ECOLE POLYTECHNIQUE FEDERALE DE LAUSANNE
SCHOOL OF LIFE SCIENCES

Master's project in Life Sciences and Technology

Development Of An Interactive Genome Browser To Visualize And Analyse Large Scale Genomic Data

Done by

Lucas Sinclair

Under the direction of
Prof. Felix Naef

and the supervision of
Jacques Rougemont, Ph.D

In the Bioinformatics and Biostatistics Core Facility, EPFL

External expert
Marcel Geertz, Ph.D

LAUSANNE, EPFL 2010

Acknowledgments

Many thanks all the members of the core facility, and in particular to Marion Leleu, Ph.D who guided the development and had the original idea of gFeatMiner.

Contents

List of Figures

1 Abstract

Genomic bioinformatics is a growing and developing field. Indeed, data analysis is becoming an integrative and essential part of any quantitative biological experiment as the technologies evolve and the wet lab methods used generate larger and larger quantities of data. Yet few standards have emerged and a plethora of analytical tools exist, none of which are established as a standard. The difficulties arise early on, even before processing any genomic data, as one first needs to visualize it. Several visualization methods exist, such as the *UCSC genome browser* [17], *IGB* [29] or *Argo* [16], but none offer a satisfying interface or set of tools.

Stemming from a pre-existing project at the bioinformatics and biostatistics core facility, this study presents a new solution to the multiple difficulties that at present beleaguer the field. A novel genome visualization tool is proposed where the user interface remains simple and incorporates a set of common statistical analysis functions. The software produced, entitled *gFeatMiner*, is capable of processing large scale genomic datasets for computing descriptive statistics and manipulate them in several ways. The program makes use of modern technologies and infrastructure paving the way for its development into an advanced data mining tool.

In the second part of this study, a practical application is worked out. Examining the genes coding for ribosomal proteins in the model organism yeast (*Saccharomyces cerevisiae*) and using several available sets of data including multiple transcription factor binding profiles *in vivo* and *in vitro*, RNA polymerase activity and nucleosome enrichment, we attempt to better understand and reveal cellular mechanisms by clustering the numerous genes together using different criteria and machine learning strategies.

2 Introduction

2.1 Context

This document is the written report following a four month internship in the Bioinformatics and Biostatistics Core Facility of the *Ecole Polytechnique Fédérale de Lausanne*. The study ran from the 22nd February to the 25th June (2010) in the context of the master project all students of the Life Science and Technology curriculum must accomplish in their 10th semester.

2.2 Motivation

Every lab or biologist requiring the services of the core facility arrives with different data and different scientific questions. Responding to them most often involves the production of custom-tailored homemade scripts for every new task or problem. In an attempt to eliminate such repetitive work, one of the present goals at the BBCF is to empower the biologist with a tool that would enable him to analyze his own data, at least in superficial way to begin with. Developing and offering a new-generation genomic data manipulation tool accommodated for use by scientists with a low computer science background would be extremely valuable, both to the individual scientists, and to the field as a whole.

To avoid asking the user to install any local software, and for maximum simplicity, the web browser was chosen as the primary channel for the tool to be created. Instead of developing a viewer system from the ground up, a promising open source project developed at Berkeley University entitled *JBrowse* [34] was adopted and modified by adding some essential functionalities such as drag-and-drop selection creation and management. Next, a data management framework into which the users can login and upload their data was wrapped around the genome browser. Finally, in addition to this interactive genomic data viewer (referred to as *GDV* and presented

in section 4 of this report), the aim was to provide a set of tools integrated inside the interface of the browser that would allow the user to quickly compute simple descriptive statistics on the data being viewed as well as operating common genomic feature manipulations. This brings us to the part this project focuses on.

2.3 Problem description

The goal of this project is to design a set of algorithms to communicate with the pre-existing genome browser containing various sets of genomic data (also known as tracks) so as to be able to compute descriptive statistics as well as manipulate the data. Specification include that the product must:

a) be able to receive input in several of the common genomic data formats, b) return the information generated back to *GDV* in a format that can be displayed in the user interface, c) maintain a rapid execution time when dealing with large datasets

This assortment of scripts – regrouped under the name of gFeatMiner – should produce graphs describing number of features, cumulative base coverage, feature length distribution or feature score distribution on a track or a subset of a track. In addition, *gFeatMiner*'s functionality should allow the creation of new tracks by operating on pre-existing tracks. These common manipulations include computing, for example, the complement of a track, the overlap or union between two tracks, the subtraction of one track from another as well as the merging of two or more tracks together.

As detailed in section 5 of this report, the product should offer a coherent framework for dealing with genomic data, making use of efficient file I/O and data processing algorithms, so that in a near future, it can develop into a full-fledged data mining tool.

2.4 Application

Today, large quantities of data are produced by different laboratories studying quantitative biology around the world. The number of open databases and freely accessible publications is quite large, but data is often collected for a single purpose and subsequently stored when the scientific question at hand is resolved. There is reason to believe there is a great potential in novel combinative analysis of different datasets. Indeed, much information regarding, for example, metabolic pathways, gene regulation or cellular life cycle might be extracted by aggregating data from the pre-existing experiments and applying intelligent statistical analysis and complex data mining.

To this end, as a second part of this study, a practical application involving the model organism *Saccharomyces cerevisiae* is carried out. In section 6, correlations between several datasets concerning the genes coding for ribosomal proteins of the yeast are explored using clustering strategies and machine learning algorithms.

3 Background

Even though the field of genomic bioinformatics can be viewed as a novel and emerging, a plethora of tools, databases, and file formats have already been created and are in common use. This background section introduces and defines some of these developments and other concepts.

3.1 Visualization

Visualizing genomic data is not a new need, and several tools offering varying degrees of functionality and user-friendliness can be found online. Examining other similar products before starting

a new project is always a good idea and an excellent opportunity to learn from the mistakes of others, identify the missing functionalities and forge an opinion on what the state of the art currently is. Below, we present a few of the major tools freely accessible on the web and compare their usability.

3.1.1 UCSC genome browser

The UCSC genome [17] browser is one of the most popular genome browser around. Developed by the University of Santa Cruz, it has been in service since the year 2000 and includes genomic sequences for 46 species. A screenshot can be seen in figure 1.

It provides some very useful tools for aggregating data from various sources and carrying out analysis on the data that is uploaded. Indeed, a diverse collection of annotation datasets can be found, including gene predictions, gene-expression, mRNA alignments, etc. It is also capable of more advanced processing. For example, using comparative alignments, it can produce graphs representing the evolutionary relationships among species. It offers a few genome analysis tools such as BLAT [25], liftOver and Gene Sorter, etc. as well.

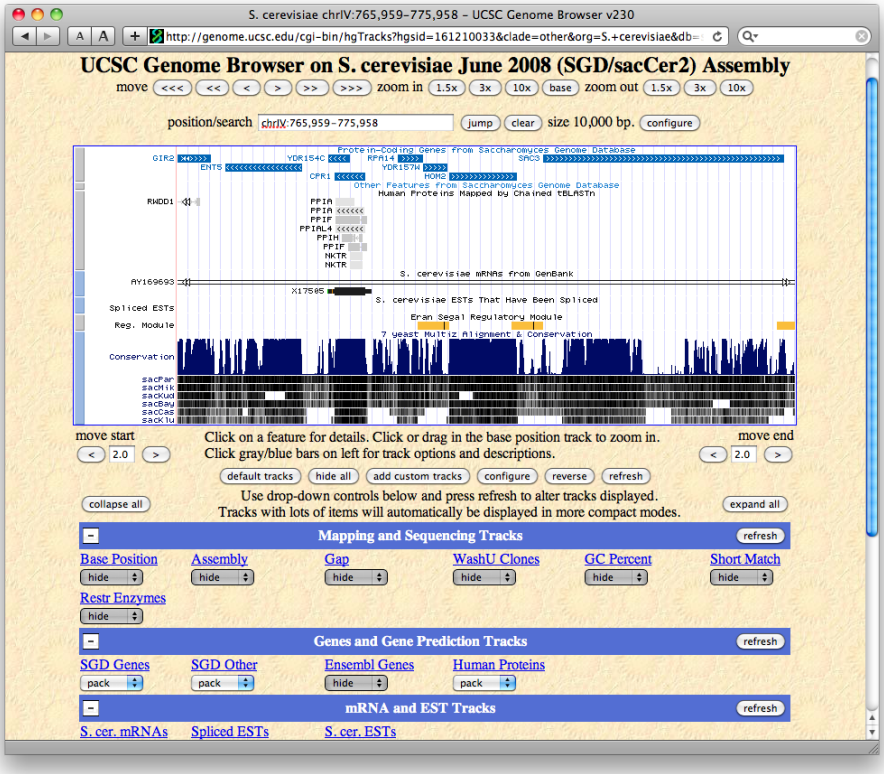

Figure 1: UCSC's main interface, browsing the genome of *S. cerviase*, as rendered by WebKit [8].

However, it is far from perfect. The wikipedia entry on the UCSC genome browsers states "The browser is a graphical viewer optimized to support fast interactive performance" [6]. Unfortunately this is not so. The interface is far from responsive, as each change of scale or panning requires a page reload. For the user accustomed to rich internet applications, navigation is most

awkward and frustration is immediate. For a simple overview of some newly generated data and some simple statistical information, UCSC is definitely not optimal. Sadly, many biologists and bioinformaticians have learned to deal with UCSC and now visualize their data exclusively using this browser.

3.1.2 Argo genome browser

The Argo genome browser [16] is another visualization tool in free access since 2007, and developed by the Broad Institute. It can read a certain number of formats, as long as they are formatted exactly to the specifications of the software. It has some special advanced features like comparative perspective for viewing dot plots of multiple aligned sequences.

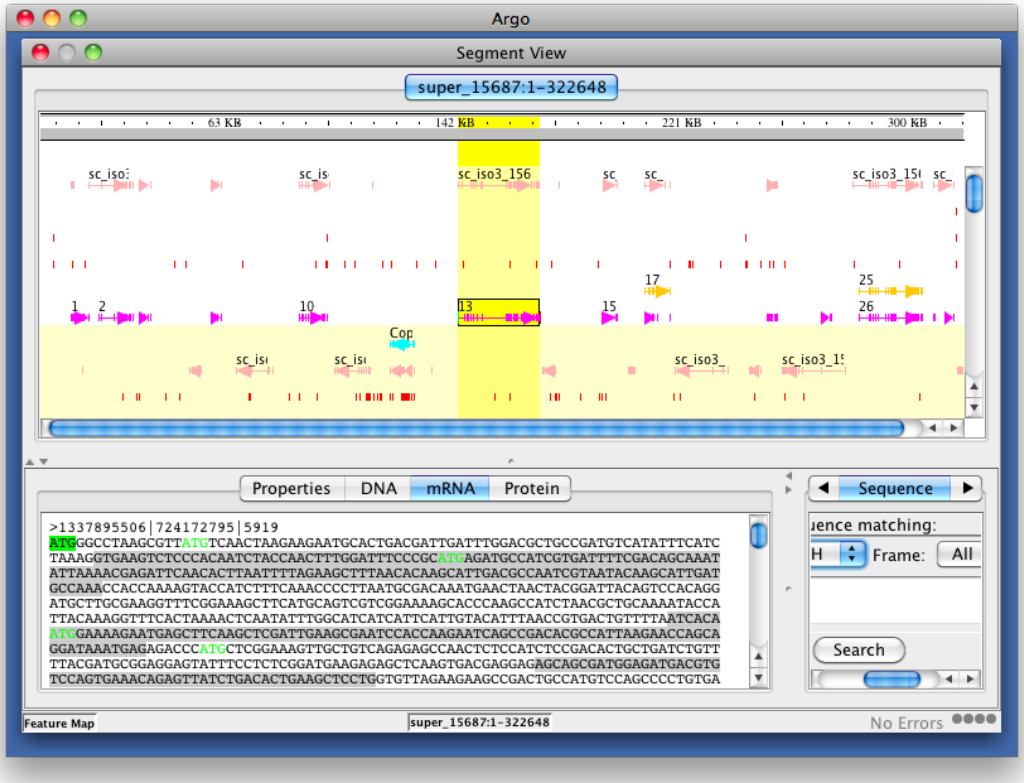

Figure 2: AGB's main interface displayed as a window inside a window, browsing the genome of *Ciona savignyi*.

As it uses the java technology and does not run inside a web browser, AGB can provide better interactivity and does zoom and pan dynamically. However, it does not respect many of the standard interface paradigms. The user must learn the program thoroughly, and become familiar with the frequent errors it produces and how to fix them.

The Argo website states: "Though other genome browsers with similar feature sets exist, we believe Argo provides a more flexible and intuitive user interface" [15]. A noble goal – but once again, in our opinion, not attained. A telling example is the fact that one of the most basic

user interface guideline [1] is violated when AGB makes use of a window inside a window. A screen-shot of this effect can be seen in figure 2.

3.1.3 Integrated genome browser

The integrated genome browser [29] is our third example of genomic visualization software. Its development was funded largely by Affymetrix Inc. It was released to the community in 2005. The technology driving this tool is similar to that of AGB, as the program must be downloaded and subsequently executed inside a JVM [14].

The number of formats it reads is somewhat smaller than AGB, but it provides automatic access to online resources using the DAS [27] or the QuickLoad [20] protocol, which is a very welcome feature. Additionally, it is also good at handling data from tiling array results, as well as viewing alignments data.

Unfortunately, once again, the biologist or bioinformatician will be repelled by the overwhelming number of interface elements like sliders, check-boxes, drop-down menus and tabs, all used in the incorrect contexts. A screen-shot of the main interface can be seen in figure 3. Profuse documentation is provided in a 100 page-long manual, but it is not an optional read. Without studying this software thoroughly beforehand, it is definitely hard to use.

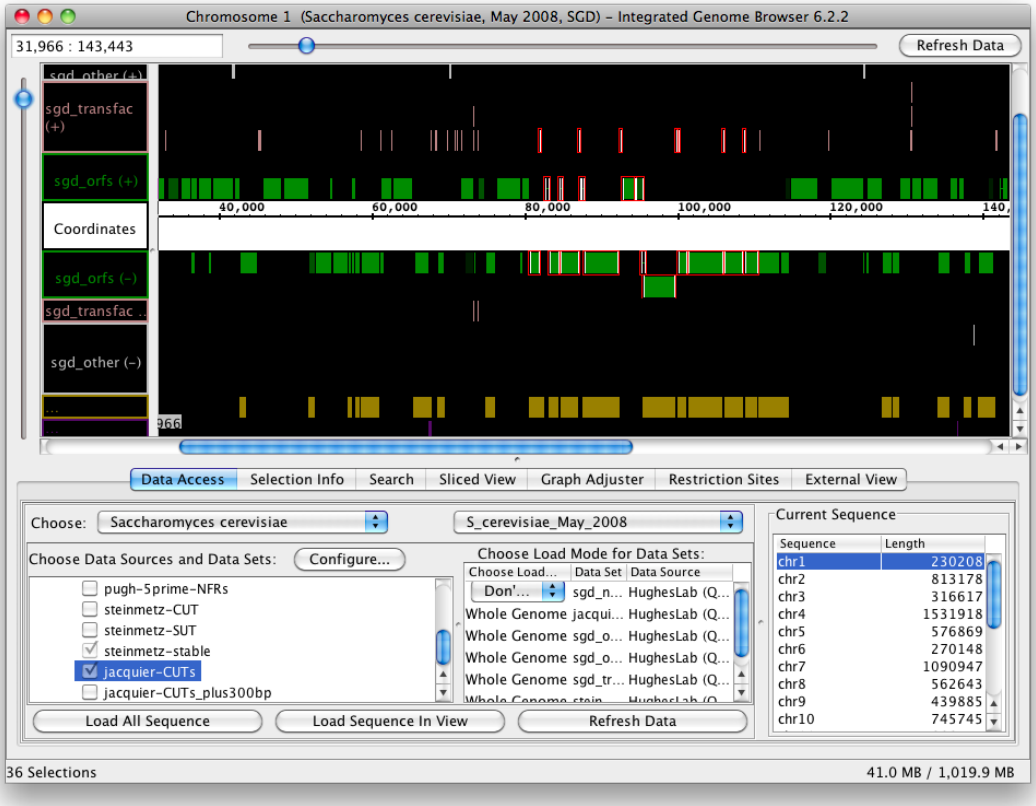

Figure 3: IGB's main interface, browsing the genome of *S. cerviase*.

3.2 Data processing/mining

3.2.1 Galaxy

Galaxy [13] [21] distinguishes itself from the tools previously presented, as it is not, as such, a genomic data browser. Nevertheless, it is very versatile in the functionality it provides and is better described as a genomic data manipulation tool. Freely accessible for use on the web, Galaxy is a project that began in 2006 at the Pennsylvania State University. A large part of the genomic data processing in Galaxy is implemented using a library entitled `bx-python` [35] which is part of the larger ESPERR [36] project.

