UiO : Department of Informatics
University of Oslo

Tools for Genome-wide Analysis of Genomic Divergence

Torkil Vederhus
Master's Thesis, Spring 2013

Abstract

The recent revolution in genomic sequencing has created new opportunities for exploring the connection between genomic variation and biological traits. By sequencing multiple individual genomes within a species, it is possible to identify genomic regions of divergence between groups of individuals sharing particular phenotypic traits. Such a strategy have in the literature been successfully applied for studies of parallel evolution, but none of these earlier studies have made the underlying methodology or tools readily accessible. It is therefore difficult to reproduce their results or to reuse the methodology for new investigations.

I here present an open tool for doing such analyses between two groups of genomic sequences. One method calculates a cluster separation score based on a two-dimensional scaling of the pairwise differences between individuals of the population. The other method uses the Fisher's exact test score for each single-nucleotide polymorphism found. The tools reproduce earlier published results on parallel evolution in freshwater three-spine sticklebacks and a long-term evolution experiment with *Drosophila*. The cluster separation scorer gives slightly more accurate results, but the right choice of parameters is of importance in both cases.

Both methods are implemented as Galaxy tools in the Genomic Hyperbrowser web server. In theory, the tools allows anyone to perform analyses identifying connections between genomic sequence and phenotypic traits based on sequencing data. However, the complexity of the methodology and the non-uniformity of formats used to represent the relevant genomic data is a challenge in practice. Other issues related to usability includes the amount of knowledge of the underlying methods required, as well as challenges with post-processing, including visualization, of the genome-wide results.

Acknowledgements

I would like to thank my supervisor **Geir Kjetil Sandve** for interesting discussions. He is truly an expert in nudging master's students into performing their best. **Bastiaan Star and Lex Nederbragt** have proven very helpful in understanding the biological implications of my work. **Kai and Sveinung** have both proven to be excellent sources for support when implementing in HyperBrowser. Without the discussions, feedback and socializing with fellow master's students **Tobias W., Runar, Knut and Kristoffer** this thesis would be much harder to finish. I'd also like to thank **Kine Veronica Lund** for being a tough competition in the race for being the first student on the floor in the morning, and for taking the time to test the tools I have developed.

On a more personal level, I would like to thank dear **Grete** for her ever-lasting patience and support. **John, Fredrik, Tobias L.** all deserve thanks for great friendship and constructive proof-reading. Special props to **Steffen and Ole Ivar** for constructive criticism as well. I would also like to thank **Andreas and Tone** for giving me the possibility to air loose thoughts walking around the city.

This thesis marks the end of my period as a student at the University of Oslo. This period would not have been any fun without all the great people I have met through **Realistforeningen**, **Det Norske Studentersamfund**, **Realistlista** and, of course, the broad student movement. The challenges of improving the university and student life have proven more inspiring and challenging than anything else. **Sunniva and Tobias L.** deserve a special mention for the trust and guts showed when running for rector together. But most of all, a hearty thanks goes to all the people who have joined arms in fighting the good fight, by lighting walkpaths, hugging professors and so much more.

Torkil Vederhus
University of Oslo
May, 2013

Contents

1 Introduction

Identifying regions of genomic divergence across populations has proven useful for identifying the connections between natural selection and DNA [17]. A rapid decrease in cost of performing variant calling on individuals within a species has created greater possibilities for identifying such regions [27]. By dividing individuals into two groups according to a specific biological trait, measuring the consistent genomic divergence can identify genomic regions with probable links to the specific trait [17]. Three notable approaches measuring such differences on regions of the genome have been published in previous studies:

- A *self-organizing map* combined with a hidden Markov model for modeling evolution [17].
- Computing *cluster separation scores* based on multi-dimensional scaling of pairwise differences within a region [17].
- Computing a quantile of the *Fisher's exact test* score for each SNP-location within a region [5].

These analyses have shown some promising results. To my knowledge, none of the analytical tools performing such analyses previously have been made available for reproducing the results shown or for further use in new studies. The goal of this project has therefore been to implement and evaluate tools for doing such analyses within the Genomic HyperBrowser framework [26].

1.1 Research questions

- How does the different algorithms for identifying genomic divergent regions compare?
- What considerations and strategies are important to have in mind when implementing complex tools in HyperBrowser?
- Is it possible to reproduce the previously published results using tools implemented in HyperBrowser?

- Is it possible for biologists and other non-developers to perform analyses identifying genomic divergence through tools implemented in HyperBrowser?

1.2 Thesis outline

First, I give an overview of the background of my thesis in Chapter 2. In Chapter 3, I present the different methods in detail for comparison and detail the generic features of tools performing cluster separation scoring and Fisher's exact test scoring. In Chapter 4 I discuss and present the general strategy for implementing tools in HyperBrowser. Chapter 5 details The HyperBrowser specific parts of implementation, including configuring the graphical user interface, creating a data structure for sliding windows of genomic data within HyperBrowser and converting file formats. The results from using these tools on data from three-spine sticklebacks and fruit fly populations are presented in Chapter 6. Chapter 7 discusses the results, as well as the accuracy, reproducibility and usability of the tools implemented. Some areas for future work are proposed in Chapter 8.

2 Background

Bioinformatics is an interdisciplinary research area concerned with developing and improving methods related to biological data. This includes organizing, retrieving, storing and analyzing such data. The field was pioneered in the 1980s and has grown to become an important part of biological research as well as medicine. The mapping of organisms' hereditary information, the DNA, has provided scientists with an ocean of data to analyze and organize. This chapter gives a brief overview of relevant parts of bioinformatics for my project.

2.1 What is DNA?

All the hereditary information of an organism is stored in the genome. This information is encoded in deoxyribonucleic acid (DNA) [33, Chapter 11]. These DNA molecules are read by ribonucleic acid (RNA) molecules which in turn uses the information to create features of the organisms, or **traits**. DNA is thus essential for all observed forms of life and integral for our understanding of how organisms are created and developed. A section of DNA sequence in the genome pertaining to a certain trait is called a **gene**.

The DNA molecule is made up of a sequence of nucleotides [33, Chapter 11]. There are only four possible nucleotides: adenine, guanine, cytosine and thymine. These are commonly represented by the letters A, G, C and C respectively. The nucleotides appear in pairs, called base pairs. Since each nucleotide has a different shape only two types of exact pairs are possible, the A-T pair and the G-C pair.

When creating new cells DNA is replicated by splitting the base pairs in two and attaching new nucleotides to each half [33, Chapter 14]. The world is not perfect, and neither is DNA-replication; mismatched pairs like A-G or even A-A are also observed from time to time. Pairs can be switched, inserted or deleted during replication. Such mutations of the DNA are instrumental in the process of evolution, since mutations that lead to better adaption survives and are reproduced.

How do we obtain DNA sequences?

Genome sequencing is the determination of the DNA sequence of an organism. Researchers have sequenced genomes since 1979 [29], but the first entire genome was first assembled in 2001 [20]. Since then several entire genomes of humans and other species have been published as sequencing technology has improved [21]. There are several different methods and technologies available for genome sequencing, varying in length and accuracy. The output from genome sequencers is a series of sequence **reads** of various lengths [20]. These reads are snippets of continuous DNA-sequence obtained from the biological material, in essence small pieces of the full genome sequence puzzle. Due to the possibility of wrong reads, sequencers try to obtain reads that cover each position of the genome multiple times. The average **read depth** is thus used as a measure for the possible quality of the assembled genome [4]. Special assembly programs solve the puzzle by identifying and combining overlapping reads.

To make it easier to compare and compile new genomic sequences several **reference genomes** have been made for different organisms, most of which are made available through the National Center for Biotechnology Information (NCBI) RefSeq database [24]. These genomes are often assembled to be the "least common multiple" of the species, by combining DNA sequences from several individuals within the species. New versions of reference genomes may be assembled to get rid of gaps and misrepresentations; the human reference genome version maintained by the Genome Reference Consortium [41] is currently in its 37th version [42]. Reference genomes may have a read coverage of $90x$, but due to the cost of doing such thorough sequencing, a coverage of between $4 - 20x$ is more common for more specific applications. Such sequencing focuses on calling **variants**, identifying locations where the sequence differs from the reference genome [2]. A location where only a single basepair is different is called a **single-nucleotide polymorphism (SNP)**. In most cases a SNP is bi-allelic, since there are only two possible exactly matching basepairs. The most common polymorphism is called the **major allele** while the less common is the **minor allele** [2]. Some SNPs have more than two alleles, due to mismatched nucleotide basepairs. Such SNPs are called **poly-allelic** SNPs.

The relative low cost of variant calling has given researchers the possibility to sequence large amount of individuals for research. One famous example of this is the 1000 genomes project, which found genetic variation from 1092 human genomes [1]. At the University of Oslo, the aqua genome project have a goal of sequencing at least one thousand cod and salmon genomes [38].

A note on the accuracy of SNP-calling

SNP-data is most often obtained through short-read sequencing which is then mapped to the reference genome for the species. There are several technologies for doing this, often assembled in pipelines. O'Rawe et al. [22] compared several SNP-calling technologies for concordance and found that across five Illumina pipelines 57.4% variants were found by all. Meanwhile, the three tested pipelines identifying insertions and deletions only had an concordance of 26.8%. The high probability of errors in the SNP data requires extra attention when calculating regions of genomic divergence. Any analysis done based on variants is dependent on the quality of the variant calling procedures.

2.2 Why compare genomes?

Comparative genomics has proven useful for figuring out the evolutionary relationships between organisms. By comparing DNA from different species one can find out how much DNA the species share. Thus we can find their relative placing in the evolutionary history of species. Such comparisons can be challenging. The relative ordering of genes and chromosome differs from species to species, which makes aligning reference genomes to each other difficult. Mappings between reference genomes, describing positions that are homogeneous in genomes, are therefore often published along with reference genomes [27].

The DNA sequence of an individual is the blueprint or **genotype** for the individual. The study of relationships between the genotype and the actual features of the individual, the gene expression or phenotype, is a huge field spanning biology, medicine and indeed bioinformatics. The common way of establishing such connections is through the microscope, identifying molecular reactions between RNA and DNA [33, Chapter 20]. The human genome is estimated to have between 20,000 and 25,000 genes [7], making such analysis cumbersome for whole genome analysis.

The possibility of a more quantitative approach has arisen with the rapid decrease in cost of sequencing. By sequencing multiple individuals one can do genome-wide analysis that identifies genomic divergence across groups [5][17]. Simplified, if you take a large group of DNA sequence data from blond-haired individuals and compare with a group of brown-haired individuals it should be possible to identify suspected regions of the genome affecting the hair color. The researcher can then study these regions for genes that affect hair color, instead of searching the whole genome. Essentially the approach identifies regions of **parallel evolution**, regions where the same mutations can be observed across a diverse

group of individuals [30].

Two interesting studies have been published utilizing such methodologies: one comparing marine and freshwater three-spine stickleback fish (*Gasterosteus aculeatus* [17], and one comparing three populations of common fruit flies (*Drosophila Melanogaster*) [5].

Marine-freshwater divergence in three-spine sticklebacks

Jones et al. [17] first assembled a reference genome for the three-spine sticklebacks (*Gasterosteus aculeatus*). Then they assembled SNP data for twenty additional individuals from different stickleback populations across the world, both marine and freshwater. Through two different methods they identified locations in the stickleback genome with large divergence between the marine and freshwater groups, and studied them more closely. These locations included several genes with differences in expression between freshwater and marine sticklebacks. The methodology used for finding these regions can be summarized in these steps:

1. Assemble reference genome, with a read coverage of $9.0\times$, and annotate it.
2. Create "read library" with sequence read for 20 other genomes of both marine and freshwater sticklebacks, with an average read coverage of $2.3\times$.
3. Align these sequences to the reference genome.
4. Validate SNPs found by only keeping SNPs at a position where at least four different reads contained a different allele than the majority
5. Calculate the divergence of genomic regions, identifying parallel evolution within the marine and freshwater groups
6. Identify the most significantly diverting regions and combine the results
7. Do further analysis on selected regions

Bioinformatics tools and machines for performing assembly and alignment of sequences are commonly available today. This is also the case for validating SNPs. This is not the case for calculating genomic divergence, so these algorithms are of special interest. The article uses two different algorithms to identify regions of genomic divergence, as illustrated by figure 2.1:

An evolutionary approach, using hidden Markov models and self-organizing maps to try to model the possible evolutionary relationships between all individuals in phylogenetic trees. Several such "family" trees were generated for each

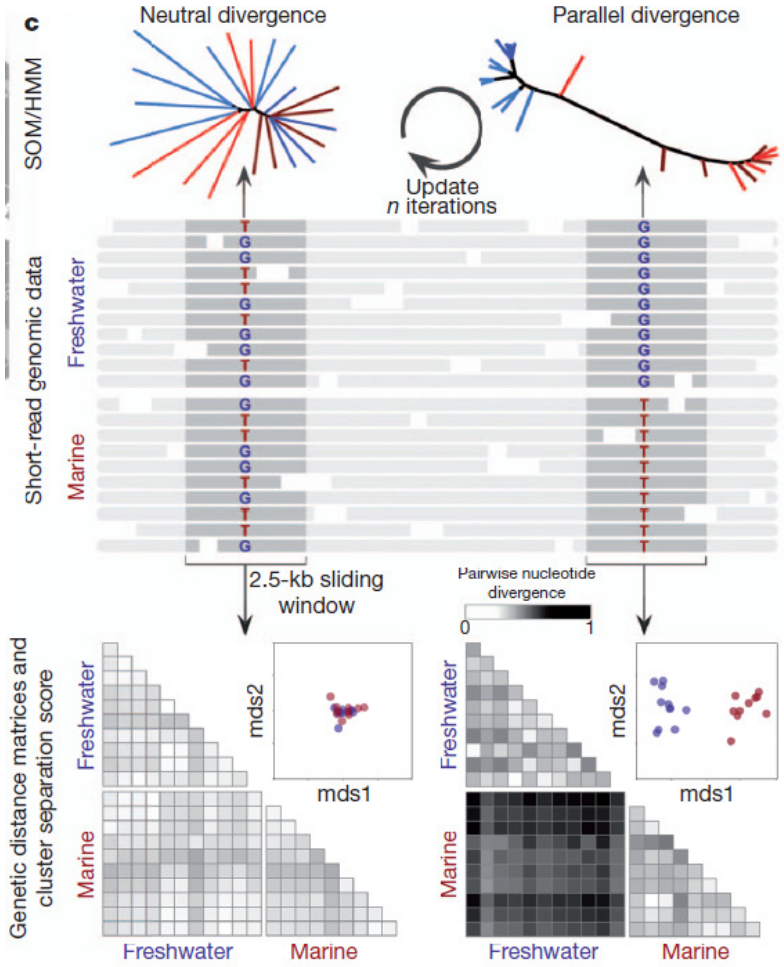

Figure 2.1: Visualization by Jones et al. [17] of methods used for identifying regions of marine-freshwater divergence in three-spine sticklebacks. Two typical cases for a window of the genome is presented, one with neutral divergence and one with parallel divergence across the groups. At the top a tree created using SOM/HMM-methods is visualized. In the middle the actual sequences are visualized for each individual. At the bottom the cluster separation score pairwise difference matrix is visualized.

region of the genome. The authors then studied the regions where trees with clear common separation between the freshwater and marine individuals occur. These regions were then used for further inspection.

A pure arithmetical based approach counting the pair-wise differences between each individual in pre-set sliding windows of the genome. Based on the distance matrix for all individuals, the authors placed the individuals in a two-dimensional space using multi-dimensional scaling, and then calculated a score expressing the between-group distance. The significance of the score was found by calculating the scores for all other possible divisions of all individuals in two groups of the same sizes.

The regions identified by the two approaches were then analyzed. Genes laying within or adjacent to the sets of regions were listed, showing that several genes with diverging marine-freshwater expression were identified. Both approaches identified the location of the *EDA* gene, which affects armor evolution in freshwater sticklebacks. The strictest set of regions, the intersection of the results from the two approaches, was then used as a basis for analysis on a new independent set of sticklebacks. 91% of the new genomic data was in concordance with the predicted marine-freshwater divergence [17].

Analysis of Drosophila populations from long-term evolution experiments

Burke et al. [5] have performed a study of genomic differences between three common fruit fly (*Drosophila Melanogaster*) populations. Two populations had experienced over 600 generations of laboratory selection for accelerated development, while the other was a control population with shared ancestors. The methodology used can be summarized in these steps:

1. Obtain pooled alleles (SNPs) for two populations under direct selection and a control population
2. Keep positions at which there were two observed alleles
3. Calculate Fisher's exact test scores on each SNP-location
4. Calculate the 5% upper quantile of all negative log_{10} values of scores within a 100kb sliding window with 2kb step.
5. Identify windows with scores larger than the upper 0.1% quantile under a normal distribution of scores.

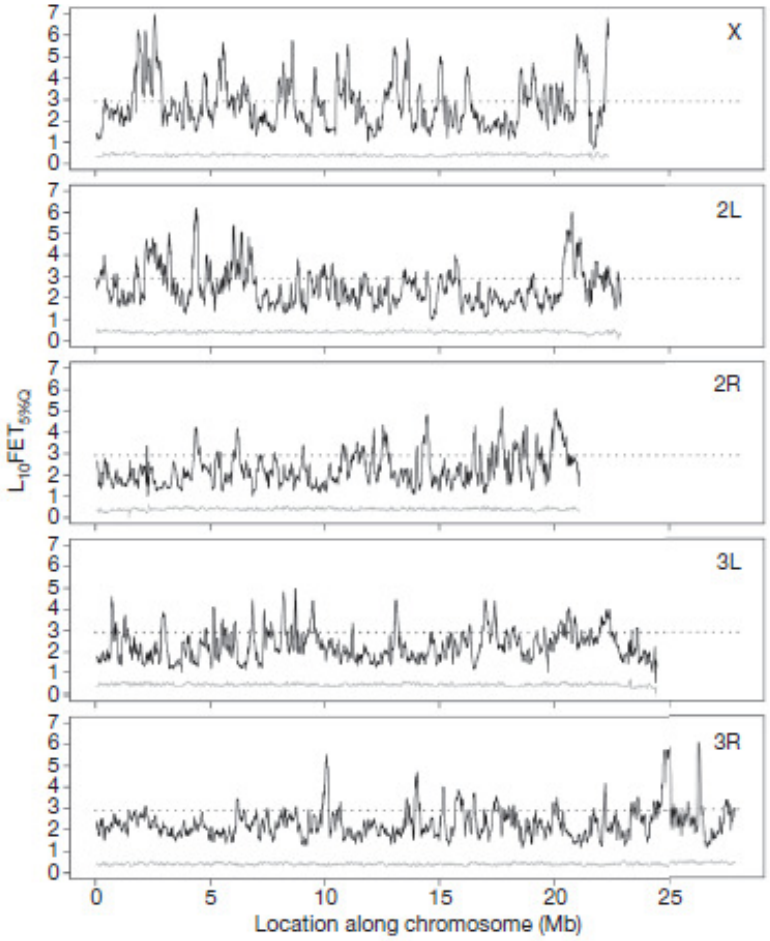

Figure 2.2: Fisher's exact test upper 5% quantile for windows of the genome as calculated and visualized by Burke et al. [5]. The gray line is the limit for significant windows based on a measure of genome-wide noise.

