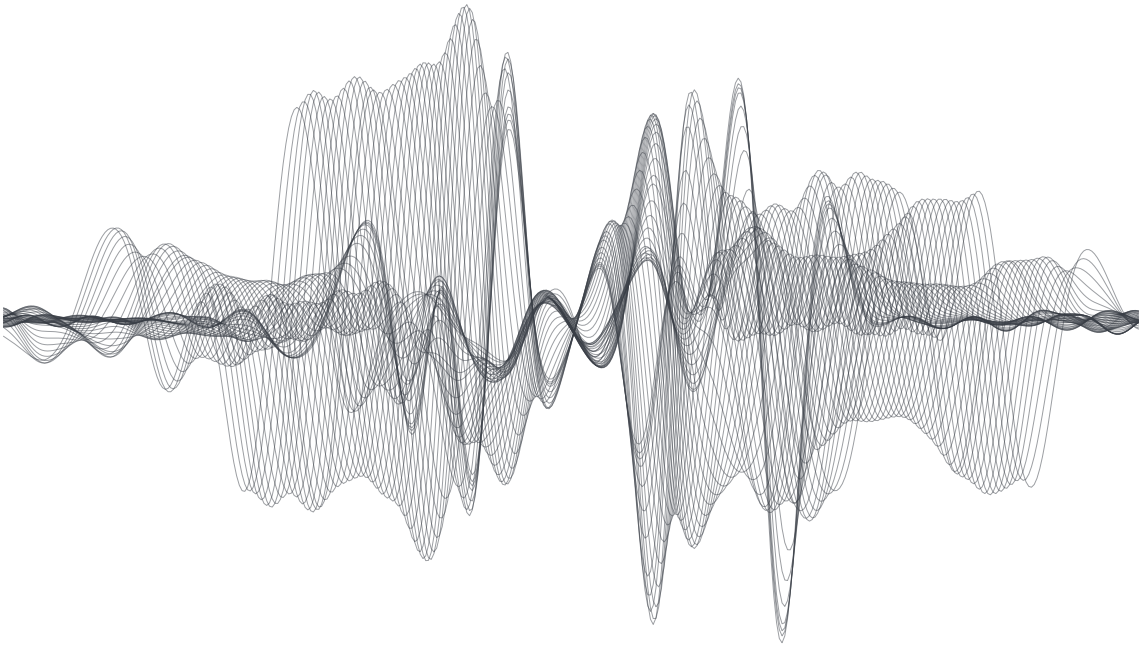
COMPARATIVE FUNCTIONAL GENOMICS
AND
ARTIFICIAL NEURAL NETWORKS
FOR THE STUDY OF THE
EVOLUTION OF CIS-REGULATION



PANOS FIRBAS NISANTZIS

Tesis Doctoral

Universidad Pablo de Olavide, Sevilla 2019

Programa de doctorado:
Biotecnología, Ingeniería y Tecnología Química

Centro Andaluz de Biología del Desarrollo

Regulación génica y morfogénesis

Supervisors:

José Luis Gómez-Skarmeta
Ignacio Maeso

Declaration of Authorship

I, Panos Firbas Nisantzis, declare that this thesis titled, "Comparative functional genomics and artificial neural networks for the study of the evolution of cis-regulation" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

As you set out for Ithaka
hope the voyage is a long one
full of adventure, full of discovery.

Laistrygonians and Cyclops,
angry Poseidon—don't be afraid of them:
you'll never find things like that on your way
as long as you keep your thoughts raised high,
as long as a rare excitement
stirs your spirit and your body.

Laistrygonians and Cyclops,
wild Poseidon—you won't encounter them
unless you bring them along inside your soul,
unless your soul sets them up in front of you.

Hope the voyage is a long one.
May there be many a summer morning when,
with what pleasure, what joy,
you come into harbors seen for the first time;
may you stop at Phoenician trading stations
to buy fine things,mother of pearl and coral, amber and ebony,
sensual perfume of every kind — as many sensual perfumes as you can;
and may you visit many Egyptian cities
to gather stores of knowledge from their scholars.

Keep Ithaka always in your mind.
Arriving there is what you are destined for.
But do not hurry the journey at all.
Better if it lasts for years,
so you are old by the time you reach the island,
wealthy with all you have gained on the way,
not expecting Ithaka to make you rich.

Ithaka gave you the marvelous journey.
Without her you would not have set out.
She has nothing left to give you now.

And if you find her poor, Ithaka won't have fooled you.
Wise as you will have become, so full of experience,
you will have understood by then what these Ithakas mean.

-Ithaka, C.P.Cavafy

UNIVERSIDAD PABLO DE OLAVIDE

Resumen

Centro Andaluz de Biología del Desarrollo

Regulación génica y morfogénesis

Doctor of Philosophy

Comparative functional genomics and artificial neural networks for the study of the evolution of cis-regulation

by Panos Firbas Nisantzis

La regulación transcripcional es el primer y quizás más importante paso de la regulación génica, una compleja serie de dinámicas que, en última instancia, controla qué cantidad de producto génico se expresará en una célula en cada momento. Los distintos niveles de actividad transcripcional y la acumulación diferencial de tránscritos resultante conducen a la diferenciación celular, la piedra angular de la vida multicelular.

La transcripción se regula mediante la interacción de distintas proteínas (factores en trans) que se unen al ADN en sitios específicos del genoma (elementos cis). Las primeras llamadas Factores de Trancripción y a los segundos Elementos Reguladores en Cis (TFs y CREs respectivamente, por sus siglas en inglés).

Estudiada desde la perspectiva de la evolución genómica, la regulación transcripcional es excepcionalmente interesante. Los CREs pueden evolucionar con relativa rapidez y se considera que sus cambios son el motor principal de la evolución morfológica en animales. Así, una pequeña modificación que afecte a un sitio de unión de un TF durante el desarrollo embrionario podría dar lugar a efectos en cascada y alterar profundamente la forma del organismo.

En este trabajo, nos enfrentamos a dos aspectos de este amplio campo de investigación.

Para la primera parte del trabajo, investigaremos la evolución de la regulación en cis en el origen de los vertebrados. Para ello, empleamos el exhaustivo conjunto de datos genómicos, transcriptómicos y epigenómicos que generamos para el anfioxo mediterráneo (Branchiostoma lanceolatum). El anfioxo es un organismo ideal para tales investigaciones ya que, por varias razones tanto genómicas como morfológicas, se puede considerar que es el linaje existente que guarda una mayor similitud con el ancestro común de todos los cordados. Contrapondremos los datos de anfioxo con la igualmente extensa colección de datos que hemos generado para el pez cebra, así como con datos adicionales de peces medaka y de ratón.

Para la segunda parte, nos centramos en el problema de identificar sitios de unión de TFs en el genoma. Aplicamos un enfoque matemático de vanguardia, una red de convolución o red neuronal (NN) que integra al mismo tiempo información de secuencia genómica con información de señal de ATAC-seq, para predecir con precisión sitios reales de unión de TFs usando únicamente experimentos de ATAC-seq. En este trabajo presentaremos nuestra NN y los resultados obtenidos al compararla con otras técnicas actuales. Estos resultados muestran que nuestro método puede resultar muy valioso en análisis genómicos actuales, ya que permite reemplazar con un solo experimento de ATAC-seq múltiples experimentos de ChIPseq, reduciendo de manera muy significativa los costes y el tiempo de los experimentos de secuenciación. Además, nuestra NN puede ser entrenada primero en una especie con datos de ChIPseq previamente disponibles y usarse después en otras especies diferentes, un enfoque que es particularmente atractivo para realizar comparaciones evolutivas, especialmente en el caso de nuevos organismos modelo.

Comparative functional genomics and artificial neural networks for the study of the evolution of cis-regulation

Transcriptional regulation is the first and perhaps most important step in gene regulation, a complex set of dynamics that ultimately controls how much of a gene's product a cell will produce at any moment in time. Differential gene transcript levels and the resulting accumulating differences drive cell differentiation which is the cornerstone of multicellular life.

Transcription regulation happens through the interplay of trans- acting proteins which bind DNA on specific cis-acting sites of the genome. We call the first Transcription Factors (TFs) and the second Cis-Regularoty Elements (CREs). The study of transcription regulation is exceptionally interesting through the lens of genomic evolution. CREs evolve faster over time and are considered the primary driver of animal form evolution. A relatively small change that influences a target site for a TF during development might have cascading effects and dramatically change the organism's form.

In this work, we face two parts of this broad research field. For the first part of the work, we will investigate the evolution of cis-regulation at the root of the vertebrate tree. To do this, we will rely on a very thorough dataset of genomic, transcriptomic and epigenomic data that was generated for the Mediterranean amphioxus (*Branchiostoma lanceolatum*) that includes a de novo sequenced and assembled genome. Amphioxus is an ideal organism for such inquiries since, for a number of good reasons, it can be considered to be very similar to what the common ancestor of all chordates once was.We will juxtapose this amphioxus dataset against an equally exhaustive dataset on zebrafish, as well as additional data, from medaka and mouse.

For the second part, we focus on the problem of identifying TF binding sites on the genome. We apply a state of the art mathematical approach, a convolution network or neural network, to integrate genomic sequence information and ATAC-seq signal information at the same time to accurately predict real TF binding sites. This can prove to be a very valuable tool in modern genomic analysis since many CHIPseq assays for different TFs can be replaced with a single ATAC-seq experiment, significantly cutting costs in time and sequencing. Furthermore, this tool can be trained in a species where data is available and then used in a

different species, an approach that is particularly appealing for evolutionary comparisons and even more so for novel model organisms. We will present the NN and results obtained when comparing it to the current state of the art.

Acknowledgements

I would like to thank my supervisor Dr.José Luis Gómez-Skarmeta for setting off this adventure by inviting me in his lab, and for maintaining it with his support and guidance. I am immensely grateful to my co-supervisor Dr. Ignacio "Nacho" Maeso for his support, mentoring, and friendship.

I would also like to personally thank Dr. Juan Tena and Dr. Silvia Naranjo who took me under their wings when I reached Sevilla and showed me the ropes of wet and dry biology. My gratitude also goes out to Dr. Manuel Irimia, for our brilliant collaboration that produced the bulk of this work here. It has been an immensely pleasant and didactic experience.

The DEVCOM network was a huge part of this chapter of my life, so I would like to express my gratitude to everyone involved in it, supervisors and students. It has really been an unforgettable experience. While on the topic of the ITN, I would like to express my appreciativeness to some people whose faces or names I do not know. That is, those responsible for the Marie Skłodowska-Curie Actions. I wish for a future with many more programs that support young scientists in basic research with livable salaries.

I would also like to thank professor Boris Lenhard for his warm welcome, the great work, and the generally great time I had in my secondment in his lab in London. The same goes for professor Gert Jan Veenstra and Dr. Simon van Heeringen for my secondment in Nijmegen and also to everyone in both labs. It was a great pleasure to work with you even for the short time of the secondments. Equally so for Dr. Héctor Escrivà, whose hospitality cannot be exaggerated, and his lab in Banyuls-sur-Mer.

Further, I would like to warmly thank all my lab-mates, all CABDites, as well as all my friends in Spain, Greece, Portugal, the Netherlands, the U.K. or wherever else in the world your lives lead you. I would also like to thank my family for their love and support, most of all my parents, for providing a rock-solid foundation in my life, encouraging me in everything I have ever attempted, and for leading exemplary lives.

Finally, I would like to thank Renata for her love and companionship, and for inspiring me to work hard and follow my dreams.

To my parents,
who are unfailingly there for me

FOREWORD: A FEW WORDS ON THE THESIS ITSELF

A doctoral dissertation is the culmination of a doctorate candidacy. The defendant is called to present the context and the results of their work, and is expected to demonstrate expertise and novelty. This author feels very strongly that the structure of the document itself is a field where expertise and novelty can be shown, and that it should be subject to improvements when those serve the dissemination of knowledge which is the goal of academic writing.

For this, and with the conviction that transparency and reproducibility are the two most important pillars of scholastic integrity, the Methods section were structured around a modern, digital type of document. These documents, also known as notebooks [1], allow a user to interweave beautifully formatted text with programming code and figures. In a single document we can present our thought process (in the text), the very actual method (the code) and the results (the figure) side by side.

On account of the methods being structured this way, someone can download the digital copy of the thesis and the assorted data and directly execute our methods on their own machine. We feel that this is a very intuitive, transparent and complete method of sharing scientific work. In fact, almost all of the figures presented in this work can be produced de-novo by running the provided notebooks and using the provided data.

The compilation script runs all of the notebooks and includes their output figures in the final thesis document. It also formats the notebooks in a print-friendly form and includes them as chapters in our Methods section of the thesis document. Because the notebooks include programming code, they are very lengthy which makes them unsuitable to include between the introduction and results, so we have included them at the end of the document.

The only analyses that are not automated like this are those corresponding to the

[1] Jupyter notebooks: https://jupyter.org/

very first steps of analyzing high-throughput data, which are too large to distribute with the thesis [2], and the neural-network tests which require too much computing time to meaningfully automate. Of course all steps have been documented and are appropriately presented.

The author feels that this approach provides more detailed than typical presentation of the methods used, without interrupting the flow of the document between the introduction and the results and that the departure from norm is thusly justified.

REPOSITORY / CONTACT INFORMATION

An online repository for the thesis can be found at

`https://gitlab.com/panosfirbas/dissertation`

Further the author can be contacted at

`panosfirbas[at]protonmail.com`

[2] Nevertheless the appropriate scripts are provided so that these steps can also be recreated

Contents

Part I

Introduction

Gene Regulation

All cells in an organism share the same genetic library. The dynamics that drive the differential usage of this library, in time and in space allowed for multicellularity, tissue specification, organ-formation, axis formation and more. In other words, without the ability to use the genetic information written on their genomes in various ways, the clones of cells would only be able to form unicellular colonies instead of the great variety of organisms that exist.

These dynamics, or regulation of gene expression (gene-regulation), are controlled by mechanisms in essentially every step of the production of a gene product (protein or RNA). They dictate how much, of which protein, at what time, will be produced. In this work, we will focus on the first and arguably most important of those steps, the regulation of transcription; how much RNA will be produced from a given gene's coding sequence. Later steps such as RNA-processing, RNA-transfer or translation are also important, but out of the scope of this work.

During development, an organism has to unpack all of its complexity out of a single cell, from a single copy of its genome [3] (and some maternal RNA that jump-starts the process). Everything needs to run smoothly at this stage since small divergences from "the program" can have big consequences for the organism. It is in this stage of an organism's life that we encounter the greatest complexity in gene

[3] This might mean a different number of copies per chromosome, depending on the species

expression patterns, and where gene-regulation mechanisms have the biggest effects. Small deviations in regulation during development can lead to significant changes in form [1] and since form can influence reproductive success, gene regulation is under heavy evolutionary pressure.

The dynamics of transcription-regulation during development and through evolution bring together three fields of research and the interesting questions that can be asked rise exponentially.

In the duration of this work, we attempted to answer a number of these questions, with a varying degree of success. Here, we present two bodies of our work that yielded the most noteworthy insights and results. One is the development of a modern informatics tool to detect where proteins bind on the DNA, an important question as will be discussed, and the other is the analysis of a rich set of epigenomic data in an attempt to shed light in some of the aspects of regulatory evolution of vertebrates.

0.1 TRANSCRIPTION REGULATION

Since the 80s when DNA elements were starting to get investigated for their role in transcription[2][3] [4],we learned to talk about enhancers, promoters, insulators, ultra-conserved regions, transcription factors, transcription-factor binding sites, chromatin accessibility, chromatin modifications and so on and so forth.

Our eyes have been opened to the vast complexities of transcription regulation and the relevant vocabulary is taking shape. Our current understanding, simplified (Fig. 1), is that some proteins (Transcription Factors) bind on somehow specific locations on the DNA molecule and help regulate gene transcription. They do this through complex interactions with other such proteins that bind on proximal or distal positions, or proteins that do not directly bind DNA but are nevertheless important for the interactions, as well as other molecules such as non-coding RNAs etc. The way that all this influences gene transcription is through the facilitation or obstruction of the binding of the proteins that actually transcribe DNA (transcriptional machinery), on the locations where they need to bind in order to start the transcription. This in essence turns transcription on or off and controls the gene's production.

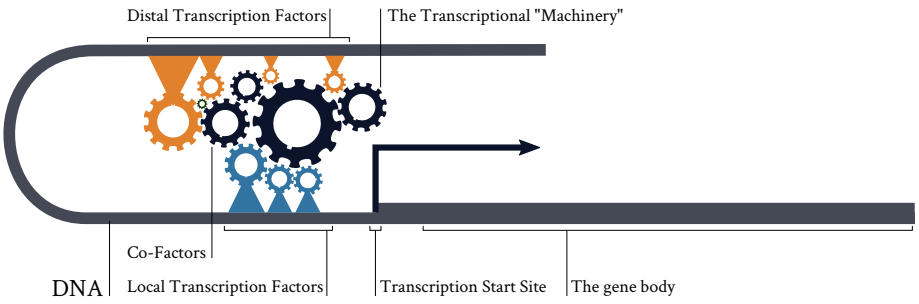


Figure 1: Transcription Regulation: A simplified model. Transcription Factors bind on specific locations of the genome and control transcription by bringing the transcriptional machinery to the gene's transcription start site.

1

Trans-Regulation

1.1 DNA-PROTEIN BINDING

We call these DNA-binding, transcription-regulating proteins Transcription Factors (TF). They act on a different molecule than themselves (trans) so we call gene regulation, when investigated through these proteins, trans-regulation.

TFs bind to DNA through physical interactions between the amino acid side chains of the protein and the accessible edges of the base pairs of the DNA molecule. Those include direct hydrogen bonds, water-mediated hydrogen bonds and hydrophobic contacts. [5]

We can categorize proteins based on the domains [1] that they consist of. TFs contain domains that make contact with the DNA molecule by presenting a protruding surface and/or a flexibly extended structure [6]. We call these 'binding domains' and they are a useful way of classifying TFs into families.

[1] protein fragments that can fold correctly without the rest of the polypeptide

In humans for example, the most abundand TF family, the C2H2 family, is characterized by zinc-finger binding domains, a protein motif that is coordinated by one or more zinc Ions. The second most abundant family, called the homeodomain family, is characterized by a type of loop-helix-loop binding domain[7]. These are just two examples from the big variety of TF families that have evolved in eukaryotes[8]. The other partner of this interaction, the DNA molecule, also presents itself in a variety of shapes and is not always found in its canonical (B-DNA) double helix form [9, 10] further complicating the dynamics of protein-DNA interactions.

BINDING SITES

In order to study trans regulation, a common first question is "where does protein X bind on the genome". Between the various shapes of both the TFs and the DNA molecule, a plethora of emerging features seem to contribute in TF-DNA readout, on multiple levels. First and foremost, at least in terms of frequency of use in research, is the strong preference for specific genomic sequence patterns that many TFs display [11–17].