It can accept input from a large number of sources and formats. You can load genomic features of all kinds including sequence data. If the input format is not recognized, Galaxy provides some easy ways to convert it on the spot. Once the data is loaded, it can be manipulated or analyzed in a wide variety of ways. For example, one can sort, filter, aggregate, compute regression or component analysis. Several datasets can be treated together or against each other. Common operations include: concatenating, merging, clustering, intersecting or subtracting. A screenshot of the interface as the user is computing the concatenation between two tracks is given in figure 4.

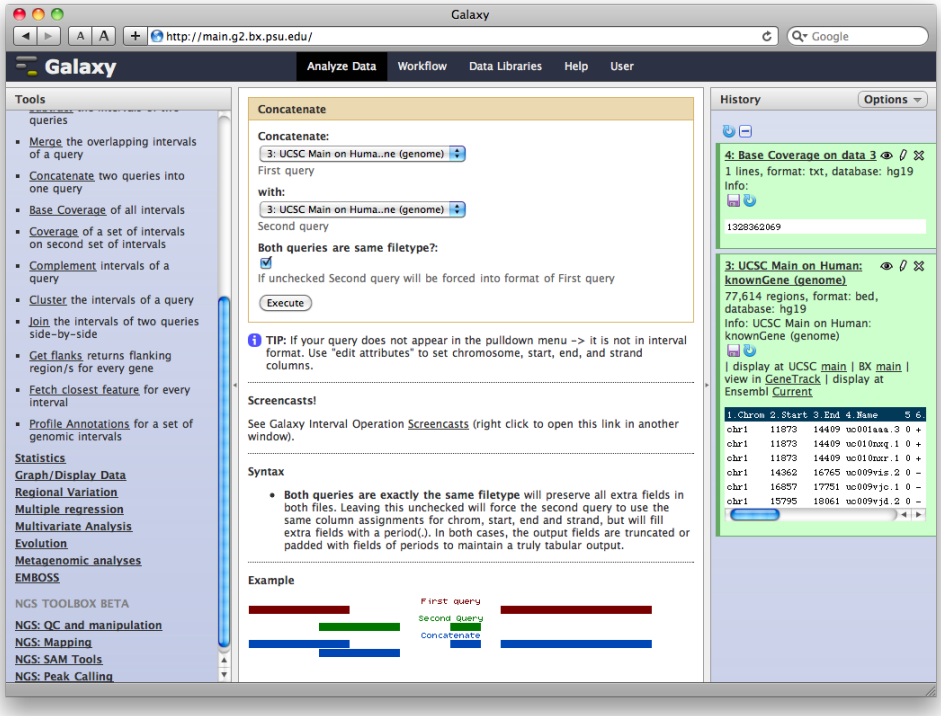

Figure 4: Galaxy's main interface, computing the concatenation between two tracks, as rendered by WebKit [8].

An interesting aspect of the Galaxy project is the effort made to increase reproducibility of the analysis carried out. This is implemented by providing the user with a complete and exhaustive history of every operation applied on the data, allowing the user to backtrack to any

stage of his manipulations, as well as branch an analysis at a given point. Moreover, the analysis history information can be exported for sharing with other users and can be reapplied to different inputs.

3.3 Formats

One of the problems of the daily life of a bioinformatician is the perpetual lack of standards in the representation of information. The chaos created by the hundreds of tab-delimited text files hinders the compatibility between tools and the exchange of data or results. Moreover, storing numerical values in their string equivalent is inherently a poor idea. As a brief overview of the problem, a few of the most common formats are described here. A complete guide to genomic formats can be found at http://genome.ucsc.edu/FAQ/FAQformat.html

3.3.1 BED

The `.bed` tab-delimited format is commonly known as a classical annotation track. It is used to describe genomic features of all kind (e.g. gene, introns, exons, etc.) and only specifies the location (chromosome, start, end and strand) of the feature inside a genome without the underlying nucleic sequence. Only the three first fields are required, but a few more optional columns can be added to describe, for example, a name or an associated score with each feature. An example of the first few lines of a typical BED file is shown below. It should be noted that the header lines of all excerpts starting with "`track`" are not part of the file specifications.

```
1 track type=bed name="S. cer. RP genes" source="SGD"
2 chr2    45975   46367   YBL092W 0.0     +
3 chr2    59818   60735   YBL087C 0.0     -
4 chr2    88521   89123   YBL072C 0.0     -
5 chr2    168426  169379  YBL027W 0.0     +
```

Listing 1: An excerpt from a typical BED file

3.3.2 GFF

The `.gff` tab-delimited format stands for general feature format and is used to represent very much the same type of objects as the BED format. The main difference of this overlapping format when compared to BED is that all nine fields are required to contain a value in order to be accepted as a valid GFF file. For example, even when feature does not have an associated score, all fields must be filled with a dot. This makes GFF one of the most rigorously defined formats but at the same time limits its adoption because of the size of the files produced. Another unique aspect of the GFF file format is the possibility of including cross-linking of features inside a file. For example, one can define a set of exons as being assigned to a parent gene.

```
1 track type=gff name="S. cer. RP genes" source="SGD"
2 chr2    SGD     gene    45975   46367   .  +  .   Name=YBL092W
3 chr2    SGD     gene    59818   60735   .  -  .   Name=YBL087C
4 chr2    SGD     gene    88521   89123   .  -  .   Name=YBL072C
5 chr2    SGD     gene    168426  169379  .  +  .   Name=YBL027W
```

Listing 2: An excerpt from a typical GFF file

3.3.3 WIG

The `.wig` tab-delimited format nicknamed wiggle format is quite different from BED as it is used to represent continuous-valued data. The wiggle format permits the display of quantitative tracks where each base pair of a given genome is associated with a floating-point number.

```
1 track type=wiggle_0 name="Nucleosomes enrichment" source="Hughes"
2 fixedStep  chrom=chr2  start=45000 span=4
3 13       61.4781229508
4 17       60.1543606557
5 21       61.5923786885
6 25       62.0665704918
```

Listing 3: An excerpt from a typical WIG file

It should be noted that the GFF format can also be used to represent quantitative data, but too often the file format is directly related to type of view produced. Taking for example the UCSC genome browser, the input of GFF file will necessarily create a qualitative view, regardless of the type information contained in the file.

3.3.4 Changing definitions

Adding to the problem of formats disparity is the fact that inside a set of files sharing the same extension, the signification of each field is ill-defined and may vary. To illustrate a case in point, let's consider the following simple one line BED file.

```
1 chr2    4        8
```

Listing 4: A one line BED file representing one unnamed feature

One wonders: what is the length of the feature described here? A naive answer from an experimentalist might be five base pairs as shown in figure 5.

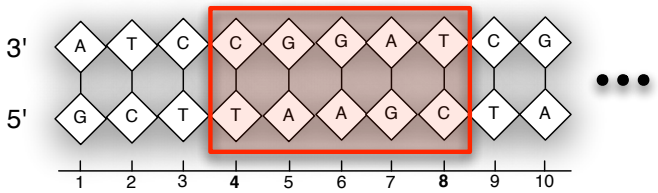

Figure 5: A naive view of nucleotide numbering on a piece of double-stranded DNA.

However, according to UCSC, the correct answer is four base pairs since the last nucleotide is never contained in the interval. This is equivalent to imagining the DNA sequence numbered on its phosphate residues instead of its sugar residues. Furthermore, a numbering convention of this kind creates ambiguity: does the count start at zero or one? Figure 6 illustrates the convention used by UCSC for interpreting genomic features and is the one we will adopt to in this report.

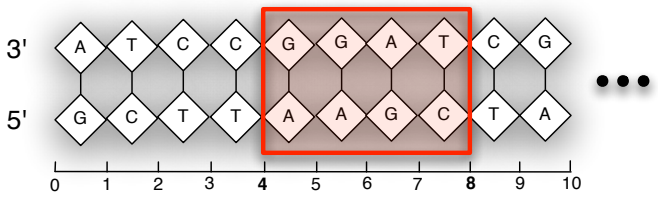

Figure 6: UCSC's standard for of numbering nucleotides on a piece of double-stranded DNA.

3.4 Techniques

The development of genomic bioinformatics is driven in a large part by advances in quantitative biology techniques. Most of the data analyzed by a bioinformatician is generated by these new high-throughput or massively parallel technologies. Therefore, understanding which processes were used to acquire the experimental data is crucial for extracting meaningful results. A few important experiments and assay types are described here.

3.4.1 ChIP-PCR, ChIP-Chip and ChIP-Seq

ChIP standing for "Chromatin immunoprecipitation" is a wet lab technique to extract and purify DNA fragments that are bound by a protein of interest *in vivo*. The protocol for performing chromatin immunoprecipitation on a sample is the following:

- The chromatin of cells of interest is cross-linked (using for formaldehyde) and fragmented (via sonication or digestion). This results in a solution containing millions of small DNA sequences (between 200 and 400 base-pairs long) all of which have conserved any protein links they previously had.
- Small antibody-bead complexes are added to the solution. The antibodies are designed to bind only a specific protein of interest, for example a transcription factor.
- The solution is immuno-precipitated, hopefully pulling out only DNA fragments that were bound by an antibody (and thus bound by the protein). Of course, at this stage, many other DNA fragments are trapped and nonspecifically precipitated along with the actual binding targets.
- The cross-linking is reversed by applying heat and DNA-protein links are broken.
- DNA is purified from the solution.

Once the DNA-fragment solution is obtained, it can be processed in various ways. The oldest type of analysis consists in amplifying the fragments using PCR and subsequently running a gel electrophoresis on the product. This type of experiment can only investigate a few genomic regions at a time depending on the sets of primers chosen.

In the more recent ChIP-on-chip technique (also known as ChIP-chip), instead of running a gel electrophoresis on the solution, the fragments are labeled with a fluorescent synthetic dye and inserted into a DNA microarray. The microarray, or chip, is specifically set-up with probes that cover the genomic region of interest. Once the DNA fragments have hybridized to their complementary oligonucleotides, fluorescence is measured and converted using various statistical

methods to relative intensity per probe. In this fashion, protein-DNA interactions on much larger portion of genome can be analyzed.

Recently, a novel way of processing the DNA fragments has been developed. In the ChIP-Seq [11] experiment (standing for "Chromatin immunoprecipitation followed by sequencing"), the fragments are processed using the next-generation massively parallel sequencing technologies from Illumina (formerly Solexa) and Life Technologies (formerly Applied Biosystems). These machines are able to perform shorts reads typically on the range of 30 to 50 base pairs on each end of every DNA fragment purified. The millions of short reads are then aligned to both strands of the genome corresponding to the species studied. One can thus attribute to each nucleotide of each strand of the reference genome a "tag count" expressed as an integer value. Inputting the tag counts into one of the many "peak finder" algorithms helps to identify the enriched regions and suppresses artifacts due to noise, background signal and repeated regions. The output consists of a list of regions where the protein of interest was potentially bound to the DNA.

The recent development of ChIP-Seq has provided consequent advances in the identification of DNA-protein interactions mechanisms and is a key technology to better grasp the structure of complex gene regulation networks.

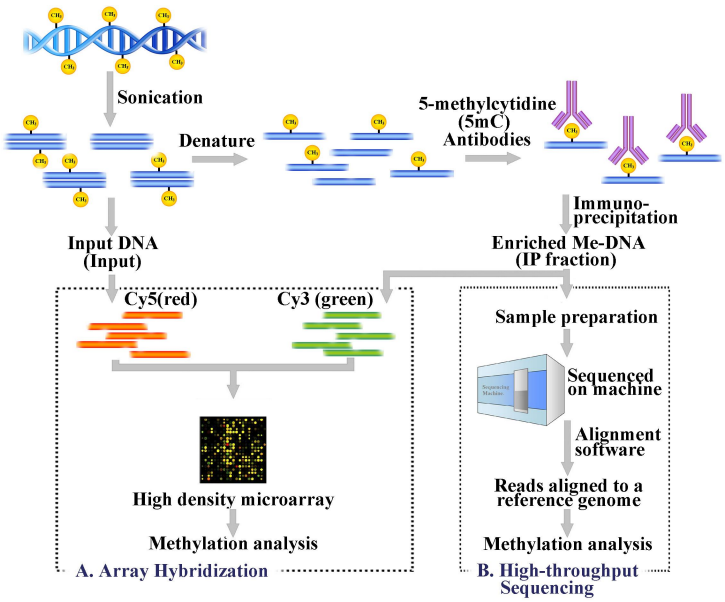

Figure 7: Schematic outline of the ChIP process followed by either array hybridization or high-throughput sequencing. Image courtesy of Wikimedia Commons.

3.4.2 RNA-Seq

RNA-Seq, also called "Whole Transcriptome Shotgun Sequencing", makes use of next generation sequencing technology just like ChIP-Seq. The difference is that this technique is used for the study of the transcriptome, not the genome. The RNA of the lysed cell is inserted into the sequencer, following some purification such as the removal of ribosomal RNA that represents about 90% of the RNA inside a given cell. The researcher can then quantify gene expression and identify post-transcriptional mutations as well as discover new transcripts.

3.4.3 Mitomi

Standing for "Mechanically Induced Trapping Of Molecular Interaction", *mitomi* [28], is a fairly novel strategy developed by Prof. Stephen Quake's Group at Stanford University. It uses cutting-edge "lab-on-a-chip", microfluidic and photolithography technologies to detect low affinity binding events in parallel. On a microarray a few milliliters square, 2400 individual chambers are created, with each one capable of trapping a small quantity of the transcription factor of interest and measuring its affinity with specific DNA sequences.

After quantifying the binding energy of a given transcription factor with a multitude of only slightly varying DNA probes, one can build a full DNA binding energy landscape over a complete genome of any eukaryote. This last step consists in building a position weight matrix based on the measurements for use in the scoring function that will assign a value to every base pair in a given genome.

4 Global context

4.1 Description

The work described in this section was essentially developed by Yohan Jaroz, Fabrice David and Bernhard Sonderegger. It is important for understanding my project, described in the next section.

The goal of BBCF's project is to build a next-generation genomic data visualization and manipulation tool that the biologist could easily use without any particular training, to view, process and analyze his data in an interactive and intuitive interface. This section focuses on the visualization part and the core interface (Genome Data Viewer or GDV). The manipulation and analysis software being developed by myself (gFeatMiner) is presented later. To give a full view of the different components and how they interact, a schematic is provided in figure 8.

Many tools for viewing and analyzing data exist, most have strengths in particular domains but lack others. Galaxy, for example, is not bad at manipulating data, but offers no visualization. UCSC has a summary viewer but doesn't offer some simple analysis functionality. None of these tools have yet produced a simple and user-friendly interface that follows standards and could be easily picked up. Our aim is to create such a tool to browse chromosomes, one that would be easier to use than UCSC, Argo or IGB. A tool that would not necessarily have many complex features but that would nevertheless be suited for a wide range of common tasks in the area of genomic studies. Ideally, intuitive, natural, hassle free, and rapid exploration of the data should be made possible.

4.2 JBrowse

We decided that the web browser would be a requirement for the product since installing software to the computer is not only tedious for the user but can add complexity to the development as multiple platforms must then be supported. Our search for existing tools in the field of genomic visualization was thus narrowed, and we chose a promising technology developed at the University of Berkley entitled "JBrowse" [34] to be the underlying viewer.

JBrowse is placed under the LGPL [18] license and hence is fully open-source and free. It offers an excellent user experience by exploiting the interactiveness of the web browser relying on technology such as JavaScript and AJAX to create a rich internet application. It produces an extremely smooth and fast scrolling of the genomic information. It accomplishes this in part by preprocessing the genomic data to display and creating hundred of small images for all

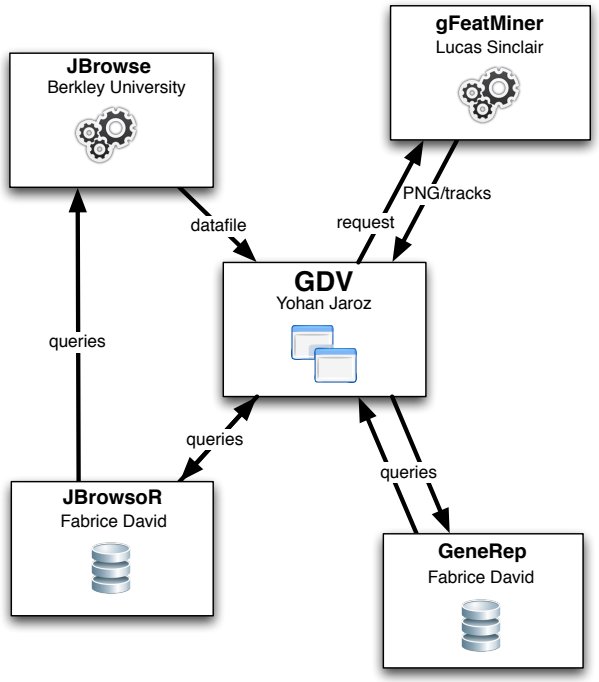

Figure 8: A schematic representing the different software pieces involved in BBCF's project interacting around GDV. Their relation types as well as their curators are specified.