Fisher's exact test is a statistical significance test that calculates the probability of a two-by-two contingency table to indicate whether the two classifications are associated [8]. In this case the classifications are the population membership for a SNP and whether the specific SNP is a minor or major allele.

The result is a few spots in the drosophila genome where the accelerated population and the control population differs significantly, as illustrated by figure 2.2. These spots are then again studied more closely by checking which genes they correspond to. The study did not find any unconditionally advantageous alleles in the population of accelerated developed fruit flies and the authors suggest that selection does not change the genetic variation in the genome.

2.3 Reproducibility in bioinformatics

One basic principle of the scientific method is to ensure reproducibility of results. Reproducible results ensures the validity of the results and eliminates the need to trust the authors of the studies. Some areas of study are inherently expensive or hard to reproduce in the field of bioinformatics. Genomic sequencing is one such part, as the technology is expensive and you would need the same biological samples for reproducing results. It is common to make sequenced data available for the public so that any analysis based on genomic sequences is easier to reproduce. However, this is not always the case, a meta-study conducted by Ioannidis et al. [15] showed that not even making data available can be called "common practice".

Making the data available is not enough to ensure reproducibility of experiments and analyses within bioinformatics. The tools created for doing analysis, the executable programs, is needed as well. Ideally, the source code of programs used should also be made available when publishing results in bioinformatics. This would give anyone the possibility of scrutinizing the code for errors and possible improvements. The earlier detailed studies on three-spine stickleback fish and fruit flies are based on analyses and algorithms that are novel in bioinformatics. The implementation of these algorithms is an important source for errors, and therefore transparency and reproducibility is especially important. The possibility to reproduce each study is therefore discussed in detail below.

Reproducibility of stickleback study: The authors of the stickleback article have made a lot of data available through its own version of the UCSC genome browser [44]. The SNP-data and reference genome are publicly available, so it should be possible to reproduce the regions of divergence found in the article using that data and their method outlined. The authors used some third-party programs which

are referred to by name, but no program packages or code is made available for doing the distance-based calculations. The reader is left to trust that the code is flawless and hence the output shown is correct.

Reproducibility of drosophila study The authors of the drosophila supplied the SNP-data and obtained Fischer exact test-scores on request, they are not openly available for download on the internet. Calculating the Fisher exact test is no complicated task, and the authors mention specific R-commands to perform both the scoring and filtering in their methods material. The sliding window analysis and preprocessing of the SNP-data into a computer readable format is however left to the reader who wants to reproduce the results.

2.4 Galaxy and The Genomic HyperBrowser

Ideally, for better reproducibility, all tools used in published studies should be available on the internet usable by anyone. Combined with the source code this means that anyone can scrutinize the code, suggest improvements, and reuse the tool for new analyses. To handle the large amount of data and resource-intensive computations several web interfaces have been created to for genomic analysis. These handle the large amount of data and resource-intensive computations by using data centers and super-computers in the back-end. The Galaxy project is one such platform, which supplies life scientists with a web interface for running, organizing and storing analyses [10]. In 2012 a total of 401 published papers cited or mentioned Galaxy, and close to 30 000 users were registered on their main server [40]. Integrating a tool in Galaxy ensure that anyone with internet access can reproduce

Several projects use the Galaxy-"engine" and have created more tools on top of that. One such project is the Genomic HyperBrowser developed at the University of Oslo [26]. The Genomic HyperBrowser aims to make it easy for life scientists to use for analysis of this data and testing hypothesizes on open web pages. In addition to the Galaxy features,HyperBrowser provides efficient handling of genomic meta-data and several tools for doing analysis and hypothesis testing on this data. Since HyperBrowser is web-based and provides user interfaces for tools it does not require more than a shallow programming-background from its users. The framework includes an growing amount of genomic sequence and annotation tracks. Together with easy-to-use links to most common bioinformatics databases a lot of data is therefore already available for analysis.

3 Methods

This chapter describes how the problem of identifying regions of genomic divergence across population can be solved. First, I discuss different methods used in earlier works, and do a brief comparison of features. Then I dive deeper into the features of performing cluster separation scoring and Fisher's exact test score. I detail important design crossroads, statistical analysis and significance calculations for both algorithms. Towards the end, I discuss some important methods for post-processing the scores for actual identification of genomic regions.

3.1 Picking methods

Several methods have been used earlier for discovering genomic divergence across populations. In the studies detailed in the background chapter, three methods are presented: a self-organizing map-based hidden Markov model, a distance based cluster separation score and a individual SNP-based Fisher's exact test.

Self-organizing map-based Hidden Markov Model

All individual genomes obtained are a product of evolution. They therefore have some kind of relationship with each other that can be visualized in family-trees. One way to discover genomic divergence between populations is to figure out probable relationship (*phylogenic*) trees between the individuals. This could be done based on regions of the genome where high degrees of variance has been found. Based on these trees it is possible to determine regions where the pre-selected populations diverge from each other. These regions are then candidates for sourcs to biological differences between the populations.

A *hidden Markov model* (HMM) [25] is useful for systems where you have unobserved states and known parameters. For the problem of identifying parallel evolution, we know the biological and genomic traits of our individuals. Factors such as geographical location might also give us some ideas on evolutionary relationships. The Hidden Markov Model is used together with a *self-organizing map* (SOM) [19]. A SOM iteratively organizes the individuals relatively to each other.

Thus we obtain a matrix describing the relative distances between each individual. These distances can then be used to generate trees which in turn will reveal divergence between our groups.

Creating tools using this method requires either deep knowledge of SOM and HMM and ways of implementing them in computer programs, or knowledge about existing data packages performing SOM and HMM. Considering the complexity of the algorithms using existing data packages seems like the most realistic and efficient scenario. Both the hidden Markov model and the self-organizing map requires several fine-tuned parameters that can not be generalized for all studies of genomic divergence. The output itself also needs to be examined in some sense. This can be done by creating scoring algorithms for divergence between individuals of each population in the tree or human inspection.

Jones et al. [17] used SOM/HMM in their stickleback study. Some weaknesses were found. In the SOM/HMM-method all calculations are done on groups of variants/SNPs, and not actual regions of the genome, which makes it more reliant on continuous assemblies of sequences. There is an abundance of patterns to find in genome-wide relationships within a species. The most likely patterns found are not necessarily the right patterns. In the stickleback study, the most interesting trees were identified by the authors and not by any automated measure. This showcases the complexity of create automated measures of the significance of a generated tree. The SOM/HMM-method successfully identified the EDA locus with known marine-freshwater divergent expression. However it was not as successful as the Cluster Separation Scorer in finding other regions of divergence [17].

Both the self-organizing map and the hidden Markov model requires several iterations of complex calculations, where the number of iterations is set by the user. Studies on the computational complexity of self-organizing maps are hard to find. The only source found indicated binomial complexity [37]. For HMM some estimating algorithms have been lowering the complexity to $O(n * 2)$ [6]. A SOM/HMM based tool will in large parts rely on existing program packages. The challenge is then to tie all parts together into a proper pipeline. A developer will find herself using a lot of time diving into code made by others. A more productive solution for creating reproducible analysis based on SOM/HMM might be to make each package available independently in a framework like HyperBrowser. Then a step-by-step guide can be made for doing the analysis. This would however put the cost of getting to know the packages on the user.

Cluster separation scoring of sliding windows

Solely based on the genomic data of several individuals it is possible to score the divergence between two populations in a window of the genome. This is what the cluster separation scoring-algorithm presented in the stickleback study does. The algorithm can gives a detailed score for each window based on any two sets of groups. This score for each window is calculated based on pairwise differences between all individuals. These pairwise differences are used to put all individuals relative to each other in a two-dimensional space using *multi-dimensional scaling* (MDS) [32]. The cluster separation score is the difference between the average between-populations distance and the average within-population distance in this two-dimensional space. A higher score implies a larger genomic divergence between the two populations in the region scored.

Calculating this score is not computationally complex: the algorithm consists of simple arithmetic steps where the most complex part is the multi-dimensional scaling. The distance metric and score-algorithm does not contain random noise, and should be easily reproducible. While the self-organizing map only identifies diverting regions, the distance-based approach also gives a similarity score for each window of the genome and hence more information for further analysis.

The p-value of a cluster separation score is highly correlated with the score itself. However, high scores might be due to high-polymorphism rates in general in the region. Thus, calculating the p-value lets us filter the high scores that are actually caused by genomic divergence between the groups. The p-value can be calculated by computing the CSS for all possible combinations of individuals into two groups of the same size. By dividing the number of scores more extreme than our with the total number of permutations we get the p-value. Based on these p-values one can set a threshold for significance. This is done by computing a false discovery rate based on the number of regions that fall above that significance threshold. These diverting genomic regions more closely for connections to biological traits.

The most complex part of this methodology is the multi-dimensional scaling, which is $O(n^3)$ with n being the size of the data set supplied [28]. This out-weighs the complexity of the other parts of calculations for each window.

Fisher's exact test of sliding window

The Fisher's exact test (FET) is a statistical significance test used for analysis of 2×2 tables showing frequency variation of variables with smaller samples [8]. The test score is useful for examining the association between two kinds of classifications. My goal is to identify association between genome variants and membership in a

population. The necessary data can thus be found by counting the occurrences of the minor allele and the major allele for each SNP-location in the genome in each population.

Fisher proved that the probability of obtaining any such table is given by the hypergeometric distribution

$$\frac{(a+b)!(c+d)!(a+c)!(b+d)!}{a!\ b!\ c!\ d!\ n!},$$

where a, b, c and d refers to the frequencies in each box of the 2×2 table and n is the sum of all frequencies. Fisher's exact test works well on unbalanced and small tables. In cases where there is not enough data for each individual one might have to combine several SNPs for each individual. Categorizing them into groups based on frequency of minor/major allele achieves this.

The aim of this project is to identify regions of divergence, and not single SNP-locations. Therefore we need a way to score windows of the genome based on the FET scores for SNPs within the window. We are interested in regions where we can find several SNPs with high ($-\log_{10}$) scores. One way to measure this based on a group of FET scores is to find the limit for a certain upper quantile of the $-\log_{10}$ of the scores. In the drosophila study, Burke et al. [5] chose to look at the limit for the upper 5% quantile of FET scores in the window. By looking at the regions where this limit exceeds the threshold for what happens one in a thousand times with random noise we can calculate a false discovery rate and determine if we have found possible regions of genomic divergence.

Fisher's exact test is by far the least complex of the three methodologies presented. Since we are dealing with sufficiently small numbers multiplication is $O(1)$, and the factorial function is $O(n)$. Therefore calculating the Fisher's exact test for each window is also $O(n)$.

Comparison and conclusion

The three methods are different in a few aspects as shown in table 3.1. Due to complexity and signs of weaknesses in earlier studies the self-organizing map-methodology was not further investigated or implemented. Both the Fisher exact test and cluster separation look like good candidates for further inspection and implementation.

Feature\Algorithm	**FET**	**CSS**	**SOM/HMM**
Human inspection	No	No	Yes
Input	SNPs	SNPs	SNPs and gene ontology?
Dependencies	None	MDS	SOM/HMM
Complexity	$O(n)$	$O(n^3)$	binomial?

Table 3.1: Features of different algorithms for identifying regions of genomic divergence.

3.2 Features of a cluster separation scorer

This section describes the general features of the cluster separation scoring algorithm. Several design crossroads are presented, such as data filtering and parameters for two-dimensional scaling. Metrics for pairwise comparisons and calculating cluster separation based on points in a two-dimensional space will also be discussed in extra detail. Strategies for calculating significance effectively are also presented. At the end, I present a method for processing scores into regions with high genomic divergence. The code implementing each part of this methodology is made available in Appendix C.1.

Method summary

Based on SNP data from two groups of individuals I want to generate a cluster separation score. The algorithm first requires a pairwise comparison between all individual datasets within the window. Multi-dimensional scaling is then used for creating a two-dimensional map positioning the individuals relative to each other. The relative positioning is the basis for computing the cluster separation score, the average difference of distance between the group of freshwater sticklebacks from marine sticklebacks relative to the within-group distances.

Calculating such a cluster separation score can hence be divided in three parts:

- Pairwise comparing all individuals with each other using a suitable metric
- Using Multi Dimensional Scaling to place the individuals relative to each other in a two dimensional space
- Calculating the difference of the in-group distances vs. the difference between the groups

Major allele	Minor allele	Both alleles observed in reads	No data
3	-3	0	-10000

Table 3.2: Table over data representation for each state of a SNP

Pairwise comparison of all individuals

A metric describing the genomic difference between two individuals as a non-negative number is an essential part of this method. This metric is dependent on the nature of your SNP data. Some sequenced data of populations only supply the pooled frequency of major and minor allele, while in others you get thorough SNP data for each individual separately. The comparison metric to use therefore needs to be chosen either by the user or by pre-processing input files. This underlines the importance of dividing our code into small focused parts, being able to switch out the pairwise comparison metric easily is a deciding factor for usability in new cases.

A comparison metric has only two specifications. All that is needed is to supply a function which takes two one-dimensional NumPy [43] arrays and returns a single number representing the difference between the two arrays. Two such comparison metrics have been developed by me:

- Count differences: The number of positions in the two arrays that have different values (and the values are not None).
- Average of differences: The sum of the absolute values of the differences between the values at each position in the two arrays.

Pairwise nucleotide differences Counting pairwise nucleotide differences is a good comparison metric if you have separate SNP data for each individual. Table 3.2 shows a data representation for different possible values at each SNP-location used by the authors of the stickleback study. [17]

The goal is to calculate the pairwise nucleotide divergence between the two individuals represented by arrays with one of the four values above at each position. A position where one array has -10000 is discarded as it is not possible to know if the two individuals differ at that position. A position where one array has the value -3 and the other has 3 increments the nucleotide divergence score by 1. But what is the case when one of the arrays have value 0? There are three options:

1. Count the difference between 3 and 0 and the difference between -3 and 0 as 1.

2. Count the difference between 3 and 0 and the difference between -3 and 0 as 0.5, or some other value between 1 and 0.

3. Count the differences between 3 and 0, -3 and 0 as discarded, just like -10000.

Option number three must be said to be the most conservative and preferable approach. I briefly ran an analysis with option 1, which yielded radically more noise in our data and therefore gave us less information. The task of our comparison metric is thus as follows: Count the number of positions where the set $a[i], b[i]$ is equal to $(-3, 3)$. Since the input arguments are NumPy arrays I have plenty of efficient vectorized operations to choose from. Subtracting or adding all positions with each other will yield 0 as sum both in the cases where $a[i]$ and $b[i]$ are $(-3, 3)$ and $(0, 0)$. Multiplying the two arrays will however yield -9 for the positions I want, and only for those positions. Counting the frequency of a specific value in an array is done by comparing the array to the value and taking the sum of the resulting array. A vectorized comparison metric is compact and easy to read:

```
def compareCount(a,b):
    product = a * b
    score = numpy.sum(product == -9)
    return score
```

Average of allele frequency differences Some SNP-data is only accessible in formats where the SNPs have been pooled into one value, the frequency of minor allele out of all identified SNPs. The task of the comparison metric is then to give quantified measures of the pairwise difference between two populations, assuming that we have several such populations. Simply computing the average of the absolute value of the difference in frequency at each position yields such a measure. Here the score is larger if the frequencies in the window differ more. This can be calculated through a simple function using vectorization:

```
def compareFreq(trackA, trackB):
    absDifferences = numpy.abs(trackA - trackB)
    avgDiff = numpy.sum(absDifferences) / len(trackA)
    return score
```

Should windows with low assembly quality be filtered?

Some windows of the genome contain gaps in the underlying sequence assembly. Depending on the metric this might cause large variance in scores and more noise

in the results. The option to filter out calculations based on too little data could therefore be handy. To perform such filtering we need information on the assembly of the individual genomes, in addition to the verified SNPs called. This raises the amount of input required to use the tool drastically, which in turn lessens usability.

Another option for avoiding low-quality windows is to demand a certain amount of valid comparisons between individuals. As long as the pairwise comparison metric increases with the amount of valid differences increases this seems unnecessary. The cases with few valid comparisons will be counted as similar, which is the conservative option in this case. The best option for securing high-quality data is to let the filtering happen when accumulating and validating SNPs, before calculating genomic divergence.

Multi Dimensional Scaling

Multidimensional scaling is a way of getting similarity data based on a set of objects. In essence, the technique tries to plot the different objects (in our case, individuals) in a geometric space. The main purpose for this has been to visualize the data, but the geometric data itself can also be used to find groupings as is our case [32].

An existing python implementation in the machine learning package of *scikit*, called *scikit-learn* [23] includes multi-dimensional scaling. The package takes a symmetric matrix as argument and returns coordinates for each individual based on that. The package is used by first creating a MDS-object with parameters for random-state, max iterations and so forth. The coordinates are then obtained calling the method `fit_transform` with the symmetric matrix as argument. The result of the call is a list of coordinates representing the point in a two-dimensional space the individual at index i is.

The most important parameter choice, and the only one which is specifically set in my code, is whether the scaling should be metric or not. Non-metric multi-dimensional scaling only takes into account the order of the similarity data, while metric multi-dimensional scaling also ensures that the relative quantity of similarities is kept. In this case the important information obtained from our pairwise comparisons is undoubtedly the relative sizes of results, not only the order. A simple test with non-metric MDS returned more noise in stickleback chromosome IV, including weaker signals in known areas of genomic divergence.

Calculating cluster separation

Based on a list of coordinates for individuals, we now want to score how much these are clustered in two preset populations. One approach to this problem is finding some kind of center for each cluster and measuring the distance between these. However, this metric does not take into account the compactness of each cluster. The algorithm for computing cluster separation scores presented by Jones et al. [17] does this, and is pretty straight forward.

$$CSS = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} s_{i,j}}{mn} - (m+n)\left(\frac{\sum_{i=1}^{m-1} s_{i,i+1}}{m^2(m-1)} + \frac{\sum_{j=1}^{n-1} s_{j,j+1}}{n^2(n-1)}\right).$$

Here, m and n are the respective sizes of the two groups that we are computing the separation score of, while $s_{i,j}$ denotes the Euclidean distance between two individuals. The first term of the calculation is intuitive, and denotes the average distance between all individuals in both groups. The second term is a weighted combination of the average within-group distance. Thus, if the individuals are clustered in the given groups far from each other we will get a higher score than if the individuals in the groups are randomly spread. Note that not all distances between individuals in each cluster is used for calculating the measure of within-cluster distance.