Every base pair has a unique hydrogen bonding signature (in the major groove, not in the minor) making it intuitive to imagine a sort of "alphabet" or code on the genomic sequence that TFs would recognize, much like tRNA molecules recognize trinucleotides. Specific sequence patterns will create specific hydrogen bonding signatures on the DNA molecule, to which different TFs will bind with different affinity. Thanks to observations derived from three-dimensional structures of protein-DNA complexes [18], the base-readout mechanism of TF-DNA binding has been very well established, but is nevertheless not the entire picture.

In some cases, the nucleotide sequence offers binding opportunities not by hydrogen bonds on the major groove, but rather by conforming the DNA molecule. A good such example is the opening of the minor groove in the complex formed between TBP and the TATA box [19, 20]. In another case, a Drosophila Hox protein is shown to bind to particular minor groove narrowings which emerge as sequence specific characteristics and create a local dip in electrostatic potential [9].

Some TFs form complexes with non-binding factors [2] which can change the binding affinity of the complex [21, 22]. Some need to be bound in multifactor complexes [23]. Some factors contain multiple independent DNA binding domains. [24]

Most of the DNA is usually found in a tightly packed chromatin state, occupied by nucleosomes. Histones, the protein parts of nucleosomes, are the most important DNA binding proteins and their dynamics and their effects on gene regulation are complex enough to warranty a section later on in this work. Some factors are heavily deterred by nucleosome occupancy [25–27] while others compete with nucleosomes [28, 29], potentially interacting with them [30–32] (Fig. 1.1).

Besides all that, the DNA might be methylated which can inhibit or promote TF binding [33].

Unfortunately then, answering the "where does it bind" question is anything but simple.

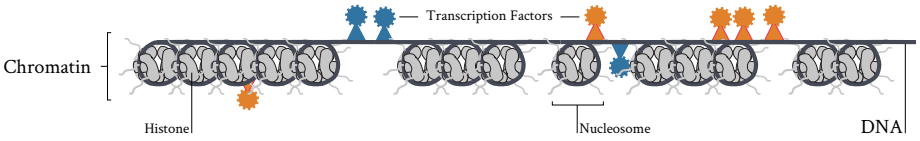


Figure 1.1: The chromatin, as viewed in the "Beads on a string" model. Different TFs have different preferences with regards to their binding sites.

1.2 WET LAB TECHNIQUES

The identification of TF Binding Sites (TFbs) has relied heavily on structural biology and more recently in modern, high-throughput technologies have helped yet another research area, identifying TFbs.

[2] Also known as co-factors

The Encyclopedia of DNA Elements (ENCODE) Consortium, an international collaboration of research groups with the goal of cataloging the functional elements of the human genome, tackles the problem of identifying TFbs with ChIP-seq[3] assays against TFs (listing about 2500 such assays in human context so far).

This technique is the currently best wet-lab approach to answer the question: "In this biological sample, which parts of the genome are more likely to be bound to protein X" with direct evidence.With ChIP-seq, we can detect regions on the genome that interact with a protein of our choice (for which an appropriate antibody must be available). Very briefly (Fig. 1.2), we cross-link proteins and DNA and then fragment the chromatin. We immuno-precipitate [4] the protein of interest while it is still bound to DNA and then isolate the DNA and sequence the fragments that were collected. The sequenced fragments are then mapped on a reference genome (Fig. 1.3). The regions of the reference genome where we detect more reads than expected are the regions of the genome that were touching our protein *in vivo*. The results of such an assay, after some statistical analysis, are regions (that can range in size up to 300bp or more) where we have high confidence that our favorite protein binds.

For some questions that a researcher might have, this resolution will be limiting. To increase the resolution and have a single-base-resolution predictions, we combine our direct-evidence data with computational methods.

[3] Chromatin ImmunoPrecipitation with sequencing

[4] We 'pinch' our protein with a targeted antibody to isolate it from the rest of the sample

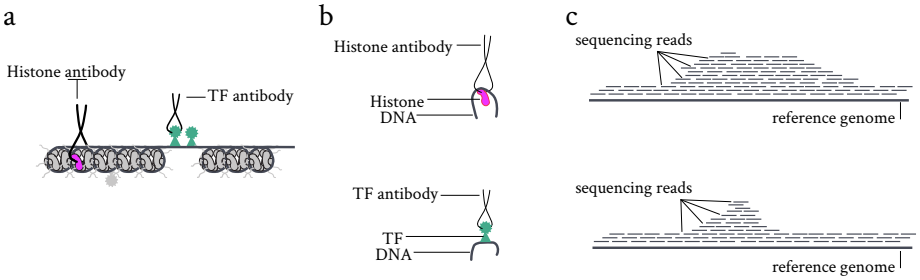


Figure 1.2: ChIP-seq simple model: With targeted antibodies (a), we isolate our protein of choice while it's still attached to DNA (b). We finally sequence this DNA and align it on a reference genome (c).

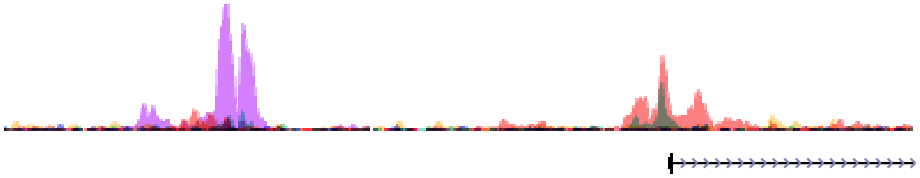


Figure 1.3: By mapping the sequenced reads from a ChIP-seq experiment on a genome, we generate a count of reads at each base-pair. These counts can be visualized as a signal over the genome. Here, various such signals are shown in different colors around the TSS of a gene.

1.3 DRY LAB TECHNIQUES

Computational methods model the TF-DNA interactions, using knowledge gained by direct-evidence experiments, and then apply the model in order to offer predictions on conditions for which direct-evidence data is not available. The goal of any such approach is to reach the predictive power of a molecular biology assay but of course this is, for now, impossible. Nevertheless, these techniques are integral to modern regulation analysis since they can be applied essentially for free in comparison to biological assays and even though the results are technically imperfect, they are still very useful for biological predictions of high quality, especially when used in tandem with biological data.

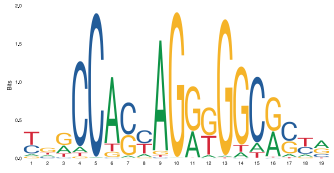
1.3.1 PWMS

The most commonly used of these models are the Position Weight Matrices (PWMs) which exploit the observation that many TFs have a strong preference for specific genomic sequence patterns [5].

A PWM then, models a certain number of bases and contains a weight for each of the 4 nucleotides in each of those positions [11](a weight per base for each position: a position weight matrix). These simple mathematical objects can lead to very fast sequence searches [34] and perhaps more importantly, can be visualized in a very intuitive way (Fig. 1.4). As a consequence they have been used extensively to identify putative sites of TFs. In fact they are often found as the basis of more complex computational methods.

In comparison to consensus sequences, an earlier approach to this concept, PWMs handle mismatches in a more plastic way, since they can score each base in each position separately instead of relying in a perfect match or not, but there are drawbacks in this model as well. For example, PWMs incorrectly assume that each position contributes independently in the binding and they don't account for position interdependencies [35].

a



b

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A	87	167	281	56	8	744	40	107	851	5	333	54	12	56	104	372	82	117	402
C	291	145	49	800	903	13	528	433	11	0	3	12	0	8	733	13	482	322	181
G	76	414	449	21	0	65	334	48	32	903	566	504	890	775	5	507	307	73	266
T	459	187	134	36	2	91	11	324	18	3	9	341	8	71	67	17	37	396	59

Figure 1.4: A position weight matrix on the right (ID #MA0139) and its logo on the left, for the transcription factor CTCF

1.3.2 OTHER SEQ-BASED APPROACHES

Plenty of effort has been invested in improving or surpassing PWMs. From improving the PWM model [36, 37], to alternative models such as dinucleotide weight matrices [38], alternative heuristic approaches [35], hidden Markov models, variable-order Bayesian networks or the TF Flexible Model [35]. Despite the

plethora of alternatives, no method has managed, yet, to replace PWMs as the basic model to detect single base resolution TFbs. PWMs are improved over time, as new direct-evidence data is collected, they are fast to implement and very intuitive owning to their ability to be represented as logos.

1.4 NEURAL NETWORKS

Artificial Neural Networks (NN) are computer systems or frameworks, not a specific algorithm themselves. They are inspired by biological neural networks and employ a variety of algorithms to work.They are enjoying an era of great development and popularity, thanks to advancements in technology, namely the increased computational power of Graphics processing units (GPU) and landmark papers such as the work by Krizhevsky et al. in [39], that firmly established NNs at the forefront of computer vision, artificial intelligence and machine learning. A few years into this explosion of NNs, NN-based programs are the best chess-playing "algorithms" and have managed, for the first time, to beat human champions in the game of Go, one of the last bastions of human intellectual supremacy over the machine.

Nevertheless, NNs are not a new idea; the theoretical base was already explored in the 19th century. In fact, the perceptron algorithm, a simple artificial NN, was at the core of the development of PWMs [11, 40].

Artificial NNs are heavily inspired by real biological neural networks, but that similarity helps mostly in making the model's design more intuitive. A more pragmatic description of the NN is as a very complex, impressively capable and efficient, linear algebra model. What can be abstracted as neurons for the benefit of human intuition, is distilled in large matrix operations. That is why NNs resurged after sufficiently powerful GPUs were developed; GPUs are exceptionally good at handling big matrix operations (this is what computer graphics boils down to as well).

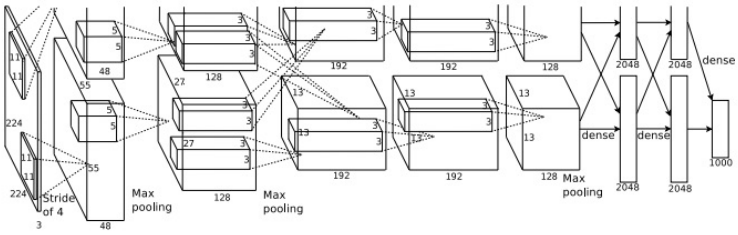


Figure 1.5: An iconic figure from Krizhevsky et al. 2012 [39] (cited 27801 times according to google scholar)

1.4.1 HOW NNS WORK

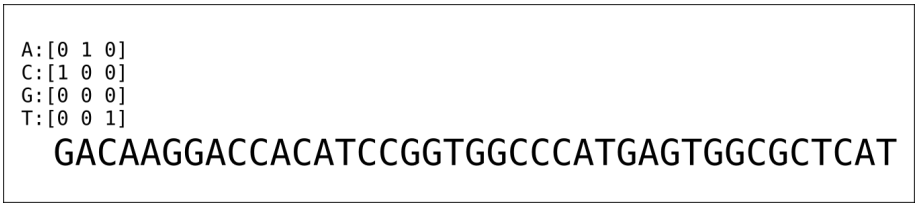
NNs are a generalized framework with many different manifestations so fully explaining them is beyond the scope of this work and the expertise of the writer. NNs were quickly employed with success in genomic analyses and became the bleeding edge of TFbs analyses so we will limit ourselves to presenting a simple/general NN as they have been employed in genetic analyses. This is a one dimensional model in comparison to the typically two dimensional image analysis models. Furthermore, we will not delve in many of the inner workings of the model such as backpropagation or batching, learning rates etc.

NNs work in 'layers', conceptually layers of neurons. Each layer reads from the layer 'below' it and outputs to the layer 'above' it. The first layer 'reads', or takes as input, the raw input (an image, a sound wave, a genomic sequence). The second layer reads, or takes as input, the output of the first layer. The third layer reads the second and so on. The last layer's output is the output of the entire model.

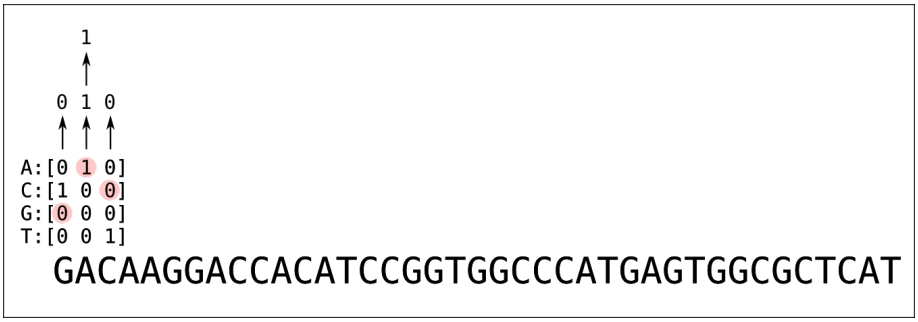
For a NN model that works with genomic sequences, the position weight matrix that was discussed above (see Chapter 1.3.1) is a great starting point to explain how such a system works.

Let us consider a simpler, 3 positions long PWM and a toy genomic sequence. To employ the PWM, we 'place' it on the first position of our sequence.

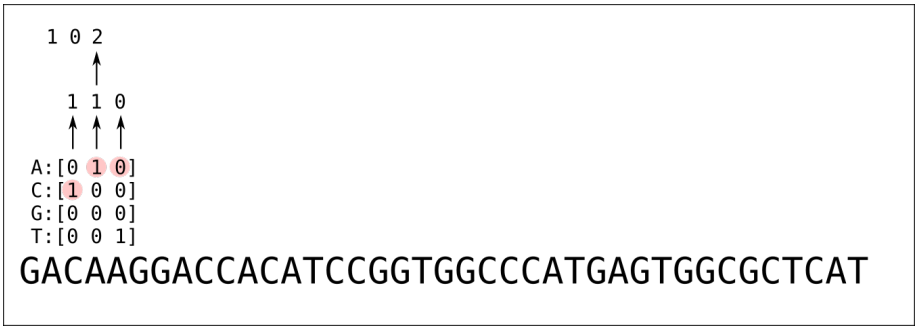
We calculate the score of the PWM in that position by getting the proper score for each of the three positions and adding them up (simplified). In our example, we get 0 for the G in the first position, 1 for the A in the second and 0 again for



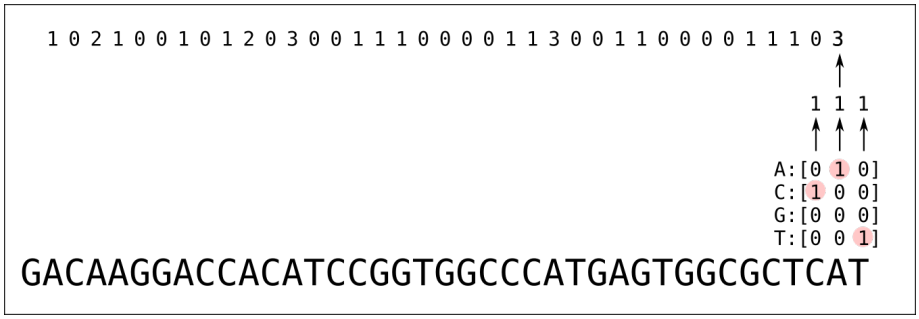
the C in third for a total score of 1.



We continue by 'sliding' the PWM one position at a time and calculating a score in each position. In our example, the second position on the sequence gets a total score of 0 and the third position gets a score of 2.

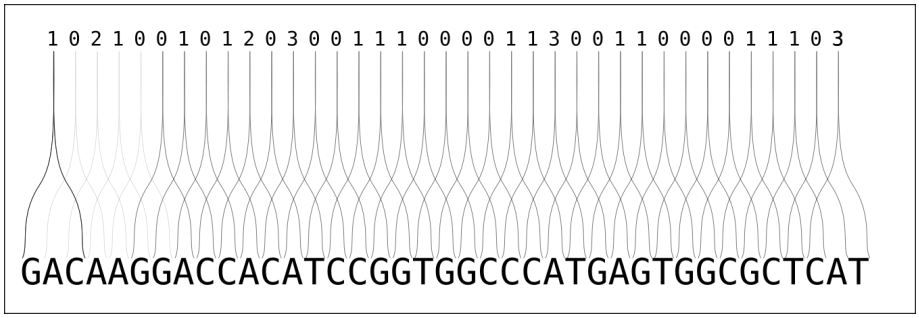


We continue sliding the PWM along the sequence until we have a score for every possible position.



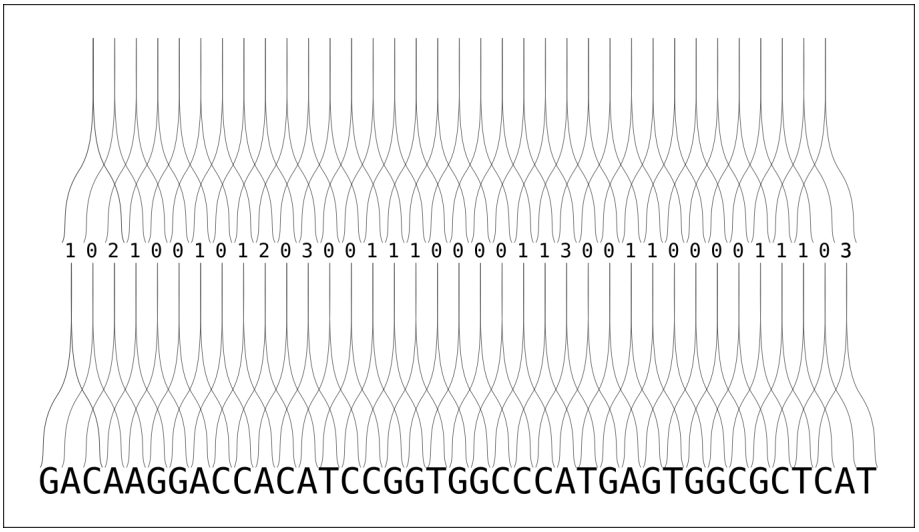
We have now created a second sequence, of numbers instead of nucleotides. The normal PWM analysis would continue to identify which of the scores on that sequence are statistically significant and output the best positions as putative binding sites for the PWM.

In each of the positions on the sequence where we placed our PWM, we read three nucleotides of the sequence and output one number. Instead of thinking in terms of sliding a matrix along the sequence, we can think about a layer of neurons stretched along our sequence.



This is what the first layer of a one dimensional NN looks like. The second layer takes the output of the first layer as its input

We can add more layers like these, or slightly different depending on the nature of the problem. A very common layer type is what is called a “fully connected layer”. That can be perceived as a single neuron that takes as input all of the neurons of



the previous layer. One of these almost always lies at the end of a neural network, serving as the output layer.