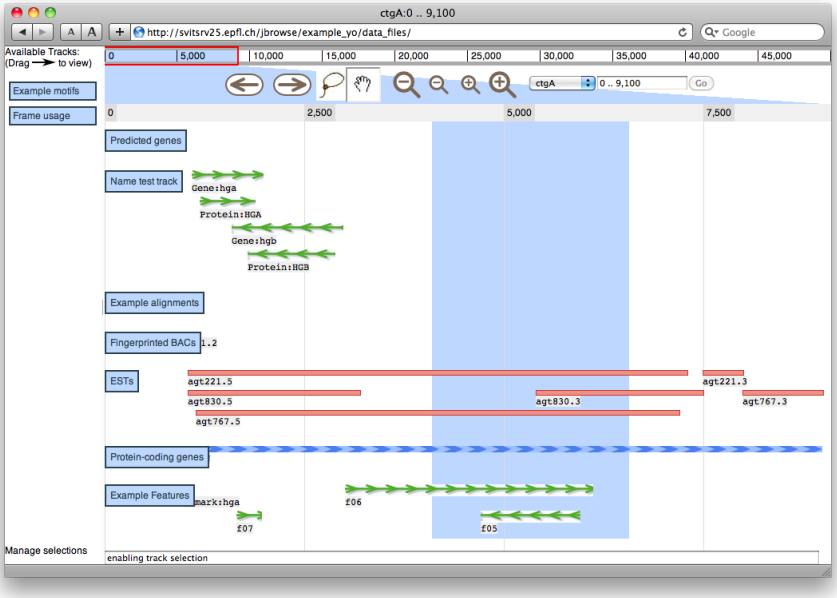

Figure 9: GDV's main interface, browsing a demonstration genome and after making a selection, as rendered by WebKit [8].

different levels of zoom off-line. The pre-rendering is done by the server while the layout and post-rendering is left to the client.

The browser can display several types of features such as qualitative data that represents features of interest on a genome, like an exon, intron, or UTR, as well as quantitative data where each base pair of a genomic region is associated with a score. At higher zoom levels, qualitative features are masked and substituted by a histogram indicating the feature density.

JBrowse also has a few disadvantages such as its makefile-driven workflow. Every time some new genomic data is added to a view, the data has to be prepared and processed using a set of logical rules, similar to that of a complier makefile.

4.3 Interface

Interface design and the user experience it creates is an important part of any project, and can seriously influence the usability and popularity of a product. Any software development team should have stringent standards on what GUI paradigms are employed. Examples of such guidelines are the famous HIG from Apple [1].

As we have demonstrated, the leading genomic visualizer today, UCSC, does not possess these qualities. A similar situation existed in the past with online map services. Most services had highly detailed maps, but there was no way to navigate them easily. Once Google Maps came along with their simple drag to move and double click to zoom, they captured almost the whole user base in a few months. This demonstrates once again that one of the key aspects for a new product after its functionality, is its the ease of use. Happily, one of the ambitions of the JBrowse project is to "extend the Google Maps experience to the genome browser". And indeed, the browser does provide a clean interface with drag to move and double click to zoom functionality, along with other simple interface paradigms.

The interface isn't everything of course, a tool can be intuitive but if doesn't solve a problem the users has or doesn't apply to the user's need, it will not be used. However, viewing genomic information is, without a doubt, an important need for many scientists around the world.

4.4 Extension

4.4.1 Selection

Some required functionality was missing from the current state of the JBrowse project, in particular the ability to make drag and drop selections on regions of interest. We added this functionality by modifying the underlying code. Multiple and discontinuous selections can be made quickly using the lasso tool. A screenshot of GDV can be seen in figure 9.

4.4.2 Data Management

JBrowse does not provide an interface to add and manage different views. This is also a requirement if one hopes that biologists and bioinformaticians will one day come to use it. JBrowse is relatively modular and can be used in a <plug-in> fashion with other services. Hence, a wrapper around the makefile process was written, also designed to be accessed from a web-browser. A screen shot of GDV's management interface can be seen in figure 10.

4.4.3 Intercompatibility

The management interface is still being developed and could in the future include options like intercompatiblity with other services. For instance, a module enabling the importation and exportation of track to and from Galaxy could be written.

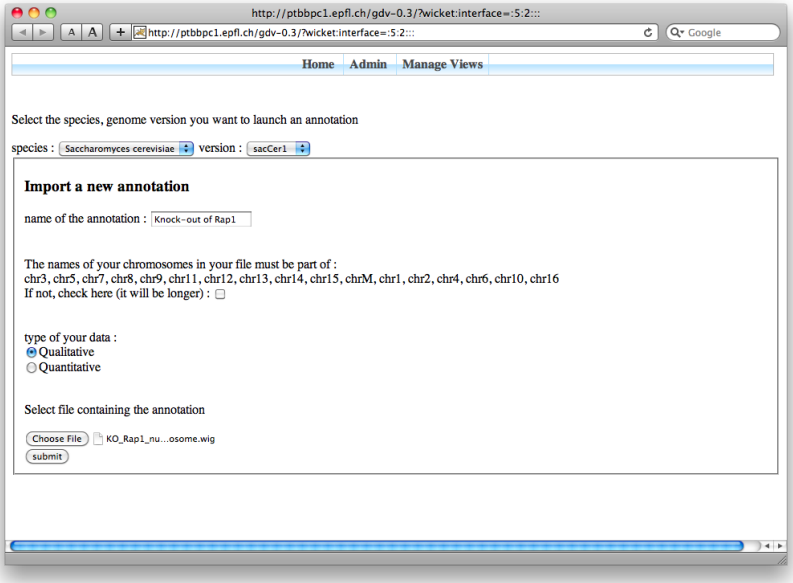

Figure 10: GDV's interface for adding and displaying tracks, as rendered by WebKit [8].

4.5 Expansion

A genome viewer is valuable but without some extra functionality bundled-in, its potential is limited. The idea is to integrate alongside GDV a set of tools that can compute descriptive statistics and manipulating the data that is being visualized. The aim is to provide something resembling the functionality that Galaxy has. This brings us to the software developed by myself under the name of genomic feature miner or gFeatMiner and is described in the next section.

5 Contributions to the project

5.1 Description

Abbreviated gFeatMiner, this tool will be bundled with the existing GDV genomic browser to enable the user to smoothly analyze and manipulate his data in different ways.

It should be composed of three modules. The first, discussed in section 5.3, must implement the computation and display of descriptive statistics. The second, discussed in section 5.4, must implement the manipulation of genomic data. The third, discussed in the future work section, remains to be done but should implement advanced analysis operations.

It should be written using modern and scalable technologies in order for it grow and be able to tackle problems like data mining in a near future.

5.2 Interface

gFeatMiner doesn't have an interface yet, but it will have to be integrated to GDV's interface in some way. To illustrate how this could be done, a naive schematic can be seen in figure 11.

Figure 11: Mock-up interface showing how gFeatMiner's commands could rest on the side of the current genome browser.

However, once integration is carried out, it should be fairly straightforward. The gFeatMiner module is wrapped around a simple server script that listens for incoming TCP/IP connections on a given port, and returns its answer through the same channel. If another program wishes to include services from gFeatMiner, there are no special libraries that need to be compiled against. Any other programming language can interact with gMeatMiner in a simple way as long as the language supports network sockets.

Once the socket is opened, gFeatMiner expects a correctly formatted request. The format itself is loosely defined in the form of a *de facto* standard INI configuration file with a number of mandatory options and a list of optional fields.

5.3 Descriptive statistics

The first goal is to provide some simple descriptive statistics over complete tracks or over a selection made by the user. Specifically, gFeatMiner should compute and graph answers to question such as the following:

- How many features are there in this track?
- What is the length distribution of the features inside the selection I made?
- What is the score distribution of the features inside these three tracks?
- What is the cumulative base coverage of these 2 tracks broken down by chromosome?

5.3.1 Request format

To understand the extent of the type of graphs gFeatMiner can produce, a good way to proceed is by describing the type of requests it accepts. The request is formulated in a text format with pairs of variables and values separated by an "=" sign and known as an INI file. This request is passed to the server script thru a network socket. An example request is provided below along with a detailed explanation of each field.

```
1  [gFeatMiner]
2  version=1.0
3  operation_type=make_plot
4  characteristic=number_of_features
5  graph_type=bar_plot
6  per_chromosome=True
7  compare_parents=False
8  wanted_chromosomes=chr1;chr2;chr5;chr6;chr7;chr8;chr9;chr10;chr11;
9  chr12;chr13;chr14;chr15;chr16;chr17;chr18;chr19;chr20;chr21;chr22;
10 chrX;chrY
11 selected_regions=chr2:0:300000;chr5:0:200000
12 track1=/home/sinclair/gFeatMiner/data/yeast/all_yeast_genes.bed
13 track1_name="hg19 refSeq genome-wide from UCSC"
14 track2=/home/sinclair/gFeatMiner/data/yeast/ribosome_proteins.bed
15 track2_name="hg19 HIV integration sites from liftOver"
```

Listing 5: An example request sent to gFeatMiner ordering it to compute the number of features on a subset of two different tracks while breaking them down by chromosome.

[gFeatMiner]: This is required as being the first line of the file, to make a request valid. This is a safeguard to reduce the probability of processing a request that was in fact not intended for gFeatMiner.

version: This field is required. The value inserted here will be compared against the current running version of gFeatMiner. If a mismatch is found, the request is not processed. This helps to avoid potential crashes due to deprecated or new features.

operation_type: This field can take the following values "`make_plot`" for generating descriptive statistics or "`return_track`" for manipulation genomic data. In this part the first option is described.

characteristic: Can take the following values: "`number_of_features`" or "`base_coverage`" or "`length`" or "`score`". Does not default to something, this is a required field. It decides what type of statistic is going to be computed.

graph_type: Can take the following values: "`bar_plot`" or "`box_plot`" or absent/false. It will default to the graph type best suited for the type of statistic requested.

per_chromosome: Can take the following values: "`True`" or "`False`". It will default to false if not specified. When enabled, this option will break down every chromosome inside a selection or a track and treat them as separate data.

selected_regions: Can be empty if no selection was made. Otherwise a list of locations and chromosomes is expected using colons and semicolons as separators. In such a case, the

statistic will then only be computed on the selection which may be discontinuous and span several chromosomes. A path to a local or distant BED file can also be used here.

compare_parents: This option can take the following values: "`True`" or "`False`". However, it is ignored if no selection was specified. In the case a selection was made and this option is enabled, gFeatMiner will compute the statistic on the selection as well as on the mother track as a whole and compare both statistics.

track1: This specifies the location, local or distant, of the first track. At least one genomic feature file must be inputted to compute a statistic. Of course, an undefined number of supplementary tracks can be specified. The order in which they are given does not influence the output. If gFeatMiner has previously processed this file, is it possible to substitute the file path by its MD5 hash.

track1_name: This field is optional but highly recommended. Without a name, the resulting graph will not have a comprehensive legend associated.

track2: Following tracks are specified according to this standard.

track2_name: Following tracks should also have a name associated.

5.3.2 Workflow

Once the request is received, it is processed in a workflow depicted in figure 13. Without going into the detail of the underlying code, this procedure parses the input files, organizes and partitions the data, computes the statistics, sets up the axes of the graph in the appropriate fashion and finally plots the results. The graph is then exported as a PDF or PNG file to be displayed by GDV.

5.3.3 Results

Here are provided, on the next pages, six different graphs produced by gFeatMiner giving a good overview of the different possible combinations. All of the examples are performed with genomic data coming from SGD [5].

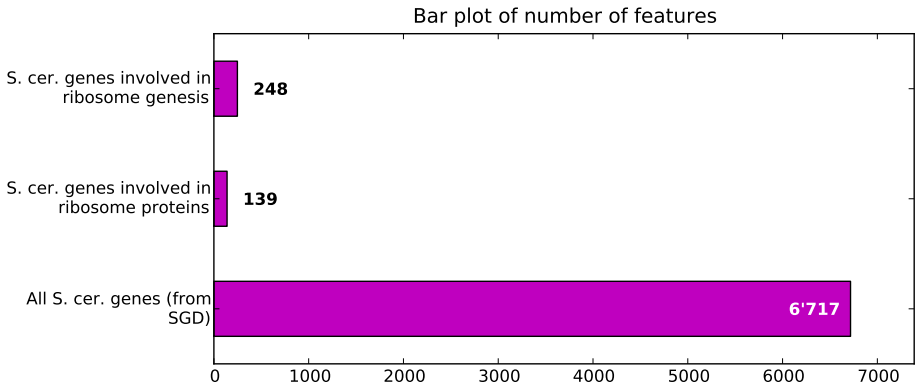

Figure 12: Graph of type E comparing the total number of all S. cer. genes against the number of genes involved in ribosomal protein and ribosome genesis.

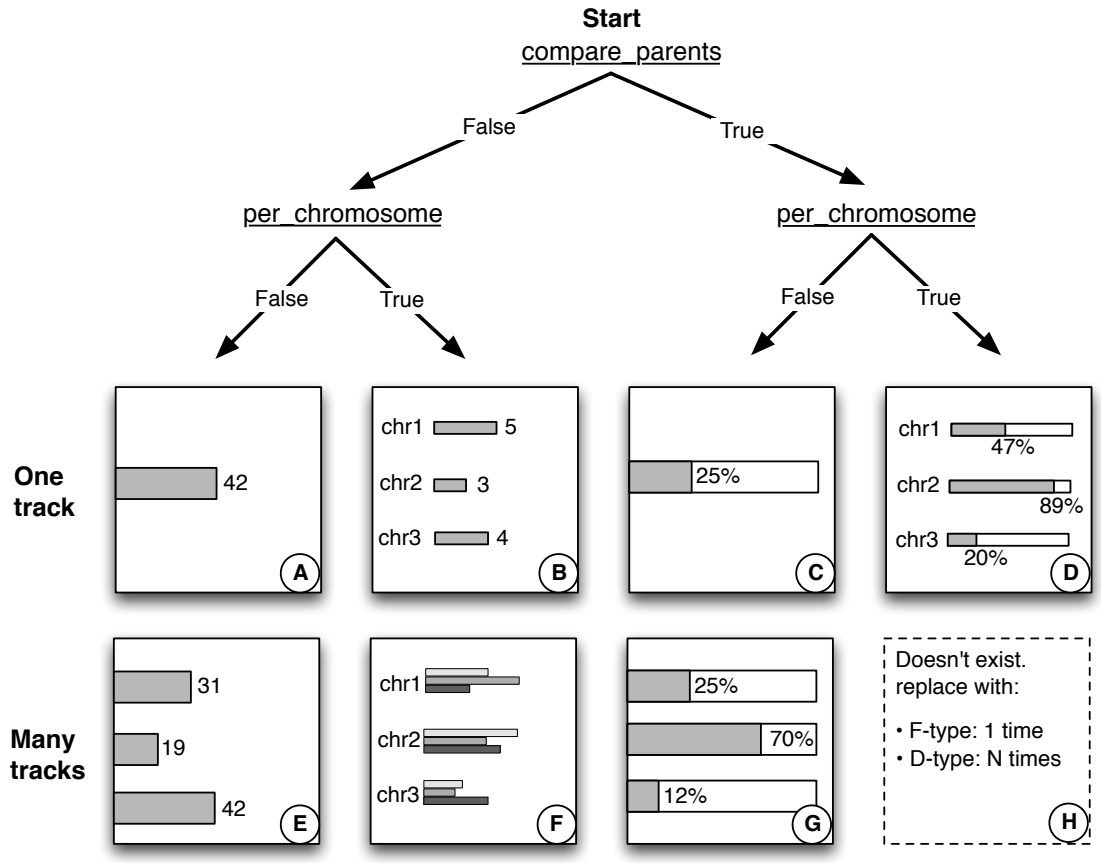

Figure 13: gFeatMiner will produce one of the 8 different bar-graphs depending on the boolean values of the two variables `compare_parents` and `per_chromsoome` as well as depending on the quantity of tracks inputted. The last case H would create a graph deemed too complex and is replaced by generating multiples other graphs. The same workflow exists for box-plot based graphs.