The algorithm relies heavily on computing the Euclidean distance between two points based on the given coordinates. We are given the coordinates between the different individuals and the indices for group A and group B. Using the indices instead of the data itself helps keeping storage to a minimum, as the same coordinates will be used for scoring differences between several permutations of groups when computing significance.

Computing significance

To get a sense of how significant our separation scores are, we need to compare our score to scores of other possible divisions of the individuals. Exhausting all possible combinations of divisions of individuals into two populations of set sizes can be very computationally expensive. The amount of possible groupings is defined as

$$\frac{n!}{r!(n-r)!}$$

, where n is the total amount of individuals and r is the size of one of the groups. In the stickleback study, with 21 individuals and two populations of 10 and 11, this

means we have 352 716 possible combinations. Future studies will probably have more individuals and more data. When calculating scores for new groups there is no need to redo the pairwise comparisons or the multi-dimensional scaling, since these are population-independent calculations. The p-value is the probability of obtaining a test statistic at least as extreme as the one observed. In our case a higher score is more extreme. By calculating many such scores one can estimate the probability of our score being the most extreme.

Shortening running time by using sequential Monte Carlo

Computing all possible scores gives us the exact p-value. This is possible in cases where the number of individuals is sufficiently low, or when we only have pooled data for populations, as is the case for the drosophila analysis detailed later.

In the case like the stickleback study computing all 352 716 possible scores for all windows of the genome is computationally expensive. A way around this is using a simple *Monte Carlo process* instead for calculating p-value. A Monte Carlo process uses repeated random sampling to obtain numerical results. For our case this means dividing the individuals in two random groupings and calculating the cluster separation score for these groupings. Even though this means that we do not have to calculate all possible scores, getting a p-value of the proper resolution requires a a large number of combinations. A p-value of less than 10^{-5} might be necessary to get a low false discovery rate for the whole genome. This means scoring at least 200 000 combinations of individuals. Calculating p-values based on 200 000 possible combinations for each window is still very computationally expensive. Halton [12] presented a solution to this problem by using sequential Monte Carlo-methods. Since we are only interested in the most significant windows, cases where our Monte Carlo simulations scores often exceed the calculated score for our populations should be dismissed early. This means that we only do 200 000 calculations for cases where our score is actually significant. The whole process can be summarized as follows:

1. Divide the individuals in two random groups with pre-set size.
2. Calculate separation score and compare to our score.
3. Repeat until either X simulations have obtained a score more extreme than ours or Y simulations have been ran.
4. Calculate p-value by dividing the number of scores higher than our score by Y.

This methods lead to a radical decrease in running time for chromosome-wide analysis, and therefore gives the user possibility to raise the resolution of p-values for the most extreme windows.

Filtering windows based on false discovery rate

The false discovery rate is the ratio of discovered windows with a p-value below a certain window that can be expected to be false positives. The number of expected false positives with a certain p-value based on a uniform distribution is simply $p \times n$ where n is the number of windows in the whole genome. We want to keep the false discovery rate below a threshold q. Benjamini and Hochberg [3] showed that the false discovery rate can be controlled by using the *Benjamini-Hochberg procedure*:

1. Sort all p-values in increasing order.
2. Start with the p-value at the end, the largest one, and iteratively check if:

$$P_i \leq \frac{i}{n} q$$

   where i is the index of the p-value, n is the total number of values and q the threshold chosen.

3. If the inequality holds, all p-values up to this point in the list can be declared positive discoveries based on the false discovery rate q.

In addition to identifying significant windows, it is useful to connect nearby windows into larger regions. This can be achieved by keeping track of scores within the pre-defined maximum distance at all times when iterating through the scores. This marks the end of a recipe for identifying regions of genomic divergence based on cluster separation scoring of SNP-data. The output, or "cake" produced by our recipe would be data on a format such as:

```
##gtrack version: 1.0
##track type: segments
##uninterrupted data lines: true
##sorted elements: true
##no overlapping elements: true
###seqid start end
chrVI 12948500 12951000
```

3.3 Features of a Fisher's exact test scorer

Calculating Fisher's exact test (FET) on each SNP-location in a sliding window of the genome is the least complex of the three algorithms presented at the beginning of this chapter. This section describes the recipe for obtaining regions of genomic divergence across groups based on Fisher's exact test scoring SNP-data. Code for each part is supplied in C.2.

A single SNP-based test

The most intuitive way of using the Fisher's exact test is creating contingency tables for each SNP location and obtaining FET scores for them. This means the headers for the column will be the membership of a SNP in either group, while the headers for rows will be whether the SNP is major or minor. This way we get a 2×2 table with SNP counts in each box. Assuming you get the SNP data for both groups in arrays as input where 3 is the major allele and -3 is the minor allele the Fisher exact score can be calculated as follows:

```
def fisherExactTest(a, b):
    a1 = a==3
    a2 = a==-3
    b1 = b==3
    b2 = b==-3
    return fisherExact(a1, a2, b1, b2)
```

As described earlier in this chapter, calculating the Fisher's exact test is pretty straight forward. The score is basically a combination of the probabilities of each cell having the given value assuming the totals for each row and column. As an example, a 2×2 contingency table for a SNP-location could look like table 3.3. The Fisher's exact test score based on this is calculated using the hypergeometric distribution detailed earlier in equation 3.1. For the example in table 3.3 the score would be

$$\frac{(9+3)!(1+6)!(9+1)!(3+6)!}{9!\ 3!\ 1!\ 6!\ (9+3+1+6)!} \approx 0.0166.$$

A Python implementation of this calculation is:

```
def fisherExact(a,b,c,d):
    import math
    f = math.factorial
    denom = f(a)*f(b)*f(c)*f(d)*f(a+b+c+d)
    nom = f(a+b)*f(c+d)*f(a+c)*f(b+d)
```

	Major allele	Minor allele
Freshwater three-spine stickleback	9	3
Marine three-spine stickleback	1	6

Table 3.3: Example of contingency table for freshwater-marine divergence in sticklebacks

```
return float(nom)/denom
```

Here, a, b, c and d are the values of each box in the 2×2 contingency table.

Scoring windows and identifying regions

Based on the scores for every SNP position in the window I want to return a score for the whole window based on this. The question we want answered is "Does this window contain sufficiently interesting differences between the two groups?". A clear majority of the values will be larger than 0.5, as minor alleles are by nature rare, and most SNP-locations will have allele distributions that are equal for both groups. These positions are not interesting for this project. A measure of how lopsided the most "extreme" SNP-locations of our window is needed. The 95% upper quantile threshold of the negative log_{10} of FET-scores was used in the drosophila study as an indicator for how interesting the differences between the two groups are in the window [5]. This can be obtained by simply using the method `scoreatpercentile` in the `scipy.stats` package with the list of scores for window and 95 as arguments.

With the 5% upper quantile scores for each window at hand, I want to identify which windows are worthy of closer inspection for genomic divergence. A conservative approach presented by Burke et al. [5] in their drosophila study is to calculate a limit based on the 0.1% upper quantile of a normal distribution around the mean of all $-log_{10}$ FET-scores obtained for the whole genome. This requires calculating a standard deviation for these scores. Burke et al. [5] chose to estimate the standard deviation through random re-sampling of 100 scores in every window. Then, the upper quartile(75%) of these scores for the whole genome was used as a measure of standard deviation for the L10FET-scores. Summarized, the limit is calculated by the following python-code:

```
limit = stats.cmedian(fetVals) + stats.norm.ppf(x) \
* stats.scoreatpercentile(FETstddev, 75)
```

Here, `x` is the user-chosen quantile limit, `fetVals` is the list of FET-scores for all windows of the genome, and `FETstddev` is the list of estimated standard devia-

tions calculated for each windows of the genome. Adjoining nearby windows into regions can be done in the exact same way as for the cluster separation scorer. My recipe for identifying regions of genomic divergence based on Fisher's exact test scores is therefore complete.

3.4 Further analysis and visualization of results

Based on the segments obtained from the recipes based on either FET or CSS there is a wide range of possibilities for further analysis. Identifying genes within the regions can be done through various browser-based databases such as the UCSC genome browser [18]. Further analysis of these genes can the molecular biologists etc. Comparing results from our recipes with each other and other results is possible through tools already made available in HyperBrowser. The `Coverage` statistic, counting basepairs falling within and outside one or two datasets is especially useful. In addition, the `Count` statistic for counting the number of points from dataset A inside the regions of dataset B can be useful for measuring accuracy of methods.

Visualizing results is an effective way of getting an immediate sense of the structure of our data. The scoring algorithms produce large amounts of data. Jones et al. [17] used a step size of 500 basepairs for their stickleback study. The stickleback genome is 675 megabases large, giving us 1.35 million windows to score. A standard laptop-screen offers a resolution of 1440×900, totaling 1.296 million pixels. Even using only one pixel per window will leave us with too little space for visualizing all windows of the genome. Even visualizing all scored windows chromosome-wise is difficult. Still, it is possible to graph the scores using graphing packages like `pyplot` [14]. We thereby let the graphing package decide what parts of information to lose. As seen in the results-chapter this leads to somewhat low readability for larger windows of the genome. One solution is plotting pixels along a *Hilbert curve* [13]. Plotting along a Hilbert curve has the benefit of giving a better picture of clusters along the genome than a linear plot [39]. A tool in the test instance of HyperBrowser [45]. The output of comparing two sets of regions in chromosome IV can be seen in figure 3.1. The majority of black pixels underlines the problem with the size of the genome, in reality we are only interested in examining the significant regions.

For visualizing regions I therefore used the `pyplot` package to only plot parts of the genome with significant regions, cutting out all white/black space. While the reader won't be able to get a proper sense of the size of the regions compared to a full genome, it gives meaningful insight into how regions overlap with genes or other regions.

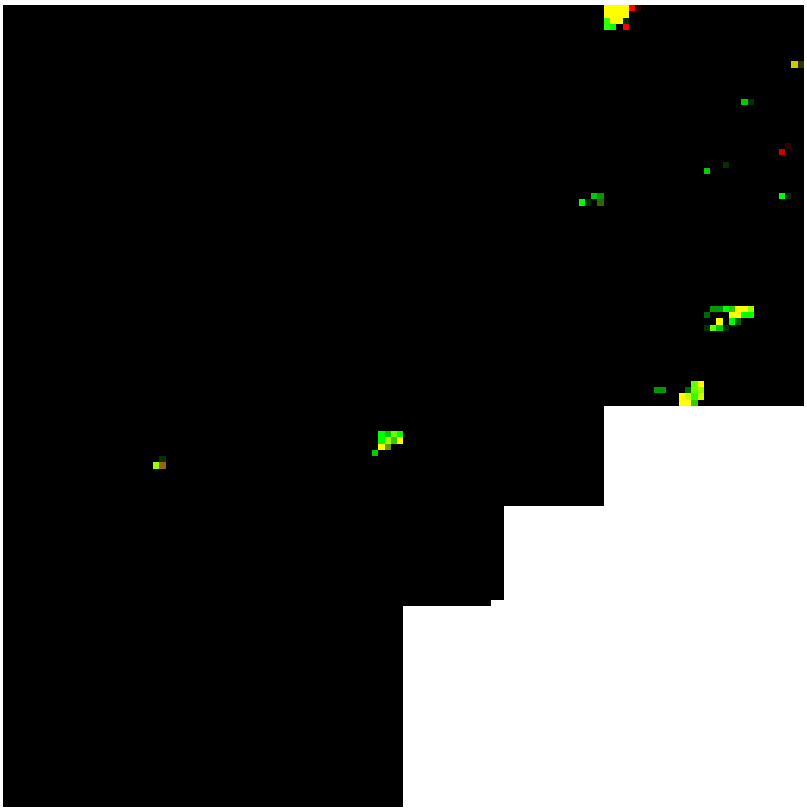

Figure 3.1: Hilbert curve visualization of two sets of regions obtained with cluster separation scoring on stickleback chromosome IV. Green indicates membership in one region, red in the other and yellow in both.

4 Implementation Strategy

There are a few general development challenges any programmer face in a project. The goal is getting an effective solution to the problem avoiding bugs. Many tools in bioinformatics deals with large sets of data and are very resource-intensive, with running times often spanning days. Running times spanning days is not unusual for genome-wide analysis. Optimized code that only saves seconds on small runs may turn out to save hours and days when scaled to larger problem. This section outlines the implementation challenges for my project, different strategies, and how I dealt with these strategies.

4.1 Creating tests before coding saves time

Software development methodology aims to increase productivity and decreasing potential for bugs and breakdowns. Methodology research on how to effectively develop applications is a growing field, driven in part by the private sector. One popular philosophy is so-called test-driven development. Janzen and Saiedian [16] performed a meta-study showing that test-driven development might increase productivity and recommends the use of it in academia. Another study assessing test-driven development at IBM shows great improvements as well [35]. Test-driven programming is a way of developing software that forces the developer to test the code very often.The development cycle is described in the book "Test-Driven Development by Example" [31] as follows:

- Add a test
- Run all tests and see if the new one fails
- Write some code
- Run the automated tests and see them succeed
- Re-factor code
- Repeat

Test-driven programming forces the developer to be sure that what has been produced so far works, and that any unwanted side-effects of any changes she does to the code is quickly discovered. The design process is bottom-up, you start by simplifying your problem drastically to its core and then expanding on it. The development process is also significant in the fact that you have to write the tests first and then code. Using test-driven programming should make it easier to identify problem areas and assess results. This way of thinking might be extra beneficial when developing new algorithms and models. By having to test often the developer is forced to think through the model itself from early on. Any mistakes in the model will be found early.

The core algorithms and calculations in my project are easily testable, and hence test-driven development was used as part of my implementation strategy. Using test-driven development proved to identify bugs earlier and eliminate time sinks that may arise when debugging large amounts of code. Test-driven development of our tool proved to be extra challenging when writing HyperBrowser-specific code. Testing whether the tool is properly integrated into the HyperBrowser, how the web interface functions and the handling of large amounts of data is cumbersome. Creating proper tests for integration into HyperBrowser requires quite some development time, and was not always prioritized. Some test-data exist in the code-base, and could probably be utilized for more efficient integration. Creating tests proved extra hard when what the code produces is a graphical user interface. In theory there is nothing wrong with creating non-automated tests defining behavior and features for graphical interface either. Using test-driven development across the board would probably save time compared to the trial-and-error approach used in the more complex parts.

4.2 Vectorization shortens running time

Array programming is programming that generalizes scalar operations to apply on vectors of data. It often shortens run times and simplifies code. The *NumPy* package [43] is the industry standard for array programming-implementation in python. In his master thesis, Lund [34] found that the NumPy package causes some trouble when developing large-scale programs. The library is most often used interactively or in short scripts where bugs can be easily discovered and corrected. NumPy therefore has a focus on usability and interprets input liberally, seldom reporting possible errors in data. This creates a demand for extra attention when utilizing vectorization. Operations should be thoroughly tested and data vetted before and after.

The basis of algorithms identifying genomic divergence in genomes is very often atomic in nature. Most commonly, tasks like counting occurrences or subtracting differences are used on large amounts of data. This makes these algorithms very suitable for vectorization. One example of the advantages of vectorization is my implementation of comparing two individuals for SNP-differences in a window of the genome. Each individual is represented by an array of integers. At each position of the array the value denotes whether the individual has the major allele (3), the minor allele (-3), both alleles have been observed (0) or no valid data has been found (-10000). We want to count how many times we can be sure that the two individuals have different alleles. A simple python implementation utilizing the fact that the product of two corresponding values is -9 if and only if we have one occurrence of -3 and one occurrence of 3 doing this is presented below:

```
def compareCount(a,b):
  count = 0
  i = 0
  asize = len(a)
  while i < asize:
    count += (a[i] * b[i] ==-9)
    i += 1
  return count
```

Since multiplication and sum is implemented for vectors in numpy this can easily be vectorized as follows:

```
def compareCount(a,b):
  product = a * b
  return numpy.sum(product == -9)
```

The python timing package *cProfiler* [36] reports that the "vanilla"-version uses 2.327 seconds to compare two five-hundred item long arrays a thousand times, while the vectorized version only uses 0.019 seconds on a regular laptop. In a chromosome-wide analysis on 21 sticklebacks using this function as a pairwise comparison metric this function was called more than 14 million times, so the benefit of vectorization is huge. As an added benefit, the code is also shorter and easier to read.

4.3 Choosing a framework

Comparing frameworks for comparative genomics and genome-wide analysis is a large task requiring a lot of experience in the field. The decision to use HyperBrowser as a framework to implement this project in was therefore a choice of convenience done before acquiring any knowledge on the subject. This section describes and discusses the factors that need to be considered when picking a framework for a project like mine, and the pros and cons of using HyperBrowser.

Reinventing wheels

A usable comparative genomics tool requires more code than the strict implementation of an algorithm. Several features needs to be implemented one way or the other:

- The user interface for running analyses
- Data handling and storage
- Presentation and storing of results

These challenges have the complexity worthy of larger stand-alone projects. They are not new problems or special for this project. For each area, several different frameworks exist offers solutions. So should I implement these myself from the ground-up? The answer boils down to a question of whether implementing these parts myself is worth the added cost. If the goal of the project was to learn and explore new possibilities within graphical user interfaces or data handling for bioinformatics tools then there would be no doubt. Reinventing the wheel might indeed be worthwhile if the design of the wheel is lacking and you want to gain knowledge on wheel-building. This is not the case for this project, I am focusing on implementing algorithms for scoring genomic divergence. Hence, the lower implementation cost of using existing HyperBrowser functionality is preferred. Another factor is that our tool is quite narrow in scope. By fitting it into a larger package of tools for genome analysis the probability for anyone to use it rises. This way we improve and build on the ”wheel“ instead of reinventing it. HyperBrowser is itself an improvement of the ”wheel“, in the sense that it is built on the Galaxy-engine for handling work-flows, user administration and integration with genome browsers and databases.

Implementing efficiently

Implementing a tool through the HyperBrowser framework naturally means relying on a large code-base created by a large amount of developers. Important design decisions have been made by the main developers of the framework, which may impact our implementation. Obtaining knowledge on the code-base is essential for any HyperBrowser developer. Especially knowledge of the following areas are important:

- How data is stored, represented and retrieved in HyperBrowser.
- How analysis tools need to be implemented for integration.
- How to create a graphical user interface for your tool within the browser.