An important detail, is that in our abstraction so far, each neuron outputs one value. In NN applications, neurons contain more than one weight matrix and output multiple values, one for each matrix. In other words, each layer (neurons of the same layer share matrices) concurrently learns multiple weight matrices and outputs one value for each to the downstream layers. In NN jargon, we call these channels, and interpret them as different channels of information.

The weight matrices of the neurons are computed in the training phase. During that phase,we present to the model a large number of sequences, each one accompanied by a label, stating if the sequence is what we are looking for or not.

For each sequence shown to the model, its values are fed to and pass through the network of neurons and the final output value of the network is produced. During the training phase, the network adapts its internal weight matrices so that its output value can best predict the real label of each sequence.

After having seen all the input data for a number of times, the model cannot improve its matrices anymore, at which point the training phase has finished. We now have a model, with inner matrices that are unknown to us (hidden layers) but

configured in a way that the model as a whole is as good as possible at predicting the label of the sequence, for example whether it is a TF binding site or not.

We can now use the model to make predictions in a set of sequences for which we do not have a known label.

The power of NNs comes from the layering of information. The first layer of our example, could be conceived as learning a number of PWMs. The second layer will be learning higher order matrices, PWMs of PWMs. The model's grip to human intuition is already waning at this point and only gets weaker as more layers are added. Nevertheless, some interesting work has been made towards investigating the inner parts of such a genomic NN, leading to PWM-logo-like visualizations of the NN's matrices [41].

These higher order 'features' that the inner layers learn, allow the model to have great plasticity. For example, it can assign more or less importance to any of the positions, it can handle "if this then that" logic and much more. Furthermore, NNs are typically employed as big windows of hundreds to a thousand base pairs.

This allows the NN to incorporate a much wider and broader sets of features that are "hinted" by the sequence. Nucleosome occupancy, DNA shape, etc, play a role in TF occupancy and can be at least to a degree inferred from, or modeled on, the genomic sequence. The NN unwittingly models many sequence-dependent characteristics , incorporating everything in its large collection of matrices. In recent applications, NNs have been used to determine and visualize TFbs [41–43] , to predict non-coding function *de novo* from sequence [44], and to predict the effects of noncoding variants [45] among others.

1.5 OTHER NON SEQ-BASED APPROACHES

A drawback of the class of models that, like PWMs or seq-based NNs, model binding based only on the genomic sequence is that they are context-ignorant. As was noted earlier, the spatially and temporally specific binding of TFs on DNA is crucial to cell differentiation and the binding sites of a factor will change drastically between different cell contexts. The same TF will be bound on different genomic

sites in different tissues. A sequence-only model's answer will always be the same since the only data it can rely on is the sequence, which remains the same between samples. Some models are designed specifically with this weakness in mind and apply methods to mitigate it [46].

We can improve our results by using multiple models in tandem, such as using the output of a tool that models DNA-shape based on sequence, to better inform our TFbs predictions. However, every such attempt will still suffer from the same shortcomings of sequence-only modeling. To further improve our investigation, we need additional biological data (from a wet lab experiment). Nucleosome occupancy, histone modification levels, transcription levels are all very useful additional contexts to inform TFbs discovery.

A very interesting option for such an endeavor are the DNA-accessibility assays DNAase [5] and ATAC-seq[6] (see later chapters). The two methods are very similar, with ATAC-seq being newer faster and cheaper. In these assays, we introduce a protein that is able to cut the DNA (a nuclease and a transposase, respectively) to our chromatin sample, and after sequencing we can deduce in which regions of the genome the protein made a cut most times. Regions of the DNA molecule that are tightly packed and inactive will not be accessible to the DNA cutting protein, but in regions with regulatory activity, the DNA molecule will be much more accessible and vulnerable. We use these techniques to identify accessible regions which we treat as a proxy to active regulatory regions (see chapter 2.2). The identification of such regions is already a great step for this analysis as we can filter for putative TFbs that fall inside those regions. Those putative TFbs have a higher chance to be real binding events that contribute in transcription regulation.

Besides this, many DNA-binding proteins also protect the underlying DNA from the cutting activity and consequently leave a "footprint" on the DNAse/ATAC-seq signal, a depression of signal in an otherwise active region.

A few models have been developed to take advantage of this phenomenon.

The Wellington algorithm [49] models protein–DNA interactions as localized imbalanced between plus aligned and minus aligned sequencing reads.

[5] DNase I hypersensitive sites sequencing [47]

[6] Assay for Transposase-Accessible Chromatin using sequencing [48]

Centipede[50] applies a hierarchical Bayesian mixture model to infer regions of the genome that are bound by particular transcription factors. Two other approaches, msCentipede[51] and Romulus[52], take further steps to improve on the idea. [50]. PIQ[53] uses machine learning techniques to model the magnitude and shape of genome-wide DNase profiles and identify occupied TFbs.

This approach of modeling the DNA accessibility data instead of the DNA sequence has its own drawbacks. For example, different TFs have different occupancy times and strengths and as a result some TFs do not bind DNA long enough, or strong enough or in enough cells in our biological sample to leave a mark on the accessibility signal.

1.6 NIMROD

Most, if not all, of the computational approaches to identify occupied TFbs that we mentioned so far, rely on PWMs for the first step of their analysis. The idea is to scan the genome for potential PWM hits and then provide a second metric, more accurate than the PWM score, to evaluate the validity of those hits.

We have further divided them in two other categories, the models that rely only on genomic sequence and those that rely on DNA accessibility data. The later might still rely on PWMs as "seeding" but to our knowledge no method integrates genomic sequence and accessibility signal in the same model.

We developed a NN to attempt such an approach [7].

We know that NNs based on just genomic sequence already give great results because they integrate signal from a large region around the putative PWMs. We also know that ATAC-seq signal can inform the detection of good TFbs. We expect that integrating the two signals in a single NN, we will manage to take

[7] We called it Nimrod, since it is our second and improved attempt at building a tool that reliably detects TFbs. Our first model, which was built around the concept of hunting footprints on ATAC-seq signal, was affectionately called Elmer after the cartoon character with questionable hunting skills. Bugs Bunny once ironically called Elmer 'Nimrod' after the great biblical hunter, reshaping the meaning of the word in modern culture.

advantage of both of the signals and the high-order features that one would expect in this context.

In the results part of this work, we will present the architecture of our model and results obtained with it.

2

Cis-Regulation

2.1 CIS-REGULATION

We've talked about TFs and how they bind on specific positions on the DNA to regulate gene transcription. These TF binding sites (TFbs) are typically found clustered in regions of the genome which can, themselves, be interpreted as functional units of the gene regulation system. These Cis Regulatory Elements (CREs) only drew scientific interest after the period of "Modern Synthesis" but by now are recognized as important players in the evolution of organisms. In fact, their effect on the expression of developmental genes distinguishes them as one of the major drivers of animal form evolution [54–56].

CREs have traditionally been categorized based on their distance to a gene's transcription start site and their observed effect on the transcription levels of their target gene. Elements laying just upstream of the TSS of a gene are called promoters and distal elements are split into enhancers and silencers. Further nuanced names can be used depending on the analysis used such as 'poised enhancers' or 'active enhancers'[57, 58].

As the community's knowledge and understanding of cis-regulation deepens, there appear to be very few differences between promoters and enhancers, challenging

the established view of separating them the two as distinct classes[59, 60].

2.1.1 PROMOTERS

Perhaps the most straight forward CRE to talk about are promoters. They are after all the most ancient and basic of elements, being found in all subdivisions of life, including viruses. We call promoter the genomic region that starts at a gene's TSS and extends upstream and on the same strand for an arbitrary length of DNA, usually some several hundred base pairs [61]. Promoters contain TFbs, sometimes with very deep evolutionary conservation (TATA box, initiator), specially so in the 'core promoter' region, the region directly adjacent to the TSS.

It is here, on the gene's promoter, that the regulatory complexity collapses into whether the appropriate RNA polymerase will bind and begin transcribing the gene or not. The metazoan promoters, perhaps as a natural consequence of having to integrate more complicated regulatory information, are more complex [61] than those of bacteria or single-cell eukaryotes. Typically we can assign one promoter per gene, although bidirectional promoters are not uncommon. In fact, there is evidence that promoter directionality is an acquired feature and not the default state [62].

Although proximal promoters may not contain all of the information required to precisely control transcription of individual genes in time and space during development, analysis of promoters alone can generate meaningful models of transcriptional regulatory networks [63]. Because of the very direct and obvious connection of any gene to its promoter (the promoter lies upstream of the TSS), promoters are the easiest class of CREs to identify and assign to a gene; we just need to know where the TSS of the gene is and look X number of bases upstream. From there we can look for TFbs in the promoter region and assign them to the gene's regulatory network.

2.1.2 ENHANCERS

While we can identify promoters by just looking upstream of the TSS of a gene, a second major category of CREs called enhancers can be found both upstream

and downstream of a TSS and at distances that can span megabases and skip over neighboring genes.

As we discuss extensively in this work, cis-regulation, and by extension enhancers, is of paramount importance to proper development and not surprisingly plenty of connections to disease have also been established [64–67], driving further interest in the field.

Researchers would originally detect these elements with enhancers trap assays, a reporter gene and its promoter are randomly inserted in a genome with a transposable element. This construct will at times find itself inserted in a genomic context where it will be regulated by some enhancer to drive expression of our reporter gene with a specific pattern. We can then determine where our construct was inserted and look around it in the genome for putative enhancer regions.

Other approaches relied on scanning the genome for TFbs (via PWM hits) and then detecting regions with statistically denser concentrations of TFbs (reviewed in [68]). Binding events outside of regulatory elements are less likely to have a significant impact on transcription [64, 69] so it makes sense to focus our effort in the regions with many putative TFbs. In a modern version of the same genomic modeling approach, neural networks are trained to not only predict TFbs but directly predict CREs.

Nowadays, we look for enhancers with genome-wide sequencing assays that take advantage of chromatin features such as accessibility or histone modifications (see following sections).

To confirm that a putative region is indeed an enhancer, we apply a similar approach where we clone our putative enhancer region next to a reporter gene and insert them together in a genome. Our putative enhancer should now be activated in the contexts where the original enhancer is also activated and it should drive expression of out reporter gene in the same domains [70].

2.2 CHROMATIN ACCESSIBILITY

While CREs contain the necessary context for TFs to bind on them, they are not always available. As was discussed in Chapter 1.1, DNA is typically 'occupied'

by nucleosomes and that makes the binding sites unavailable to most TFs. This offers an opportunity for research though because active regions of the genome should be accessible to other proteins. If we can detect the regions where DNA is accessible, we can detect the regions that contribute to regulation.

Deoxyribonuclease I (DNase I) is an endonuclease [1] that cleaves DNA relatively non specifically, although some biases have been identified [71]. In DNase-seq [47], DNase I is exploited to detect regions of the genome that are hypersensitive to it and thus, accessible. Using DNase I to determine chromatin accessibility is an old trick [72] and the principle was further refined in FAIRE-seq[73] and ATAC-seq[48].

These techniques have been widely employed in a plethora of contexts that are too many and varied to be listed here. The ENCODE project, which we have used as a metric of recent scientific interest in techniques, lists 767 DNAse-seqs experiments in human samples.

2.2.1 ATAC-SEQ

With ATAC-seq (Assay for Transposase-Accessible Chromatin), we exploit another DNA cutting protein, the transposase Tn5. Transposases are proteins that "copy-paste" or "cut-paste" *genomic elements* (transposons, or transposable elements). This ability of transposons (which often encode for their own transposase), allows these elements to evolve on a separate plane than their host organisms, like viruses or parasitic microorganisms do. Their effect on the genome is inherently mutagenic though, so transposons cannot be too aggressive otherwise they will kill their host and die themselves. Nevertheless, transposases are the most abundant genes in nature [74].

The Tn5 protein is actually a retro-transposase, a family of proteins used by viruses to insert viral retro-transcribed DNA in naked/exposed bacterial genomes. It works as a dimer, two molecules attach and cleave DNA at specific 19bp sites that surround the transposon. The two parts of the dimer come together to form a homodimer, merging the two cuts on the "main" DNA molecule and removing the intermediate piece which is now bent like a hairpin. The reverse process then

[1] Enzymes that cleave the phosphodiester bond within a polynucleotide chain

follows in a different part of the genome. The dimer cleaves the DNA and inserts the intermediate hair-pinned DNA, potentially reversed[75].

We can "load" Tn5 *in vitro* with DNA of our choosing as long as it contains the 19bp binding sites that the transposase expects. We can make small DNA pieces that include both these 19bp sequences and the DNA bar-codes that are required by next generation sequencing processes. The pieces don't need to form a hairpin, each part of the dimer will attach to a DNA piece and a cut will be created in the host's DNA after the insertion.

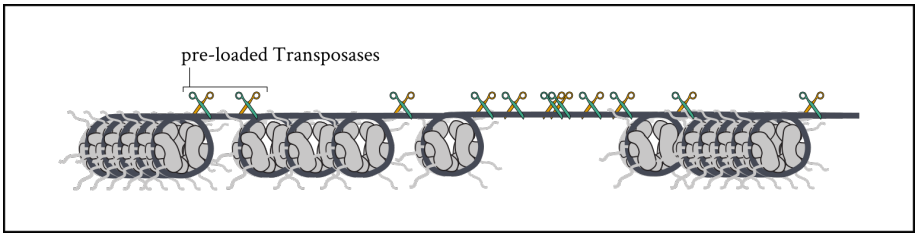


Figure 2.1: In the ATAC-seq protocol, TN5 transposase is introduced to isolated chromatin. The transposase cleaves DNA where is it accessible but not where it is packed around histones.

Exposing host DNA to a high concentration of these preloaded Tn5 proteins will result in the concurrent cleavage of the genome in the regions where the DNA molecule is accessible [75]. The fragments will also have sequencing barcodes at their ends, which the transposases inserted.

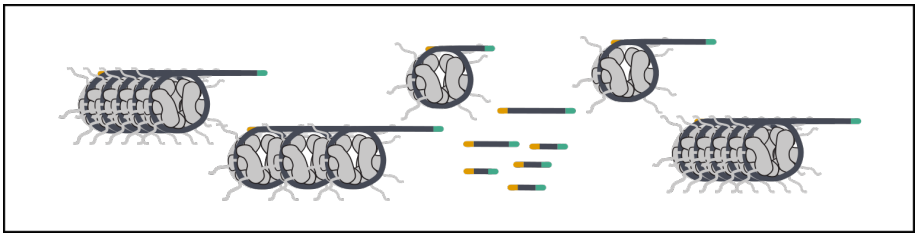


Figure 2.2: When the Tn5 cleaves DNA, it ligates high throughput sequencing bar-codes (represented here in orange and green). This will allow the sequencing in later steps. Doing the cleaving and tagging in a single step is one of the reasons why the ATAC-seq protocol is an attractive option.

The fragments can now be directly sequenced and mapped to a reference genome, giving us a genome wide signal of DNA accessibility.

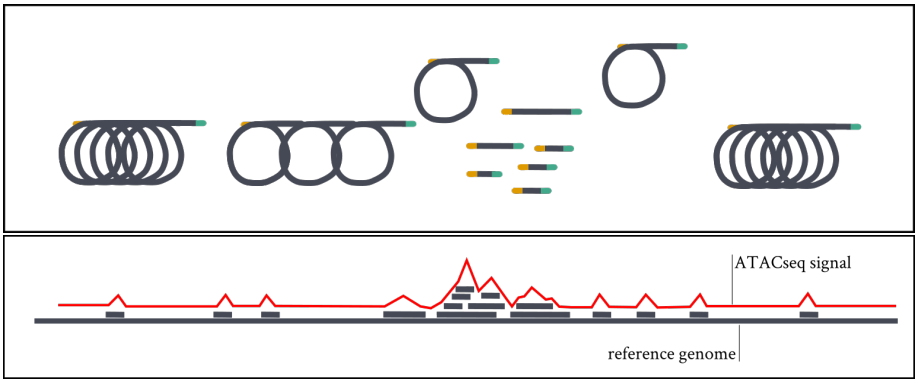


Figure 2.3: In the final steps of the ATAC-seq protocol, the fragments are isolated, sequenced and mapped to a reference genome. This way we get a count of Tn5 cleaves on every position of the genome, the ATAC-seq signal.

As was discussed, transposases cannot be too active and Tn5, true to form, has very low activity in nature. The protein used in molecular techniques had its activity amplified through a series of mutations [76].

ATAC-seq is a very attractive experiment for the simplicity of execution and richness of data it provides. Besides allowing us to determine accessible regions of the chromatin, the nature of the sequencing reads allows us to infer nucleosomal positions [77] and detect TF binding sites.

In this, we work extensively with ATAC-seq data generated on whole embryos during the early development of zebrafish and amphioxus, generated by labmates and collaborators.

Epigenetics

While DNA accessibility marks the regions that are actively participating in cis-regulation, it is only a consequence of the mechanisms that control regulation.

One of the first debates in biology was about the way in which complex organisms develop. An early popular theory was that organisms are pre-formed, perhaps existing as little homunculi inside sperm, and then simply grow during development. Preformationism lasted as the dominant theory until the end of the 19th century or early 20th, by which time enough evidence had been gathered to show that actually organisms develop based on a plan that is orchestrated by nothing more than complex chemical reactions, a process that would be called "epigenesis" [78]. The discovery that genes, as they were understood at the time, could be associated to specific regions of chromosomes solidified epigenesis as a leading theory [79] and the identification of DNA as the main information-holding molecule changed the landscape even further.

In 1970 Laskey and Gurdon [80] show that the DNA of a somatic cell nucleus was competent to direct embryogenesis when introduced into an enucleated egg. That means that somatic cells inherit the entirety of the genomic information of the zygote. Yet the divergent phenotypes between differentiated cells and the communication of those to daughter cells were also undeniable. The information is in the DNA, but different cells have the same DNA, so how are they different? This is the burning question behind the term epigenetic, as it is understood today. Riggs et al, in 1996 [81] defined it as "the study of mitotically and/or meiotically heritable changes in gene function that cannot be explained by changes in DNA sequence".

One of the first[78] such heritable[82] changes was DNA methylation, a process by which methyl groups are added to the DNA molecule[83]. This alteration of the DNA molecule can influence gene expression in at least two ways. One is to interfere with TF binding by making the methylated site inhospitable to the TF. The second way is by recruiting proteins that are associated with chromatin modifiers to make the chromatin environment repressive [83].

2.3 HISTONE MODIFICATIONS

Chromatin modifiers are enzymes that apply a range of post-translational modifications of the most abundant DNA-binding proteins; the histones. The DNA molecule is packed tightly in the nuclei of eukaryotes, wrapped around octamers of the "core" histones (one copy of H3-H4 tetramer and two copies of H2A-H2B dimer) whose amino-terminal tails pass over and between the DNA superhelix to contact neighboring particles[84].