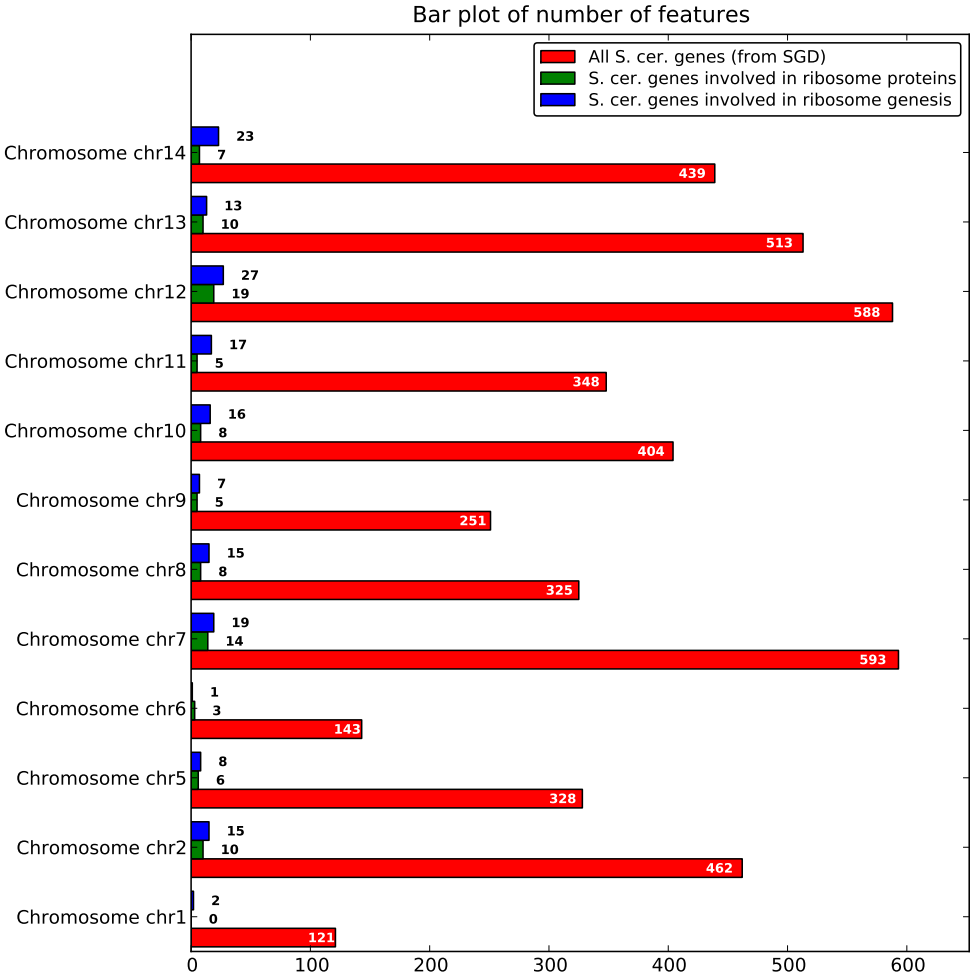

Figure 14: Graph of type F displaying the quantity of genes located on the first 14 chromosome of S. cer. among all genes (red), ribosomal protein genes (green) and ribosome genesis genes (blue).

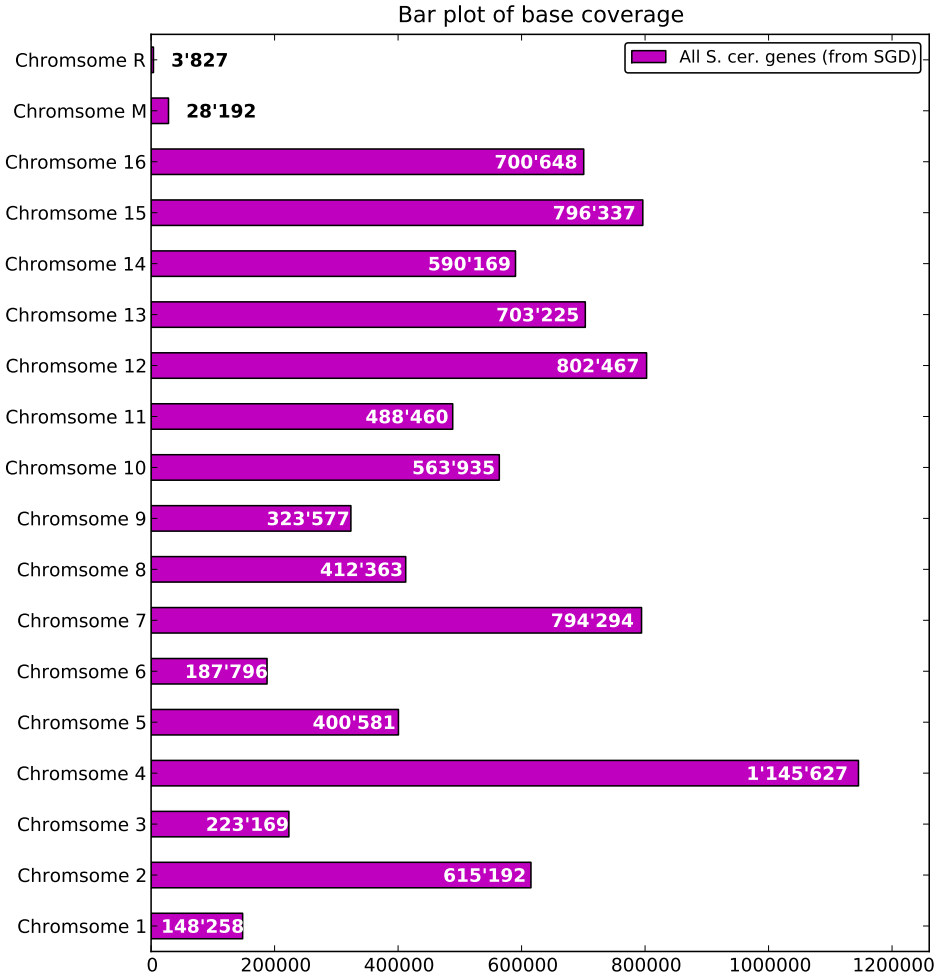

Figure 15: Graph of type *B* displaying the cumulative base coverage of all S. cer. genes by individual chromosomes. Chromosome R stands for the 2-micron DNA. Chromosome M stands for the mitochondrial DNA.

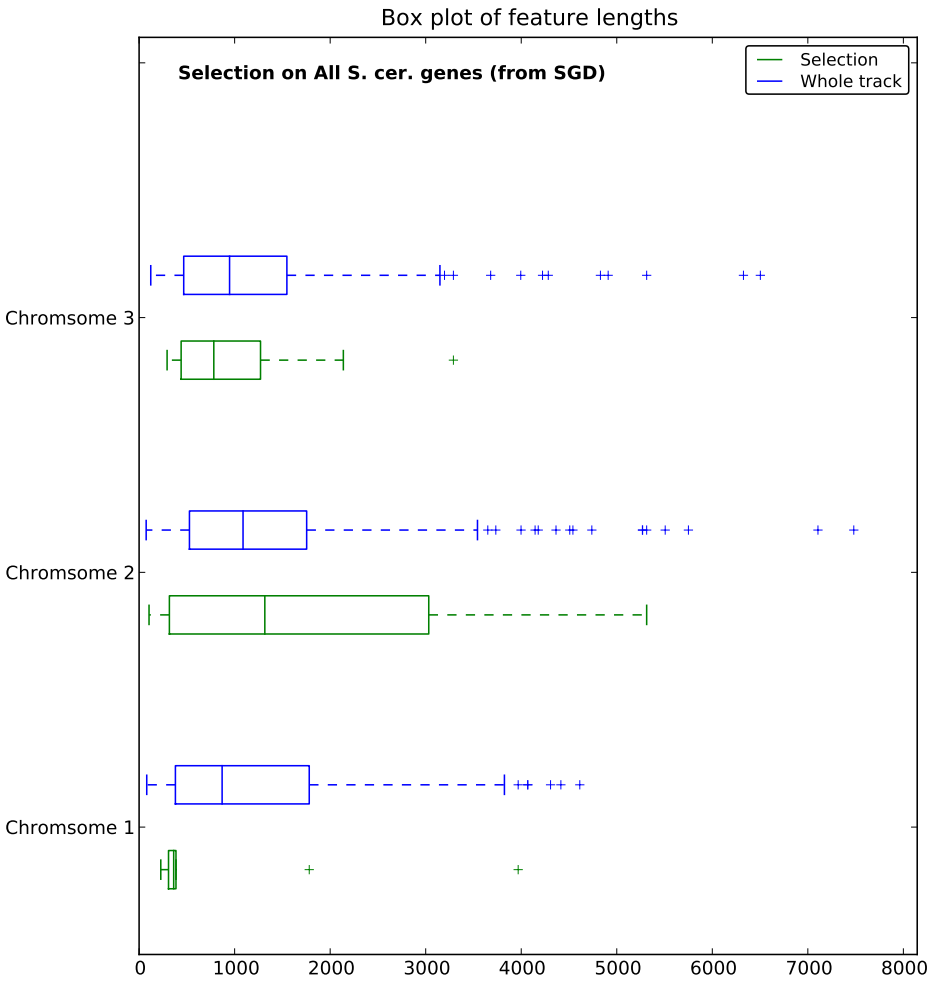

Figure 16: Graph of type *D* displaying the distribution of feature lengths broken down by chromosome on a discontinuous selection spanning regions of all S. cer. genes against the same distribution computed on the complete chromosomes.

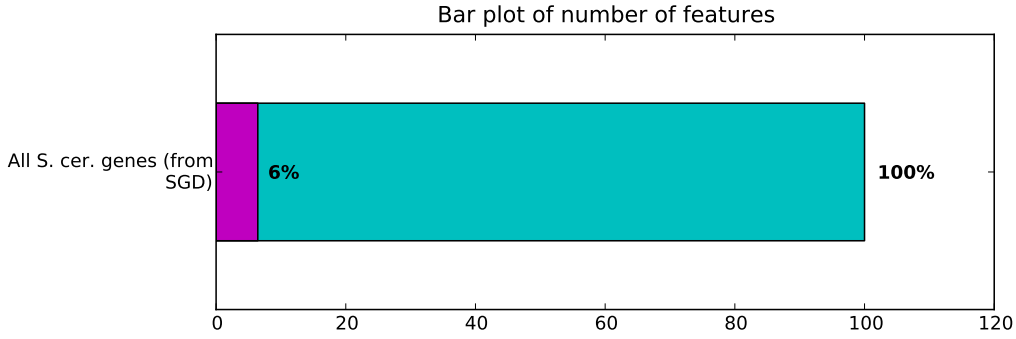

Figure 17: Graph of type *C* comparing the fraction of genes located on the mitochondrial and 2-micron chromosomes against all other chromosomes.

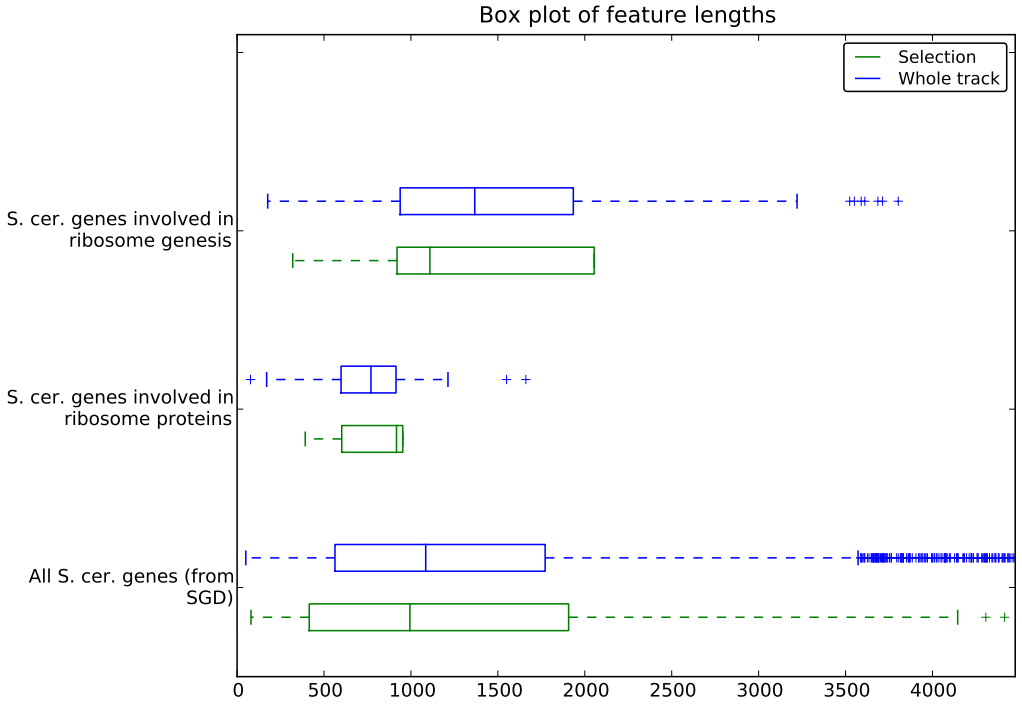

Figure 18: Graph of type *G* displaying the distribution of feature lengths on three different tracks. The distribution is first computed taking only the features contained in the selection made by the user and secondly taking the whole genome.

5.4 Genomic data manipulation

As a second goal, gFeatMiner should also be able to manipulate the genomic data in various simple ways. For example, it should be capable of answering the following demands:

- Provide a new track built by merging two other tracks.
- View the overlapping intervals of two tracks.
- Create a track that is the complement of another track.

A more exhaustive list of operations can be seen in figure 19. These types of data transformations can be useful for exploring relationships between different tracks and for revealing non-obvious correlations. This, in turn, can lead to the creation of new hypotheses on the underlying genomic processes. For example, computing the complement of a track describing genes will yield the intra-genenic locations. Similarly, computing the overlap between a list of genes and a list of protein binding locations can rapidly reveal the promotor sites of the genes bound by the given protein.

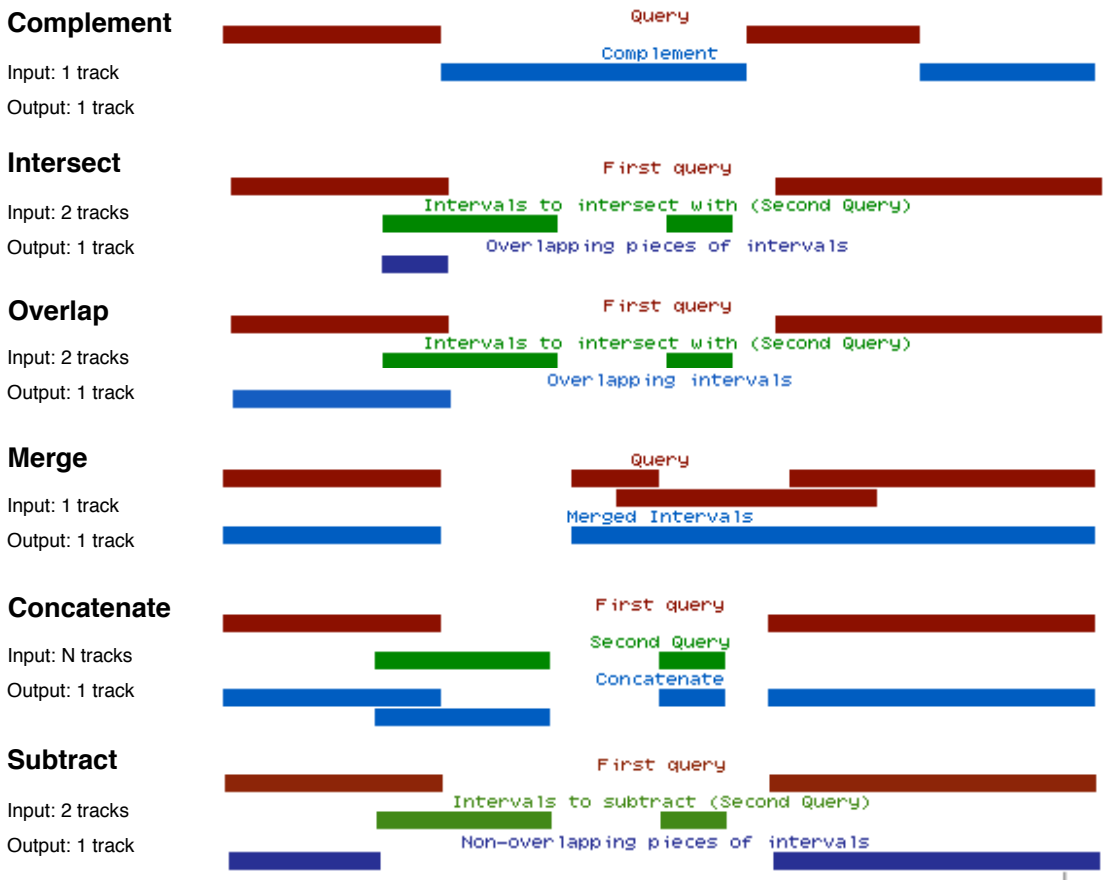

Figure 19: A listing of possible operations that can be performed on genomic data with their visual representations. Images courtesy of the Galaxy website.