In general there are three strategies for gaining knowledge of program packages and frameworks. The most common strategy when implementing in large and established projects, is to read the supplied documentation. If such documentation does not exist or is non-complete one might have to read the actual code, or find test code using the functionality provided. This strategy can easily become a time sink, as a developer might find herself delving deeper and deeper into complex code. If possible, the best solution is to ask developers on the same framework directly, or even the core-developers.

HyperBrowser documentation can to some extent be found on a private webpage for the project. This Wikipedia-style page does contain basic information on the most important areas mentioned above, but the documentation of existing statistics and tools is lacking. Some test code is also available in the project codebase. These are very useful, but only test select parts of the code. The project is quite grown large and navigating through different classes can be cumbersome.

During this project, the by far most effective way of finding solutions to implementation challenges has been asking other hyperbrowser developers themselves for specifics. A low threshold for contacting and getting answers from other developers is an essential part of the implementation strategy for this project. In retrospect, the close availability of core-developers is one of the major reasons why implementing in HyperBrowser was a good choice.

Analysis should be easily reproducible

As detailed in the background chapter, one of the major flaws of earlier genome-wide analysis of divergent genomic regions across populations is that the analyses

are not easily reproducible and transparent. Inspecting previous work and exposing flaws and possible improvements is an important part of science. In bioinformatics it is common to present input and output of analyses, but the calculations and code that give the output from the input shown are only described and not supplied. In a perfect world, anyone should be able to supply input data to a tool and run it. For complex analysis there are often a lot of less important parameter choices that have to be made. While these parameters might affect the results, they are often deemed to be of little importance and not mentioned in published articles. For total transparency, anyone should be able to run the exact same code in the exact same environment with the exact same input as the published results.

HyperBrowser provides such total transparency. A HyperBrowser tool is accessible from anywhere with an internet connection. This means that anyone can make use of the same code and environment. HyperBrowser also handles data storage, as well as connections to other large genomic data libraries. Any user can upload their own data and run analysis. In addition, one can save analysis runs and share them with anyone. This gives anyone the possibility to check exactly which parameters were used to get the results shown, and the ability to redo such analysis.

The tool itself is often the last part of the puzzle when reproducing results in bioinformatics, as both the datasets used and the results are readily shared as supplementary material or even in separate genomic browsers. Datasets can be shared using the HyperBrowser itself and demonstration pages can be created through a simple "what you see is what you get"-editor integrating tool-runs, datasets and results in a simple walk-through. These features ensures that analysis is not only possible to reproduce, but easy to reproduce.

Handling and representing genomic data

Whole genomes contain large amounts of data. The human genome consists of three billion DNA basepairs. Only storing the basepair type of each requires a lot of storage. For analysis done on whole populations even more data is required. Writing and reading from disk is a time-intensive task, and will often affect running time adversely if not handled properly. It is tempting for any developer to create a data representation tailored to our specific algorithms. While implementing my tools in HyperBrowser has some clear advantages, the disadvantage of having to deal with a complex code-base and data structure may seem large. One could imagine only taking advantage of the "outer" features of the HyperBrowser, the graphical user interface and job-handler, using simple data structures specialized for our tool. Such an implementation was developed at the beginning of the project,

but quickly dropped. When trying to run the non-integrated implementation on chromosome-wide amounts of data required more than a week compared to 24 hours for a fully integrated approach. The security of proper data handling and structures that HyperBrowser proved essential to completing this project. Experience from previous masters projects underlines that creating own data structures creates more trouble than joy [34]. All tools are therefore as fully integrated into HyperBrowser as possible.

The HyperBrowser framework provides storage on disks maintained by the University of Oslo, ensuring safety and stability. On disk the data is divided by genome and chromosome as well as data type. This ensures that one does not need to load more data into memory than what is necessary for the specific analysis. When storing and retrieving the data, *memmaps* are used to map data from physical locations on disk. The genomic data is represented as data arrays within the framework, with one array used for every column of a genomic track. The arrays are retrieved through *TrackView*-objects which themselves implement much of the functionality you would expect from a regular ordered collection. They are both iterable and slice-able into smaller parts. The actual data arrays are then retrieved by specific methods for each kind of genomic data you are looking for, i.e. start positions of an interval, values or ends. A more thorough presentation of the implementation of data representation for this project can be found in the implementation details chapter.

A solution for larger or future problems?

This project aims to solve a specific problem within comparative genomics. The task of finding out all other possible problems one can solve by solving our problem is huge, and not a part of the project. However, the value of my project increases if it is easy for others to build on and test for other uses.

By integrating a tool in HyperBrowser a user will be able to use the results from the tool as input for further analysis in other tools made available. Several tools are available through the HyperBrowser framework, ranging from simple file format conversion to visualization of the relationship between multiple tracks of annotations on a genome. This makes any tool implemented in HyperBrowser more powerful, complex analysis-pipelines can be created without changing frameworks. An additional advantage is that HyperBrowser is in constant development. Any new improvements of the framework might be an improvement of the tool. As an example, an implementation of parallelization the tool proposed would have a significantly lower running time, without any change being done to the code itself.

My code is modularized such that every function for the most part focuses on

doing one thing and one thing only. This ensures that other developers can borrow specific functions from my project without having to import the whole project. This way my project contributes more to the rest of the community.

5 Implementation Details

This chapter entails the nitty-gritty details of implementing a tool for identifying genomic divergence across populations in HyperBrowser. First, a data structure for sliding windows is presented. Then I present a HyperBrowser-statistic making use of existing functionality for genome-wide analysis. I also present my implementation of a graphical user interface for the tool through HyperBrowser. Where applicable, the alternative to a HyperBrowser-specific implementation is described. Finally, I discuss the jungle of file formats existing in bioinformatics, and decide on what format to use for the purposes of my tools.

5.1 An implementation of sliding windows in HyperBrowser

The term *sliding windows* is used in computer science as a term for creating intervals of a dataset where the beginning of each interval occurs *step* points away from each other. In most cases *step* will be smaller than the sizes of the intervals. This way the intervals, or windows, will overlap. Sliding window analysis is a popular form of doing genome-wide analysis where the goal is identifying connected regions of the genome. While demanding more computational resources, using sliding windows eliminates the risk of missing information at the borders of windows. For finding interesting regions in large datasets at unknown positions sliding windows is preferable. We do, however, need to know the probable size of the regions we are looking for as window size does not vary when doing a sliding window analysis.

Existing functionality not complete

When doing an analysis through the HyperBrowser you are often given the option to pick the size of *bins*. A bin is a partition of the genome, so bins can be a set amount of basepairs, whole chromosomes or a whole genome. In HyperBrowser a `TrackView`-object is created for every bin and the analysis is run on these bins separately. This is exactly what we want to do with sliding window analysis, only with overlapping bins.

A solution for doing sliding window analysis could be running several analyses with bins of the right window size, but with different start points for partitioning each time. A sliding window analysis with a window size of 2500 basepairs and a step size 500 of basepairs, could be a combination five separate HyperBrowser analysis with bins of size 2500 basepairs and starting points of 0, 500, 1000, 1500 and 2000 respectively. The results of these analyses could then be combined into a complete sliding window analysis.

Before doing analysis on bins, HyperBrowser creates a `TrackView` for every bin, which all retrieve their relevant data by reading the whole chromosome tracks from disk and keeping the relevant window data for the bin. For the sliding window implementation mentioned above, the same data would be retrieved from disk several times. A more efficient way is therefore to load the whole chromosome into a `TrackView` and then slide along that `TrackView` doing analysis on the windows.

Some functionality for sliding along a `TrackView` existed in HyperBrowser: A `TrackViewSlider` takes a full `TrackView` and mimics a `TrackView` on only a portion of that full `TrackView`. This was achieved by calling `slideTo(start, end)`, with start and end representing positions in the genome and not indices of items in the tracks. For most cases that `TrackViewSlider` would just slice the `TrackView` ignoring any previously stored data. The only exception was when the arguments called are just a single increment of the arguments from the previous call to `slideTo`, start+1 and end+1. In those cases, the slider just checked whether the first item of the former window was to be discarded and whether a new item should be added. For sliding window analysis a step size larger than one is often necessary, but the windows will still overlap.

The Sliding Window class

Since most of the functionality needed for a sliding window trackview was already implemented in the `TrackViewSlider`, creating a subclass of the `TrackViewSlider` was deemed the best option for implementing sliding windows in full. There is no need in the algorithms to slide backwards along the track, the only need is to slide "right" and to do so in larger steps than one.

It is important to have in mind that the basepair index does not correspond to array index in the data structure. The next item of our value array might be the value for a basepair position several thousand basepairs away. Therefore we can't just take the next 500 values and "drop" the first 500 values when we want to move a sliding window 500 basepairs. We need to check the address of the next value and see if it is within our target interval. A new method called `slideRight` accomplishes this, with "step" as the only necessary argument. The first task when

sliding right is to calculate the indices of our new window. This is done by using two quick while loops:

```
newLeftIndex =  prevLeftIndex
newRightIndex = prevRightIndex

while newLeftIndex < len(fullTrackView)\
        and fullTrackView[newLeftIndex] < prevStart+step:
   newLeftIndex += 1

while newRightIndex < len(fullTrackView)\
        and fullTrackView[newRightIndex] <= prevEnd+step:
   newRightIndex += 1
```

When these indices have been calculated the rest is only a case of getting the right slices of each track. If the new left index is smaller than the old right index our windows overlap and we don't need to re-retrieve the whole window. E.g. for the `startList` and `endList`-arrays one would do this:

```
slideTV.startList = slideTV.startList[leftIndexStep:] +
                     fullTV.startList[prevRightIndex:newRightIndex])
slideTV.endList =  slideTV.endList[leftIndexStep:] +
                    fullTV.endList[prevRightIndex:newRightIndex])
```

Here the `slideTV`-object is the current window while `fullTV` is the `TrackView` object for the whole chromosome. The rest of the arrays are updated in the same way if applicable, along with a few `TrackView`-specific attributes so that the `SlidingWindow` object acts similarly to any other `TrackView` object.

5.2 Implementing statistics in HyperBrowser

After dividing data in bins using `TrackView`-objects, the HyperBrowser framework uses a special form of class to do actual analysis on each bin. Those classes dealing with actual analysis of genomic data are called "statistics". Some standards for implementing statistics exists. Any classes following the standards which are added to the HyperBrowser code-base are automatically made available for HyperBrowser users supplying fitting data. Implementing a statistic also means that one takes advantage of all existing and future functionality for data handling, processing and storage.

Standards for creating HyperBrowser statistics

The standards for creating HyperBrowser statistics are presented on the HyperBrowser wiki, and can be summarized as follows:

- The main class should inherit the class `MagicStatFactory` and have name equal to the file ending on Stat.
- Another class inheriting the `Statistic`-class with the same name as the main class just with Unsplittable appended to the end.
- The `Unsplittable` class should have a _compute method which does all the main computations

In addition to these three mandatory rules one can also create a Splittable statistic. This statistic describes how to split a problem into subproblems and combine the results from these subproblem into a result for the "mother" problem. This splitting essentially does the same as binning, with no possibility to create sliding windows. Hence, all our analysis for the whole track is done in the unsplittable-statistic.

Results are returned in different ways, but in general one will get a dictionary with a label identifying the bin the analysis is run on as key and the result from that analysis as value. If more than one analysis is done at the same time a dictionary of dictionaries will be created, and if the analysis is genome-wide one will just get a value.

Several templates for statistics have already been made, and most basic operations have been covered by earlier work. Most statistics are descriptive and often run on a single track, just returning information about the track at hand. Some statistics also does overlap analyses and hypothesis testing for similarities. My project involves a bit more complex calculations, including using packages for multi-dimensional scaling and specialized statistical methods which have not been implemented before.

Implementations of statistics for genomic divergence analysis

I created one statistic for each method, the `CategoryClusterSeparationStat` and the `FisherExactScoreStat`. The two statistics are very similar, as the same sliding window partitioning and data retrieval is performed in both cases. For my tools, I decided to let one track represent a population of individuals. Analyses is therefore performed on two tracks at a time. A statistic for operating on two tracks is already created through the `MultipleRawDataStatistic` class in HyperBrowser, which was

therefore used as a superclass in my implementation. The tracks need to be non-dense as I only care about SNPs, which do not occur at all positions of the genome. It is also necessary to both allow more than one data point at each basepair address, since more than one individual is represented by a single track.

To ensure that the tracks allows overlaps I implemented `_getTrackFormatReq` as well as the `_compute`-method, returning an object of the class TrackFormatReq initialized with the parameter `allowOverlaps` set to True. The `_compute` method can then obtain TrackViews by through the `self.children` variable.

I then use the `SlidingWindow` implementation detailed earlier to operate on a sliding window of these trackview objects. The parameters detailing the size and step of the windows as well as other parameters used in the cluster computations themselves, are obtained through the **kwArgs-argument dictionary sent with the initialization of the statistic. I iterate through the `TrackView` object sliding the sliding window along, and for each window a cluster separation score and a corresponding p-value is calculated. Correspondingly a fisher exact test score and a corresponding standard deviation for the `FisherExactScoreStat`-case. These results are returned as a dictionary with the string version of the starting position as key and a tuple of the two values as value.

5.3 Creating graphical user interfaces in HyperBrowser

As detailed in the implementation of sliding windows I want to operate on objects of `SlidingTrackView` and keep the user away from splitting the data into bins. This cannot be assured through the regular "Run analysis"-template and is the reason why a special HyperBrowser tool is created for calculating cluster separation. Luckily, creating such tools in HyperBrowser is pretty simple and did not add much to the total workload of any project.

Requirements for creating a graphical user interface

To create a separate graphical user interface for a hyperbrowser tool which executes and analysis one needs to create a subclass of the `GeneralGuiTool` class implementing the following methods:

- **getToolName(), isPublic() and getOutputFormat()** self-explanatory methods
- **getInputBoxNames()**, which defines the number of and names of the input boxes

- **getOptionsBoxX()** where X is an integer starting at 1 and going until all input boxes have its own method. The return value of this method defines the type of input box, i.e. history element, genome or just string input.
- **execute()** Takes the choices done as input as executes the code, the results should either be printed as HTML or written to file

In addition to these methods, several methods exist for customizing the appearance and usability of the tool.

Executing genome-wide cluster separation scoring

This section details how our HyperBrowser tool executes cluster separation score analyses of a sliding window of the genome. First, our tool requires the following input from the user through a list choices supplied as an argument to execute:

1. The reference genome for the SNP-data.
2. The gtrack file for the first group of SNP-data
3. The gtrack file for the second group of SNP-data
4. How should the atomic comparisons in the scoring process(see below) be done? [Count differences]
5. Size of windows to compute separation scores on, this affects computation time and should depend on density of data [2500 basepairs]
6. Step between windows, which in effect decides how much each window should overlap the previous and next window [500 basepairs]
7. How many instances of scores should be observed before we can stop calculating the p-value(since the score then is insignificant) [10]
8. How many simulations of scores on different groupings should be done at a maximum when calculating p-values [50000]
9. Output formats. The choices are HTML, which gives the data straight in text in the browser, or tabular, which then can be used by downloading the file or further analysis in the browser [tabular]

The tracks supplied are only identifiers, not the real track names used in the rest of the hyperbrowser. The track data is retrieved through calling a method within

the `ExternalTrackManager` called `getPreProcessedTrackFromGalaxyTN` with the genome and track identifier as arguments. To compute separate analysis for each chromosome, we then iterate through the chromosome list of the genome. This list is obtained through using the `getChrList` of the `GenomeInfo` class. For each chromosome we then run the analysis by using the `runManual` method of the `GalaxyInterface` class. The `runManual`-method requires five arguments:

- A list of track names
- An analysis definition, written in a special format of ”Name: description ([arg=value of argument] [arg2= value of argument2]) -> NameOfStatistic-Class“, the name and description is obsolete when using runManual, so in our case the analysis definition is: "Dummy: dummy name ([wStep=%g] [wSize=%g] [func=%s] [mcT=%g] [mcR=%g])-> CategoryClusterSeparation-Stat" % (windowStep, windowSize, compare, mcTreshhold, mcRuns)
- The region to run on, in our case the chromosome gotten from the chromosome list
- The bins to use, in our case we are doing analysis on the whole chromosome so we use the wild-card ”*“ to do an analysis on the whole region
- Genome, which has been supplied by the user

The results are then returned in the dictionary format and we iterate through the dictionary writing a new line for each item to file with start position, score and p-value. This format makes further processing of results easy. The implementation for the Fisher’s exact test scoring is similar, only excluding the parameters for sequential Monte Carlo and comparison metric.

5.4 Representing SNP-data uniformly

Our tool acts on sets of single-nucleotide polymorphisms (SNPs) discovered when sequencing genomes. SNP-data for different species is made available through specialized web sites, with a great variation in file formats. These SNPs needs to be translated into a uniform representation when doing analysis. This section gives a brief overview over the current jungle of file formats for representing genomic data. I then present a simple file format for use when doing analyses on sets of SNPs as well as analyze possibilities for converting to this file format.

No industry standard exists

There are several different types of formats for genomic tracks. The most common format for presenting genomic data is as tabular data with self-defined headers. Supplying tabular data with self-defined headers effectively means creating your own representation of data more or less disregarding other standards. The lack of such common standards makes it harder to reproduce results as tools have to be specialized for a specific format for each analysis. This is the case for the drosophila and stickleback studies detailed in the background chapter, where the authors have created their own headers for representing data. Since these genomic track often aim to describe vast amounts of data choosing a compact format is important. The most used current general formats with a standard for headers are General Feature Format (GFF), Browser Extensible Data (BED) and Wiggle Track Format (WIG). [11] All of these formats lack some functionality and are not mutually compatible, but they are very similar in that they are tabular based and have a narrow focus.

Two new proposed track formats are introduced by Gundersen et al. [11] in a study of file formats published in *BMC Bioinformatics*:

- **GTrack**
  GTrack is a tabular format with specific headers for type of content, making it possible to convert any of 15 different formats to GTrack by setting the relevant information in the header. This makes GTrack very powerful for future use. GTrack allows only one kind of feature to be in focus in the annotation track. This means that we will have to create several annotation track files for the same genome.

- **BioXSD**
  BioXSD is an XML-type format which allows combining features with overlapping intervals etc. in a single file. This format requires more processing which is unfortunate when dealing with massive datasets.