Most nucleosomes also recruit either histone H1 or high mobility group (HMG) proteins (sometimes both) which bind to the outside of the nucleosome to form a particle known as the chromatosome[85].

The first connection of histone modifications to gene regulation was proposed surprisingly early in 1964 when, with a working model that considered histones as proteins that bind to DNA and repress RNA transcription, Allfrey and Mirsky[86] presented their findings that acetylation and methylation of histone takes place post-translationally and that acetylation specially "may affect the capacity of the histones to inhibit ribonucleic acid synthesis in vivo"[86].

Since then, the nucleosome was described and a lot of the details of the above model were illuminated. The N-terminal tails of histones, where most modifications take place, can undergo a large variety of modifications (at least eight distinct types) but acetylation, methylation and phosphorylation have dominated scientific interest [87]. Besides modifications of the N tails, reports of modifications located within the globular domain of histones are surfacing [88].

In the 'beads on a string' model, the DNA molecule is first wrapped around histones and then wrapped into higher order structures facilitated by the histones. The nucleosomes can now be thought of as a sequence of units which, besides

the raw genomic sequence, also vary in nucleosomal structure, thanks to modifications. The beaded string now forms a new 'primary structure' of chromatin which can give rise to a multitude of higher order structures[89].

The tight packaging essentially blocks access to the DNA molecule, which is a double edged sword. On one hand, foreign DNA such as retroviral elements are silenced. On the other hand, of course, the cell needs access to the DNA for many processes. To overcome this, cells employ specialized chromatin remodeling complexes which either reorganize nucleosomes directly in an ATP dependent manner, or toggle modifications on the histone tails which can alter nucleosome structure, stability and dynamics. [90]

To study these modifications in the modern context, we typically perform ChIP-seq assays with antibodies that recognize the modified histone. This isolates the fragments of DNA that were in proximity to the target histone. We then sequence the fragments and map them on a reference genome to acquire genomic tracks of said modification. Regions of the genome that harbored our target modified histone in a significant percentage of the cell population of the sample will give a stronger signal in the sequencing experiment.

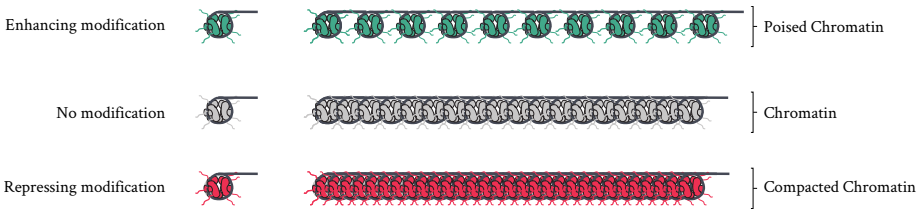


Figure 2.4: Histone modifications influence the way that nucleosomes interact with other nucleosomes and consequently influence the local chromatin state. A modification that makes the nucleosomes pack together tightly (here, in the bottom row) will solidify the chromatin and repress local gene expression since it doesn't allow DNA to be accessed by proteins. In contrast, an enhancing modification (top row), would relax the packing of the chromatin, facilitating the binding of transcriptionally relevant proteins on the DNA.

2.3.1 H3K4ME3

The tri-methylation of the fourth lysine of histone 3, one of the post translational modifications of histones that were just discussed, is ubiquitously found on the promoters of eukaryotic genes that are undergoing transcription [91]. It is mediated by the Set1 protein and the COMPAS (Complex Proteins Associated with Set1)complex [92] which are recruited in the promoter by the RNA polymerase II elongation factors Paf and FACT[91].

H3K4me3 activity is conserved in eukaryotes but not without differences. Specifically in chicken, it was observed that while H3K4me3 can be used to classify genes in active and inactive and preferentially associates with the transcribed regions of genes, it can still be detected in lower quantities on inactive genes[93].

In humans, H3K4me3 is mediated by Set9, which competes with histone deacetylases and precludes the H3k9 methylation by Suv39h1[94]. Up to 75% of our genes were found to be tri-methylated even without detectable RNA elongation [95].

Nevertheless, H3K4me3 was one of the epigenetic marks most employed by the ENCODE project in its attempt to identify functional elements on the human genome. At the time of writing, 168 different ChIP-seq assays against H3K4me3 are available for human cells and tissues on the ENCODE project's page and 106 for mouse. These assays were conducted in order to characterize promoter regions. It is believed that nucleosomes that are modified by H3K4me3 form a chromatin state that somehow facilitates transcription initiation.

In zebrafish too, H3K4me3 has been shown to be enriched at transcriptional start sites, an association which correlates with gene expression [96, 97].

We contribute to the effort to investigate this modification by providing H3K4me3 assays in two new developmental stages of zebrafish, significantly enhancing the resolution of the available data during the development of the popular model organism.

2.3.2 H3K27AC

As we discussed earlier, TFs bind on DNA in sites both proximal and distal to TSSs. The regions of these distal binding sites usually behave as 'hotspots', offering binding sites for multiple TFs. We can think of those regions as functional units of the gene regulation system and we can categorize them depending on further characteristics such as chromatin state, or chromatin accessibility (see later chapter).

H3K27ac is another heavily investigated histone modification (ENCODE: 97 in human, 93 in mouse). Acetylation has long been associated with transcriptional activation, in contrast to histone deacetylation which is generally thought to have roles in transcriptional repression[90].

Originally investigated in yeast [98] and later in human and mouse [99], H3K27ac was initially reported to be highly enriched at promoter regions of transcriptionally active genes [100] but later was also established as an important mark to detect active enhancers[57, 58, 101], including distal cis regions which seemingly increase gene transcription levels. H3K27ac has also been investigated in early zebrafish development where it was shown to positively correlate with gene expression [97].

In this work, we present H3K27ac assays in two new developmental stages of zebrafish, significantly enhancing the resolution of the available data during the development of the popular model organism.

2.4 UNDER THE LIGHT OF EVOLUTION

The study of animals and their form is the simplest, most intuitive first step towards studying life and has kept scholars busy since the times of Aristotle. Unfortunately comparative anatomy would only be rekindled in the West after the stupor of ecclesiastical absolutism of the middle ages. We urge the reader to read E. W. Gudger's 1934 paper on renaissance and the pioneers of ichthyology in the early 16th century [102]. Among them Pierre Belon who, among other works, published a comparison of a human to a bird skeleton, and is singled out as the "prophet" of modern comparative anatomy.

By the time Darwin was publishing the origin of species, the concepts of homology and analogy were well understood by his contemporaries. That was before the general acceptance of phylogeny [103]. As the era of molecular biology dawned, the concept of homology was expanded to refer to proteins, which were initially investigated based on their chemical similarities [104] and later on based on their sequence. And just like that, we started talking about homologous genes.

Molecular phylogenies confirmed some morphological phylogenies, clarified some and revolutionized others. We discovered that the coding sequences of genes display remarkable conservation in the tree of life with some of them, particularly those involved in transcription and translation, being traceable all the way to the universal common ancestor.

On the cis-regulatory level things are less straight forward. While some conservation can be observed between close species (e.g. human and mouse), and in rare cases over longer distances (e.g. in all vertebrates) a high degree of turnover is also evident and cis-regulation conservation seemingly disappears over longer evolutionary distances (e.g. human-fly)(see Chapter 10). Overall the conservation of cis-regulatory elements is poorly understood.

In this work, we investigate the evolution of cis-regulation in a branch of the tree of life where intriguing questions remain unanswered. In the following sections, we introduce this branch, our model organism, and what we hope to learn from our analyses.

2.4.1 ON THE TREE OF LIFE

The vertebrate pattern is one of life's successful discoveries. Thanks to their high morphological diversity, they occupy a huge variety of niches. They come in all forms and shapes and they are of course of special interest to us since we ourselves belong in this group. Outside of vertebrates, life can seem quite alien but inside this group our similarities are obvious, vertebrates have a spine like us, most of

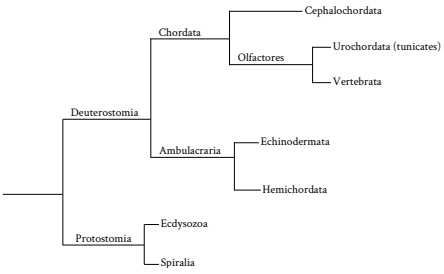


Figure 2.5: Some major animal groups in the vicinity of vertebrates

them have some form of hands and legs, they have an obvious head and mouth. They include the vast majority of animals that humans are typically familiar with: fish, amphibians, reptiles, birds and mammals. Non vertebrates are mostly jelly like, worm-like, or insects, forms that we cannot empathize with very much.

The most vertebrate-like but non-vertebrate organisms are the cephalochordates [2] and the tunicates. These two groups together with vertebrates form the group of chordates. Tunicates, also known as urochordates, are marine organisms that start their lives as tadpole-looking larvae. Most of them later undergo a dramatic metamorphosis into a sack- like filter-feeding organism that is often attached to rocks or stones.

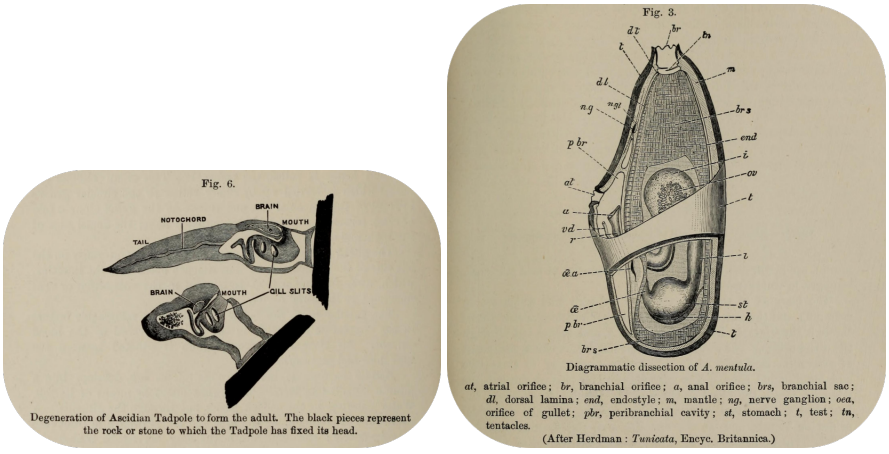


Figure 2.6: The metamoprhosis, on the left, transforms the tadpole to the adult form, on the right. Illustrations from "A guide to the shell and starfish galleries", Department of Zoology, British museum (Natural history) 1901.

After molecular comparisons became available, it was shown that tunicates are the closest organisms to vertebrates [105]. This came as a bit of a surprise since amphioxi look much more like a 'minimal' vertebrate than urochordates who look

[2] of which Amphioxi are the only extant members

quite alien-like in their adult form. This should work as a good reminder that small and alien looking life forms aren't less evolved than us.

2.4.2 AMPHIOXUS

Amphioxus is a small, fish-like, benthic organism that lives in shallow waters around the earth. In fact, about 35 different species have been described, in three genera; Branchiostoma, Epigonicthys and Asymmetron[106].

Amphioxus is a slow-evolving organism and considering how much more it looks like a vertebrate in comparison to urochordates who are closer to us, it is tempting to consider that this is probably what the common ancestor chordate looked like.

Amphioxi possess typical chordate characteristics, such as a dorsal hollow neural tube and notochord, a ventral gut and a perforated pharynx with gill slits, segmented axial muscles and gonads, a post- anal tail, a pronephric kidney, and homologues of the thyroid gland and adenohypophysis. However, they lack typical vertebrate-specific structures, such as paired sensory organs (image-forming eyes or ears), paired appendages, neural crest cells and placodes [107]. [108]

The many morphological homologies between amphioxus and vertebrates as well as its place on the phylogenetic tree are two great motivations to research amphioxus, but its genomic simplicity can be even more intriguing than its morphological.

2.4.3 WHOLE GENOME DUPLICATIONS

Among other vertebrate-specific genomic characteristics such as their large intergenic distances[109, 110] or their high methylation-dependent regulation of embryonic transcriptional enhancers [111], vertebrates have undergone two rounds of whole genome duplication (WGD). Twice at some point near the root of the vertebrate tree (details are still being debated) the whole genome was kept in duplicate. These genome duplication events, have been considered to be major driving forces in the evolution of vertebrate and plant complexity and form [112, 113]. Interestingly, teleosts, a group of vertebrates with a third duplication, shows further greater complexity, comprising nearly 50% of all vertebrate species[114] and 99% of all extant fish species [115]. Despite efforts thought, scientists have not

managed to show this expected increase in morphological diversity in the fossil record [116, 117], therefore the link between WGD and increase in animal shape complexity remains disputed.

Amphioxus doesn't share this genomic characteristic and only has a single, evolved, copy of the ancestral proto-chordate genome [118, 119]. As a consequence, for the vast majority of genes that we can detect in amphioxus we can find from a single up to eight copies of genes in zebrafish and up to four copies in mouse and human. That's because teleosts have undergone a third, fish specific genome duplication. The genes that are kept in multiple copies are vary often related to gene regulation and development which is how, as we discuss in Chapter 10.2, WGDs might increase the evolutionary potential of organisms.

Once a gene is retained in duplicate, it is interesting to consider what might happen to the two ohnologues [3]. The *classical*, as of 1999, model of evolution of duplicate genes predicts that one of the copies will slowly accumulate deleterious mutations until it degenerates. Force et al. proposed an alternative model, the DDC model and supported their position with relevant observations [121]. The Duplication-Degeneration-Complementation (DDC) model, suggests that mutations on the duplicated CREs of duplicated genes can increase the probability that the pair of genes will be kept, and that by sharing deletions of such elements, the copies will partition the ancestral expression landscape.

The copies, they hypothesize, might each lose function in some of the ancestral domains, a process called 'subfunctionalization'. Alternatively, one copy would retain all functions while the other loses more and more expression domains, which is named 'specialization'. The two might also both keep all ancestral expression ('redundancy') or one or both might discover new niches of expression ('neofuncionalization').

A further elaboration, the DDI model [122] (Duplication–Degeneration–Innovation) argues that duplicated genes evolve simpler regulatory landscapes during the process of subfunctionalization, which makes their regulatory landscapes more receptive to change and thus increases their diversification potential.

Little is know about the actual patterns of gene or cis evolution after WGDs.

[3] Gene duplicates resulting from WGD, in honor of Susumu Ohno who first cosidered the implications of gene duplication in evolution [120]

Although intuitively attractive, the DDC and DDI hypotheses have been difficult to test for the vertebrate WGDs due to lack of genome-wide transcriptomic and regulatory data from appropriate outgroups. Amphioxus with its lack of WGDs is a great candidate model organism for these questions so probing those will be one of our main goals. In the discussion part, we will explore some of the relevant work and see how our results align with that and the proposed models.

3

Objectives

To sum up, the following are some of the objectives that we will try to address in this work:

1. To what degree is cis-regulation conserved in chordates? Is the anatomically and morphologically conserved chordate body plan reflected on conserved cis-regulation?
2. How do duplicated genes evolve after a WGD event? Do our observations fit with existing models? What happens to their cis-regulatory landscape?
3. Histological and morphological complexity has increased in vertebrates. Does this reflect on cis-regulation complexity and can we quantify it?
4. Can a NN that integrates genomic sequence and ATAC-seq signal competently classify TF binding sites ?
5. Would such a model be able to generalize its TF binding site classification ability in a cross-species manner?

Part II

Results: The origins of vertebrate gene regulation

4

Introductory Analyses

The results presented in this part titled "The origins of vertebrate gene regulation", were created as part of our collaboration in the work that was publish in [123]. Here we will focus on the analyses that we contributed to that work, revolving around ATAC-seq and cis-regulation.

4.1 THE GENOMES

Genomic analyses are applied on reference genomes which are at times updated so it serves to clarify which exact reference genomes where used here. For amphioxus, we work with the novel genome for the Mediterranean amphioxus Branchiostoma *lanceolatum* which was assembled by collaborators as part of our work in [123]. For the other organisms our choices are listed in Table 13.5

The genome assemblies themselves are used here as tools but we must not forget that they are the product of hard biological data and thus a valuable resource themselves so at least a brief analysis is called for.

Having undergone two rounds of whole genome duplication, vertebrates started their journey through evolution with four times the genetic material of their single copy ancestor. This is reflected in the their current genome size, but not always and in varying degrees. For example, even though teleosts have undergone a third whole genome duplication, both zebrafish and medaka have much smaller genome sizes than mice. Interestingly teleosts, the vertebrate group with most variety of species among vertebrates, have smaller genomes than tetrapods in general. Medaka has a particularly small genome, perhaps mostly on account of the much smaller number of repeated elements that it contains. In fact if we discard the regions of the genome that we can mark as repeats, medaka has a slightly larger "effective" genome size.

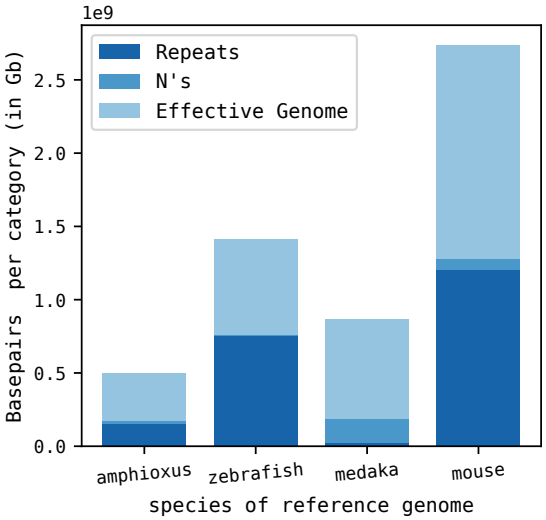


Figure 4.1: Sizes of genomic categories in our reference genomes. The repeated regions are based on publically available repeat-masks for each genome. The amphioxus repeats were annotated by our collaborators as prts of our publication [123]. "N's" refers to the number of unidentified bases in the reference genome. Effective genome is everything that is left after marking repeats and Ns [notebook: 13.4]

Amphioxus has a smaller genome than all the mentioned vertebrates and is composed by repeated elements in slightly smaller but similar degrees to zebrafish and mouse.

4.2 INTERGENIC REGIONS

The genome size differences are reflected on the amount of space that can be attributed to each gene. Considering intergenic distances [1] as a measure of that, we see a clear increase between amphioxus and vertebrates.