5.4.1 Request format

The request method uses the same procedure as the descriptive statistics module expect for a few fields that specify the type of operation to perform. An example request with detailed explanations on the new variables is shown below.

```
1  [gFeatMiner]
2  version=1.0
3  operation_type=return_track
4  manipulation=overlap
5  output_format=bed
6  selected_regions=chr1:10000:50000
7  track1=/home/sinclair/gFeatMiner/data/hg19/refseq_ucsc.bed
8  track1_name="hg19 refSeq genome-wide from UCSC"
9  track2=/home/sinclair/gFeatMiner/data/hg19/hiv_bushman.bed
10 track2_name="hg19 HIV integration sites from liftOver"
```

Listing 6: An example request sent to gFeatMiner ordering it to compute and return the overlap between two tracks.

operation_type: This field can take the following values "`make_plot`" for generating descriptive statistics or "`return_track`" for manipulation genomic data. In this the second option is described.

manipulation: Can take the following values: "`complement`" or "`overlap`" or "`overlap_pieces`" or "`merge`" or "`internal_merge`". Does not default to something, this is a required field. It decides what type of operation is going to be carried out.

output_format: Can take the following values: "`bed`" or "`gff`" or "`wig`" or absent/false. It describes how the newly created track should be formatted at the output. It will default to BED.

selected_regions: Can be empty if no selection was made. Otherwise a list of locations and chromosomes is expected using colons and semicolons as separators. In such a case, the operation will then only be computed on the selection which may be discontinuous and span several chromosomes. A path to a local or distant BED file can also be used here.

5.4.2 Workflow

The workflow is relatively straightforward - forking the execution depending on the number of inputs the operation takes as show in figure 20. For example, a complement operation will take only one track as an input, while an overlap will take exactly two. Finally, a merge operation may take an arbitrary number of inputs.

5.4.3 Algorithms

Keeping in mind that gFeatMiner should be able to process large quantities of data rapidly, every component must be written in an optimized fashion. To make an illustrative example, let's take a closer look at the algorithm that computes the overlap between a track A and a track B. A naive way of solving this problem might be to use two nested loops comparing every entry in the track A to every entry in track B. This method would work with small files, but produces an

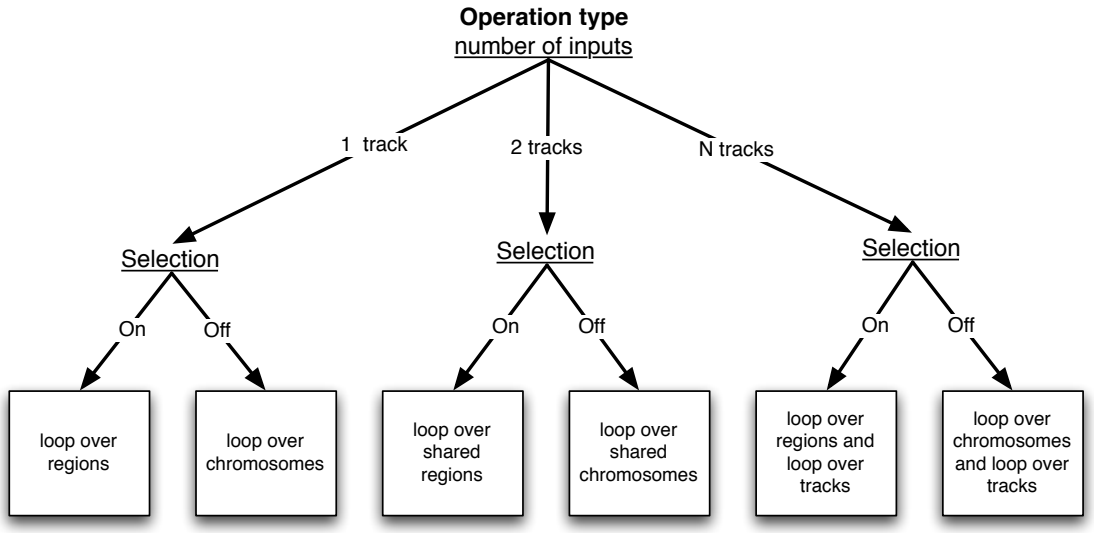

Figure 20: gFeatMiner will compute the type of operation requested according to the number of inputs the operation accepts and the presence or not of a selection.

$O(n^2)$ algorithm if one accepts that the number of feature are on the same order of magnitude in both tracks. Happily, there is a better way to compute the overlap between two tracks and avoid making most of the comparisons that the first algorithm would. If both tracks are sorted at the input and read in linear fashion it is easy to identify points where one can assert that no features in track B will ever overlap with any of the features to come in track A.

An algorithm that solves this problem in a $O(n)$ fashion or $O(n \cdot log(n))$ if the inputs are not sorted was published in 2006 under the name of fjoin [32]. This idea was recoded and implemented in gFeatMiner. A code snippet is provided in appendix section B.2.

5.5 Technologies and Infrastructure

Just as important as the functionality of gFeatMiner are the technologies on which it is built and the infrastructure it provides. gFeatMiner was coded to remain comprehensible and extensible by any other programmer. It is also built to remain computationally fast as it grows and is easy to debug.

5.5.1 Interpreted language

The first design decision taken was to write all of gFeatMiner's code in Python [38]. This language appeared in 1991 and has become a widely popular language with an enormous user base. Python is a general-purpose high-level programming language whose design philosophy emphasizes code readability. It is well documented and the large community has written hundreds of packages extending the functionality of the language.

5.5.2 Graphic library

To generate all of the graphs regarding the descriptive statistics part, an excellent python package entitled matplotlib [23] was used. Specifically designed for representing scientific data, it produces

publication quality figures. Matplotlib also strives to keep a simple syntax and copies the object-oriented standards borrowed from the Matlab language, making it easy to adopt for newcomers.

5.5.3 Mathematical packages

The Numpy [9] and Scipy [24] packages for python provide methods and functions covering numerous domains of scientific computing from statistical methods to modulable data containers. Moreover, most of the underlying code in these packages is written in C and will optimize the execution of common mathematical tasks. These packages are used throughout gFeatMiner, the python language itself being inherently high-level and ill-suited for heavy number crunching.

5.5.4 Optimized file access

As described in section 3.3, one of the problems plaguing the field of bioinformatics is the lack of standards in file formats and the fact that storing numerical values in their string representation is fundamentally a bad idea. This is why gFeatMiner, while taking several different formats as input, will first convert any track it sees to a rapid and efficient data storage before processing it. This file is then stored in the file system and can be reloaded instantly if the user requests a second manipulation on the same track.

The fast and efficient data storing format is the well-known HDF5 [3] standard. In essence, the HDF5 architecture permits the representation of any type of hierarchical datasets in an optimized manner from the viewpoint of disk space and access time. All read and writes from an HDF5 file are heavily cached and processed in batch. To implement this technology inside the python framework of gFeatMiner, the excellent pytables [7] package is used.

HDF5 is not to be confused with a database system like MySQL. There is no special request language to pull data out of the storage and every original BED or GFF file has an equivalent stand-alone HDF5 file representing the same information. Creating a data structure to represent genomic features is quite straightforward as shown in the code snippet below and data retrieval is accelerated by executing most requests via "in-kernel queries".

```
1 class Feature(IsDescription):
2         start = Int64Col(pos=1)
3         end = Int64Col(pos=2)
4         score = Float32Col(pos=3)
5         strand = StringCol(pos=4,itemsize=1)
6         name = StringCol(pos=5,itemsize=64)
```

Listing 7: An example of a small class describing an HDF5 table to store genomic features which have a start an end a score and a name.

Several other additions could be made to the HDF5 data storage system. The first, netCDF [19], is useful when the files containing the data are located on a different machine than the one running gFeatMiner. This python package will optimize the access and sharing of HDF5 files over the network. The second is the SZIP [33] library which can compress and inflate the data stored in an HDF5 file on the fly and in a lossless manner.

File I/O is major issue when dealing with large sets of genomic information. Indeed, the speed of algorithm is heavily dependent on information retrieval and caching. Most of the other tools that process genomic data in various ways almost always parse the original text file at every execution. gFeatMiner parses every file only once.

5.5.5 Inline machine code

Often in the processing on genomic data, one needs to iterate over large chunks of data and operate a few calculations, but no optimized function for the particular task exists in the standard mathematical packages, and coding the iteration in python would result in unacceptable slow-downs. This is when inline machine code comes to use. Using the weave package (included in scipy), one is easily able to write small functions in C++ code that can rest inside a python file as a multi-line string. These machine code functions are able to share and take control of python variable using on-the-fly converters. A code example demonstrating this is provided in appendix section B.1.

However, sometimes python is perfectly suited for the job and adding inline machine code doesn't add much speed. An example of an optimized python function is provided in appendix section B.3.

5.5.6 Streamlined tests

Another useful python package that helps build a project in a consistent and scalable fashion is the doctest module (included in the standard python library). This extension allows the programmer to write prototypical tests alongside his code. In practice, this consists in specifying the expected output of a function given a statically defined input. An undefined number of tests can be assigned to each function in a project to enable the frequent execution of a series of automated test on subparts of a program. The code is thus easily controlled and notifications appear if a part of an algorithm breaks due to badly thought out modifications. Naturally, as the code base grows, previous parts of the software rely on consistent behavior of individual functions which must remain unchanged.

5.5.7 Revision control

Finally, as is common with software projects, the entire code base is placed under a revision control system, in this case the popular SVN [30] solution. This creates a repository for every file involved in a project and will automatically back-up every script while tracking modifications in a reversible manner. This type of technology can also be extremely useful when several programmers want to contribute to the same project. This is not the case for gFeatMiner at the moment, but could be one day.

5.6 Performance

It order to compare the speed of execution of other tools providing similar functionality. The table below presents results obtained by timing manipulations of genomic data using gFeatMiner, bedtools [31] and bx-python [35].

All the tests were executed on a Dell computer equipped with four x86_64 intel processors clocked at 2.4 [Ghz], 24 [Gb] of RAM and running Fedora release 12. Input files were all randomly generated BED files containing a million features. An excerpt of such a file is shown below.

```
1 chr1    54      139
2 chr1    57      121
3 chr1    140     233
4 chr1    247     299
```

Listing 8: An excerpt from a randomly generated BED file.

Base coverage	
Number of features	Size of file
10^6	21.5 MB
Algorithm	Time [sec]
gFeatMiner parsing	12.87
gFeatMiner run	1.94
bx-python	42.10
bedtools	4.34

Figure 21: A table comparing the execution speed of different tools computing the same operation, namely the cumulative base coverage of a track.

Complement	
Number of features	Size of file
10^6	21.5 MB
Algorithm	Time [sec]
gFeatMiner parsing	13.02
gFeatMiner run	5.23
bx-python	55.61
bedtools	2.93

Figure 22: A table comparing the execution speed of different tools computing the same operation, namely the complement of a track.

Examining figure 21 and 22 one may note that gFeatMiner gives excellent results even outperforming bedtools (a set of tools coded entirely in C++) on some tasks. gFeatMiner's execution time is separated into two distinct phases. The first, "parsing", accounts for the creation of the HDF5 entry in the file system corresponding the BED file. This operation only needs to be done once for every new file presented and any subsequent operation on the same input will skip directly to the second "run" phase.

One must also keep in mind that while bedtools is very fast, its versatility is severely limited. The bedtools set of programs are restricted to computing 17 types of manipulations on a single type of file format. Bx-python, which regroups the algorithms that power Galaxy, compute results in times of an order of magnitude slower than the other tools.

5.7 Future work

The gFeatMiner project is in a functional state, however more work is needed in order to reach a fully working state. The integration on GDV's side and the development of the data mining aspect of the project remain outstanding tasks. An other useful addition discussed here is parallelism.

5.7.1 Parallelism

Parallelism could be an interesting extension to attain increased execution speed. As of today such services typically run on octa-core servers (i.e. computers equipped with eight central processing units), but most code that is executed is only able to take advantage of one processor at a time. One must note that not all computational operations can be divided in a such way, and some problems are proven to be unparallelizable. However, in the domain of genomic information processing, most of the operations are embarrassingly parallelizable. For example, when computing the overlap between two tracks, there is no restriction on dividing the number of features found on a genome into an arbitrary amount of subgroups and computing the overlap individually on the subgroups. Indeed, once the outputs are reassembled, the result is exactly the same as that of a function that would compute the overlap in one go.

Once again there are some excellent python packages that will provide such functionality to a pre-existing code base, requiring only minor code refactoring. A common implementation is the multiengine interface provided in ipython [4]. Using such technologies, an eight-fold increase in execution time could be achieved at a low cost. Furthermore, once algorithms have been recoded

in such a way, they can also be parallelized between multiple machines and use an arbitrarily large number of processors at the same time.

5.7.2 Data mining

One of the goals for the gFeatMiner project is to attack the complex problems of data mining and advanced analysis.

The objective is to identify regions and/or groups of genomic feature that will maximize one or several measurements using clustering and decision trees. These measurements might include: preference scores, GO-terms, SNP families, motifs, density of genes, conservation levels, etc.

A simple example using a preference score function on a set of genomic tracks and a list of peaks extracted from a Chip-Seq experiment could be the following: what combination of genomic tracks maximizes the proximity of my peaks to the upstream regions of the genes involved? This would be a good area for automatization since it is impossible for the user to test every combination manually.

With the infrastructure and technologies used, *gFeatMiner* should be up to the challenge.

6 Application with real data

6.1 Description

For the second part of this project, we turn to a practical application in which I will be combining and analyzing genomic data collected by different laboratories working on the model organism yeast, *Saccharomyces cerevisiae*. Also known as the Brewer's yeast, S. cer. is a type of budding yeast intensively studied in molecular and cell biology; it has become one of the better understood examples of eukaryotic cells. S. cer is also of interest for the study of the cell biology of higher-level eukaryotes. For instance, many human proteins were first discovered by homology in the budding yeast.

6.2 Context

The project focusses on the ribosomal proteins of S. cer. and more specifically on the genes coding for these proteins. Using various quantitative datasets on a few of the known transcription factors involved in ribosome genes production, in combination with information concerning the nucleosome enrichment of the yeast genome, as well as the transcription level of these genes, we will attempt to extract statistical correlations. This part of the project performs an example of the type of advanced analysis that will, one day, be automatic in gFeatMiner.

All the data from the different experiments were collected under similar conditions. The yeast cells are in normal growth phase and no special conditions or environmental stress were applied to the colonies.

Ribosomal proteins are a crucial part of any cell's life. In the growth phase of S. cer., it is assumed that approximately half of the RNA polymerase transcription activity is devoted to RP genes [40].

6.3 Datasets

Six different genomic datasets from diverse sources were used – some available in the public domain, others coming from as yet published research.

6.3.1 Ribosomal genes

The first essential dataset is, of course, the complete list of gene coding for ribosomal proteins in S. cer. In total, 139 genes scattered across every chromosome have been annotated to code for ribosomal proteins in S. cer. The promoter regions of these genes are going to be of particular interest to us.

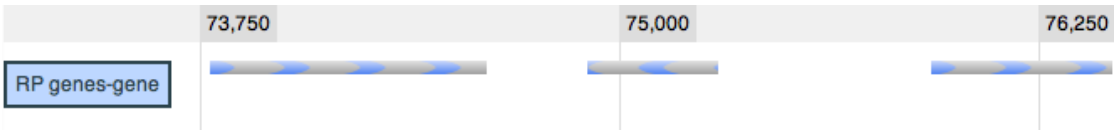

Figure 23: Excerpt from the file `"ribosome_proteins.wig"`.

6.3.2 ChIP-Seq for Rap1, Fhl1 and Ifh1

This data was obtained through ChIP-Seq using antibodies designed to bind a c-Myc tagged protein. Mutated strains expressing such fusion proteins for Rap1, Fhl1 and Ifhl1 were used. A control experiment where no specific tag is included is also provided. The ChIP-Seq laboratory technique is described in section 3.4.1.

These three transcription factors are known to regulate the transcription of most of the 139 ribosomal genes [39]. Rap1 binds many of the RP gene promoters but also functions as a repressor in other cases and is thus not the specific regulator of transcription. However, Fhl1 and Ifhl1 bind in an almost exclusive fashion to RP gene promoters, the latter being recruited by the former. The level of association of Ifhl1 is the best predictor of transcriptional activity.

This data is yet to be published and is treated as private information provided by the Shore Lab at University of Geneva.

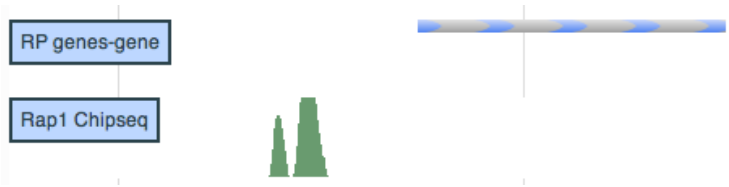

Figure 24: Excerpt from the file `"Rap1_chipseq.wig"`.

6.3.3 RNA Polymearse

This data is also produced by a ChIP-Seq experiment using antibodies against RNA polymerase II. One can thus use this information as a good indicator of the level of transcriptional activity of any given gene found in the yeast genome under normal conditions.

This data was obtained by the Snyder lab at Yale University and is in the public domain [26].