Although a myriad of different tabular formats exists a standard for referencing a location in a genome is used by all the common file formats. As the name implies, a single-nucleotide polymorphism, consists of exactly one basepair. A specific basepair in the reference genome is almost always referenced through two fields in file representations:

- The chromosome id, also referred to as seqid (sequence id)

- The number of basepairs from the beginning of the chromosome (most often ”start“ is used as header)

A third field is also often used for categorizing locations, the sequencing bin of the position. Sticking to this standard will significantly lessen any burden in converting files to and from our data. No other addressing scheme has been identified.

Picking our standard

To pick a standard I need to define what information is needed for my tool. The information needed can be divided in two parts, the location of the specific SNP and the value of a specific SNP.

Location A SNP is called on a specific address of the reference genome for an individual which is part of one of two groups. Therefore we need to have a seqid-field for identifying the chromosome and a start-field for the position of the SNP within the chromosome. The membership of the SNP can be determined by using one file for each group of SNPs. With more than one individual in a group we would then lose information about which individual the SNP is from. This information is useful for cases were there is a need for changing membership for an individual when calculating cluster separation scores.

Value When a single-nucleotide polymorphism has been identified and validated every individual can be categorized in four categories. The individual can have:

- the major allele
- the minor allele
- both alleles can be observed
- data can be missing

A field with the header ”value“ is often used for representing genomic tracks in common file formats and will suit the uses for this project wll. Values are most often represented as numbers, which is perfectly suitable for my case.

The chosen format

That leaves me with these required fields for our target track format:

- Seqid, or chromosome
- Start, or basepair address

- Value of the SNP
- Individual id, if possible

Gtrack has been shown to be a flexible file format which accepts custom headers [11]. Another important factor is that a lot of HyperBrowser-functionality has already been implemented for gtrack files. Looking at the table in the study by Gundersen et al. [11] describing the different sub-formats of gtrack available we see that Valued Points fulfills all mu needs except the membership id. Gtrack does however support extra columns with custom names, which is useful for our individual id. Such a format is also easily convertible to other formats through tools already made available in the HyperBrowser tool. A "Value Point" gtrack format is therefore my format of choice for this tool. The beginning of such a file would look like this:

```
##gtrack version: 1.0
##track type: valued points
##uninterrupted data lines: true
###seqid start value genomeid
chrIV 171 3.0 12
chrIV 171 -10000.0 13
```

Converting SNP data to my format

Converting unstructured tabular data to gtrack files is possible by using a tool in the HyperBrowser. This tool simply lets you create new headers for the tabular format and choose which columns to keep to use in your gtrack file. No editing of any values can be done, except correcting sequenceid names, which is useful for cases where the chromosome identifiers lack the necessary "chr" before the chromosome name. It is not possible to split data in one row into several rows either. For both the SNP data of the stickleback study and the drosophila study this means I need to do some additional data-massaging to convert the data into my chosen format.

Use case: Converting stickleback SNPs

The data used for the stickleback article is published on the internet. The published data actually contains all SNPs calls of a certain quality, and it is possible to separate the SNPs based on individuals Downloading the SNP data from their table browser yields a tabular file like this:

```
#bin chrom chromStart chromEnd name score strand
thickStart thickEnd reserved blockCount blockSizes
chromStarts expCount expIds expScores
682 chrI 12750156 12750157 chrI:12750157-R:A_A:G 1000 +
12750156 12750157 0 1 1 0 22
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,\
-3,0,0,3,0,-3,3,-10000,-10000,3,3,-10000,-10000,\
3,3,-10000,3,-3,3,-3,-3,3,
```

Most of the fields are not interesting for my tools and can easily be filtered out. The interesting fields for this project are:

- chrom, which corresponds to seqid
- chromStart, which is the basepair address
- expScores, which is a comma-separated list of all scores for 22 different genomes
- expIds, which is a comma-separated list of all ids for 22 different genomes. Note that this list is identical for all SNP data from the sticklebrowser.

The only challenge here other than renaming the fields and creating proper headers for the gtrack format is to create 22 separate lines for each line, so that each individual SNP for every individual is separated. Note that a special property of this dataset is that although the data only present SNPs for 21 individuals, 22 scores and ids are used. The 12th score(id 11) is always -10000, seemingly due to visualization purposes relating to the tools used by the authors [44].

Use case: Converting drosophila SNPs

The data used for the drosophila article was supplied by Burke et al. [5] as a part of the supplementary material. In the article they pool SNPs based on treatment and hence it is not possible to separated the SNPs based on individuals. Instead they have supplied the number of SNP calls for the major allele and the number of SNP calls for the minor allele. The file format is as follows:

```
#chromosome position N.major.AC0 N.minor.AC0 N.major.C0 N.minor.C0
N.major.AC01\
N.minor.AC01 major.flavor minor.flavor reference.flavor\
L10FET.AC0vC0 L10FET.AC0vsAC01 DPGP.hap DPGP.nA DPGP.nC\
```

```
DPGP.nG DPGP.nT DPGP.biallelic DPGP.major DPGP.minor DPGP.MAF
1 2L 5762 5 4 12 4 3 1 C T T 0.404264907824556 0\
TCCTCCTTTCTTNCCCTTCTCTTTCTCTCCTTTTCTTTC 0 16 0 22 1 T C\
 0.421052631578947
```

Again, most of the fields are not interesting for my tool, so I have to filter out most of it. Obviously, chromosome corresponds to our seqid field and position to start. The other interesting fields are all the fields starting with either N.major or N.minor. These are counts of SNPs called for either the major allele or the minor allele for each treatment. I only need to know the proportion for each allele to do our computations. Hence, one line of data in the above mentioned data format leads to three lines of data in my target gtrack file where the individual id is set to either ACO, CO or ACO1 based on treatment and the value is set to the proportion of major allele SNPs of all called SNPs for that specific treatment and position.

6 Results

I have implemented tools for identifying regions of genomic divergence through scoring cluster separations and Fisher's exact test. This chapter details the results obtained from using the developed HyperBrowser tools on stickleback and drosophila data. First, I present results identifying regions of marine-freshwater genomic divergence in sticklebacks. Results from doing cluster separation scoring and Fisher's exact test significance scoring is presented and compared with each other and with previously published results. I then present results from identifying regions of genomic divergence across three drosophila populations using cluster separation scoring. The results are compared to previously published Fisher's exact test scores before candidate regions for genomic divergence are briefly presented. At the end of the chapter I discuss the performance and reproducibility of the analyses.

6.1 Marine-freshwater divergence in stickleback genomes

The process of identifying regions of genomic divergence in HyperBrowser requires three steps: The first step is to obtain SNP-data in the right format for input into our tools. One may either use the cluster separation scorer or Fisher's exact test scorer to score the genomic divergence in each window. At the end the user has to process the results by filtering out the most significant regions and analyze these.

Obtaining validated SNP data

Validated SNPs for 21 stickleback individuals are supplied by Jones et al. [17] through a dedicated web page[44] in the format described in the implementation chapter. The data have to be downloaded to a computer and then uploaded to HyperBrowser through the Galaxy "get data"-tool. The tool presented in the implementation chapter for converting stickleback SNP-data into correct input data for CSS-analysis is available in the HyperBrowser as "Convert from Jones et al. format to gtrack". The tool requires the user to pick which individuals belong in

each population, and has to be run once for every population. I divided the SNP-data into the same two populations used in the stickleback study, data with id 1-10 belonging to the first population and 12-21 in the second population.

Result from cluster separation scoring stickleback data

Trying to do a genome-wide analysis computing cluster separation scores for all stickleback data fails due to the amount of data. When HyperBrowser preprocess data from files into tracks, it loads all data into memory. In our case this means more than 50GB of data loaded into the virtual memory. Therefore I did the analysis chromosome-wide, which have the added advantage of being able to doing analysis on several chromosomes at the same time, as opposed to sequentially.

I calculated cluster separation scores for windows with step 500 and size 2500. The metric for pairwise comparisons for this kind of data is `countDifferences`, counting pairwise nucleotide differences as described in the methods chapter. For significance scoring I required a minimum of 10 runs, and a maximum of 200 000 runs. This gives us a least possible p-value of $\approx 10^{-6}$ Most of the parameters are exactly the same as the ones picked in the stickleback study, the only exception being the parameters for the significance scoring. Jones et al. [17] chose to score all possible combinations of two populations, instead of using the more efficient sequential Monte Carlo-method.

The output of the tool is a gtrack file with a score and p-value for every 500th basepair of the chromosome like this:

```
#seqid start score p
chrIV 1038000 0.654720449248 0.769230769231
chrIV 1038500 0.60881305367 0.909090909091
chrIV 1039000 0.632245461451 1.0
```

Figure 6.1 shows Cluster Separation Scores and p-values for a small window of chromosome IV, including the EDA-gene which controls stickleback armor and is a known biological trait that differs between marine and freshwater fish. Figure 6.2 shows how the scores vary and give no meaningful information for chromosomes with little genomic variation. I calculated Pearson's r correlation coefficient chromosome-wise for the cluster separation scores using my HyperBrowser tool and the scores made available by *Kingsley Lab Stickleback Genome Browser* [44]. The correlation between the sets of scores vary a lot between chromosomes, with higher correlation in chromosomes where significant regions are identified, and almost no correlation in windows such as chromosome XV. The correlation scores are shown in Appendix B.2.

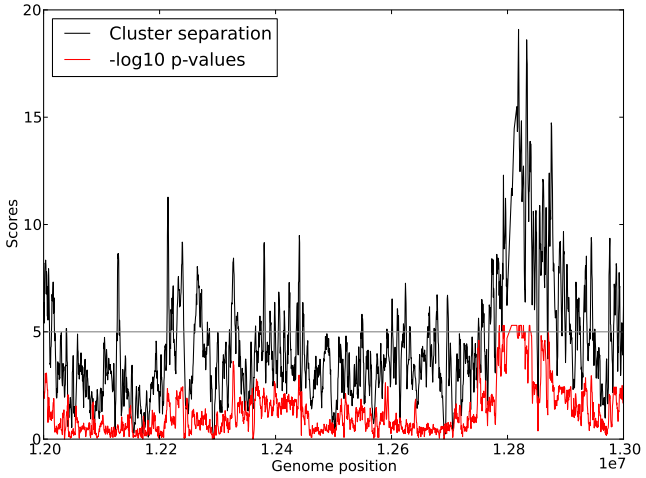

Figure 6.1: Cluster separation scores and corresponding $-\log_{10}p$-values for a small part of stickleback chromosome IV. The grey line is the genome-wide p-value limit for windows kept with a False Discovery Rate of 0.05, the EDA gene is found in the area where the p-values cross this limit.

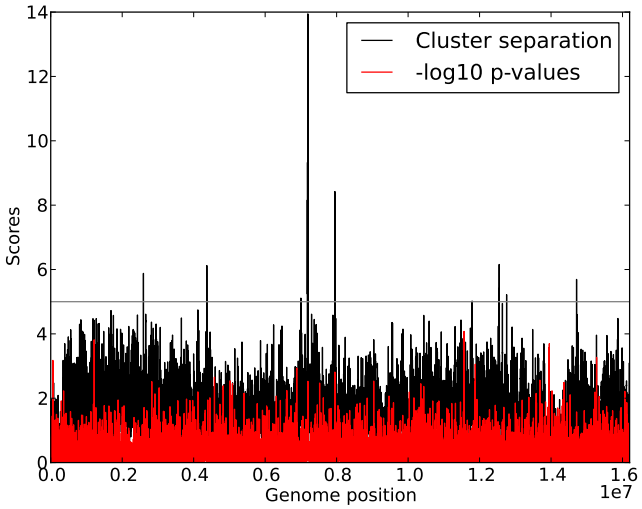

Figure 6.2: Cluster separation scores and $-\log_{10}p$-val for the whole stickleback chromosome XV. Most scores vary wildly with no scores above 14 and no p-values are close to the threshold of 5 needed for significance under a False Discovery rate of 0.05.

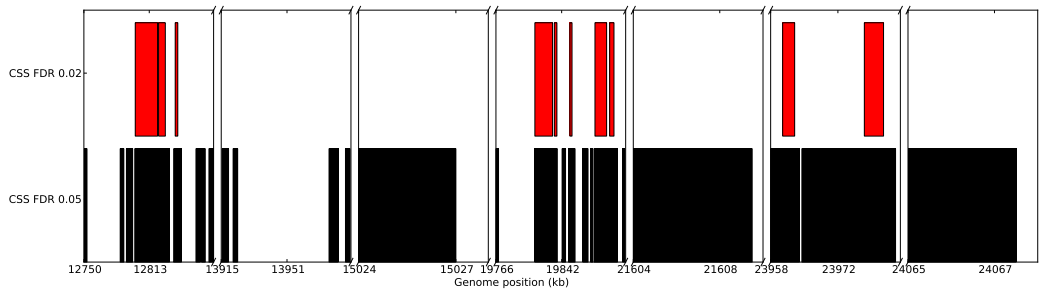

Figure 6.3: Regions of stickleback chromosome IV identified as genomic divergent using cluster separation scoring under a false discovery rate of 0.02 and 0.05 respectively

To identify the most divergent regions I used the tool presented for doing such analysis based on a false discovery rate. Picking 0.02 as our false discovery rate gives us 105 windows spread across 23 regions, all of which have a p-value of 1./200000. 2 of these windows can be expected to be false positives. For these windows, no randomly permuted data gave a more extreme score than those found when dividing the sticklebacks in marine and freshwater-groups. We also did analysis with a false discovery rate of 0.05, which returns 637 windows spread across 105 regions. The regions found under the two different FDR-values are visualized for chromosome IV in figure 6.3. Note that I did not include regions and data from chrUn in my false discovery analysis, as these regions are not contiguous but rather a collection of all regions that the assemblers were unable to place in chromosomes. The regions found by using a false discovery rate of 0.05 of course overlap all regions found with a FDR of 0.02, but while some of the regions are "expanded", most extra windows are "new" regions not discovered with 0.02.

Fisher's exact test analysis on stickleback SNPs

A genome-wide Fisher's exact test-analysis was performed using the same grouped SNP-data as for the cluster separation scoring-tool. Burke et al. [5] argues for using 100kb windows when performing FET-analysis. Therefore I performed two analyses, one with 100kb windows and 2kb steps and one with 2.5kb windows and 500 basepair steps. Figure 6.4 shows scores for the part of chromosome IV including the EDA-locus. The values for 2.5 kb windows vary a lot more, but the EDA locus is clearly visible in both. Figure 6.5 shows how none of the values for chrXVI cross the threshold, and how the scores vary a lot for both window sizes.

Again we need to identify the most extreme regions for closer examination. By using the tool created for filtering fisher exact test scores based on the 99% upper

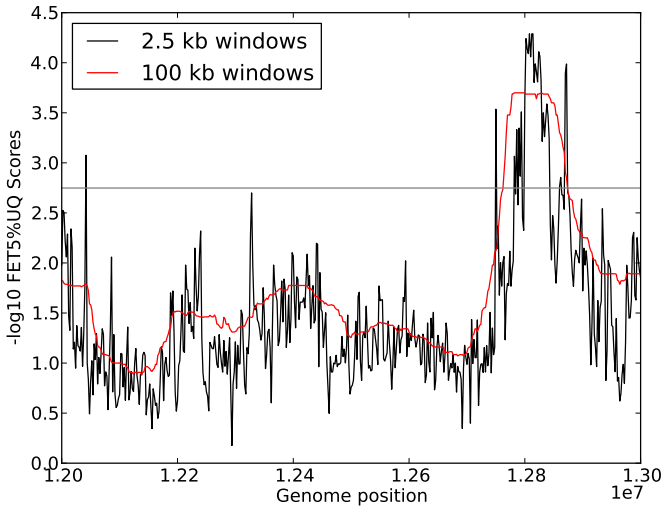

Figure 6.4: Graph showing -log10 Fisher's exact test 5% upper quantile scores for sliding windows of size 2.5 kb and 100 kb in stickleback chromosome IV. Grey line signifies the threshold for interesting regions based on a normal distribution over the mean of genome-wide scores.

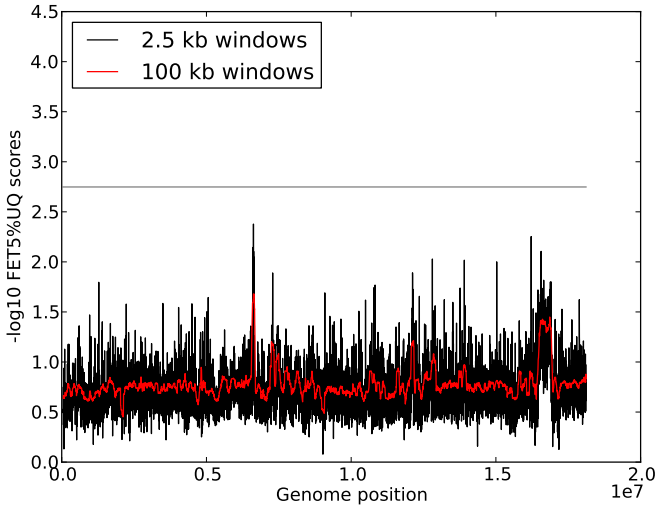

Figure 6.5: Graph showing -log10 Fisher's exact test 5% upper quantile scores for 100 kb and 2.5 kb sliding windows of stickleback chromosome XVI. The grey line at the top is the threshold for interesting regions.

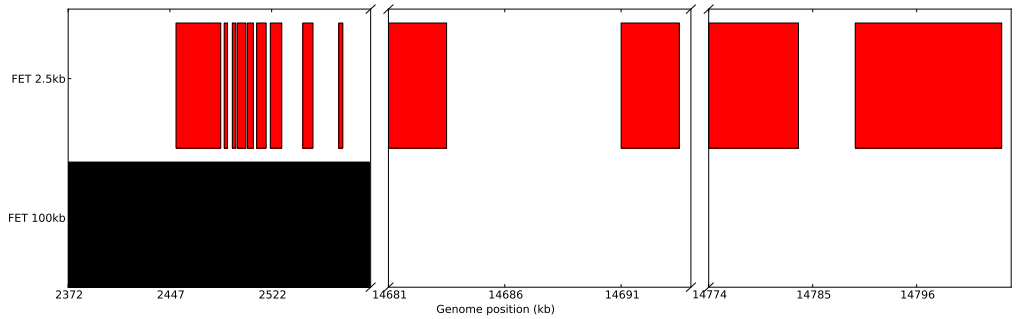

Figure 6.6: Regions of stickleback chromosome XIX identified as genomic divergent using Fisher's exact scores and a threshold taken from the upper q quantile of a normal distribution around the mean of the genome-wide FET-scores. Analyses performed with 2.5kb windows and 100kb windows are compared, with limits that give comparable sizes of output.

limit of a normal distribution over the mean of the scores we get a limit of scores of 2.33 and 7 regions covering 514 windows are identified. These windows are visualized along with a greedier 90% upper limit and a more conservative 99.9% limit for chromosome IV in figure 6.6. For the 99.9% limit both regions identified were actually found in chromosome IV, one of them covering the EDA location. The FET results for 2 500 basepair long sliding windows are regions obtained by taking the 10^{-9} upper percentile, yielding a limit of 2.7008. The high percentile was needed to get a narrow amount of regions, as results with lower limits returned regions covering several million basepairs.