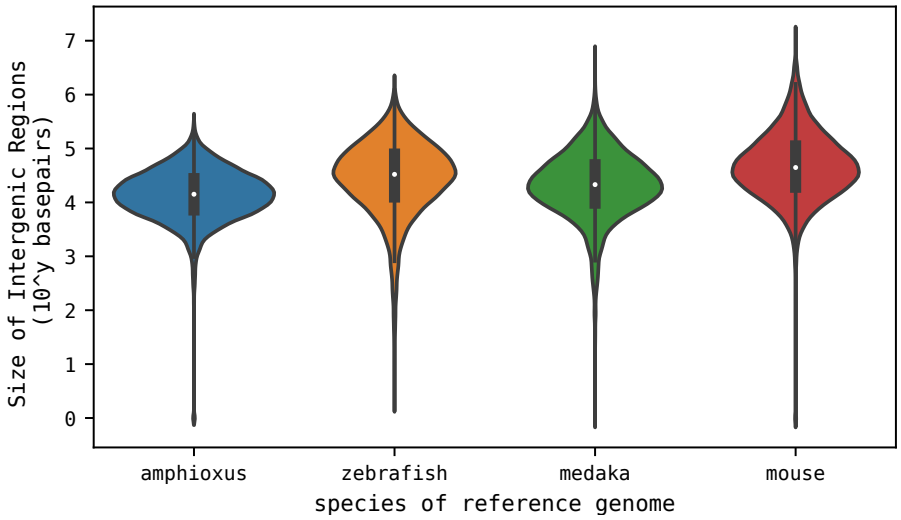


Figure 4.2: The distributions of Intergenic sizes, the distance between consecutive TSSs, for orthologous genes in our reference genomes [notebook: 13.5]

[1] simply the distances between successive genes (see 13.6 and 13.8)

4.3 GREAT REGIONS

A different way to assign genomic regions to genes, is to use the GREAT method [124]. We applied this method (see Chapter 13.7) to our genomes and observed the same dynamics once more.

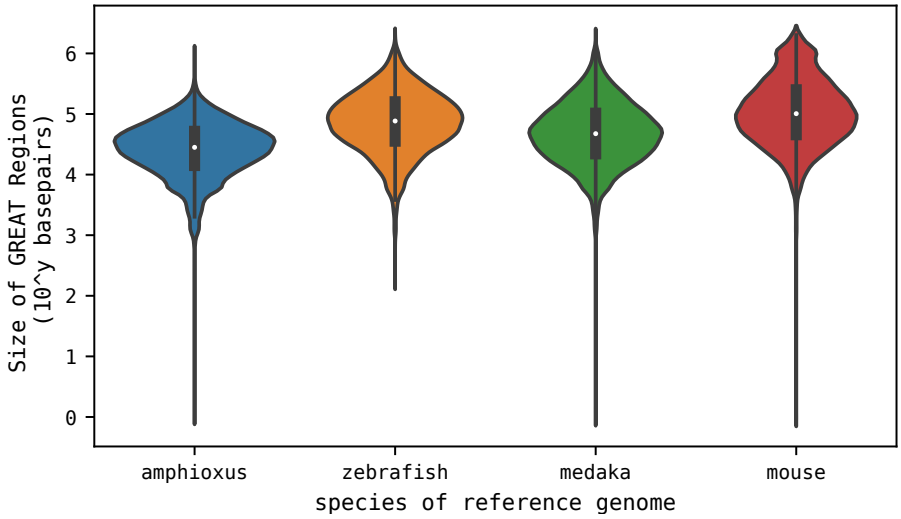


Figure 4.3: The distributions of GREAT region sizes for homologous genes in our reference genomes [notebook: 13.5]

4.4 HISTONE MODIFICATION CHIP-SEQ

As mentioned before, H3K4me3 is a marker of active promoters, while H3K27ac marks promoters as well as active enhancers. Processing the high-throughput sequencing data provides us with ChIP-seq "signal", a measurement of ChIP-seq activity for each base-pair of the reference genome, measuring the degree to which each base-pair was collected in the molecular biology experiment.

This signal varies in strength between different regions of a genome, but forms "peaks" over pertinent regions. We aim to detect these peak regions which is the ultimate fruit of such an assay. These are our putative-enhancers or promoters.

Sometimes a peak from one region of the genome will have lower signal from a different "richer" region but is nevertheless a region displaying significant activity and should be detected. This means that we cannot simply set a threshold on the signal to detect significant regions but instead we need to employ more robust statistical methods.

We have at our disposal a set of ChIP-seq datasets, for these two types of histone modification, obtained in a number of developmental stages in amphioxus and zebrafish. We applied a peak-calling pipeline on these datasets, employing previously described methods and proved bioinformatic tools (see 13.9).

This gives us a set of ChIP-seq peaks, for two different marks in a variety of developmental stages.

4.4.1 WIDTH OF PEAKS

At first glance (Fig: 4.4), looking at the width of the peaks, little differences can be detected between amphioxus and zebrafish.

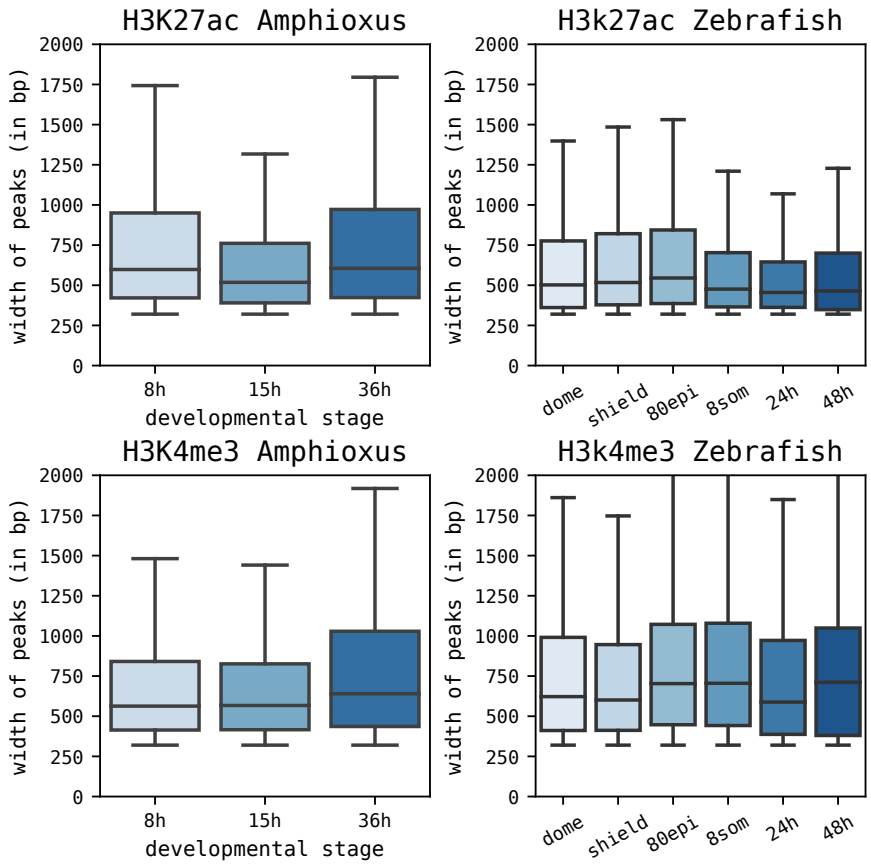


Figure 4.4: Distributions of peak widths from the ChIP-seq assays. The H3k27ac experiments are in the top row and H3k4me3 in the lower row. Amphioxus on the left, zebrafish on the right.

4.4.2 NUMBER OF PEAKS/ GENOME COVERAGE

We went on to compare the number of putative elements. Having duplicated its genes at some point in its evolutionary past, one could expect zebrafish to have more transcriptionally active promoters at any developmental stage. Nevertheless, the H3K4me3 experiments yielded similar numbers of elements between the two species (Fig: 4.5), furthermore declining in similar fashion through the course of development.

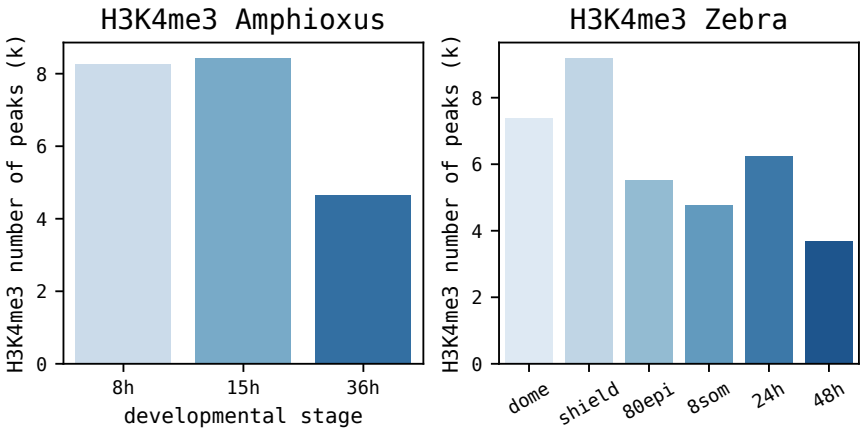


Figure 4.5: Number of peaks, regions of statistically significant signal, in the H3K4me3 assays

Differences only start to appear when we compare the number of putative enhancers (Fig: 4.6). The number of H3K27ac-marked regions is higher in zebrafish and increase as development progresses in comparison to the regions marked in amphioxus whose numbers stay stable.

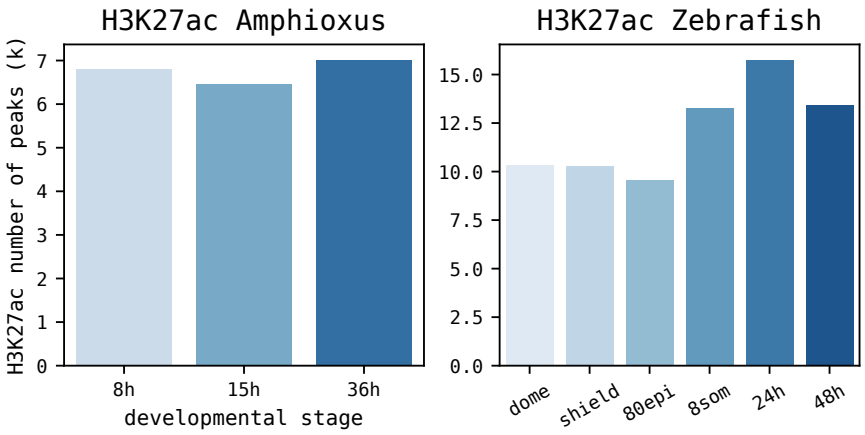


Figure 4.6: Number of peaks, regions of statistically significant signal, in the H3K27ac assays

Despite the smaller count of cis elements marked by these two experiments, a larger percentage of amphioxus's genome is covered by them (Fig: 4.7).

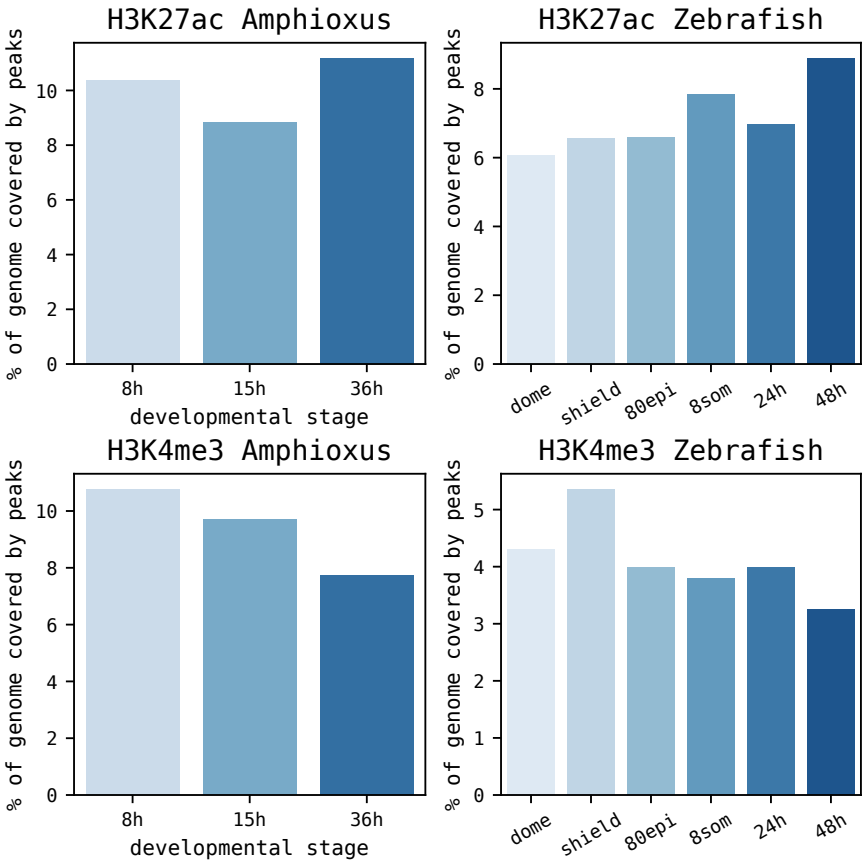


Figure 4.7: The percentage of the reference genome that is covered by regions with statistically significant ChIP-seq signal in our assays.

4.5 ATAC-SEQ

Similarly to ChIP-seq, we applied a peak-calling pipeline to our list of ATAC-seq experiments to obtain ATAC-seq peaks in a number of developmental stages of amphioxus, zebrafish and medaka, as well as two cell lines of mouse. [notebook: 13.10]

The numbers of ATAC-seq peaks, like the putative enhancers detected through H3K27ac, are higher in vertebrates than amphioxus (Fig: 4.8), and increasing throughout development (more plots at 13.10).

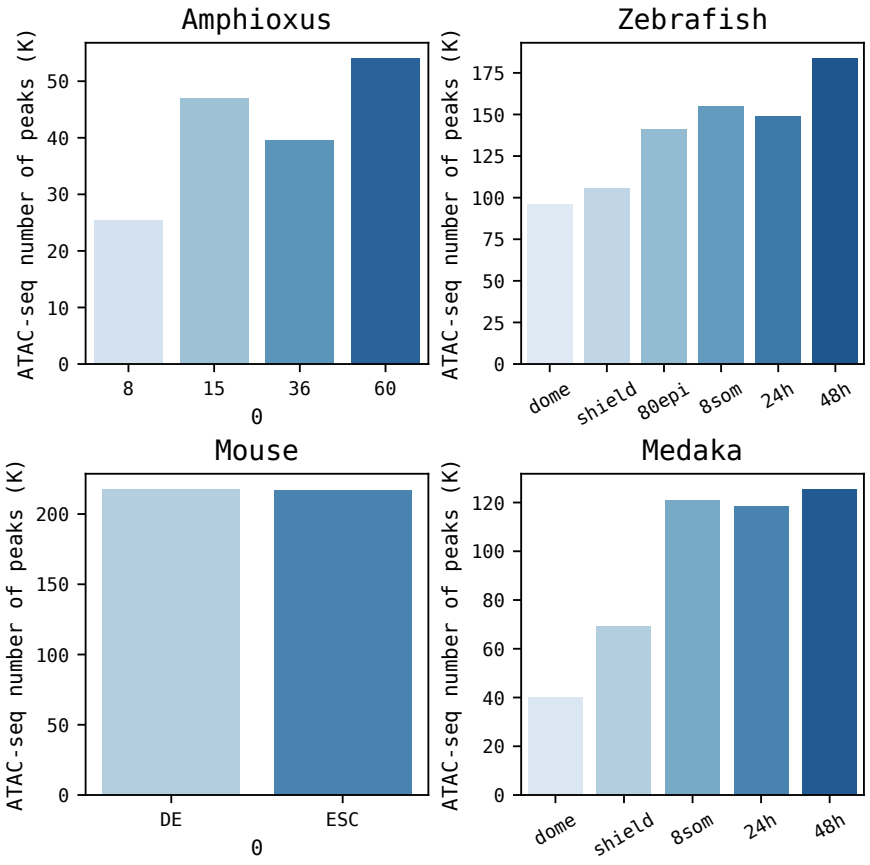


Figure 4.8: The number of detected ATAC-seq peaks in our experiments [notebook: 13.10]

4.6 CRE-TSS DISTANCES

A more interesting inquiry than the raw number of peaks is how those peaks are distributed on the genome. Since ATAC-seq marks all regulatorily active regions of the genome, we need other clues to inform us on their function.

Such a clue can be obtained from the genome. A peak that overlaps a TSS is likely the gene's promoter while a peak at a large distance is likely an enhancer or

silencer.

To examine the distribution of peaks around TSSs, we plotted (Fig: 4.9) the cumulative distributions of distances from our various experiments. In this plot, the y axis spans from 0 to 1 and measures the percentage of total peaks that are found at an equal or smaller distance than the value on the X axis.

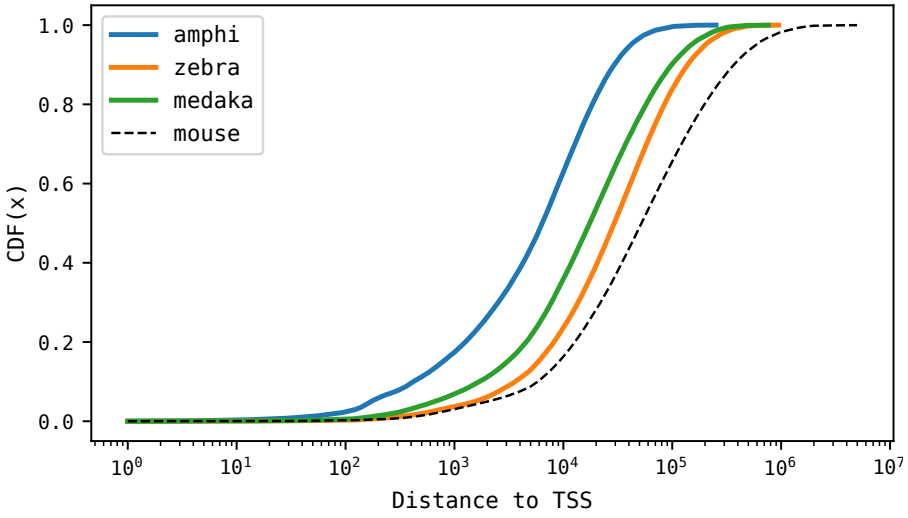


Figure 4.9: The cumulative distribution function (CDF) of the distances of ATAC-seq peaks from TSSs. For each point in X, Y percentage of peaks have X or less distance to a TSS. i.e. roughly 20% of zebrafish peaks and 80% of amphioxus are found at a distance of 10^4 or less from a TSS. [notebook: 13.11]

We can see a clear difference between amphioxus and the vertebrates, but the vertebrates themselves are also clearly in order of genome size. Maybe this is simply the effect of larger genomes. To clarify, we normalized all peak-TSS distances by the average intergenic distance of each genome. In the resulting plot (Fig: 4.9), the vertebrates behave very similarly to each other while amphioxus is still clearly distinct.