The raw data is provided in the form of two files containing a score for every nucleotide along the forward and reverse strands of every chromosome of the yeast genome. Instead of extracting peaks in the usual way with software like Quest [37] or MACS [41], I used a simple method that consists in averaging the forward and reverse strands at every base. Using such a strategy is reasonable in this case, as peak detection algorithms are commonly implemented to identify regions of binding and define zones of interest using no *a priori* information except the tag counts

generated by the alignment of the ChIP-Seq reads. However, in the context of this experiment the zones of interest along the genome are predefined and they need not be extrapolated. interest along the genome are predefined and they need not be extrapolated. Suppressing a further step of treatment and remaining closer to the raw data is actually a benefit.

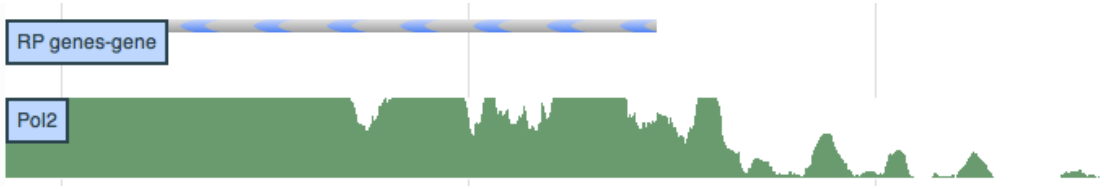

Figure 25: Excerpt from the file `"PolII_peaks.wig"`.

6.3.4 Mitomi for Rap1 and Fhl1

Two extra tracks describing the in-vitro binding affinity of Rap1 and Fhl1 are provided from another laboratory. These results are unpublished at present and are considered private data owned by the Maerkl Lab at EPFL.

The data was obtained using the mitomi laboratory technique as described in section 3.4.3. As a result one obtains binding predictions for two transcription factors based only on the DNA sequence of the genome while ignoring the chromatin structure.

Ifh1 is not present in the dataset for the simple reason that it does not bind directly to the DNA interacting instead with a domain on Fhl1.

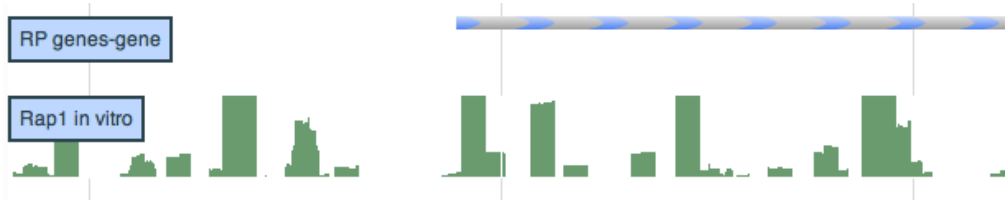

Figure 26: Excerpt from the file `"Rap1_invitro.wig"`.

6.3.5 Nucleosome enrichment

This data was gathered by Hughes lab at the University of Toronto, and was published as supplementary data in 2008 [10]. The laboratory technique used consists in digesting the free DNA of yeast cell with micrococcal nuclease and subsequently hybridizing the fragments to a tiling array. MNase is special kind of exonuclease that is able to digest double-stranded DNA around nucleosome linker regions, but only induces single strands nicks inside in the nucleosome itself [2]. These fragments of DNA can be purified and analyzed using microarrays to determine the genomic positions of nucleosomes.

Two tracks are provided, one WT where the yeast's genome was not modified and one for a knock-out of Rap1. The knock-out is taken from the collection of temperature-sensitive mutant repository of the essential genes of Saccharomyces cerevisiae [12]. Since ribosomal proteins are essential for the cell's life, a classic knock-out is not possible.

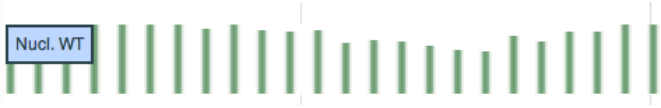

Figure 27: Excerpt from the file `"WT_nucleosome.wig"`.

6.3.6 Binding site footprints

This data was collected by a lab at the University of Washington, and published in a paper in 2009 [22]. The strategy used to profile regulatory protein occupancy on genomic DNA consists in mapping DNase I cleavages using massively parallel sequencing. This technique called *digital genomic footprinting* essentially counts the frequency of DNAse cleavage and uses this information to predict protected regulatory protein footprints where cis-regulatory factors are highly likely to bind.

Since this experiment is done in-vivo it is sensitive to the chromatin structure of the DNA. One can therefore expect that it will not predict potential protein binding sites that are hidden by histones. Indeed, if the DNA in question is only favorable in sequence to protein interactions but is not exposed sufficiently in-vivo, it will not be identified.

Figure 28: Excerpt from the file `"dna_footprints.png"`.

6.4 Upstream regions

6.4.1 Regions of interest

In order to characterize the patterns in which the three transcriptions factors of interest modulate the expression of S. cer. ribosomal genes, we specifically focus on the upstream regions of the 139 RP genes. Using an unvarying region of interest defined as 500 [bp] downstream and 1000 [bp] upstream of a gene's TTS, we will extract the quantitative values of all our datasets within this window. Since every RP gene is associated with precisely one region of interest, this results in the creation of 139 windows or miniature tracks made up of only 1500 base pairs. Two examples of these windows, for a gene lying on the positive strand of a piece of double-stranded DNA, and for a gene lying on a negative strand, are provided in figures 30 and 29. Windows that are extracted from a negative strand are flipped so that every track follows the transcription direction.

6.4.2 Interference

A problem that may arise when using fixed windows in the upstream regions of genes is the interference from promoters of neighboring genes. If a given part of a chromosome is shared between two windows belonging to different RP genes, there is no way of telling which of the

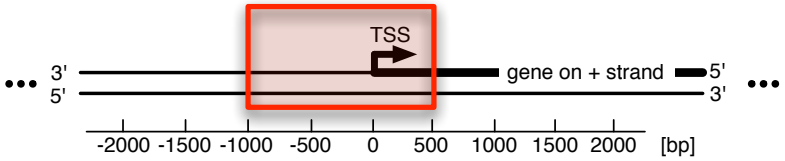

Figure 29: A region of interest defined as window spanning 500 [bp] downstream and 1000 [bp] upstream of a gene's TTS lying on the plus strand of chromosome.

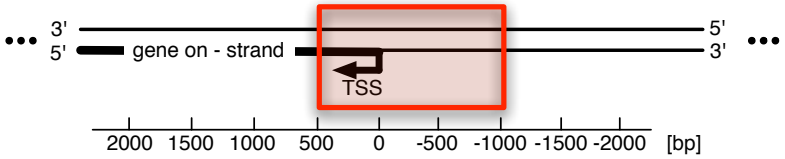

Figure 30: A region of interest defined as window spanning 500 [bp] downstream and 1000 [bp] upstream of a gene's TTS lying on the minus strand of chromosome.

two genes the transcription factors are enhancing. An example of a such a situation is shown in figure 31.

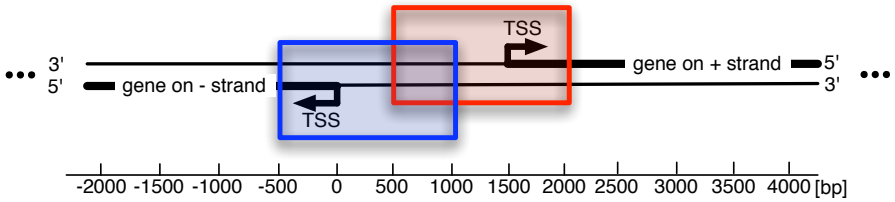

Figure 31: Two regions of interest interfere with each other. In a few cases, two ribosomal genes lie in close proximity on opposite strands causing difficulties in the analysis of transcription factors bound to their upstream regions.

To assess the frequency of such events, two graphs quantifying the number of RP genes subject to interference according to the distance from their TSS are shown in figures 32 and 33. The cumulative distributions are computed for all RP genes against all RP genes themselves and, subsequently, against all S. cer. genes.

The result shows that only four pairs of RP genes interfere with each other within the regions of interest defined. Considered to be outliers, these eight genes are removed from further analysis. The exact references to these genes are given below.

```
1 YDL083C on chr4- at 307789      with YDL082W on chr4+ at 308424
2 YJL190C on chr10- at 75301      with YJL189W on chr10+ at 75932
3 YMR142C on chr13- at 551206     with YMR143W on chr13+ at 551927
4 YOL040C on chr15- at 253576     with YOL039W on chr15+ at 254296
```

Listing 9: Exact names of the eight RP genes suffering from interference from an other RP gene start inside their upstream regions.

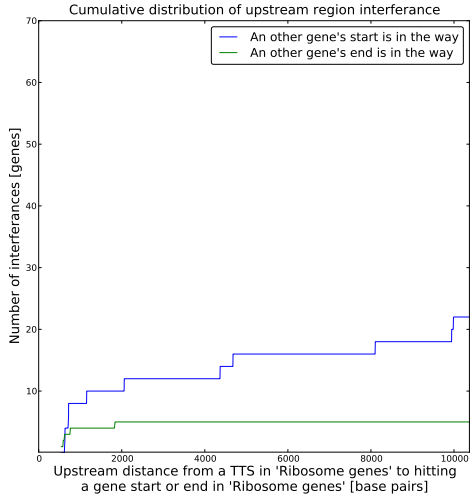

Figure 32: Cumulative distribution of the distance from every ribosomal gene to the next ribosomal TSS or TTS.

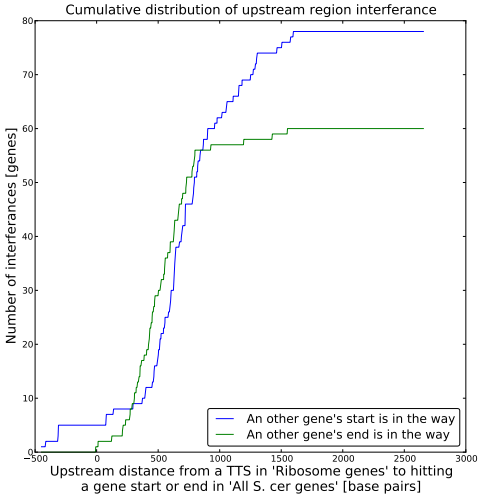

Figure 33: Cumulative distribution of the distance from every ribosomal gene to the next TSS or TSS on the S. cer. genome.

6.5 Clustering

6.5.1 Criteria

Once every region of interest is established and the relevant values from the six different dataset extracted, our goal is to regroup similar kinds of windows into clusters. Inside such a category, the continuous values of each window are averaged together and compared to all the other categories. Two questions arise at this step:

1. Which criteria are used for comparing windows among each other and judging on their similarity ?

2. Which criteria are chosen to assign a window to a specific category rather than another ?

6.5.2 Scoring

The first question can be handled by establishing a function that reads the data from a window and assigns a score to it. Once every window has a unique value associated to it, one can decide into which category the window must fall. Clustering data points placed in a one dimensional space is then a simple matter.

Scoring a window can be done in various way. Most of the time the scoring function will only examine one of the datasets inside the region of interest, e.g. the Rap1 values. One might, for example, decide to regroup all windows with a high level of Rap1 together and, in another category, cluster those with an absent or low level of Rap1. In such a situation, the scoring function can simply return the integral under the curve of the Rap1 signal as a score for each window.

Or, one might want to combine multiple scoring functions and attribute several scores to each window. For instance, a window might be tagged with two scores: the strength of its Rap1 signal and the distance of the Rap1 peak from the TSS.

6.5.3 K-means

The second question is dealt with by establishing a function that, given one or more scores per window, classifies them into a fixed number of categories. An excellent candidate for this function is the K-means machine-learning method. This clustering strategy will partitions n observations into k groups by trying to minimize the following cost function:

$$\sum_{i=1}^{k} \sum_{x_j \in S_i} \|x_j - \mu_i\|^2 \tag{1}$$

The first constraint it that the number of categories be necessarily smaller than the number of observations: $k < n$. Next, every observation $x_j \subset (x_1, x_2, ..., x_n)$ is a vector in d-dimensional space. In our case, an observation corresponds to a list of d scores associated to one upstream window. Every observation is assigned to a particular category $S_i \subset \{S_1, S_2, ..., S_k\}$. Finally, μ_i represents the mean of points included in S_i. The K-means algorithm thus clusters the observations so as to minimize the sum of squares within each category.

To better understand the functioning of the algorithm, an example using $n = 15$ observations in a two-dimensional space and $k = 3$ groups is provided in figures 34, 35, 36 and 37.

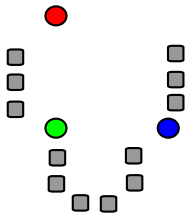

Figure 34: Illustration of the K-means algorithm at step A. Image courtesy of Wikimedia Commons.

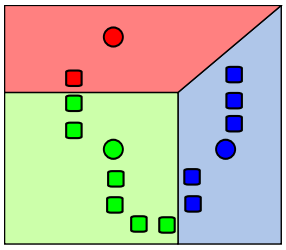

Figure 35: Illustration of the K-means algorithm at step B. Image courtesy of Wikimedia Commons.

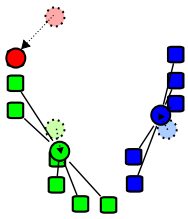

Figure 36: Illustration of the K-means algorithm at step C. Image courtesy of Wikimedia Commons.

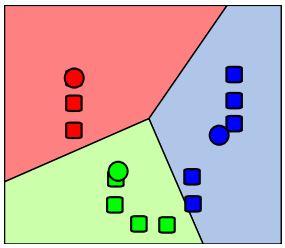

Figure 37: Illustration of the K-means algorithm at step D. Image courtesy of Wikimedia Commons.

- At step A, every gray square represents an observation in an undefined space. In our case, the x-axis might represent the strength of the Rap1 signal and the y-axis the distance

from the Rap1 peak to the TSS. Next, k initial centers are created by selecting k arbitrary observations from the dataset.

- At step B, every observation is assigned to the nearest of the k centers. This step separates the data space into k surfaces.
- At step C, each of the centers are moved from their original location to the centroid created by the observations that were assigned to it at the previous step.
- Steps B and C are repeated until convergence.

6.5.4 Summary

A schematic summary of the clustering process is provided in figure 38.

6.6 Results

With the set of scripts developed it is possible to explore and combine the data in many different ways. One is able to perform one or two dimensional clustering on any of the tracks in the six datasets. Figures 39 to 43 present different situations and describe the potential implications concerning the biological processes that govern the binding of the three transcription factors of interest on the promoter regions of RP genes in S. cer.

In every cluster configuration the number of categories was set to 3 and a graph summarizing which windows are assigned to which category is included.

7 Conclusion

7.1 Closing words

As the age of computational biology approaches, tools like gFeatMiner will become increasingly important as petabytes of data overwhelm the field. The ability to aggregate and analyze data from diverse sources will be valuable to generating new hypotheses and discovering new biological processes.

The tools constructed were designed and implemented to facilitate their integration, maintenance and future development. Particularly attention was paid to the choice of language and libraries used, as well as the documentation of the code.

Given the short time span of the project, I feel I have come a long way and reached the projected goal. I found the internship highly interesting and engaging, and was pleased to be confronted with some real-world problems in the field of genomic bioinformatics.

7.2 Contact

The author of this document can be contacted at the following address: lucas.sinclair@me.com

Figure 38: Summary of the clustering process using only four windows and two categories. In the first step, four upstream regions of interest are defined from the list of RP genes and the values for Rap1 (in red) and Fhl1 (in blue) extracted. In the second step, each window is evaluated by two different scoring functions: one computing the strength of the Rap1 peak and the other computing the distance of the Rap1 peak from the TSS. In the third step, each original window is plotted as a point in a two dimensional space according to the scores it receives. In the fourth step the k-means algorithm is applied and two categories are formed. In the fifth step, a profile is built for each category by averaging the values of the original windows together.