Comparison of results

Jones et al. [17] identified regions of genomic divergence using cluster separation scoring and a self-organizing map. As noted in the methods-chapter, visualizing such large amounts of data is complicated as a single pixel will represent several windows. Figure 6.7 shows the regions of genomic divergence identified in stickleback chromosome XIX by different methods. FET with 100kb clearly is both to broad and have the worst accuracy, while CSS with a false discovery rate of 0.02 seems to narrow. The regions obtained by cluster separation scoring with a false discovery rate of 0.05 overlaps nearly the same amount as the Fisher's exact test 0.99 quantile results with windows of 100kb. This is made more clear in table 6.1, which shows the pairwise overlap between the four different methods and Jones' set. The most interesting column in that table is definitely the last, where the

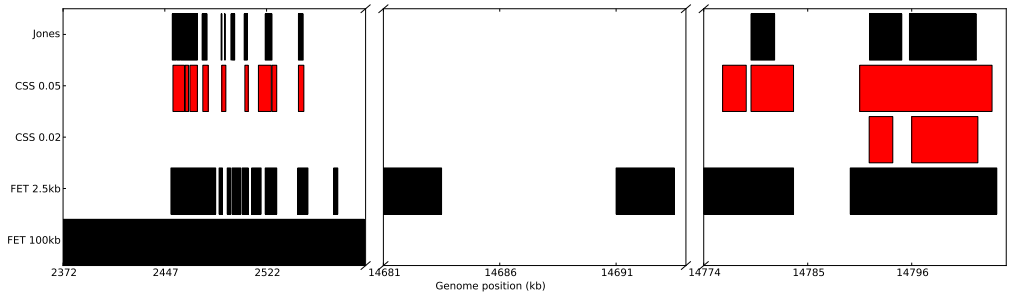

Figure 6.7: Comparison of regions identified in chromosome XIX in stickleback by four different methods. At the bottom, the regions identified by our tool using Fisher's exact test with window size 100kb and 2.5kb respectively, in the middle the regions identified using cluster separation scoring and two different false discovery rates is shown, and at the top the regions supplied by Jones et al. [17] as the union of their results from CSS (FDR0.02) and SOM/HMM-methods.

amount of coverage of previously published results is measured. FET with 2.5kb fares best, but covers almost twice as many basepairs as CSS with a false discovery rate of 0.05.

Obtaining the names of genes that fall within a set of genomic regions is possible by downloading relevant gene data from genome browser such as Ensembl [9]. Identifying which genes that fall within our regions does not tell us anything about whether we've identified the *right* genes. Identifying genes with diverting marine-freshwater expressions, that is biological differences, is a complex task outside the scope of this project. In the supplementary material of Jones et al. [17] all the regions of the strict set are annotated with adjacent and overlapping genes. The notes also includes information on whether the genes are known to have marine-freshwater divergent expression and whether the gene contains SNP-locations where there is a clear marine-freshwater division, i.e. where all marine sticklebacks have the major allele while all the freshwater sticklebacks have the minor allele. Table 6.2 gives an overview over all genes that fell within Jones' strict set and whether they were covered by the my different analyses based on tools implemented in HyperBrowser.

6.2 Genomic divergence across drosophila populations

Genome-wide resequencing data from Drosophila Melanogaster populations that have experienced over 600 generations of laboratory selection have been provided

Overlap	CSS 0.02	CSS 0.05	FET 100k	FET 2.5k	Jones
CSS 0.02		$\frac{126}{585} \approx 22\%$	$\frac{108}{1728} \approx 7\%$	$\frac{122}{1055} \approx 7\%$	$\frac{101}{368} \approx 28\%$
CSS 0.05	$\frac{126}{126} \approx 100\%$		$\frac{545}{1055} \approx 52\%$	$\frac{470}{1728} \approx 27\%$	$\frac{330}{368} \approx 90\%$
FET 100k	$\frac{108}{126} = 86\%$	$\frac{470}{585} \approx 80\%$		$\frac{891}{1055} \approx 85\%$	$\frac{331}{368} \approx 90\%$
FET 2.5k	$\frac{122}{126} \approx 97\%$	$\frac{545}{585} \approx 93\%$	$\frac{891}{1728} = 52\%$		$\frac{364}{368} \approx 99\%$
Jones	$\frac{101}{126} \approx 81\%$	$\frac{330}{585} \approx 56\%$	$\frac{331}{1728} \approx 19\%$	$\frac{364}{1055} \approx 35\%$	

Table 6.1: Overlaps between the four different methods for identifying regions of genomic divergence.
CSS X=Cluster separation score with a false discovery rate of X.
FET=Fisher's exact test regions found by using either 2 500 or 100 000 basepair sliding windows.
Jones=The strict set of regions obtained by Jones et al. [17] through intersecting their CSS-analysis with regions obtained from SOM/HMM.
Each spot in the table gives the ratio of how much the row overlaps the column. Numbers are coverage in kilo basepairs.

by Burke et al. Comparing the genomic data for these populations with a control drosophila-population using my tools can provide insight in which genomic regions the selection have influenced.

Obtaining and converting SNP-data The SNP-data was provided by contacting Burke et al. as a txt-file with counts for minor and major allele for each SNP-location for each population. Since no information on individuals is provided we can not run a fisher exact test score similar to the one done on sticklebacks, but by using the average difference of minor allele frequency for each window as a pairwise comparison metric it is possible to use our cluster separation scorer. The SNP-data was therefore computed by computing the minor frequency allele at each location for each population. Data for three populations were provided, two accelerated populations and one control population. Since this leaves us with only three ways of dividing the populations into groups significance testing through permuting the possible divisions is not worthwhile and we won't get any significance testing of our cluster separation scores.

Comparison of fisher scores and cluster separation scores for drosophila Cluster separation scoring was performed using windows of 100 000 basepairs, to get best possible comparison with the FET-scores published by Burke et al. [5]. Figure 6.8 shows how the Fisher's exact test scores and cluster separation score follow

Gene	CSS FDR 0.02	CSS FDR 0.05	FET 2.5kb	FET 100kb
TNS1[1]	yes	yes	yes	yes
IGFBP5	no	yes	yes	yes
IGFBP2	no	yes	yes	yes
STK11IP[5]	no	yes	yes	yes
Cx44.2	no	yes	yes	yes
INHA	no	yes	yes	yes
SPEG[2]	no	yes	yes	yes
CTDSP1	no	yes	yes	yes
OBSCN	yes	yes	yes	yes
ATP1A1[1]	yes	yes	yes	yes
FILIPIL	yes	yes	yes	yes
Muc5B	no	yes	yes	no
EDA[1]	yes	yes	yes	yes
GARP[7]	yes	yes	yes	yes
GJB1	no	no	yes	yes
CPEB4	no	yes	yes	no
BNIP1	no	yes	yes	no
NXT2	no	yes	yes	no
PPARA	yes	yes	yes	yes
TTN	no	yes	yes	no
ATGL	yes	yes	yes	yes
SULT4A1[1]	yes	yes	yes	yes
TM4SF19[3]	yes	yes	yes	no
MPDU1	no	yes	yes	no
NLRC5[4]	no	yes	yes	yes
MYHC	no	yes	yes	yes
Total coverage*	18/32	31/32	32/32	25/32

Table 6.2: Genes identified within the strict set of regions obtained by Jones et al. [17] and whether they fall within the regions obtained through the implemented Hyperbrowser tools. Superscript in gene name indicates known parallel marine-freshwater non-synomous substitutions within the gene, and **boldface** indicates the significant marine-freshwater expression differences. Data obtained by cross-checking the table supplied in the supplementary material of stickleback study with our regions manually. *Note that some non-diverting genes covered by all methods are excluded from the table due to lack of space, they are however included in the total coverage-count.

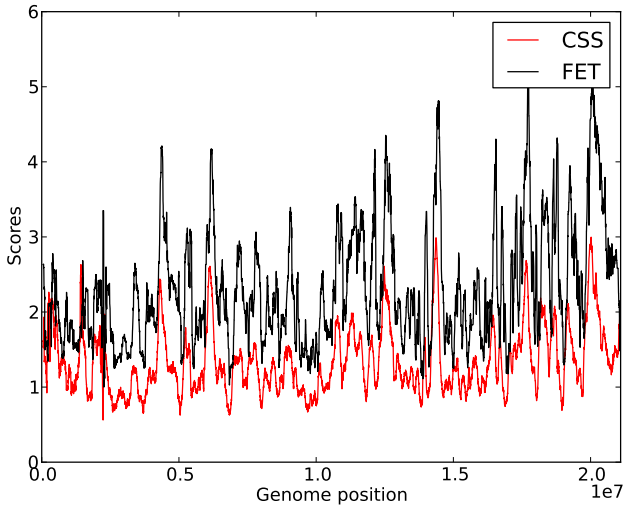

Figure 6.8: Cluster separation scores for chromosome 2R of drosophila, compared with previously published results by Burke et al. [5]. To get a proper visual sense of similarity, the cluster separation scores are multiplied by 10. The graph clearly shows how the two methods identify similar peaks.

each other closely. Since we do not have any p-values for these cluster separation scores we have to sort to using the method used for FET-scores when trying to identify the most significantly different regions. By finding regions which have a more extreme score than 99.9% of what falls within a normal distribution around the mean of all scores we obtain the following two regions:

```
###seqid start end
chr2L 22978000 23084000
chrX 22220000 22522000
```

The lower limit of average allele frequency difference found that identified these two regions covering 105 windows was 0.35, which intuitively seems quite large for a non-biologists. Burke et al. [5] did not find any genes with diverging expression in their analyses, so overlap analysis is not possible.

6.3 A note on reproducibility and performance of analyses

Since all of my analyses are done online through the Genomic HyperBrowser, they are easily reproducible. All jobs run in HyperBrowser are collected in an openly

available history, and a special Galaxy Published page is created for each analysis. A list of links to these analyses are available in Appendix A.1. The pages makes it possible for anyone to import the history into a Hyperbrowser branch and re-run the analyses themselves. Anyone can therefore inspect each element and tinker with parameters on the same data as one wishes. It is not uncommon for bioinformatics analyses running on large datasets for several days before returning results. Most of our tools finish within a minute. The exceptions are the Fisher's exact test scorer and the cluster separation scorer. The longest FET-run (chrUn of the stickleback) took just under an hour of running time on the HyperBrowser. The Cluster Separation Scorer is by far the most resource-intensive, the longest running cluster separation scoring with analyses on chromosome I of the stickleback chromosome taking 45 hours. The major deciding factor is the calculation of p-value, a calculation that is dependent on the maximal number of Monte Carlo permutations. This implies a trade-off between running time and resolution of the p-value.

7 Discussion

This chapter discusses the accuracy of identifying regions of genomic divergence with the tools developed. The effect of parameter choices and possible sourced for inaccuracies are presented. The usability of my tool is an important factor for achieving the goal of more reproducible analyses in bioinformatics, section 7.4 discusses general and specific usability issues. A conclusion of the project is presented at the end of the chapter.

7.1 Discussion of results from analyses

In chapter 3 I presented three different algorithms for identifying genomic divergence. Two of them have been implemented in HyperBrowser, and results from doing analyses with these tools were presented in chapter 6. All analyses are done with data which has already been used for similar analyses that have been published. The results, or scores for each window, are filtered and analyzed after scoring, which gives us several levels to assess the quality of my results on:

- Correlation of resulting scores with previously published results
- Overlap of significant regions with previously published results
- Genes identified within the regions, and their properties

In this section I use these methods for assessing the quality of analyses done on stickleback-data and drosophila-data respectively.

7.1.1 Quality of stickleback analyses

Appendix B.2 gives an overview over the chromosome-wise Pearson r-correlation scores between cluster separation scores from my analyses and the scores obtained from the stickleback study performed by Jones et al. [17]. The scores vary wildly, with really low correlations in some chromosomes. This is not surprising, there is quite some noise in the cluster separation scores in areas with little divergence, as observed in Figure 6.2. The correlation of scores therefore does not give us

much information on the ability of my method to identify correct genes. Table 6.1 shows the pairwise overlaps for four different analyses done to identify marine-freshwater divergence in sticklebacks. The table includes overlaps with the strictest set of regions published by Jones et al. [17].

Fisher's exact test with 2500 basepair windows (FET2.5kb) achieves most overlap with the strict set previously published. The $\approx 10\%$ improvement compared to cluster separation scoring with a false discovery rate of 0.05 (CSS0.05) is somewhat offset by the fact that the FET2.5kb-regions are broader. FET2.5kb covers almost twice as many regions as CSS0.05 (1055kb vs 585kb). The results from CSS0.02 and FET100kb underlines the major impact of window size and limits for identifying regions, both of these perform notably worse than their counterparts using the same algorithm. Figure 6.7 visualizes this well, the resolution of CSS0.05 is clearly better than FET2.5kb. Both CSS0.02 and FET100kb miss regions obtained by Jones et al. [17] strict set. Visualization of all other regions are available in B.1, confirming the trends shown in figure 6.7.

Table 6.1 gives a more detailed overview over which genes within the previously identified regions by Jones et al. [17] that are also identified by my methods. This has some inherent weaknesses. Genes my analyses identify that haven't been previously identified are not listed. Such identification requires analyses of differences in expression for properly measuring the quality of results. All genes with a different expression across marine-freshwater populations are identified by both CSS0.05 and FET2.5kb.

The results from filtering **cluster separation scores with a false discovery rate of 0.02** fails to identify several of the genes with difference in expression. This might indicate that the p-value threshold required by a false discovery rate of 0.02 therefore seems to be too low. In effect the threshold obtained by using the Benjamini-Hochberg-procedure is 1/200000; all regions where none of the randomly picked groupings get a score equal to or more extreme than the marine-freshwater groupings. There is however a probability that one or more of the random groupings done while using sequential Monte Carlo for calculating p-value are the same marine-freshwater groupings. More than 0 scores equal or more extreme to the one obtained is therefore always possible, and setting the threshold at 0 is too conservative.

Using **Fisher's exact test with a window size of 100kb** also fails to identify several genes with marine-freshwater divergent expressions. Note that the limit chosen for filtering these scores was the 99% upper quantile of the normal distribution around the mean of the scores. This set of results covers the largest proportion of the genome, and still performs worse. 100kb windows were used by Burke et al. [5] in the drosophila study, but clearly perform worse among the stickleback analyses.

The significantly diverging SNPs seems to be surpressed by the large amount of non-significant SNPs that can be found in most 100kb windows of the genome. A way of avoiding this problem might be to set a stricter upper quantile-score for each window than the 95% that is now used.

For more information on the importance of window size in the stickleback analyses I also ran a **cluster separation score analysis with 100k windows**. With a false discovery rate of 0.05, 8 regions covering 2 174 000 bases of the genome were identified. This is the largest set of regions obtained. 91% of Jones' strict set is overlapped by these regions, only marginally better than CSS with 2.5kb windows. More importantly, several genes are not identified by the 100kb CSS analyses. The broader analysis fails to identify small regions of genomic divergence in chromosome II and VII, where actual genes with marine-freshwater divergent expression exist (`Muc5b`).

All in all, the cluster separation scorer together with a sensible false discovery rate is the most accurate. Both methods identify regions of genomic divergence accurately under the right parameters. The size of sliding windows seems to be an important factor of the accuracy of my analyses. In contrast to the filtering options of false discovery rate and normal distribution upper quantile, the sliding window size is a parameter that needs to be decided at the beginning of analyses. A smaller window size seems preferable, but several sizes should be tested when performing analyses.

7.1.2 Quality of analyses done on drosophila SNPs

For the drosophila SNPs I performed cluster separation scoring on 100kb windows based on a pairwise comparison of allele frequency. The correlation of these scores with FET scores calculated by Burke et al. [5] are given chromosome-wise in table 7.1, and a figure visualizing the scores on top of each other are supplied in figure 6.8. Again the correlation scores can't be said to give us much information, except from that their is a somewhat clear positive correlation between the scores.

Burke et al. [5] did not supply regions with significant genomic divergence in their supplementary material, but the score peaks clearly identify the same regions in figure 6.8. Neither significance scoring or standard deviation estimation is possible based on the output from cluster separation scoring drosophila data. Therefore none of my tools for identifying regions based on scores can be used. The authors of the drosophila study did supply a list over all SNP-locations with a -log10 FET score of more than 5, 1945 positions in total. To get some sort of comparison I extracted the top 0.1% windows in terms of cluster separation score and checked how many overlapped with these positions. 23.6% (826/3491) of the points fall

Chromosome	Pearson's r correlation
chrX	0.619
chr2L	0.5
chr2R	0.528
chr3L	0.34
chr3R	0.49

Table 7.1: Pearson's r correlation between cluster separation scores from HyperBrowser tool and FET-scores published by Burke et al. [5] for Drosophila Melanogaster.

within the regions. The regions cover 2.7% of the genome, so there is an indication that the methods correlates. Unfortunately, no further analysis and comparison is possible without extensive gene analysis like what was done for sticklebacks by Jones et al. [17].

7.2 Possible weaknesses in analyses

As detailed, the results of my analyses are broader than the results obtained in previously published studies on sticklebacks and drosophila. This section discusses possible imperfections in analyses using my tools, and to some extent the risk of these imperfections.

The quality of SNP-data and assembly The quality of analyses based on SNPs naturally hinges on the quality of SNPs. The quality of reference genome assembly plays a natural part, gaps in the genome is not considered when running analysis. In both the stickleback and drosophila study sequences that have not be determined to be a part of a specific chromosome are left out, and hence I might be missing areas of significant genomic divergence. Improving the quality of variant callers and algorithms for verifying and filtering SNPs is not within the scope of this project. However, all else being perfect, this could be the deciding factor in the accuracy of my results. As described in chapter 2 the accuracy of variant callers are not perfect, and the probability of errors are large. Due to the quantitative nature of the algorithms chosen for my tool, single SNP-locations with errors should not have a very large impact on the results. More important is achieving the best possible assembly of the genome, increasing the amount of data to analyze.

