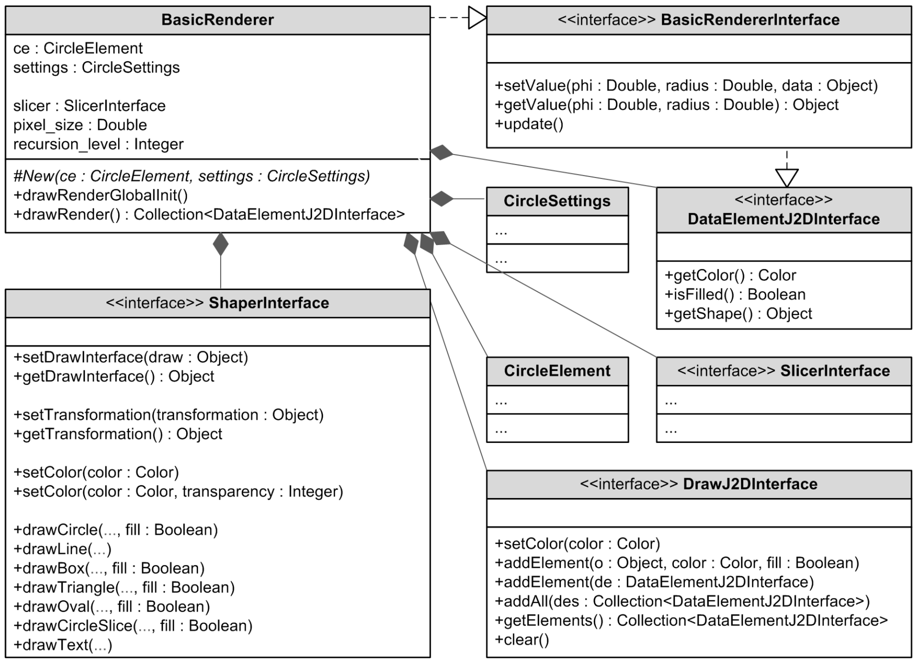

BasicRenderer
ce : CircleElement
settings : CircleSettings
slicer : SlicerInterface
pixel_size : Double
recursion_level : Integer
#New(ce : CircleElement, settings : CircleSettings)
+drawRenderGlobalInit()
+drawRender() : Collection<DataElementJ2DInterface>
<<interface>> BasicRendererInterface
+setValue(phi : Double, radius : Double, data : Object)
+getValue(phi : Double, radius : Double) : Object
+update()
CircleSettings
...
...
<<interface>> DataElementJ2DInterface
+getColor() : Color
+isFilled() : Boolean
+getShape() : Object
<<interface>> ShaperInterface
+setDrawInterface(draw : Object)
+getDrawInterface() : Object
+setTransformation(transformation : Object)
+getTransformation() : Object
+setColor(color : Color)
+setColor(color : Color, transparency : Integer)
+drawCircle(..., fill : Boolean)
+drawLine(...)
+drawBox(..., fill : Boolean)
+drawTriangle(..., fill : Boolean)
+drawOval(..., fill : Boolean)
+drawCircleSlice(..., fill : Boolean)
+drawText(...)
CircleElement
...
...
<<interface>> SlicerInterface
...
...
<<interface>> DrawJ2DInterface
+setColor(color : Color)
+addElement(o : Object, color : Color, fill : Boolean)
+addElement(de : DataElementJ2DInterface)
+addAll(des : Collection<DataElementJ2DInterface>)
+getElements() : Collection<DataElementJ2DInterface>
+clear()

Figure E.6: UML class diagram of BasicRender

E.2.3 The circle visualization

The circle visualization has a main renderer (`MainRender` class) that combines multiple renderers into a single visualization. This main renderer processes the information within the tree structure by walking recursively through the tree structure (calling himself (indirectly) with a child node as tree root).

The main renderer decides whether the current root node will be described using: a single icon renderer, the circle renderer, or the phi-ball renderer. The main renderer is also responsible for additional visualizations like the inner and outer renderer. All its decisions are based on the customization settings (see section 5.4.2), and the information in the root node (active/inactive and number of direct children).

Both the circle and phi-ball visualizations rely on the main renderer to draw their child nodes, using an indirect recursive call with the child node as new root node. The returned result is scaled and transformed to the size and location of the child node (using the `SlicerInterface`).

Appendix F

Additional property matrix visualization information

F.1 The implementation

This section describes the implementation details of the property matrix visualization. The first part provides an overview of the implemented features and the various components. The chapter is concluded with a number of screenshots showing the final implementation within the DNAVis2 toolset.

F.1.1 The data structure

The property matrix visualization combines the high level data structure used by the CBB and the circle visualization, with the low level annotation data within the GFF files. The internal structure of this data structure (*AttributeProvider* class) makes heavy use of hash tables [12]. The hash-keys are represented by the chromosomes, BINs, BACs, annotation types and annotations within the dataset, and provide a lists of BACs or Annotations depending on the used hash-table. There are eight hash tables in total, as shown in table F.1.

Hash table	**Hash-key**	**Resulting information**
chromosomes2BacsList	Chromosome	All BACs in the chromosome
bins2BacsList	BIN	All BACs in the BIN
types2BacsList	Annotation types	All BACs containing annotation type
keywords2BacsList	Keyword	All BACs containing keyword
chromosome2KeywordsList	Chromosome	All keywords in the chromosome
bins2KeywordsList	BIN	All keywords in the BIN
bacs2KeywordsList	BAC	All keywords in the BAC
types2KeywordsList	Annotation types	All keywords of annotation type

Table F.1: Hash table, has-key, and the resulting information

F.1.2 The visualization components

The property matrix visualization consists of a number of components. The most important components will be discussed in this section.

The cross-reference calculations

The cross-reference calculations depicted in table 6.1 makes heavy use of Java's `Collections` framework [27] to compare the listings that are returned by the data structure described above. Most of these cross-reference calculations are based on calculating the intersection of two sets (read lists).

The current implementation has an option to store the result of a cross-referencing to speed up the redrawing, or to do the cross-referencing on the fly while redrawing. The first option is obviously faster but uses far more system memory, which can become a problem when visualizing big datasets.

The filters

The raw text and annotation type filters (see section 6.4) are currently integrated in the cross-reference calculations. Not the cleanest solution but the easiest and fastest, because annotation type can already be filtered when requesting the lists from the data structure (*AttributeProvider* class). The raw text filter is applied to the cross-reference data, just before it is returned to the visualization. (Changing a raw text filter does not require a recalculation of the cross-referencing data when the data storing option is enabled.)

The renderers

The property matrix visualization uses multiple renderers. All these renderers are based on the `ViewRenderer` class, provided by the existing DNAVis2 implementation. This ViewRenderer class provides the JOGL interfaces [21], needed to use OpenGL [31, 2] within the DNAVis2 environment. All the renderers draw directly to the screen.

F.1.3 The propery matrix visualization

The property matrix visualization uses the same multiple viewports technique as the CBB visualization to combine multiple renderers into a single visualization. The dot-plot renderer is the most prominent renderer, it displays the cross-reference elements. The row and column elements are placed at adjacent sides of the dot-plot, using the top bar and left bar renderers.

A small viewport in the top-left corner is used to provide an overview of the zoom level and scroll location, using the mini-matrix renderer as described in section 6.1.3.

The final renderer is used in a viewport that is placed on top of the other viewports, this renderer highlights the cursor position as described in section 6.1.4, and can visualize a selection box.