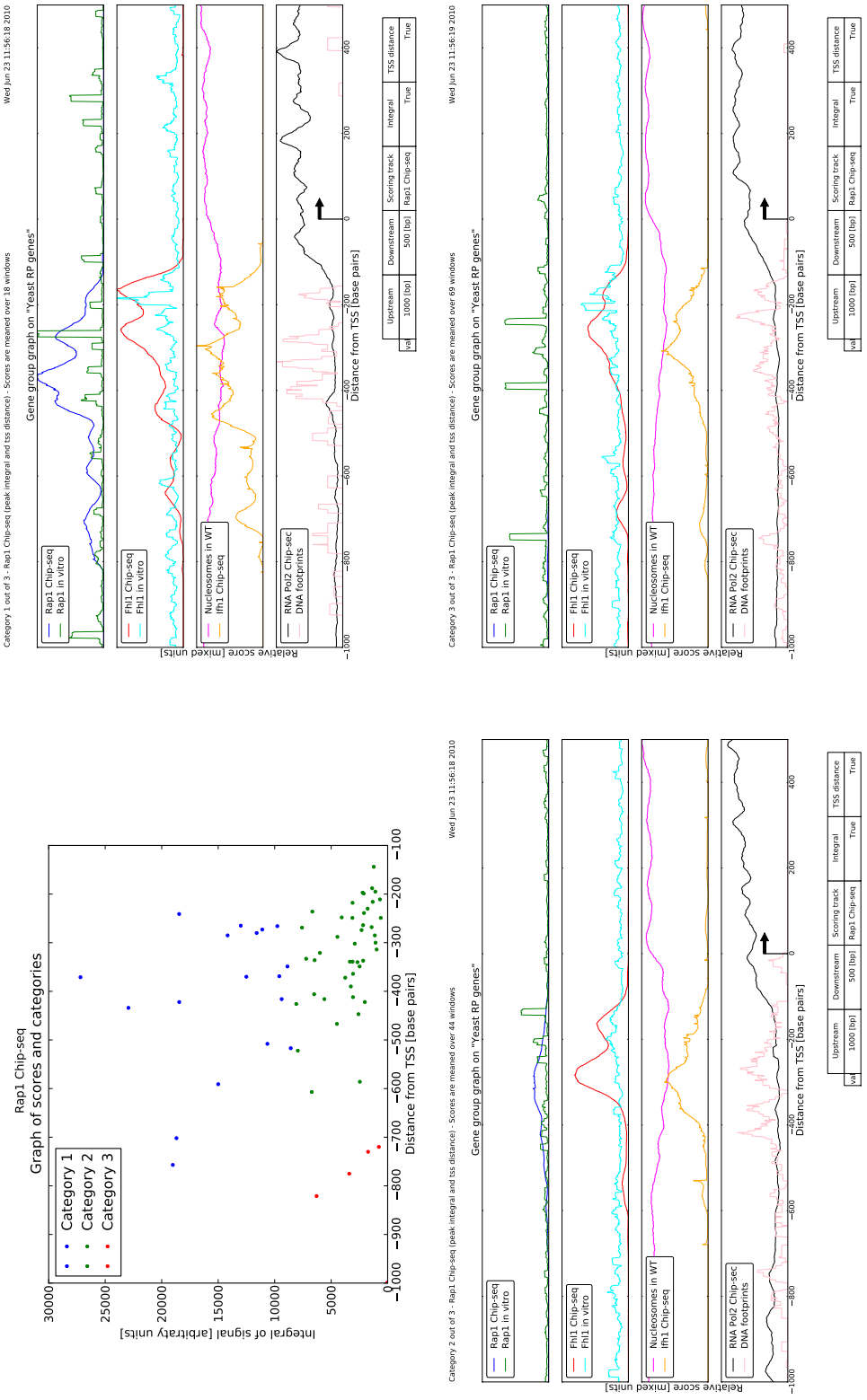

Figure 39: The 131 windows of interest are clustered into 3 categories according to the intensity of the Rap1 signal and the distance from the TSS of the Rap1 peak. Strong intensities of Rap1 are correlated with strong intensities of Fhl1 and Ifhl1, however the presence of Rap1 is not mandatory for the presence of the two other transcription factors. The strength of the Rap1 signal cannot be used as a predictor of the transcriptional activity, as the PolII signal is constant across categories. The absence of Rap1 in the third category cannot be explained either by the mitomi track or the nucleosome data.

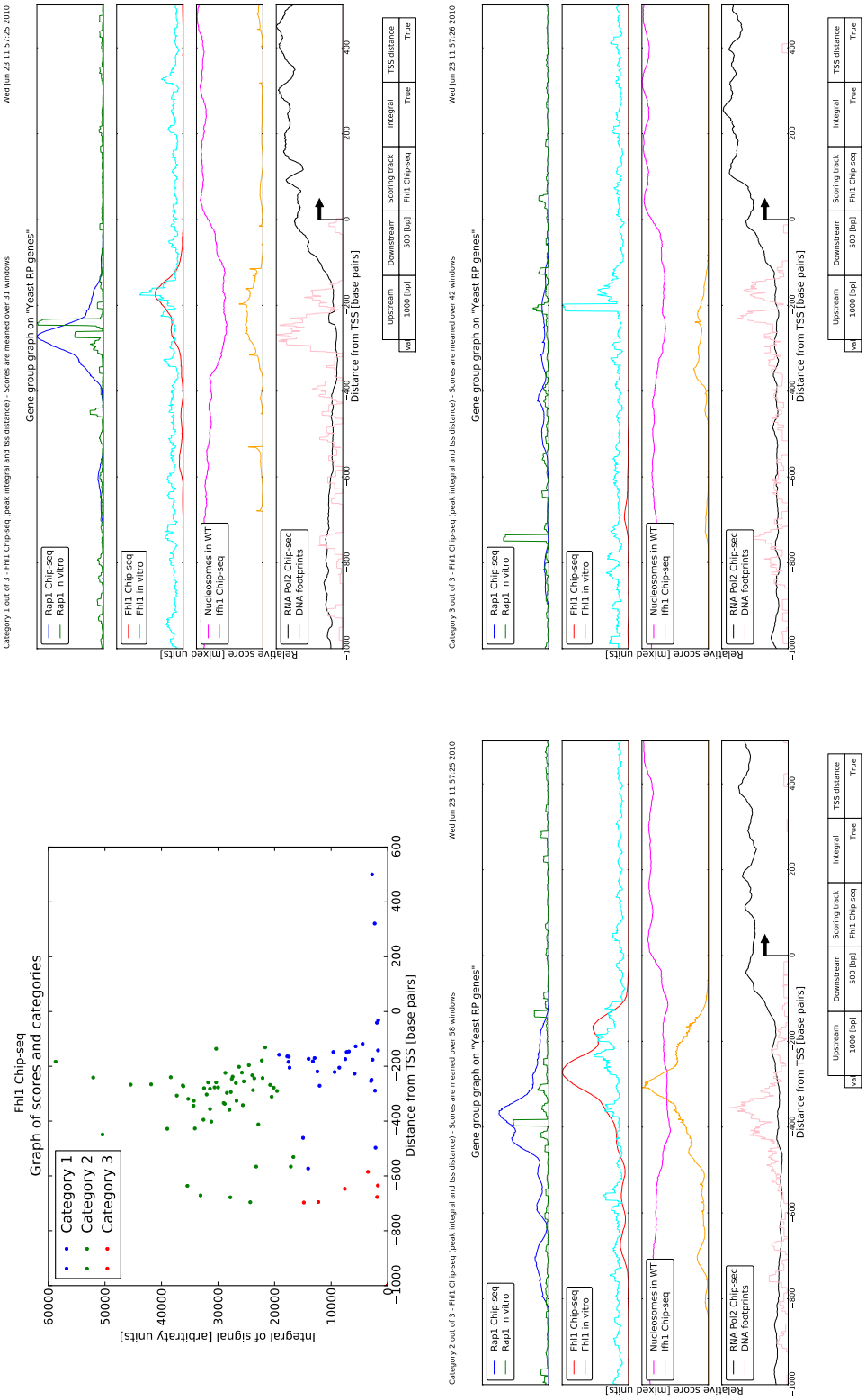

Figure 40: The 131 windows of interest are clustered into 3 categories according to the intensity of the Fhl1 signal and the distance from the TSS of the Fhl1 peak. The intensity of the Fhl1 signal is strongly correlated with the intensity of the Ifh1 signal. This is not surprising as Ifh1 interacts with a domain on Fhl1. The presence of Fhl1 can predict, to an extent, the presence of Rap1 but does not affect the level of transcription as the PolII signal is constant across categories. A slight opening in nucleosome enrichment can be seen in categories 1 and 2 where Fhl1 is strongest.

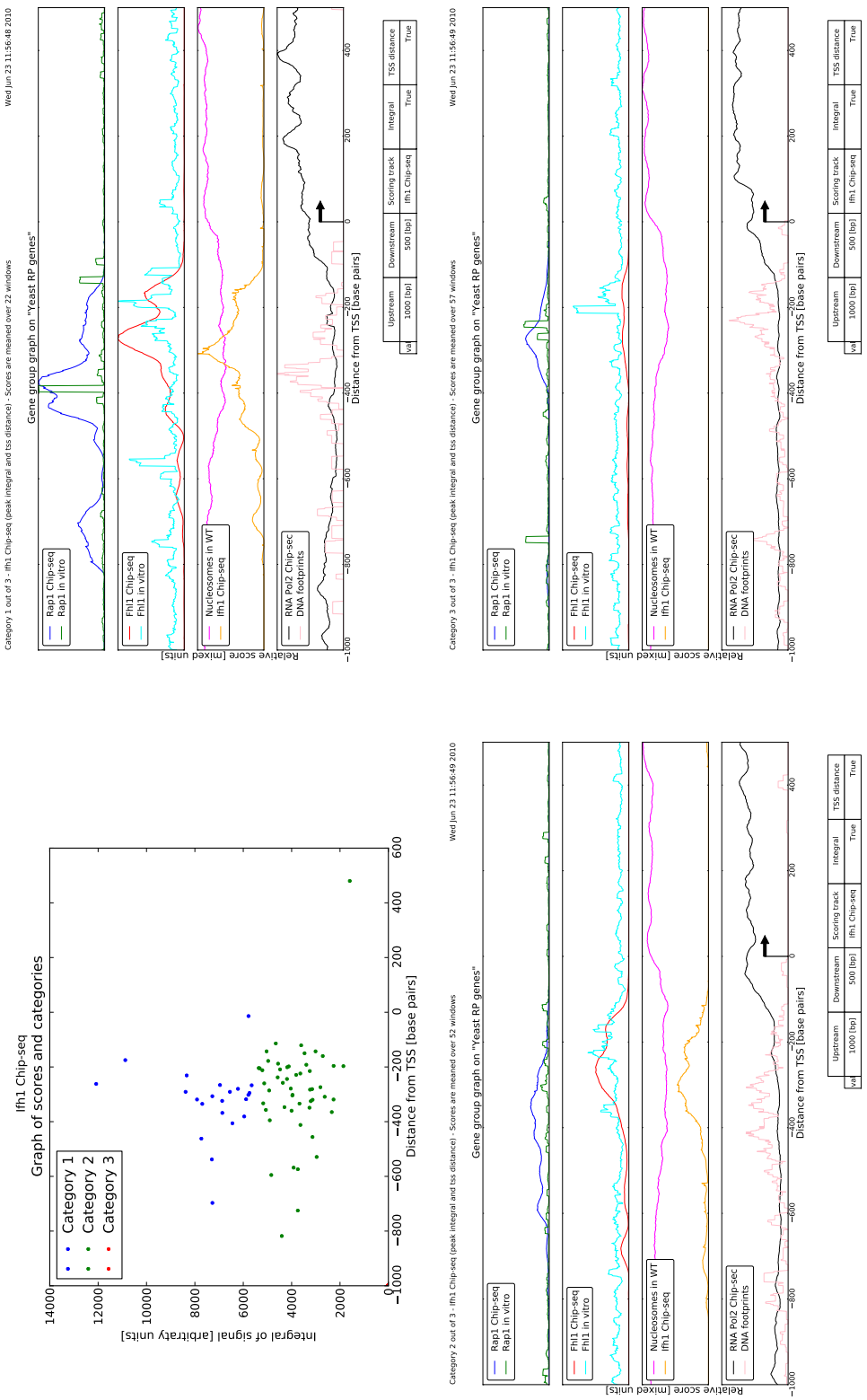

Figure 41: The 131 windows of interest are clustered into 3 categories according to the intensity of the Ifh1 signal and the distance from the TSS of the Ifh1 peak. The strong correlation between the Ifh1 and Fhl1 signals appears once again along with a colocalization with the Rap1 transcription factor. The mitomi and DNA footprints data do not seem to explain why binding is absent in the third category.

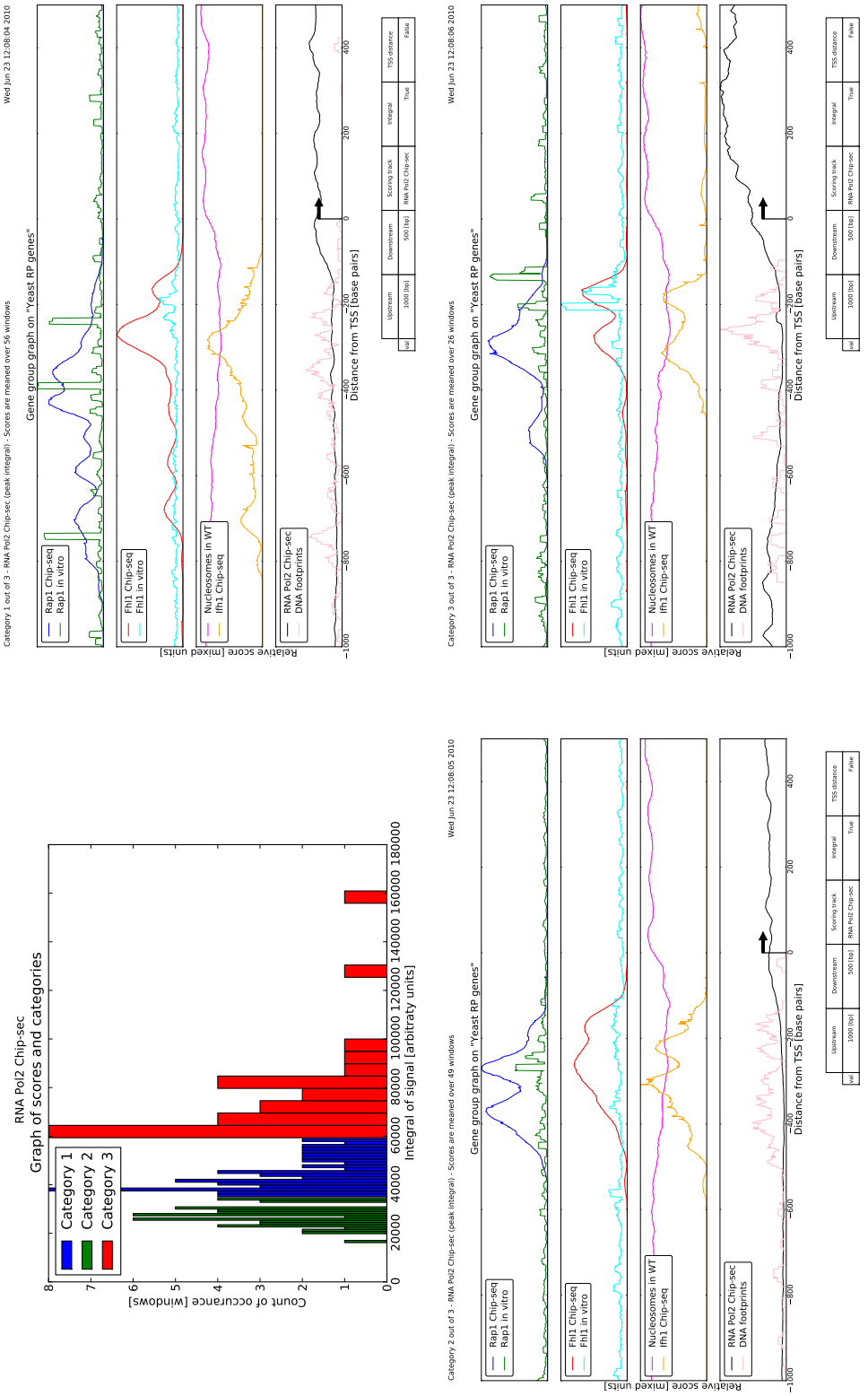

Figure 42: The 131 windows of interest are clustered into 3 categories according to the intensity of the PolII signal. The strength of the Rap1, Fhl1 and Ifhl1 signals don't seem to noticeably affect the level of transcription. However, the third category, where transcription is highest, seems to indicate that when the three transcription factors are bound at precisely -180 or -320 base pairs from the TSS, the polymerase activity is upregulated. Indeed, most of the windows where any of the transpiration factors are bound further upstream are clustered in the first or second category where the PolII signal is weakest.