Methods for filtering scores I have implemented two methods for filtering out the most divergent regions of the genome based on certain parameters. Both depend on statistical measures of the scores on each window, either the standard deviation of a score or the p-value. The data supplied for drosophila was pre-categorized into the three populations. This makes it impossible to filter cluster separation scores using my tools, a weakness of the CSS-methodology. More importantly, it underlines the importance of proper data for doing these kind of analyses. Setting the threshold proved to be non-trivial for either method. An objective for these analyses is to narrow down the amount of genetic material to analyze for divergence. When using Fisher's exact test score with 2 500 basepair windows, filtering was based on a upper quantile of set of a normal distribution based on all scores. To achieve a set of regions similar in size to the other results this limit had to be set to be the 10^{-9}% of this distribution. Conversely, for the Fisher's exact test scores done on 100kb windows, the 1% upper quantile was used. This makes picking the right parameters difficult for any user. By using the false discovery rate as threshold, the size of results are part of the calculations, and more conform results are obtained across window sizes. Implementing a filtering tool based on a false discovery rate for Fisher's exact test scores is therefore a possible improvement.

Implementation weaknesses and bugs Implementing tools in HyperBrowser has some advantages, and some disadvantages. One disadvantage is the inability to perform genome-wide analysis on sticklebacks due to the amount of memory required when pre-processing the data. This issue will probably be fixed in future versions of the HyperBrowser, independently of my tools. The risk of bugs in the code rises with the amount of code. For the core algorithm implementations tests have been created and run to ensure correct behavior. This is not the case for the HyperBrowser specific implementation, due to the complex test data structures required. However, several large-scale analyses have been performed using the tools with intended behavior. An important risk factor for the HyperBrowser-specific implementations is the fact that large parts of the code is dependent on the behavior of other parts of the code-base. A change in these parts might break my tools. A way of mitigating this risk is through audits of my code performed by core HyperBrowser developers.

7.3 Identifying regions of genomic divergence in other species

My methods and analyses are based on previous studies on three-spine sticklebacks and fruit flies. The methodologies are not species-specific. Scoring genomic divergence based on SNPs is not contingent on species. With the rapid rise of available SNP-data for individuals within species creates many possibilities for interesting use of my tools.

From a strict informatics perspective, there are two obstacles for performing analysis on other species based on my tools. First, the reference genome for the species has to be integrated in HyperBrowser. Many reference genomes are already integrated, and the core developers respond quickly on requests for new genomes. Secondly, the SNP-data needs to be in the right format. As detailed in chapter 5 file format is a common issue in bioinformatics. The file format used is simplistic however, and existing HyperBrowser tools exists that can convert from any tabular-based format into mine. I have also created two tools for converting more specific formats, the formats used in the stickleback and fruit fly studies.

A more complex issue that may arise when trying to perform analyses on other SNP-data is the choice of pairwise comparison metric in the cluster separation scorer. Two such metrics have been implemented, both of which should be usable in most studies. While creating new pairwise comparison metrics is perfectly possible, it currently requires a HyperBrowser developer. The possibility of supplying the code for a comparison metric as input for the tool could be a consideration for future versions of the tool.

All in all, these issues are by making it easy to understand how the tool functions and lowering the threshold of contacting developers. Using the tools on data from other species is certainly possible and could give interesting new information.

7.4 Is the tool possible to use for the target audience?

The target audience for my tools are biologists with some insight in bioinformatics. If the interface and requirements for using my tool is too complex for this target audience the value of the tool is greatly diminished. By implementing the tool in Galaxy-based HyperBrowser there is an advantage of an interface similar to what many biologists have seen before (Galaxy). To get some insight in the usability of the tool I asked a biostatistics student to perform some tasks while observing the behavior. Five tasks were given:

1. Download stickleback SNPs for one chromosome from the stickleback study-website
2. Upload stickleback SNPs to HyperBrowser
3. Get significant regions of genomic divergence using HyperBrowser tools for cluster separation scoring
4. Get significant regions of genomic divergence using HyperBrowser tools for Fisher's exact testing
5. Compare the tools using the HyperBrowser analysis-tool

The tasks vary from being totally independent from HyperBrowser to using specific tools created through this project. The feedback can be divided in three parts: issues specific to my tools, issues stemming from the HyperBrowser framework and general usability issues in the field of on-line bioinformatics tools.

General usability issues in bioinformatics Throughout all the tasks the problem when navigating was the overwhelming amount of options and information given. This requires the user to be able to quickly disseminate what is relevant and irrelevant for the task at hand. The front page of the stickleback study-website reads as a long-form FAQ-section [44], this proved useful, but might alienate impatient users with specific purposes. In general the amount of information, links and options provided at each page seems to be a general issue of bioinformatics tools, often overwhelming the users. Placement of search-boxes and help-links is therefore essential. Such design decisions is in most cases decided by the framework used, in my case Galaxy provides most of the design features.

Usability issues in HyperBrowser When navigating the HyperBrowser, the student quickly found the Help-page. This help-page focuses on using the perform analysis, which arguably is the main feature of HyperBrowser. Guidance is given using quite complex example workflows, which means that figuring out simple tasks like uploading data can prove cumbersome. Tools are divided in two sections, one for HyperBrowser-specific tools and one for general Galaxy-tools. This separation might seem intuitive to developers, but proved confusing to the student. In each section there are several headlines for groups of tools, 14 for HyperBrowser and 37 for the galaxy section. Several of the groups have seemingly overlapping generic names, like "Export/import", "Get Data" and "Send data". Fewer sections, or alternatively higher depth, would probably increase the usability, and seems to

already be on its way on other instances of HyperBrowser. The most helpful feature for finding specific features is the search box on the top of the tool frame. When the user found it, it quickly became the preferred way of navigating the HyperBrowser. By guessing the names of tools to use she eventually found all tools needed. This gives importance to proper naming of tools, avoiding acronyms and using several keywords.

Doing analysis with pre-existing HyperBrowser tools required heavy guidance, which implies that the bonus of already implemented analysis tools might not be as large as assumed in the implementation strategy.

Usability issues for my tools Some tool-specific areas of improvement was found during testing with the biostatistics student. Giving proper feedback when no fitting data to do analysis through my tools is found in the history of the user is important. A simple FAQ should also be available on the same page as the tool. Using text-boxes instead of check-boxes or drop-down menus actually improved readability. The most important lesson made from testing the usability of my tools was probably that naming is key. Tools should have names including the most probable keywords for the purpose of the tool.

Usability conclusion The major factors for the usability of mu tool is dependent on the framework, which is improved on continuously by other developers. Compared to running command-line tools or coding scripts my tool gives the user a lot more feedback without requiring much programming knowledge. My tools are therefore a good step on the way for letting biologists be biologists and at the same time ensuring transparency and reproducibility of analyses. Some small adjustments can be large improvements usability-wise. Examples of this is naming of tools and providing a basic FAQ for a tool on the tool-page.

7.5 Conclusion

Two methods for identifying regions genomic divergence have been implemented in the HyperBrowser framework for on-line analysis. Results from analysis done on stickleback and drosophila data shows that both tools identify regions of genomic divergence. The cluster separation scoring methodology is slightly more accurate than the Fisher's exact test scorer. The sliding window size used in both types of analyses has been identified as an important factor for accuracy of results. The major obstacles for using of the tool is the requirements for providing data in the right format and the general complexity of using bioinformatics tools. The tools

can be improved through parallelization, better memory-handling and improved interface design of the framework. By completing this project, I have provided an opportunity for biologists to do transparent and reproducible genome-wide analysis for identifying regions of genomic divergence.

8 Future Work

I have shown that both cluster separation scoring and Fisher's exact test are possible to implement and usable for identifying regions of genomic divergence across populations. This forms the basis for some new research questions and possible improvements of the tools.

Improvements of the tools Since both tools score sliding windows of the genome independently of each other there is a potential for optimization through parallelization. Some parallelization features are already implemented in the Hyperbrowser, but implementing for my tools requires some modifications. By calculating windows in parallel the total running time will be dependent on the window which takes the longest time, and the whole analysis for the stickleback genomes should decrease sharply. The memory issues forcing the user to do chromosome-wise analyses on sticklebacks could easily be solved by only preprocessing and storing SNPs for one chromosome at the time behind the scenes, and should be worth looking into in the future.

Implementing the self-organizing map-method used by Jones et al. [17] described in chapter 3 in combination with my tools might yield even more accurate region boundaries. A HyperBrowser tool for listing annotated genes within supplied regions of a genome might also be helpful. Biological analysis is required for further comparison of the accuracy of the methods, and their biological implications.

New applications Finding new species to do this analysis on is definitely an interesting future field of study. Genomic sequences of individuals are popping up everywhere, at the University of Oslo a research group at the institute of biology recently got a grant for sequencing 1000 individual cod and salmons [38]. Using my tools to perform analyses on such data could help biologists acquire new knowledge on parallel evolution within the species.

A

Overview of analyses and tools created in HyperBrowser

A.1 Analyses

I performed several different analyses using my tools on HyperBrowser branch available at `http://hyperbrowser.uio.no/comparative`. Each of the analyses are made available through a Galaxy Published Page, and all jobs can therefore be imported and inspected by anyone. Below follows a list over analyses and corresponding Galaxy Published Pages. The pages can also be found by clicking your way from the HyperBrowser branch->"Shared Data"->"Published pages". For source code of analyses done please see appendix C.

- Cluster separation scoring sticklebacks:
  `http://hyperbrowser.uio.no/comparative/u/torkilve/p/stickleback-css`
- Fisher's exact test scoring on sticklebacks:
  `http://hyperbrowser.uio.no/comparative/u/torkilve/p/stickleback-fet`
- Calculating overlaps between different results for sticklebacks
  `http://hyperbrowser.uio.no/comparative/u/torkilve/p/overlaps`
- Calculating cluster separation scoring on drosophila:
  `http://hyperbrowser.uio.no/comparative/u/torkilve/p/drosophila-css`

A.2 Tools implemented

All tools described in this thesis are available on a special HyperBrowser branch at `http://hyperbrowser.uio.no/comparative`. The tools can be found under the "Restricted and experimental tools" section. Below follows a list of tools and their purposes. For info on usage, some of the tools have a FAQ on their page, so click your way in:

- **Cluster Separation Score** A tool for calculating cluster separation scores between two groups of individuals on windows of the genome.
- **Significant C S S Regions** Filtering cluster separation scores obtained by "Cluster Separation Score"-tool into regions
- **Fisher's exact test S N P tool** A tool for scoring windows of the regions based on a Fisher's exact test on every SNP-location across two groups of individuals.
- **Filter Fisher scores** Filtering fisher scores into regions based on results obtained from "Fisher's exact test S N P tool"
- **Convert Stickleback Snps to Gtrack** Converting data in Jones et al. [17] format to gtrack valued points for CSS and FET-scoring.

B

Detailed data for stickleback analyses

The stickleback analyses provided large amount of data. This chapter includes visualization of all regions found by the four different methods described in Chapter 6. A table showing the Pearson's r correlations between my calculations of cluster separation scores and Jones' is also provided.

B.1 Regions produced by the different methods

In Figure B.1 through B.4 regions produced by cluster separation scoring with a false discovery rate of 0.02(CSS FDR0.02) and 0.05(CSS FDR0.05)and results from Fisher's exact test with 2.5kb windows and 100kb windows are visualized together with the strict set of regions provided by Jones et al. [17]. Note that chromosomes and regions with no results are not shown.

B.2 Correlations of CSS with Jones' CSS

Table B.1 show the computed Pearson's r correlation score for each chromosome between Cluster Separation Scoring performed by my tool and the scores made available by Jones et al. [17]. The same parameters were used to get a best possible match.

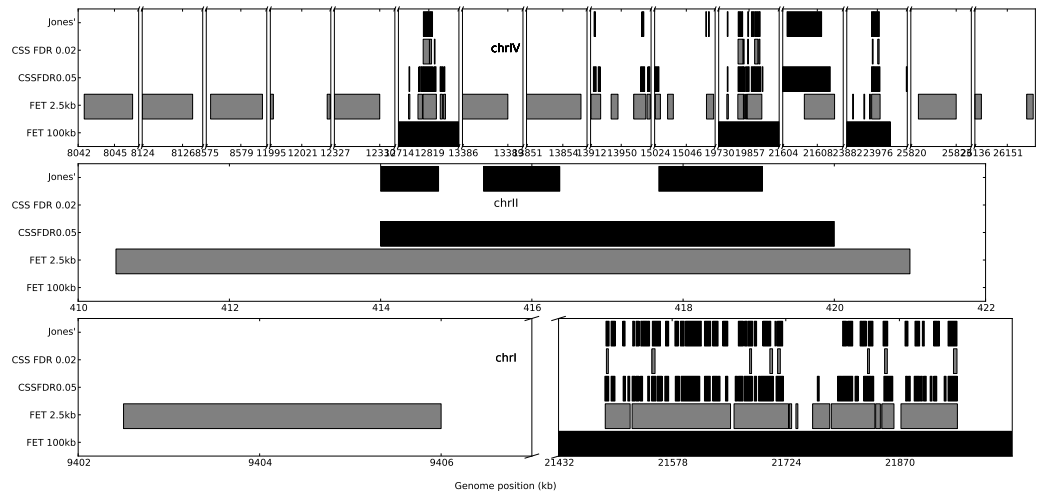

Figure B.1: Chromosome I through IV

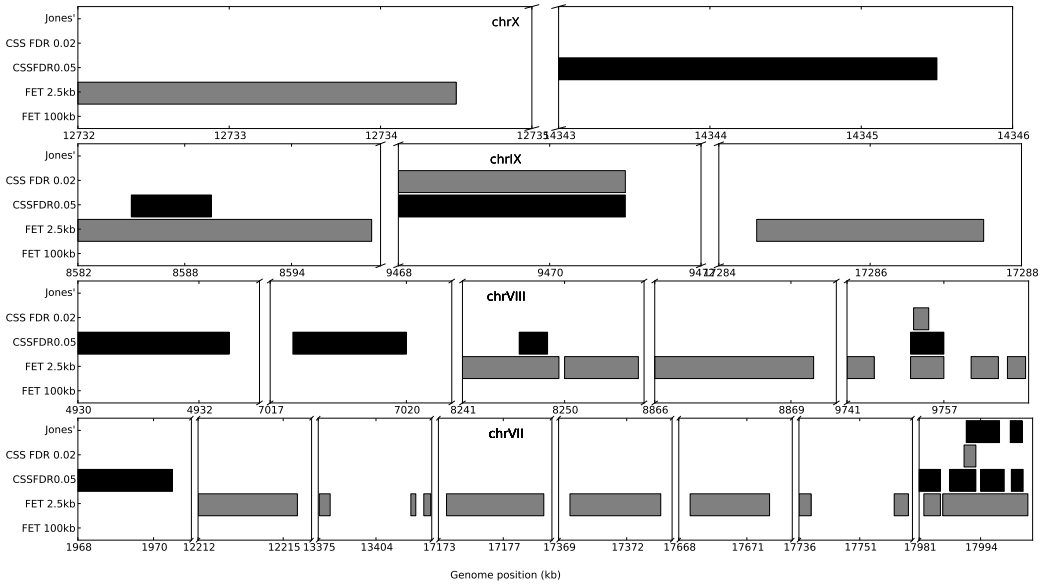

Figure B.2: Chromosome VII through X

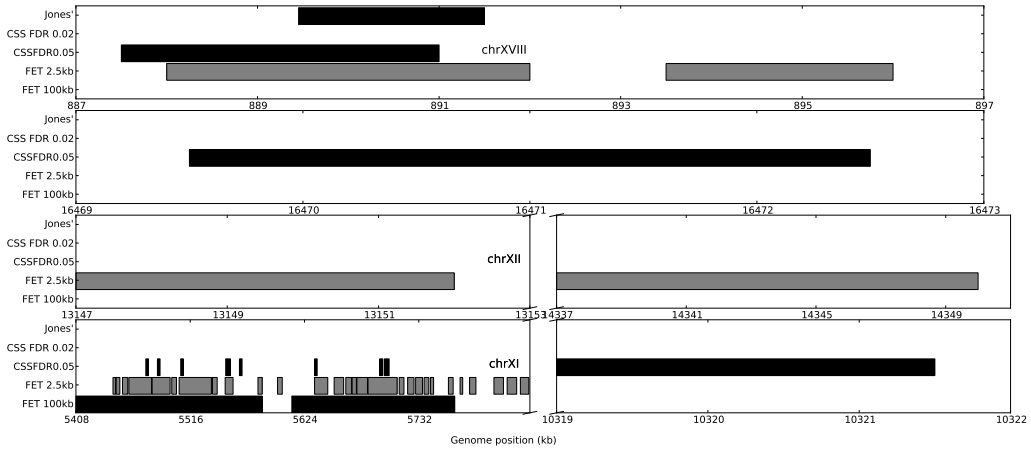

Figure B.3: Chromosome XI through XVIII

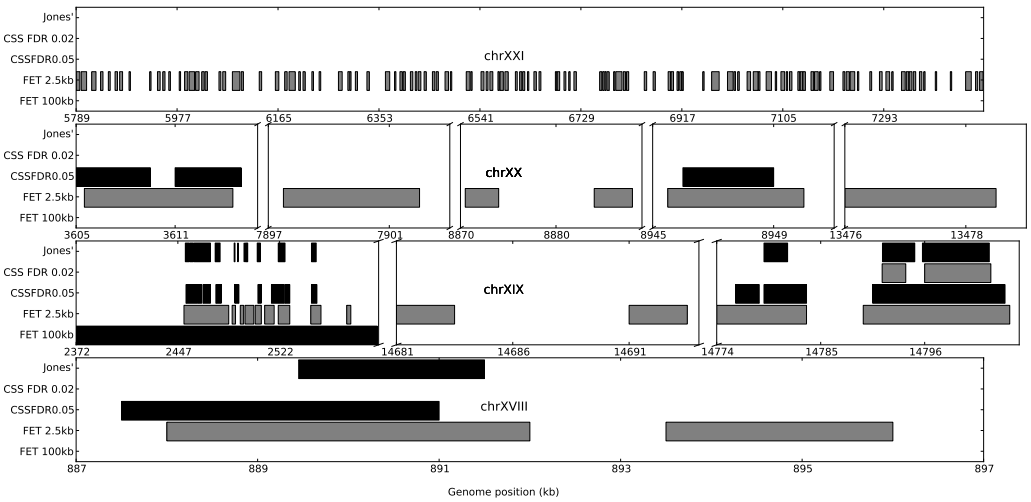

Figure B.4: Chromosome XVIIII through XXI

Chromosome	Correlation
chrI	0.74
chrII	0.24
chrIII	0.15
chrIV	0.11
chrVII	0.73
chrVIII	0.59
chrIX	0.53
chrX	0.28
chrXI	0.7
chrXII	0.62
chrXIII	0.33
chrXIV	0.13
chrXV	0.11
chrXVI	0.32
chrXVII	0
chrXVIII	0.1
chrXIX	0.56
chrXX	0.69
chrXXI	0.88
chrUn	0.56