It looks like cis regulatory landscapes of vertebrates have been expanded, with cis regulatory elements being found at larger distances from promoters.

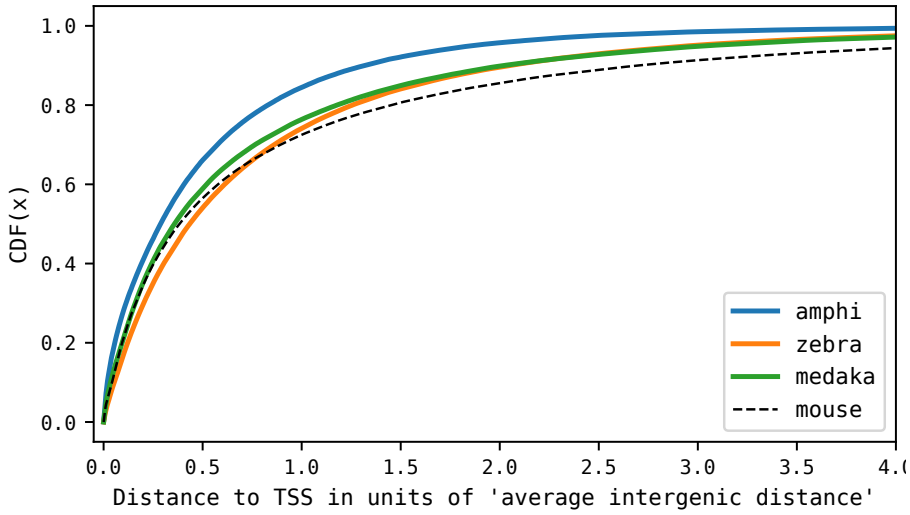


Figure 4.10: Like in 4.9 but now distances to TSS are normalized by the average intergenic distance. i.e. 70% of ATAC-seq peaks in vertebrates and more than 80% in amphioxus are found at a distance of less than the average intergenic distance. [notebook:13.11]

4.7 HIGHER REGULATORY CONTENT

The next question arising through our train of thought is if vertebrate genes are controlled by more cis regulatory elements than their amphioxus homologues. To investigate this we need to assign cis elements to genes, which is not a simple task. It is known that cis regulatory elements can be found in surprisingly long distances from their target gene(s), even with other genes found between them. Lacking adequate direct, biologically-collected data with regards to which element interacts with what gene, we necessarily have to adopt a suboptimal scheme.

A typical such approach is to assign each putative element to its nearest gene on the genome. We employed an improvement on this concept, the GREAT method (more in chapter 13.7).

With this, we can count the number of assignable cis regulatory elements per gene and examine their distribution.

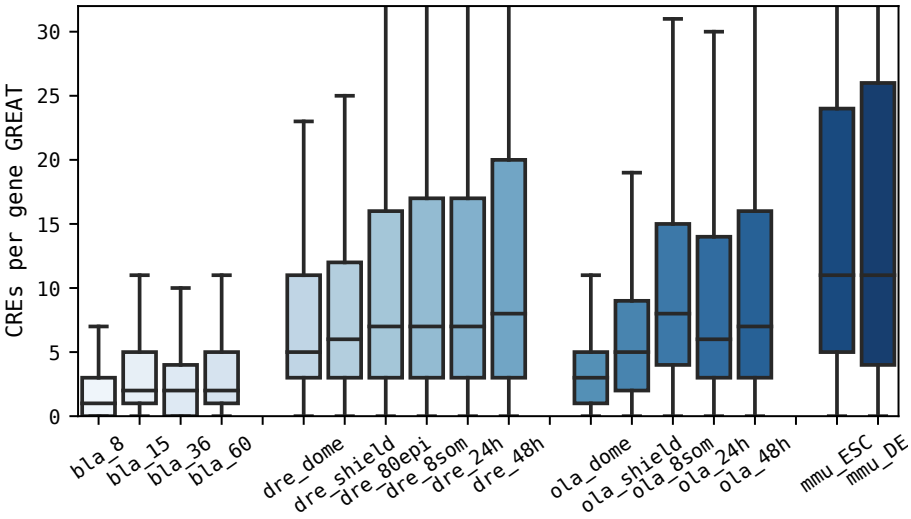


Figure 4.11: Distribution of the number of ATAC-seq peaks within each gene's regulatory landscape (as estimated by GREAT in each of our experiments. For each species (from left to right: amphioxus(ola), zebrafish (dre), medaka(ola), mouse(mmu)), we plot stage or cell line [notebook: 13.12]

4.7.1 MATCHED GENOMIC REGION SIZES

We find more ATAC-seq peaks per gene's GREAT region in vertebrates, but, as we commented in previous sections, genes in vertebrates also have larger genomes and larger intergenic regions (Example in Chapter 6.0.1, Fig. 6.2). Can the increase in regulatory content be singularly explained by the increase of space?

We split genes in amphioxus and zebrafish in categories based on the size of their GREAT region (Fig. 4.13), by dividing that size by 10000. This way, category 1 is all genes with a GREAT region of under 10kb, category 2 is GREAT regions larger than 10kb but smaller than 20kb and so on. We repeated the same process with intergenic regions instead of GREAT regions in Fig. 4.12.

In both cases, we found that zebrafish genes consistently had more ATAC-seq peaks, even at comparable genomic sizes, indicating that besides the increase in genomic size, the higher regulatory content is also driven by an increase of CRE density.

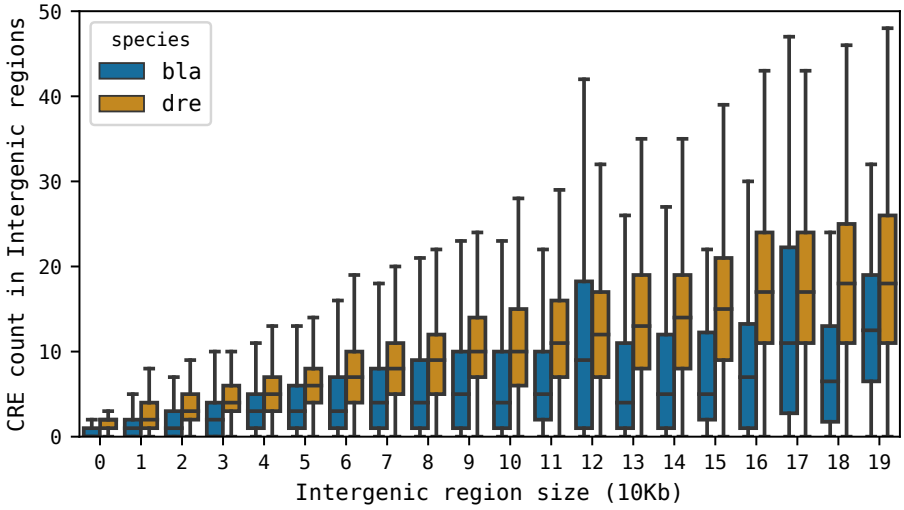


Figure 4.12: The counts of ATAC-seq peaks in the intergenic regions of genes, on the Y axis. Genes are grouped based on the size of their intergenic region. i.e. category 1 is all genes with a intergenic region of under 10kb, category 2 is intergenic regions larger than 10kb but smaller than 20kb and so on.

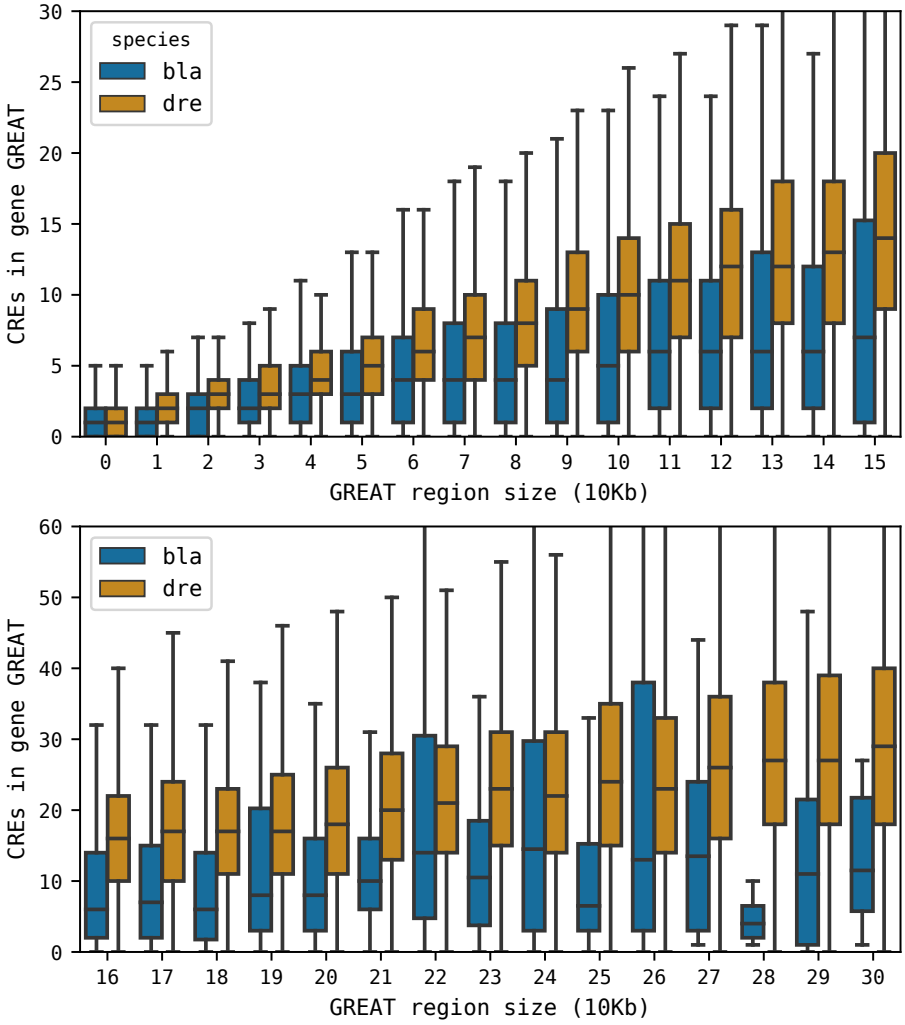


Figure 4.13: The counts of ATAC-seq peaks in the GREAT regions of genes, on the Y axis. Genes are grouped based on the size of their GREAT region. i.e. category 1 is all genes with a GREAT region of under 10kb, category 2 is GREAT regions larger than 10kb but smaller than 20kb and so on.

4.7.2 DOWNSAMPLING

To examine the possibility that this increase is a result of richer, more deeply sequenced experiments, we down-sampled vertebrate experiments. That is, we randomly discard a portion of the reads of a rich experiment and repeat the peak-calling procedure. We removed more and more of reads from a zebrafish and a medaka experiment and found that we had to lower the coverage of vertebrates to 20% of our richest amphioxus coverage in order to break the effect.

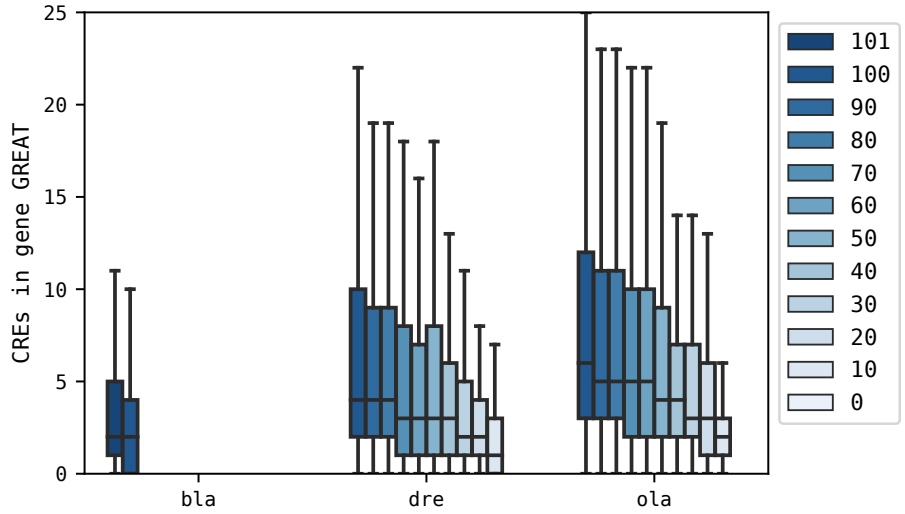


Figure 4.14: The counts of CREs at increasing levels of downsampling [see notebook: 13.14]. At 100, is the mean coverage (sequencing reads per effective genome size) of our two richest amphioxus replicates. At 101 is the peaks found in the slightly richer of those two replicates alone. In zebrafish and medaka, we tried increasingly harsh downsampling in order to bring the coverage of those experiments at lower levels than the ones from amphioxus.

5

Conservation of cis regulation

5.1 NACC

Looking for evidence of regulatory conservation, we applied Neighborhood Analysis of Conserved Co-expression (NACC) [125], a method developed to compare heterogeneous, non-matched sample sets across species. This allowed us to use all of our available RNAseq data without having to worry about matching them across the species. In total we had 49 Amphioxus, 31 zebrafish, 65 mouse and 52 human samples. They consist of assays on embryonic stages, organs, cell lines etc. The full list can be found in 13.22.

The method investigates to what degree gene neighborhoods, genes that are expressed similarly across a set of tissues, remain as a neighborhood in another species where we can detect the orthologous for each gene. For each gene, we get a NACC score which is smaller the more conserved the gene's neighborhood is. By plotting the distributions of the NACC values for all of the genes and contrasting it to the same distribution made with randomized orthologies (Fig. 5.1) we can see a clear conservation of gene regulation between amphioxus and human.

Using human as a reference species we applied the NACC analysis to mouse, zebrafish and amphioxus to reveal an interesting pattern of the regulatory similarities as revealed by NACC, increasing with evolutionary proximity (Fig. 5.2).

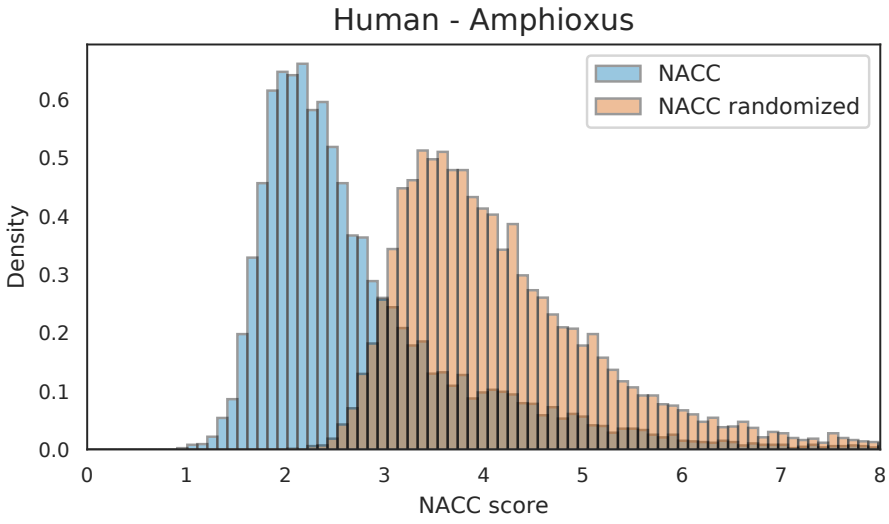


Figure 5.1: Distributions of NACC values for orthologous genes (in blue) or random orthology assignments (red) between Human and Amphioxus. Lower NACC values imply higher conservation of relative expression.[see notebook:13.17]

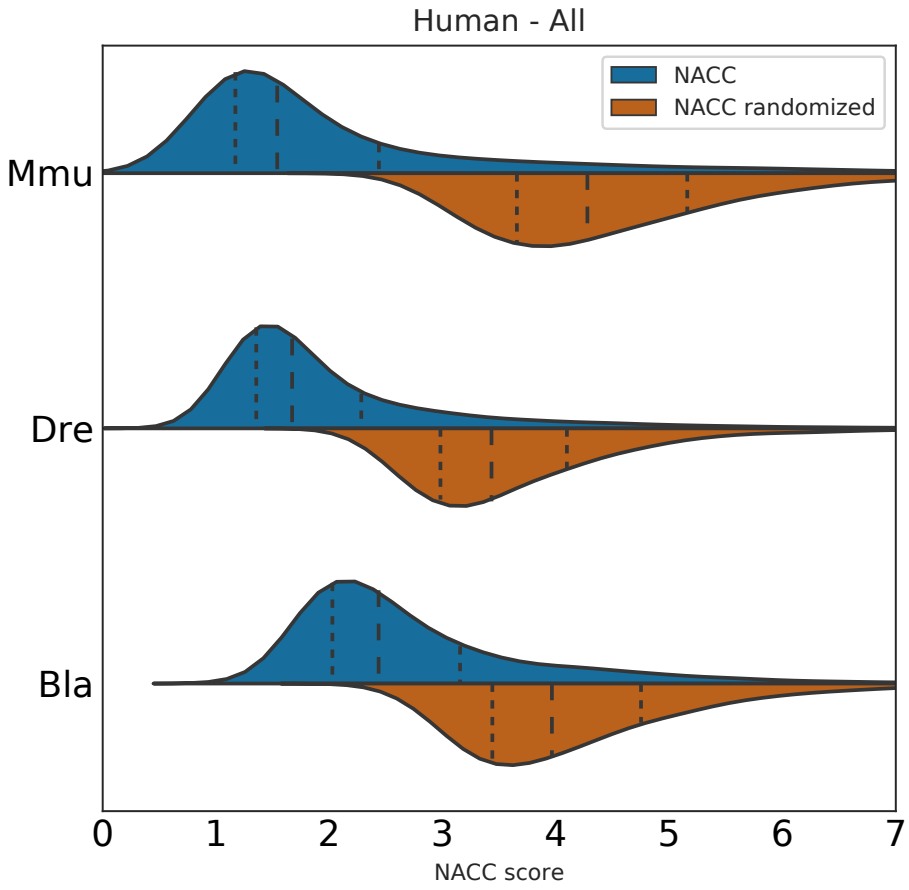


Figure 5.2: Distributions of NACC values for orthologous genes (in blue) or random orthology assignments (red) in three chordate species (mouse, zebrafish and amphioxus) against human. Lower NACC values imply higher conservation of relative expression. When we randomize the orthology connections, the distribution shifts to the right, meaning that on average genes that are co-expressed in one species, stay co-expressed in the other species as well. [see notebook:13.17]

5.2 THE PHYLOTYPIC PERIOD

The phylotypic period is a characteristic shared by vertebrates when morphological differences are their minimum. That is, vertebrates start their development with different forms, become more similar to the others at their phylotypic period and then diverge to the great spectrum of vertebrate form variety.