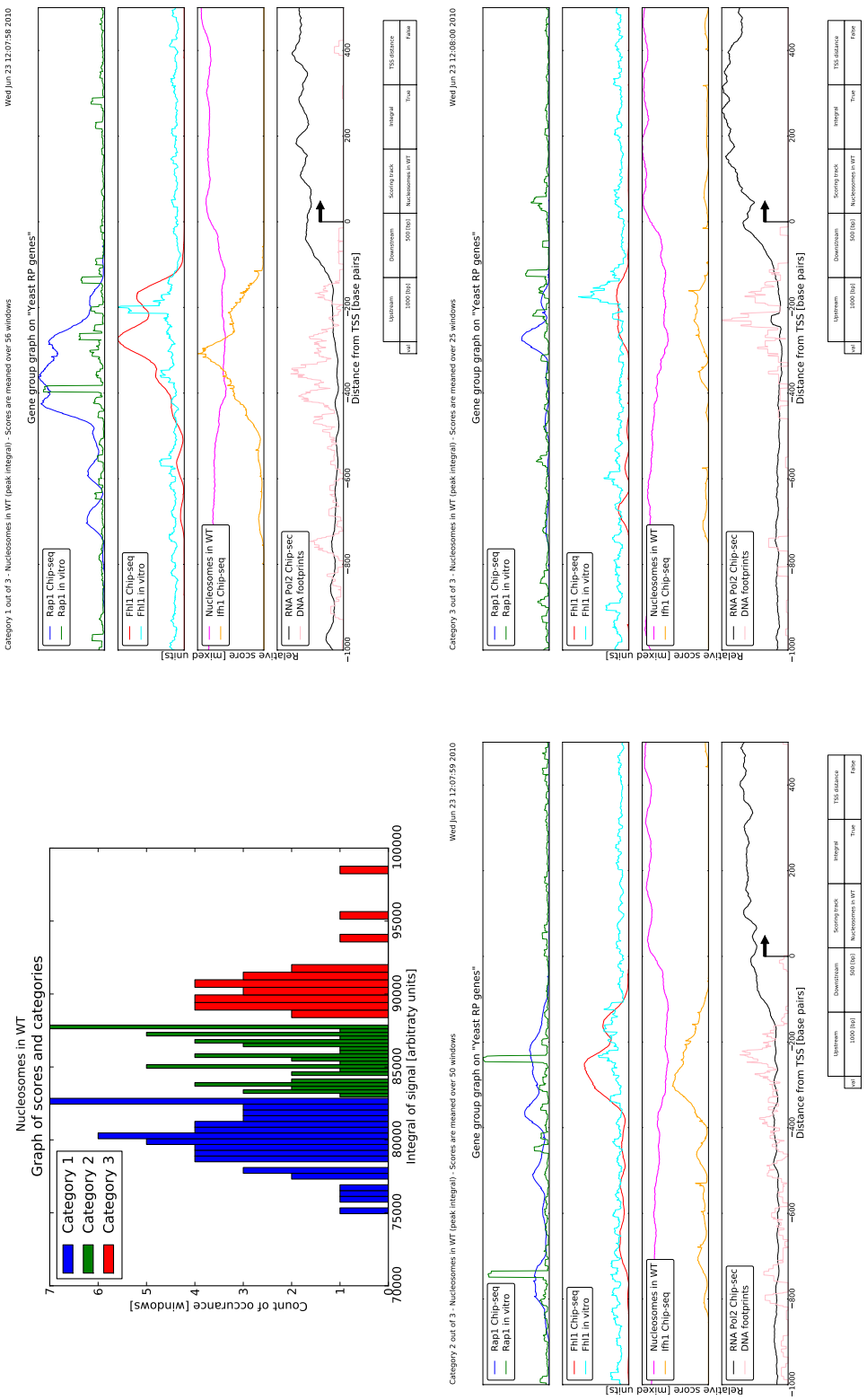

Figure 43: The 131 windows of interest are clustered into 3 categories according to the intensity of the nucleosome signal. As expected, the nucleosome enrichment data strongly predicts the level of binding of the three transpiration factors. The first category where the integral of nucleosome signal is lowest indicates a looser packing of DNA, and thus induces higher binding of Rap1, Fhl1 and Ifh1.

References

[1] *Apple Human Interface Guidelines: The Apple Desktop Interface*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1993.

[2] Nat meth; micrococcal nuclease-southern blot assay. 2(9):719–720, 2005.

[3] Hdf5 is a data model, library, and file format for storing and managing data. http://www.hdfgroup.org/HDF5/, 2009.

[4] IPython's multiengine interface. http://ipython.scipy.org/doc/manual/html/parallel/parallel_multiengine.html, 2010.

[5] SGD project - Saccharomyces Genome Database. http://www.yeastgenome.org/, 2010.

[6] Wikipedia, the free encyclopedia. Jul 2010.

[7] Francesc Alted, Ivan Vilata, et al. PyTables: Hierarchical datasets in Python. http://www.pytables.org/, 2002–.

[8] Apple. Open source - WebKit. http://developer.apple.com/opensource/internet/webkit.html, 2010.

[9] David Ascher, Paul F. Dubois, Konrad Hinsen, James Hugunin, and Travis Oliphant. *Numerical Python*. Lawrence Livermore National Laboratory, Livermore, CA, ucrl-ma-128569 edition, 1999.

[10] Gwenael Badis, Esther T Chan, Harm van Bakel, Lourdes Pena-Castillo, Desiree Tillo, Kyle Tsui, Clayton D Carlson, Andrea J Gossett, Michael J Hasinoff, Christopher L Warren, Marinella Gebbia, Shaheynoor Talukder, Ally Yang, Sanie Mnaimneh, Dimitri Terterov, David Coburn, Ai Li Yeo, Zhen Xuan Yeo, Neil D Clarke, Jason D Lieb, Aseem Z Ansari, Corey Nislow, and Timothy R Hughes. A library of yeast transcription factor motifs reveals a widespread function for rsc3 in targeting nucleosome exclusion at promoters. *Mol Cell*, 32(6):878–87, Dec 2008.

[11] Artem Barski and Keji Zhao. Genomic location analysis by chip-seq. *J Cell Biochem*, 107(1):11–8, May 2009.

[12] Shay Ben-Aroya, Candice Coombes, Teresa Kwok, Kathryn A O'Donnell, Jef D Boeke, and Philip Hieter. Toward a comprehensive temperature-sensitive mutant repository of the essential genes of saccharomyces cerevisiae. *Mol Cell*, 30(2):248–58, Apr 2008.

[13] Daniel Blankenberg, Gregory Von Kuster, Nathaniel Coraor, Guruprasad Ananda, Ross Lazarus, Mary Mangan, Anton Nekrutenko, and James Taylor. Galaxy: a web-based genome analysis tool for experimentalists. *Curr Protoc Mol Biol*, Chapter 19:Unit 19.10.1–21, Jan 2010.

[14] Egon Börger and Wolfram Schulte. Defining the java virtual machine as platform for provably correct java compilation. pages 17–35, 1998.

[15] Reinhard Engels. Argo Genome Browser website. http://www.broadinstitute.org/annotation/argo/, 2010.

[16] Reinhard Engels, Tamara Yu, Chris Burge, Jill P Mesirov, David DeCaprio, and James E Galagan. Combo: a whole genome comparative browser. *Bioinformatics*, 22(14):1782–3, Jul 2006.

[17] Rhead et al. The UCSC genome browser database: update 2010. *Nucleic Acids Res*, 38(Database issue):D613–9, Jan 2010.

[18] Free Software Foundation. GNU Lesser General Public License. http://www.gnu.org/licenses/lgpl.html.

[19] Vicente Galiano, Héctor Migallón, Violeta Migallón, and Jose Penadés. Pypnetcdf: A high level framework for parallel access to netcdf files. *Advances in Engineering Software*, July 2009.

[20] genoviz. Creating a quickload directory. http://sourceforge.net/apps/trac/genoviz/wiki/Creating%20a%20QuickLoad%20Directory.

[21] Belinda Giardine, Cathy Riemer, Ross C Hardison, Richard Burhans, Laura Elnitski, Prachi Shah, Yi Zhang, Daniel Blankenberg, Istvan Albert, James Taylor, Webb Miller, W James Kent, and Anton Nekrutenko. Galaxy: a platform for interactive large-scale genome analysis. *Genome Res*, 15(10):1451–5, Oct 2005.

[22] Jay R Hesselberth, Xiaoyu Chen, Zhihong Zhang, Peter J Sabo, Richard Sandstrom, Alex P Reynolds, Robert E Thurman, Shane Neph, Michael S Kuehn, William S Noble, Stanley Fields, and John A Stamatoyannopoulos. Global mapping of protein-dna interactions in vivo by digital genomic footprinting. *Nat Methods*, 6(4):283–9, Apr 2009.

[23] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[24] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.

[25] W James Kent. Blat–the blast-like alignment tool. *Genome Res*, 12(4):656–64, Apr 2002.

[26] Philippe Lefrançois, Ghia M Euskirchen, Raymond K Auerbach, Joel Rozowsky, Theodore Gibson, Christopher M Yellman, Mark Gerstein, and Michael Snyder. Efficient yeast chip-seq using multiplex short-read dna sequencing. *BMC Genomics*, 10:37, 2009.

[27] Robin Dowell Lincoln D. Stein, Sean Eddy. Distributed sequence annotation system (DAS). http://www.biodas.org/documents/spec-1.53.html, March 2002.

[28] Sebastian J Maerkl and Stephen R Quake. A systems approach to measuring the binding energy landscapes of transcription factors. *Science*, 315(5809):233–7, Jan 2007.

[29] John W Nicol, Gregg A Helt, Steven G Blanchard, Jr, Archana Raja, and Ann E Loraine. The integrated genome browser: free software for distribution and exploration of genome-scale datasets. *Bioinformatics*, 25(20):2730–1, Oct 2009.

[30] C. Pilato, Ben Collins-Sussman, and Brian Fitzpatrick. *Version Control with Subversion*. O'Reilly Media, Inc., 2008.

[31] Aaron R Quinlan and Ira M Hall. Bedtools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, 26(6):841–2, Mar 2010.

[32] Joel E Richardson. fjoin: simple and efficient computation of feature overlaps. *J Comput Biol*, 13(8):1457–64, Oct 2006.

[33] Michael Schindler. Szip is a freeware portable general purpose lossless compression program. http://www.compressconsult.com/szip/, 2010.

[34] Mitchell E Skinner, Andrew V Uzilov, Lincoln D Stein, Christopher J Mungall, and Ian H Holmes. Jbrowse: a next-generation genome browser. *Genome Res*, 19(9):1630–8, Sep 2009.

[35] James Taylor. bx-python. http://bitbucket.org/james_taylor/bx-python/wiki/Home, 2010.

[36] James Taylor, Svitlana Tyekucheva, David C King, Ross C Hardison, Webb Miller, and Francesca Chiaromonte. Esperr: learning strong and weak signals in genomic sequence alignments to identify functional elements. *Genome Res*, 16(12):1596–604, Dec 2006.

[37] Anton Valouev, David S Johnson, Andreas Sundquist, Catherine Medina, Elizabeth Anton, Serafim Batzoglou, Richard M Myers, and Arend Sidow. Genome-wide analysis of transcription factor binding sites based on chip-seq data. *Nat Methods*, 5(9):829–34, Sep 2008.

[38] G. van Rossum et al. Python Language Website. http://www.python.org/, 2010.

[39] Joseph T Wade, Daniel B Hall, and Kevin Struhl. The transcription factor ifh1 is a key regulator of yeast ribosomal protein genes. *Nature*, 432(7020):1054–8, Dec 2004.

[40] J R Warner. The economics of ribosome biosynthesis in yeast. *Trends Biochem Sci*, 24(11):437–40, Nov 1999.

[41] Yong Zhang, Tao Liu, Clifford A Meyer, Jérôme Eeckhoute, David S Johnson, Bradley E Bernstein, Chad Nussbaum, Richard M Myers, Myles Brown, Wei Li, and X Shirley Liu. Model-based analysis of chip-seq (macs). *Genome Biol*, 9(9):R137, 2008.

A Glossary

Standard abbreviations found in the biologist's lexicon as well as the computer scientist jargon will be used in this report. A non-exhaustive list of this vocabulary is provided here.

A.1 Biological

TF: Transcription factor.
RP: Ribosomal proteins.
BP: Base pair.
GO: Gene ontology.
TSS: Transcription start site (a location on a chromosome).
TTS: Transcriptional termination site (a location on a chromosome).
UTR: Untranslated region (located on an mRNA).
ORF: Open reading frame (potentially encodes a protein).
WT: Wild-type (non-mutated strain).
KO: Knock-out (muted or special strain, one or several genes silenced).
S. cer: Saccharomyces Cerevisiae (standard budding yeast model organism).
MNase: Micrococcal nuclease (special DNA digestion enzyme).

A.2 Proteins

Rap1: Ras-proximate (repressor-activator) one.
Fhl1: Fork-head like one.
Ifh1: Interacts with fork-head like one.
Hom1: High mobility group/box one.

A.2.1 Informatics

Track: One genomic data set organized in a linear fashion covering one or several chromosomes.
I/O: Input and output.
SVN: Subversion (revision control system).
JVM: Java Virtual Machine.
DAS: Distributed Annotation System.
MD5: Message Digest algorithm, version 5 (cryptographic hash function).
UCSC: University of Santa Cruz's genome browser.
AGB: Argo genome browser (Broad Institute).
IGB: Integrated genome browser (Bioviz).
GDV: Genomic data viewer (BBCF's project).
SGD: Saccharomyces Genome Database.
PWM: Position weight matrix.

A.3 File Formats

BED: RegionMiner Genomic Regions File (Genomatix Software GmbH).
GFF: General Feature Format.
WIG: Wiggle format.
FASTA: Nucleotide sequences (a.k.a. Pearson format).

HDF5: Hierarchical data format, version 5.
INI: *De facto* standard for configuration files.
PDF: Portable document format.
PNG: Portable network graphics.

A.4 Other

BBCF: Bioinformatics and Biostatistics Core Facility (PTBB in French).

B Code

In this appendix section, a few snippets of relevant python code are provided.

B.1 Base coverage

This function takes a list with n rows and 2 columns. Column one is the feature start, column two is the feature end. It returns the base coverage calculated in C.

```
def get_base_coverage_in_c(f):
    '''This will contain the result'''
    res = [0]

    '''Declare C++ function'''
    code = """
    int dist = 0;
    int me = -1;

    for (int i=0; i<f.length(); i++) {
        if (int(f[i][1]) <= me) {}
        else {
            if (int(f[i][0]) < me) {
                dist += int(f[i][1]) - me;
                me = int(f[i][1]);
            } else {
                dist += int(f[i][1]) - int(f[i][0]);
                me = int(f[i][1]);
            }
        }
    }

    res[0] = dist;
    """

    '''Execute it'''
    err = weave.inline(code, ['f', 'res'], type_converters=weave.converters.
        blitz)

    '''Return results'''
    return res[0]
```

B.2 Pieces of overlap

Takes two tracks and computes the overlap by pieces using a modified version of fjoin.

```
def make_overlap_pieces(track1, track2, chr):
    def go():
        """fjoin's main loop. On each iteration, advance
        the lagging stream, and scan the opposite window.
        Returns the overlapping features."""
        Wx = []
        Wy = []
        X = SentinelIterator(track1)
        Y = SentinelIterator(track2)
        x = X.next()
        y = Y.next()
        while x[0] is not sys.maxint or y[0] is not sys.maxint:
            if x[0] < y[0]:
                scan(x, Wx, y, Wy, 0)
                x = X.next()
            else:
                scan(y, Wy, x, Wx, 1)
                y = Y.next()
    def scan(f, Wf, g, Wg, lastPick):
        """Scans window Wg for features overlapping f.
        May remove features from Wg. May add f to Wf."""
        g_index = 0
        while g_index < len(Wg):
            g_current = Wg[g_index]
            if leftOf(g_current, f):
                Wg.pop(g_index)
            else:
                g_index = g_index + 1
                if overlaps(g_current, f):
                    output.append([chr, max(f[0],g_current[0]), min(f[1],
                        g_current[1]), f[2] + " with " + g_current[2], 0.0, "
                        ."])
        if not leftOf(f, g):
            Wf.append(f)

    def leftOf(a, b):
        """Returns true iff a is left of position b"""
        return (a[1] < b[0])

    def overlaps(a, b):
        """If a and b overlap returns. Otherwise, returns None."""
        return min(a[1],b[1]) - max(a[0],b[0]) > 0

    class SentinelIterator:
        def __init__(self, obj):
            self.custom_iterator=obj.__iter__()
        def next(self):
            try:
                return self.custom_iterator.next()
            except StopIteration:
```

```
49                  return [sys.maxint, sys.maxint]
50
51      '''This will contain the result'''
52      output = []
53
54      '''Call fjoin function'''
55      go()
56
57      '''Return results'''
58      return output
```

B.3 Overlap

Takes two tracks and computes the uncommutable overlap.

```
1   def make_overlap(track1, track2, chr):
2       '''This will contain the result'''
3       output = []
4
5       '''Prepare variables'''
6       continue_loop = True
7       X = SentinelIterator(track1)
8       Y = SentinelIterator(track2)
9
10      '''Read first features'''
11      y = Y.next()
12      if y == [sys.maxint, sys.maxint]: continue_loop = False
13      x = X.next()
14      if x == [sys.maxint, sys.maxint]: continue_loop = False
15
16      '''Main loop'''
17      while continue_loop:
18          open_window = y[0]
19          close_window = y[1]
20
21          '''Extend the y window as long as possible'''
22          while True:
23              if y == [sys.maxint, sys.maxint]: break
24              y = Y.next()
25              if y[0] > close_window: break
26              if y[1] > close_window: close_window = y[1]
27
28          '''Read features from x until overshooting the y window'''
29          while True:
30              if x[0] >= close_window: break
31              if x[1] > open_window: output.append([chr, x[0], x[1], x[2], x
                    [3], x[4]])
32              x = X.next()
33              if x == [sys.maxint, sys.maxint]:
34                  continue_loop = False
35                  break
36
37      '''Return results'''
38      return output
```