Table B.1: Pearson's r correlation for cluster separation scores on stickleback chromosomes

C

Source code

Below follows python code used for calculating cluster separation scores and Fisher's exact test scores. The rest of the source code, including all HyperBrowser-specific code is available as a zip-archive at `http://hyperbrowser.uio.no/dev2/static/downloads/TorkilMasterSourceCode.zip`

C.1 Source code for divergence scoring

Code Listing C.1: Methods for Cluster Separation Scoring and significance computations

```
from sklearn.manifold import MDS
import numpy as np
import math


def separationScore(trA, trB,drosophila=False, debug=False, l=100):
    """Main method for calculating cluster separation score"""

    #Obtain data
    tracks, aSize, bSize = getValueArrays(trA, trB)

    if aSize == 0:
        return 0, 0, 0

    #Perform pairwise comparisons
    matrice = compareAll(tracks, drosophila, l)
    if type(matrice) == type(False):
        return 0, 0, 0

    #Perform multi-dimensional scaling
    mdsScore = calculateMDS(matrice)
    if type(mdsScore) != type(np.array([])):
        return 0,0,0
    assert len(mdsScore) == aSize+bSize
    return computeCSSindexes(mdsScore, xrange(aSize), xrange(aSize, aSize+bSize)), mdsScore, aSize


def calculateMDS(matrice, s=False):
    if type(matrice) ==type(False) or np.sum(matrice) == 0:
        return 0
    mds = MDS(metric=True)
    mdsScore = mds.fit_transform(matrice)
    return mdsScore

def dist(matrice, a, b, results=None):
    return math.sqrt((matrice[a][0]-matrice[b][0])**2 + \
                (matrice[a][1] - matrice[b][1])**2), results
```

```
38
39 def computeCSSindexes(mdsScore, gAi, gBi, results=None):
40     betweenDist = 0.
41     m = len(gAi)
42     n = len(gBi)
43
44     #Calculate between-group distance
45     for a in gAi:
46         for b in gBi:
47             d, results = dist(mdsScore, a, b, results)
48             betweenDist += d
49
50     betweenVal = betweenDist/(m*n)
51
52     #Calculate within-group distances
53     if m > 1:
54         aDist = 0.
55         for i in range(m-1):
56             d, results = dist(mdsScore,gAi[i],gAi[i+1], results)
57             aDist += d
58
59         aVal = aDist/(m*m*(m-1))
60     else:
61         aVal = 0
62
63     if n > 1:
64         bDist = 0.
65
66         for i in range(n-1):
67             d, results = dist(mdsScore, gBi[i], gBi[i+1], results)
68             bDist += d
69
70         bVal = bDist/(n*n*(n-1))
71
72     else:
73         bVal = 0
74
75     abVal = (m+n)*(aVal+bVal)
76
77     return betweenVal - abVal
78
79
80 def getValueArrays(trackviewA, trackviewB):
81     Avals = trackviewA._valList.astype(np.float64)
82     Bvals = trackviewB._valList.astype(np.float64)
83     if Avals.size == 0:
84         return Avals, 0, 0
85     aIds = trackviewA._extraLists["genomeid"]
86     bIds = trackviewB._extraLists["genomeid"]
87     aSize = len(set(aIds))
88     bSize = len(set(bIds))
89
90     aTracks = [Avals[aIds==x] for x in list(set(aIds))]
91     bTracks = [Bvals[bIds==x] for x in list(set(bIds))]
92     return aTracks+bTracks, aSize, bSize
93
94 def compareAll(tracks, drosophila = False, limit=100):
95     size = len(tracks)
96     d = np.zeros([size, size])
97     j = 0
98     for track in tracks:
99         i = 0
100        for trackB in tracks:
101            if not drosophila:
102                sc, validated = compareCount(track, trackB)
103                d[j][i] = sc
104            else:
105                sc, validated = compareFreq(track, trackB)
106                d[j][i] = sc
107
108            i += 1
109        j += 1
110    return d
111
112
```

```
113
114
115 def compareCount(a,b):
116     product = a*b
117     added = a+b
118     score = np.sum(product==-9)
119     validated = np.sum(added>-1000)
120     return score, validated
121
122 #Slooooooooooooow version to show the point of vectorization
123 def compareCount2(a,b):
124     count = 0
125     i = 0
126     asize = len(a)
127     while i < asize:
128         count += (a[i]*b[i])==-9
129         i +=1
130     return count
131
132
133 def compareFreq(trackA, trackB):
134     score = np.sum(np.abs(trackA-trackB))/float(trackA.size)
135     return score, len(trackA)
136
137
138 def significanceAll(mdsScore,score,treshhold, aSets):
139     """Method that performs significance computations for all permutations of the window"""
140     hits = 0
141     results = {}
142     nScores = 0
143     indices = xrange(mdsScore.shape[0])
144     for a in aSets:
145         b = np.delete(indices, a)
146         newScore = computeCSSindexes(mdsScore, a, b, results)
147         nScores += 1.
148         if newScore > score:
149             hits += 1
150             if hits > treshhold:
151                 break
152     p = float(hits)/nScores
153     return p, nScores
154
155
156
157
158 def significanceTreshold(mdsScore, aSize,score=0, runs=5000, treshold=10):
159     """Method that estimates significance based on sequential Monte Carlo"""
160     import random
161     hits = 0
162
163     nTracks = mdsScore.shape[0]
164     results = {}
165
166     nScores = 0
167     #Sequential Monte Carlo
168     while hits < treshold and nScores < runs:
169         indices = xrange(nTracks)
170         gAi = random.sample(indices, aSize)
171         gBi = np.delete(indices, gAi)
172
173         newscore = computeCSSindexes(mdsScore, gAi, gBi, results)
174         if newscore > score:
175             hits += 1
176         nScores += 1
177     p = float(hits+1)/nScores
178
179     return p, nScores
```

C.2 Fisher's Exact Test Scorer Source Code

Code Listing C.2: Python implementation of Fisher's exact test scoring genomic regions

```
from scipy.stats import fisher_exact, scoreatpercentile
import numpy as np
import math

def fisherScorer(trA, trB):
    """Method which extracts value arrays and sizes of groupings before calculating
    fisher scores"""
    Avals = trA._valList.astype(np.int)
    Bvals = trB._valList.astype(np.int)
    aStarts = trA._startList
    bStarts = trB._startList

    aSize = 0
    bSize = 0
    for a in aStarts:
        if a == aStarts[0]:
            aSize += 1
        else:
            break
    for b in bStarts:
        if b == bStarts[0]:
            bSize += 1
        else:
            break

    if aSize == 0 or bSize == 0:
        return 0, 1
    return fisherScorerEach(Avals, Bvals, aSize, bSize)

def fisherScorerEach(Avals, Bvals, aSize, bSize):
    """Calculates fisher exact scores for every position"""
    i = 0
    j = 0
    d = 0
    scores = []
    while i+aSize <= Avals.size and j+bSize <= Bvals.size:
        f = fisherExactTest(Avals[i:i+aSize], Bvals[j:j+bSize])
        if np.isinf(f) or np.isnan(f):
            d += 1
        else:
            scores.append(-1*math.log(f,10))
        i += aSize
        j += bSize
    scores = np.array(scores)
    stddev = np.random.choice(scores, 100).std()
    return scoreatpercentile(scores, 95), stddev


def fisherExactTest(a, b):
    """Translate SNP-data into contingency table"""
    a1 = np.sum(a==3)
    a2 = np.sum(a==-3)
    b1 = np.sum(b==3)
    b2 = np.sum(b==-3)
    return fisherExact(a1, a2, b1, b2)



def fisherExact(a,b,c,d):
    """Calculate fisher exact test score based on contingency table"""
    import math
    f = math.factorial
    denom = f(a)*f(b)*f(c)*f(d)*f(a+b+c+d)
    nom = f(a+b)*f(c+d)*f(a+c)*f(b+d)
    return float(nom)/denom
```

Bibliography

Journal papers

[1] Gonçalo R Abecasis, David Altshuler, Adam Auton, Lisa D Brooks, Richard M Durbin, Richard A Gibbs, Matt E Hurles, and Gil A McVean. "A map of human genome variation from population-scale sequencing". In: *Nature* 467.7319 (Oct. 2010), pp. 1061–1073.

[2] André Altmann, Peter Weber, Daniel Bader, Michael Preuss, Elisabeth B Binder, and Bertram Müller-Myhsok. "A beginners guide to SNP calling from high-throughput DNA-sequencing data". In: *Human genetics* 131.10 (Oct. 2012), pp. 1541–1554.

[3] Yoav Benjamini and Yosef Hochberg. "Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 57.1 (Jan. 1995), pp. 289–300.

[4] David R Bentley et al. "Accurate whole human genome sequencing using reversible terminator chemistry". In: *Nature* 456.7218 (Nov. 2008), pp. 53–59.

[5] Molly K Burke, Joseph P Dunham, Parvin Shahrestani, Kevin R Thornton, Michael R Rose, and Anthony D Long. "Genome-wide analysis of a long-term evolution experiment with Drosophila". In: *Nature* 467.7315 (Sept. 2010), pp. 587–590.

[6] S. Dey and I. Mareels. "Reduced-complexity estimation for large-scale hidden Markov models". In: *IEEE Transactions on Signal Processing* 52.5 (2004), pp. 1242–1249.

[7] "Finishing the euchromatic sequence of the human genome". In: *Nature* 431.7011 (Oct. 2004), pp. 931–945.

[8] R. A. Fisher. "On the Interpretation of x2 from Contingency Tables, and the Calculation of P". In: *Journal of the Royal Statistical Society* 85.1 (Jan. 1922), pp. 87–94.

[9] P. Flicek, M. R. Amode, D. Barrell, K. Beal, S. Brent, D. Carvalho-Silva, P. Clapham, G. Coates, S. Fairley, S. Fitzgerald, L. Gil, L. Gordon, M. Hendrix, T. Hourlier, N. Johnson, A. K. Kahari, D. Keefe, S. Keenan, R. Kinsella, M. Komorowska, G. Koscielny, E. Kulesha, P. Larsson, I. Longden, W. McLaren, M. Muffato, B. Overduin, M. Pignatelli, B. Pritchard, H. S. Riat, G. R. S. Ritchie, M. Ruffier, M. Schuster, D. Sobral, Y. A. Tang, K. Taylor, S. Trevanion, J. Vandrovcova, S. White, M. Wilson, S. P. Wilder, B. L. Aken, E. Birney, F. Cunningham, I. Dunham, R. Durbin, X. M. Fernandez-Suarez, J. Harrow, J. Herrero, T. J. P. Hubbard, A. Parker, G. Proctor, G. Spudich, J. Vogel, A. Yates, A. Zadissa, and S. M. J. Searle. "Ensembl 2012". In: *Nucleic Acids Research* 40.D1 (Nov. 2011), pp. D84–D90.

[10] Jeremy Goecks, Anton Nekrutenko, and James Taylor. "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences". In: *Genome Biology* 11.8 (Aug. 2010), R86.

[11] Sveinung Gundersen, Matúš Kalaš, Osman Abul, Arnoldo Frigessi, Eivind Hovig, and Geir K. Sandve. "Identifying elemental genomic track types and representing them uniformly". In: *BMC Bioinformatics* 12.1 (Dec. 2011), p. 494.

[12] John H. Halton. "Sequential Monte Carlo Techniques For The Solution Of Linear Systems". In: *Journal of Scientific Computing* 9 (1992), pp. 213–257.

[13] David Hilbert. "Ueber die stetige Abbildung einer Line auf ein Flächenstück". In: *Mathematische Annalen* 38.3 (1891), pp. 459–460.

[14] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing In Science & Engineering* 9.3 (2007), pp. 90–95.

[15] John P A Ioannidis, David B Allison, Catherine A Ball, Issa Coulibaly, Xiangqin Cui, Aedín C Culhane, Mario Falchi, Cesare Furlanello, Laurence Game, Giuseppe Jurman, Jon Mangion, Tapan Mehta, Michael Nitzberg, Grier P Page, Enrico Petretto, and Vera van Noort. "Repeatability of published microarray gene expression analyses". In: *Nature genetics* 41.2 (Feb. 2009), pp. 149–155.

[16] D. Janzen and H. Saiedian. "Test-driven development concepts, taxonomy, and future direction". In: *Computer* 38.9 (2005), pp. 43–50.

[17] Felicity C. Jones, Manfred G. Grabherr, Yingguang Frank Chan, Pamela Russell, Evan Mauceli, Jeremy Johnson, Ross Swofford, Mono Pirun, Michael C. Zody, Simon White, Ewan Birney, Stephen Searle, Jeremy Schmutz, Jane Grimwood, Mark C. Dickson, Richard M. Myers, Craig T. Miller, Brian R.

Summers, Anne K. Knecht, Shannon D. Brady, Haili Zhang, Alex A. Pollen, Timothy Howes, Chris Amemiya, Broad Institute Genome Sequencing Platform & Whole Genome Assembly Team, Eric S. Lander, Federica Di Palma, Kerstin Lindblad-Toh, and David M. Kingsley. "The genomic basis of adaptive evolution in threespine sticklebacks". In: *Nature* 484.7392 (Apr. 2012), pp. 55–61.

[18] W. James Kent, Charles W. Sugnet, Terrence S. Furey, Krishna M. Roskin, Tom H. Pringle, Alan M. Zahler, Haussler, and David. "The Human Genome Browser at UCSC". In: *Genome Research* 12.6 (June 2002), pp. 996–1006.

[19] T. Kohonen. "The self-organizing map". In: *Proceedings of the IEEE* 78.9 (1990), pp. 1464–1480.

[20] Eric S. Lander et al. "Initial sequencing and analysis of the human genome". In: *Nature* 409.6822 (Feb. 2001), pp. 860–921.

[21] Elaine R. Mardis. "A decade/'s perspective on DNA sequencing technology". In: *Nature* 470.7333 (Feb. 2011), pp. 198–203.

[22] Jason O'Rawe, Tao Jiang, Guangqing Sun, Yiyang Wu, Wei Wang, Jingchu Hu, Paul Bodily, Lifeng Tian, Hakon Hakonarson, W. Evan Johnson, Zhi Wei, Kai Wang, and Gholson J. Lyon. "Low concordance of multiple variant-calling pipelines: practical implications for exome and genome sequencing". In: *Genome Medicine* 5.3 (Mar. 2013), p. 28.

[23] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (Oct. 2011), pp. 2825–2830.

[24] K. D. Pruitt, T. Tatusova, G. R. Brown, and D. R. Maglott. "NCBI Reference Sequences (RefSeq): current status, new features and genome annotation policy". In: *Nucleic Acids Research* 40.D1 (Nov. 2011), pp. D130–D135.

[25] L. Rabiner and B.-H. Juang. "An introduction to hidden Markov models". In: *IEEE ASSP Magazine* 3.1 (1986), pp. 4–16.

[26] Geir K. Sandve, Sveinung Gundersen, Halfdan Rydbeck, Ingrid K. Glad, Lars Holden, Marit Holden, Knut Liestøl, Trevor Clancy, Egil Ferkingstad, Morten Johansen, Vegard Nygaard, Eivind Tøstesen, Arnoldo Frigessi, and Eivind Hovig. "The Genomic HyperBrowser: inferential genomics at the sequence level". In: *Genome Biology* 11.12 (Dec. 2010), R121.

[27] Andrea Sboner, Xinmeng J. Mu, Dov Greenbaum, Raymond K. Auerbach, and Mark B. Gerstein. "The real cost of sequencing: higher than you think!" In: *Genome Biology* 12.8 (Aug. 2011), p. 125.

[28] Vin de Silva and Joshua B Tenenbaum. "Global versus local methods in nonlinear dimensionality reduction". In: *Advances in neural information processing systems* 15 (2003), pp. 705–712.

[29] R Staden. "A strategy of DNA sequencing employing computer programs." In: *Nucleic Acids Research* 6.7 (June 1979), pp. 2601–2610.

[30] J Zhang and S Kumar. "Detection of convergent and parallel evolution at the amino acid sequence level". In: *Molecular biology and evolution* 14.5 (May 1997), pp. 527–536.

Other written references

[31] Kent Beck. *Test Driven Development: By Example*. 1st ed. Addison-Wesley Professional, Nov. 2002.

[32] I. Borg. *Modern Multidimensional Scaling: Theory and Applications*. Springer, Aug. 2005.

[33] Robert Brooker, Eric Widmaier, Linda Graham, and Peter Stiling. *Biology*. 1st ed. McGraw-Hill, Mar. 2007.

[34] Eivind Gard Lund. "An Extensible Framework for Comparative Analysis of Annotations". MA thesis. University of Oslo, 2011. URL: `https://www.duo.uio.no//handle/10852/8901`.

[35] E.M. Maximilien and L. Williams. "Assessing test-driven development at IBM". In: *25th International Conference on Software Engineering, 2003. Proceedings*. 2003, pp. 564–569.

Online references

[36] *26.4. The Python Profilers - Python v2.7.4 documentation*. URL: `http://docs.python.org/2/library/profile.html#module-cProfile` (visited on 04/27/2013).

[37] *3.2.3 Self-organizing maps*. URL: `http://www.girardin.org/cgv/report/report-27.html` (visited on 04/30/2013).

[38] *Biotek 2021 Research Grant Awarded to CEES Research Team - Centre for Ecological and Evolutionary Synthesis*. URL: http://www.mn.uio.no/cees/english/research/news/cees/biotek2021.html (visited on 04/30/2013).

[39] *cortesi - Visualizing binaries with space-filling curves*. URL: http://corte.si/posts/visualisation/binvis/ (visited on 04/30/2013).

[40] *Galaxy Project/Statistics - Galaxy Wiki*. URL: http://wiki.galaxyproject.org/Galaxy%20Project/Statistics (visited on 04/27/2013).

[41] *Genome Reference Consortium*. URL: http://www.ncbi.nlm.nih.gov/projects/genome/assembly/grc/ (visited on 04/30/2013).

[42] *GRCh37.p12 - Assembly - NCBI*. URL: http://www.ncbi.nlm.nih.gov/assembly/GCA_000001405.13/ (visited on 04/30/2013).

[43] Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. 2001-. URL: http://www.scipy.org/.

[44] *Kingsley Lab Stickleback Genome Browser*. URL: http://sticklebrowser.stanford.edu/ (visited on 04/27/2013).

[45] *The Genomic HyperBrowser (test version)*. URL: http://hyperbrowser.uio.no/test/ (visited on 04/30/2013).