Previous comparative analyses among vertebrate transcriptomes [126] also showed a developmental stage of maximal similarity in gene expression, coinciding with the so-called vertebrate phylotypic period, in agreement with the hourglass model [127, 128]. However, similar comparisons with tunicates and amphioxus have thus far not resolved, at the transcriptomic level, a phylotypic period shared across all chordates [129].

As part of our work in [123], our collaborators tested whether a period of maximal gene expression similarity exists between amphioxus and vertebrates. They did pairwise comparisons of RNAseq data from developmental time courses in amphioxus, zebrafish, medaka, frog and chicken and revealed a consistent period of higher similarity between amphioxus and all vertebrate species (Fig. 5.3), corresponding to the 4-7 somite neurula (18-21 hpf).

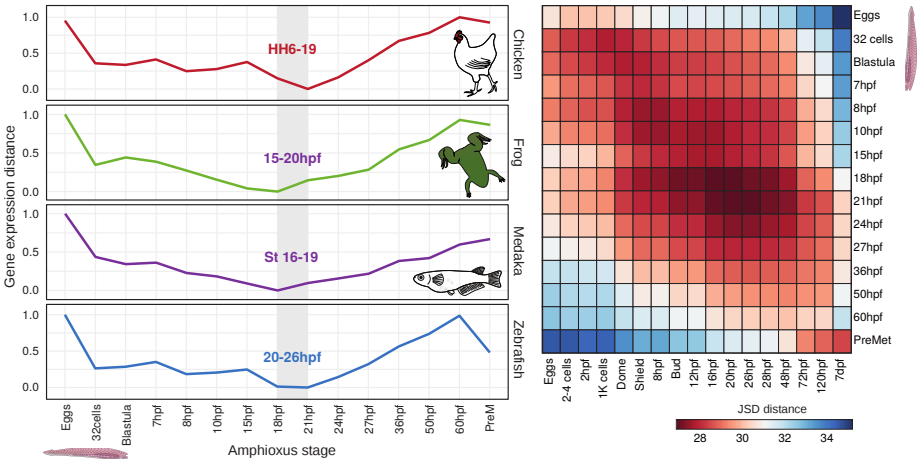


Figure 5.3: Left) Stages of minimal transcriptomic divergence (Jensen-Shannon Distance, JSD) to each amphioxus stage in four vertebrate species. The grey box outlines the 'phylotypic' period of minimal divergence, with the corresponding vertebrate periods indicated (the range given by the two closest stages). Dispersions correspond to the standard deviation computed over 100 bootstrap resamplings of the ortholog set. Right) Heatmap of pairwise transcriptomic distances (Jensen-Shannon metrics) between amphioxus and zebrafish stages. Smaller distance (red) indicates higher similarity.

Continuing our investigation of cis-regulatory conservation from Chapter 5.1, we set out to investigate if the transcriptomic phylotypic dynamics are reflected on the cis-regulatory level. If gene transcription levels are ultimately controlled by TF-DNA binding events, one would expect to find similar TF binding sites in the regulatory landscapes of genes that are expressed similarly.

For our analysis (see 13.15), we looked into atac peaks that are open in each developmental stage, and counted statistically significant hits for a large set (see chapter 13.1) of Position Weight Matrices. Having counts for all the PWMs at each stage, allows us to compare the stages to each other based on their cis-regulatory content.

After normalizing the PWM counts, we compute the correlation levels between stages. If two stages are highly correlated it means that the same PWMs are important or unimportant in those stages.

We present our correlation values in a heatmap, just like in (Fig. 5.3) (a), to showcase how our analysis revealed similar dynamics

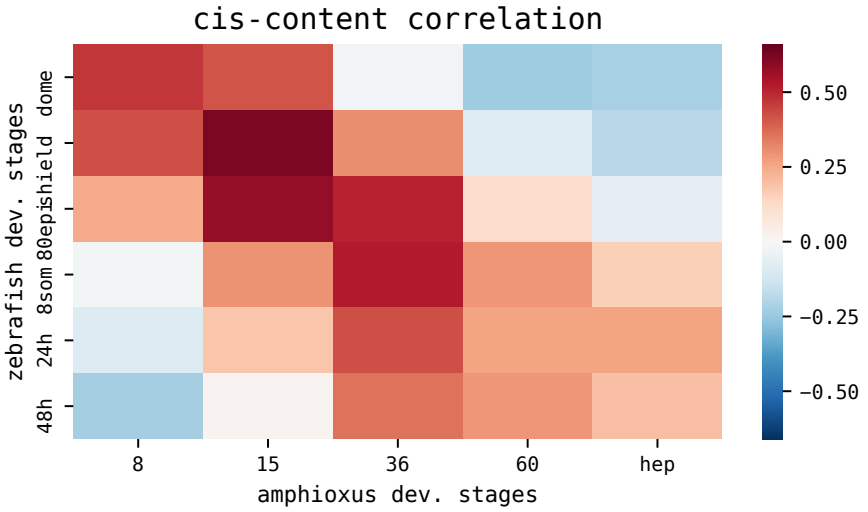


Figure 5.4: Zebrafish and amphioxus pairwise correlation of relative TF motif enrichment z-scores in ATAC-seq peaks active at different developmental stages. Four developmental stages and hepatic tissue from amphioxus are compared against six developmental stages of zebrafish. [see notebook:13.15]

We can also visualize our analysis as a line plot, again to showcase the similarities to the RNA-based comparisons. Our results are consistent with the hourglass model, with the two most similar stages (in terms of cis-regulatory content) being those directly preceding the RNA phylotypic period.

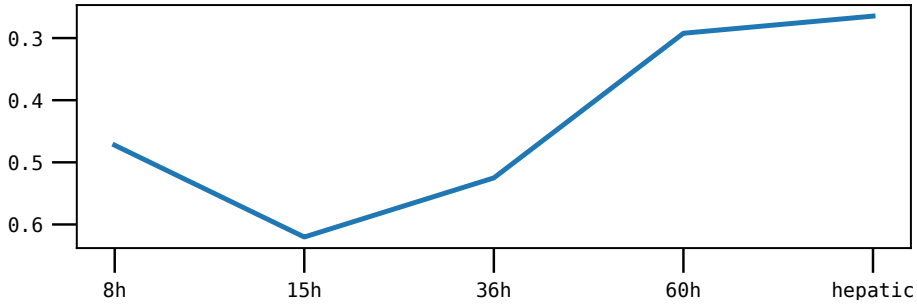


Figure 5.5: For each amphioxus stages (on the x axis), we plot the maximum similarity (the y axis is inverted) to any zebrafish stage. This similarity is what was plotted in Fig. 5.4 [see notebook: 13.15]

5.3 GENE MODULES

5.3.1 THE WGCNA ANALYSIS

Having established that at least on some level, there's conservation of gene networks between amphioxus and vertebrates, we wanted to investigate deeper. As part of our work for [123], our collaborators applied a gene clustering analysis on 17 and 27 RNAseq datasets in amphioxus and zebrafish respectively. The Weighted Gene Correlation Network Analysis clusters together genes that are highly correlated, meaning they have similarly low or high transcriptomic levels across the various tissues/samples/organs.

This analysis yielded 25 and 23 clusters in amphioxus and zebrafish, with varying size (number of genes in each cluster). We manually annotated those clusters, based on their overall gene expression and GO enrichment profiles. As an example, in Fig: 5.6 we show the annotation of two modules in each species. The two on the left were labeled as 'cilium' and the two on the right "Neural tube" and "Brain" respectively.

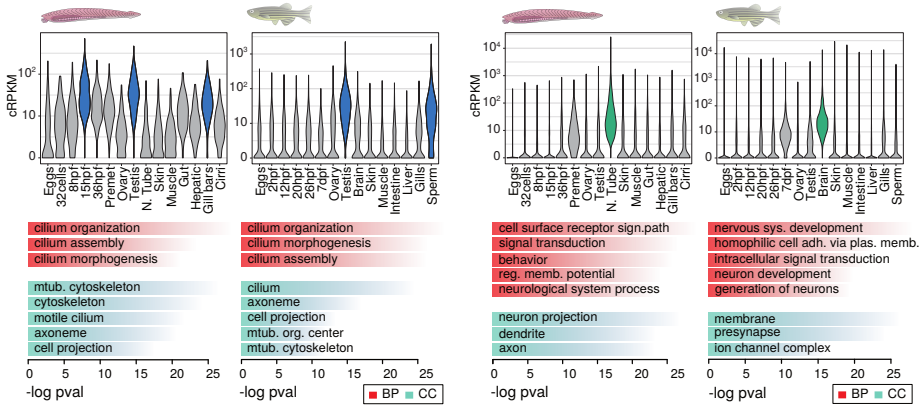


Figure 5.6: For two pairs of modules (from left to right, Cilium-amphioxus, Cilium-zebrafish, Neural Tube-amphioxus, Brain-zebrafish), the distribution of expression values using the cRPKM (corrected for mappability Reads Per Kbp and Million mapped reads) metric for all genes within the given module across each sample on the top. In the bottom, the enriched GO terms within each module (BP, Biological Process; CC, Cellular Component). From our work in [123]

5.3.2 HOMOLOGOUS GENE CONTENT

Having detected modules of genes in both species, and noticing how in some cases their activity is detected in the same homologous tissues, questions arise. For example; *are the two muscle modules composed of the same (orthologous) genes?* We did pairwise comparisons between the modules of the two species, examining if a given zebrafish module is enriched in orthologous genes that are included in any of the amphioxus modules, and present these pairwise enrichments in a heatmap 5.7. By further clustering the rows and columns of this matrix, we can reveal how some modules of genes have remained tightly co-regulated from their common ancestor to amphioxus and zebrafish.

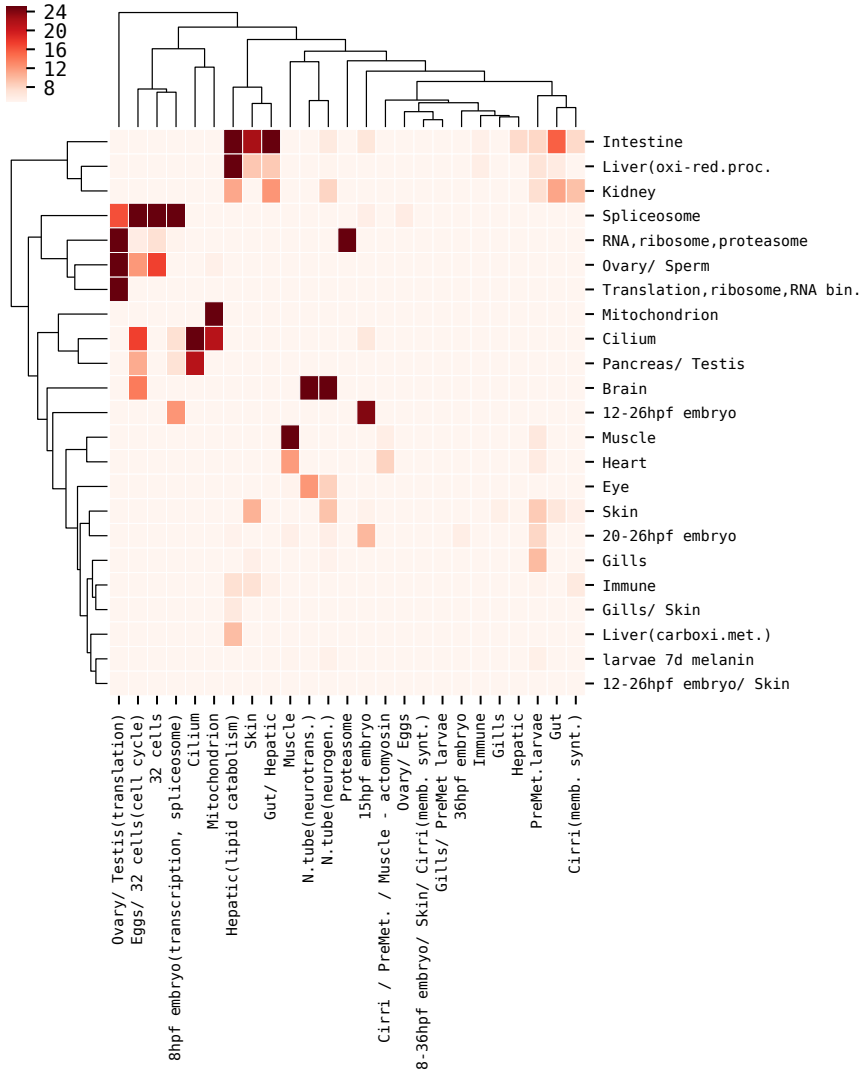


Figure 5.7: Heatmap showing the level of statistical significance of orthologous gene overlap between WGCNA modules in the two species as derived from hypergeometric tests. (-log10(pvalue)) [see notebook: 13.16]

5.3.3 CIS-REGULATORY CONTENT

We continued on the same spirit of comparing the modules between the species, but this time we wanted to investigate to what degree the modules are similar on their cis-regulatory context. Similarly to what we did for the phylotypic cis-regulatory analysis, we counted instances of significant PWM hits inside a genomic region assigned to each gene. After summing the counts for each module and properly normalizing, we can compare modules just like we compared developmental stages in Fig. 5.4.

We tested the correlation coefficient of these relative motif enrichment scores for all of our intra-species pairs of modules and found significant positive values for a large fraction of the pairs that also displayed high homology conservation (in Fig. 5.7). In such cases, the most enriched TF motifs within each cluster were highly consistent between amphioxus and zebrafish and included TFs with well-known roles in tissue-specific development and differentiation e.g. Rfx for cilia, Hox for brain, Hnf1a for liver and gut, Ghrl for skin, Mef2 for muscle, and Elf1 and Spic for immune function. Some of these are shown in Fig. 5.9, and the rest can be computed in the relevant analysis notebook (chapter 13.16).

In summary, our results show a high level of transcriptomic and cis-regulatory conservation underlying basic cellular processes and differentiated tissues in adult amphioxus and vertebrates.

Figure 5.8: Heatmap of all pairwise correlations between the modules of the two species, based on the relative TF motif z-scores for each module. Modules are ordered according to the clustering in Fig. 5.7. [see notebook: 13.16]

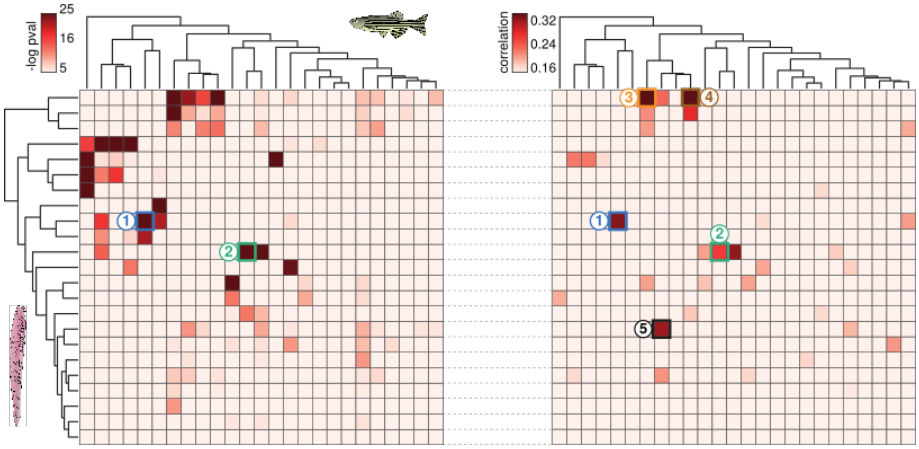


Figure 5.9: Putting these two heatmaps next to each-other, shows how some pairs of modules are highly conserved both at the gene and cis-regulatory levels. The heatmap from Fig. 5.7 is on the left and the heatmap from Fig. 5.8 on the right. Some interesting case are highlighted. From our work in [123].

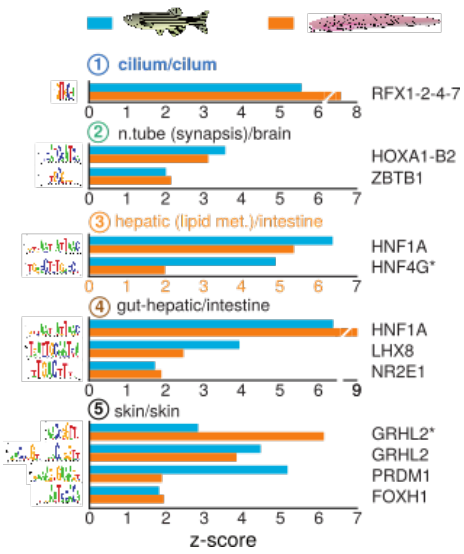


Figure 5.10: Examples of TF binding site motifs with high z-scores from highly correlated pairs of modules between zebrafish and amphioxus.From our work in [123]

6

Regulatory content and gene fate after WGD

6.0.1 GENE FATE AFTER WGD

As was mentioned before, Whole Genome Duplications are significant events in the developmental history of organisms. Comparing the slow evolving amphioxus's genome which contains no whole-genome duplications to vertebrate genomes that contain at least two, allows us to investigate what happens to genes after a WGD event.

Based on the reconstructed homologous gene families, we split genes in categories based on how many copies of the gene have been retained in mouse, to keep comparisons between vertebrates consistent. This yielded four categories of genes. The first one, where the gene has been retained in a single copy in mouse ('1-1'), is enriched for genes that can be labeled as "House Keeping" (see 13.12), that is genes that are involved in basic cellular functions that are shared acroos cell types. The other three categories, where genes are retained from 2 to 4 copies, are in contrast enriched for genes implicated in transcriptional regulation or development.

By splitting the genes in these categories we can elucidate how Trans-Dev genes are more likely to be retained in multiple copies, as was discussed in 2.4.3, and

how these genes are more likely to have larger regulatory landscapes with more cis-regulatory elements both in amphioxus and in vertebrates (Fig. 6.1) The increase in zebrafish was not attributable to the third WGD of teleosts, because using published ATAC-seq datasets we found an even stronger pattern for mouse (see the notebook in chapter 13.12).

Plotting only the 'Housekeeping' and 'Trans-Dev' labeled genes also highlights the same dynamic (Fig. 6.2)

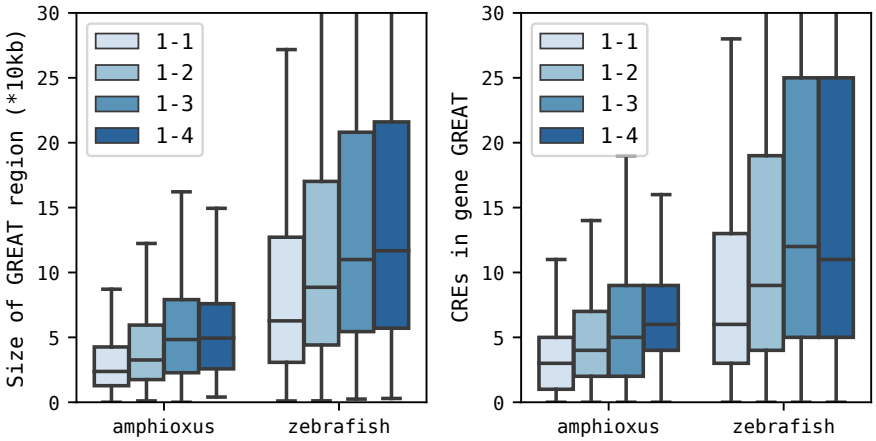


Figure 6.1: The size of gene GREAT regions (left) and counts of CREs in gene GREAT regions (right) split in categories based on WGD retention. Genes found in a single copy in both amphioxus and mouse are denoted as '1-1'. Genes found in two copies in mouse but in a single copy in amphioxus are '1-2' and so on.[see notebook: 13.12]

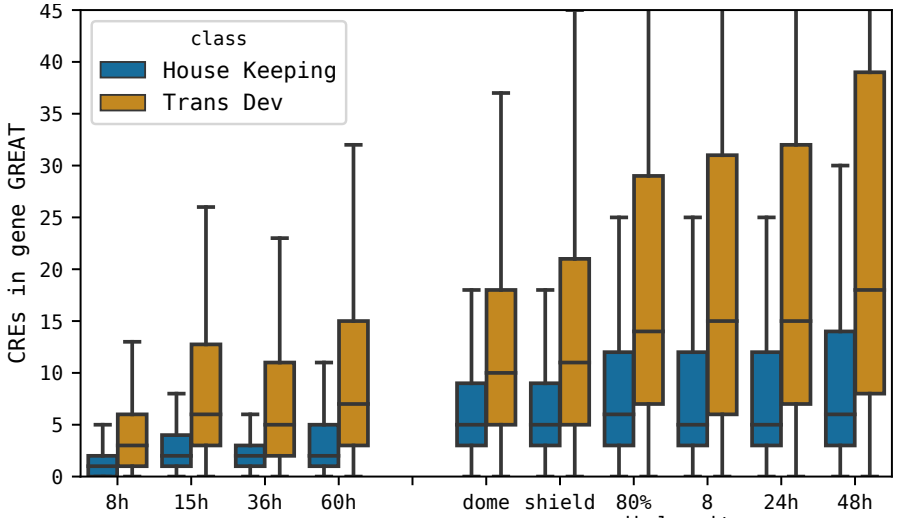


Figure 6.2: The distribution of counts of CREs in gene GREAT regions, split in Transdev and Housekeeping subsets [see notebook: 13.12]

6.0.2 CRES PER PARALOG

Interestingly, the number of CREs is very uneven between ohnologs [1]: the paralog with the lowest number of associated CREs generally has a comparable number to the amphioxus ortholog, but dramatic regulatory expansions were observed for some ohnologs (Fig. 6.3). The same patterns were detected for all amphioxus and zebrafish developmental stages (see notebook: 13.12).

[1] Gene duplicates resulting from WGD, in honor of Susumu Ohno who first considered the implications of gene duplication in evolution [120]

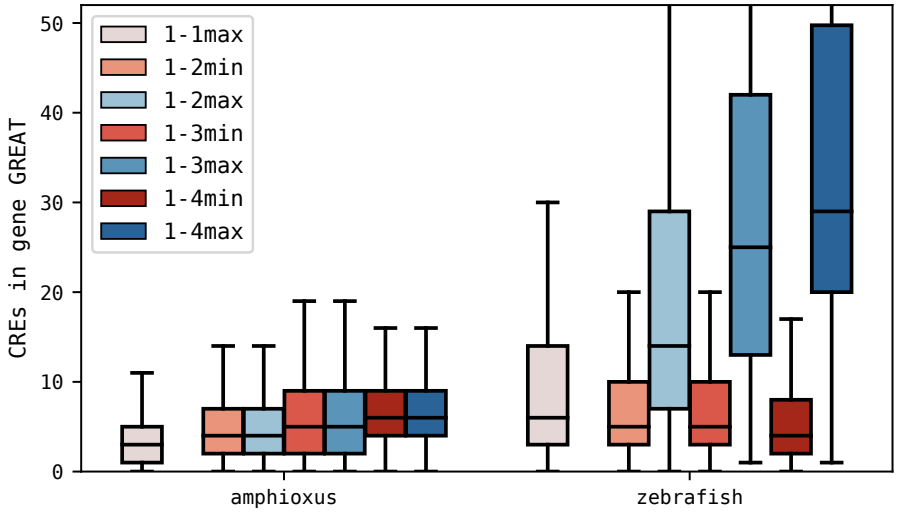


Figure 6.3: The distribution of counts of CREs in gene GREAT regions, split in categories based on WGD retention. Only the minimum and maximum count per gene family is shown [see notebook: 13.12]

6.0.3 INCREASED REGULATORY COMPLEXITY IN FUNCTIONALLY SPECIALIZED OHNOLOGS

DDC predicts that individual duplicate genes would each have more restricted expression than an unduplicated outgroup, but their summation would not. To investigate this, our collaborators focused on seven homologous tissues and two equivalent developmental stages in amphioxus, zebrafish, frog and mouse, and marked the expression of each gene in each sample as on or off based on fixed cut-offs.

By counting an expression-bias metric that they called *delta zebra*, for each gene and plotting the distribution of these values, they investigate differences between amphioxus and vertebrates (Fig. 6.4). The metric is defined as the number of domains expressed in zebrafish minus the number of domains expressed in amphioxus. The left skewing of the distribution plotted in the middle of Fig.6.4 for example, shows how the majority of ohnologs in zebrafish lose expression domains in comparison to their amphioxus counterparts.

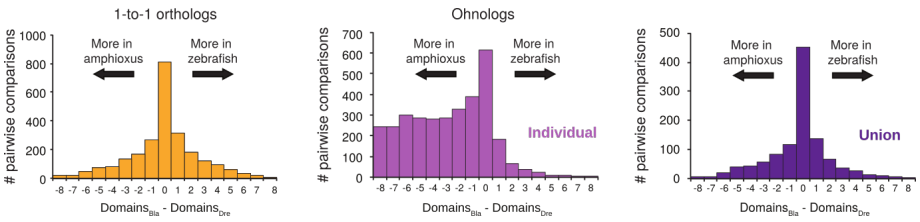


Figure 6.4: Distribution of the difference in positive domains between zebrafish and amphioxus for 1-to-1 orthologs (left), individual ohnologs (middle) and the union of all vertebrate ohnologs in a family (right). See Fig. 6.5 for schematic of how the values are calculated.

For genes that are conserved in single copies in vertebrates (Fig. 6.4 left), the distribution of values is centered around 0, indicating that this group of genes tends to retain the ancestral expression domains. In contrast, when vertebrate genes from families with multiple copies were compared to their single amphioxus ortholog, the distributions were strongly skewed, with many vertebrate genes displaying far more restricted expression domains (Fig. 6.4 middle). Remarkably, the symmetrical pattern was fully recovered when the expression of all vertebrate members was combined or the raw expression values summed for each member within a paralogy group (Fig. 6.4 right).

Although the above findings are consistent with the DDC model, they are also compatible with an alternative model in which a subset of duplicate genes becomes more 'specialized' in expression pattern while one or more paralogs retain the ancestral broader expression. To distinguish between these alternatives, we analyzed a subset of multi-gene families in which both the single amphioxus ortholog and the union of the vertebrate ohnologs were expressed across all nine compared samples.

We then identified (Fig. 6.5 c): (i) gene families in which all vertebrate paralogs were expressed in all domains ('redundancy'), (ii) gene families in which none of the vertebrate members had expression across all domains ('subfunctionalization'), and (iii) gene families in which one or more vertebrate ohnologs were expressed in all domains, but at least one ohnolog was not ('specialization').

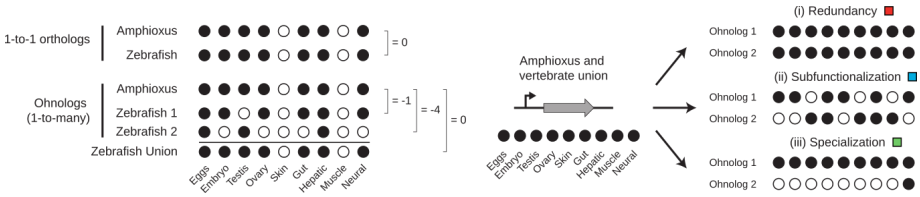


Figure 6.5: Left: Schematic summary of the analysis shown in Fig. 6.4. Expression is binarized (on or off) for each amphioxus and vertebrate gene across nine comparable samples, based on an arbitrary expression cut-off (normalized cRPKM>5). For each vertebrate gene, the number of positive expression domains is subtracted from the number of domains in which the single amphioxus ortholog is expressed. Black/White circles represent on/off expression, respectively. Right: Schematic summary of the analyses shown in Fig. 6.6, representing the three possible fates after WGD: Redundancy, all ohnologs are expressed in all domains; Subfunctionalization, none of the ohnologs are expressed in all domains; Specialization, at least one of the ohnologs in expressed in all domains, but at least one is not.

We obtained very similar results for the three studied vertebrate species (6.6 a): between 80 and 88% of gene families fell into either subfunctionalization or specialization, meaning they show a loss of ancestral expression domains in at least one member. Moreover, we found specialization to be consistently more frequent than subfunctionalization as a fate for vertebrate ohnologs.

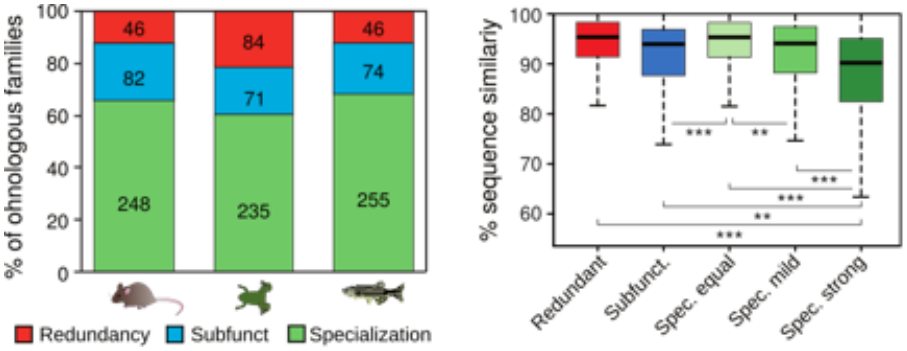


Figure 6.6: Left: Distribution of fates after WGD for families of ohnologs inferred to be ancestrally expressed in all nine studied domains for each vertebrate species. Right: Distribution of the percentage of nucleotide sequence similarity between human and mouse for different classes of ohnologs based on their fate after WGD. Ohnologs from specialized families are divided into "Spec. equal" (maintaining all expression domains), "Spec. mild" (which have lost expression domains, but maintained more than two), "Spec. strong" (with two or fewer remaining expression domains)

Interestingly, ohnologs that have experienced strong specialization (defined as having two or fewer remaining expression domains) showed the fastest rates of sequence evolution and the highest dN/dS ratio between human and mouse, whereas genes from redundant families and those ohnologs from specialized families that retain ancestral expression displayed the lowest levels of sequence divergence (Fig.6.6, right).

Surprisingly, specialization and subfunctionalization were not correlated with an obsious loss of CREs as the DDC model would assume. In fact, we observed that these categories of genes were more likely to have more CREs found inside their cis landscapes (Fig. 6.7)

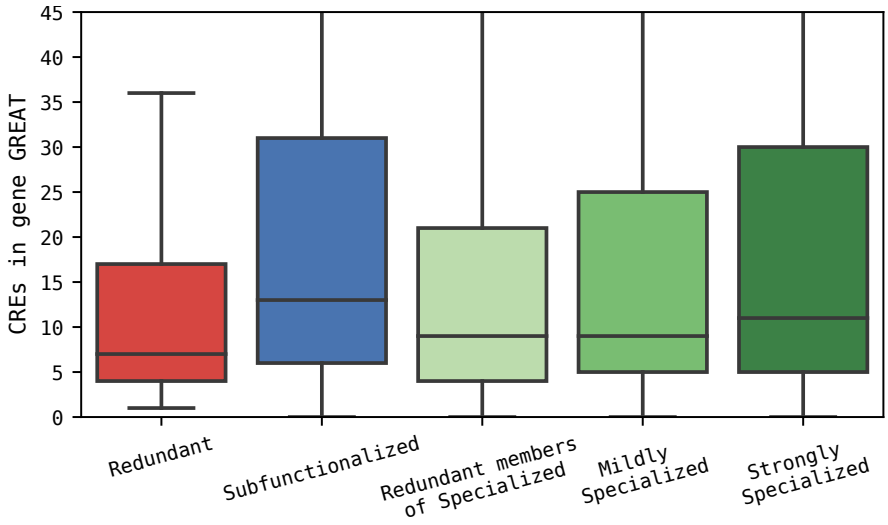


Figure 6.7: Here we plot the distribution of counts of CREs in gene GREAT regions. We split the genes in categories based on WGD fate, as was discussed in the text and shown in the previous figures [see notebook: 13.12]

Furthermore, we found that ohnologs from specialized families that have lost expression domains showed significantly more associated regulatory elements than those with the full ancestral expression. In fact, we observed a strong positive relationship between the number of ancestral expression domains lost and the number of putative regulatory elements associated with specialized ohnologs (Fig. 6.8)

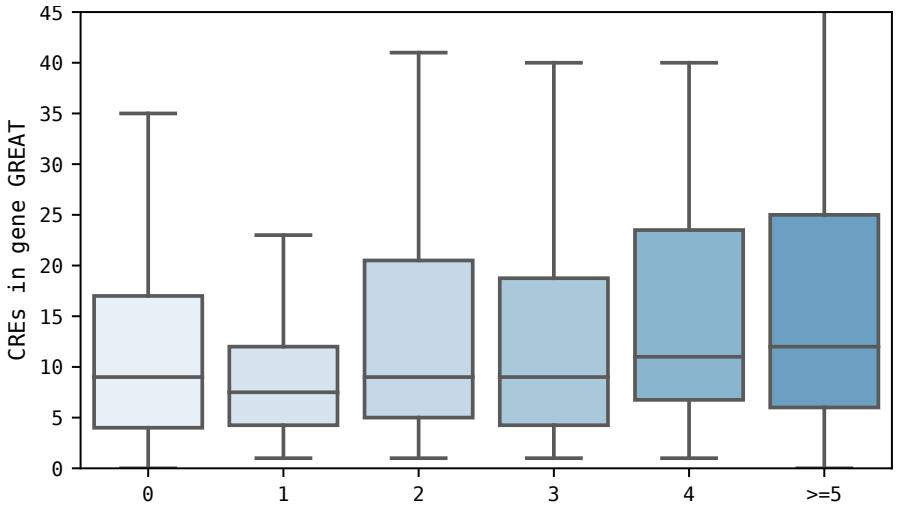


Figure 6.8: The distribution of counts of CREs in gene GREAT regions, by number of expression domains lost when compared to amphioxus homologue [see notebook: 13.12]

This implies that specialization of gene expression after WGD does not occur primarily through loss of ancestral tissue-specific regulatory elements, but rather by complex remodeling of regulatory landscapes involving recruitment of novel tissue- specific regulatory elements.

Part III

Results: Detecting TF binding with a Neural Network

As was mentioned before, an important goal of current transcription regulation research is to determine where proteins are bound on the genome. This changes from one biological context to the next so ChIP-seq which gives us the best attainable answer to our question needs to be repeated in any context that we want to investigate in detail.

Besides the expense in consumables and time, ChIP-seq might be restricted by antibody availability and does not offer base-pair resolution. To obtain such resolution, we employ computational approaches, of which the most commonly used are the PWMs.

When we use a PWM for a DNA-binding protein, we obtain a large number of sites on the genome, each accompanied by a PWM-score indicating the predicted certainty that the protein is indeed bound on that location. Since this is a computational technique, these are putative binding sites and since this is a sequence only approach, these putative sites are applicable to all biological contexts. In a context for which we have available ChIP-seq data, we can categorize the putative sites into true sites, those that overlap with peaks of the ChIP-seq signal, and false sites, those that do not overlap. This kind of exercise, classifying a set of elements into two groups, is called binary classification.

The problem is then posed as such: *"For a set of putative PWM sites, how good is any computational method at classifying them between true and false?"*.

Since our proposed model includes ATAC-seq data, we refine the problem to *"For a set of putative PWM sites that are found inside ATAC-seq peaks, how good is any computational method at classifying them between true and false?"*.

Classification is a task in which NNs excel, thus we expected that a NN model that is designed similarly to previous similar attempts but also incorporates ATAC-seq signal, should be able to better detect binding sites thanks to the additional information that the accessibility assay carries.

A NNs proficiency at any task depends on its architecture (nature) and its training (nurture). The architecture of a NN refers to a number of choices with regards to the design of the network; how many layers should there be? how big should the neurons be? which optimization algorithm should be used?

The training of the NN is the process in which the NN that we designed is given data in order to update its internal matrices. A number of choices need to be made

here as well; which data should we use for training? how much of the data should we give to the model at each step ? When should we stop the training?

In both architecture and training, at the current state of NNs, there is no recipe on how to make most of the choices. Consequently most of our design choices such as the optimization algorithm, activation functions, the number and design of layers, the widths of the neurons at each layer, were made empirically, after determining some possible choices based on relevant literature and testing many of them. The model can be examined in detail in the relevant python script found in the repository of our implementation[2], made with TensorFlow [130]. The repository also contains accompanying python code to facilitate the training and testing of new models.

In chapter 7 we present the main architectural choices of our model, the number of neurons, number of layers, how they are connected etc.

6.1 TRAINING CONCEPTS

After settling on these design choices, there is further parameters that will influence the ability of our model to classify TFbs. During the training phase, the network updates its internal matrices depending on the data that we give it as a training set. The quality of data, as well as the patterns in which the data is given and the speed with which we configure the model to update itself all play an important role in the success of the training phase. Furthermore, there is no set point at which the training finished so terminating the training phase is itself a problem to be solved.

6.1.1 CHOICE OF DATA

Regarding our choice of data, since we designed our model around ATAC-seq signal, we found it appropriate to also design the dataset around the question that a researcher with an available ATAC-seq set would ask: "Which of the PWM

[2] https://gitlab.com/panosfirbas/nimrod/blob/master/nimrod/model.py

hits inside ATAC-seq peaks are real". Filtering for PWM hits inside ATAC-seq peaks is a reasonable first filter for our putative PWM hits on the whole genome since they are most likely to be transcriptionally relevant. In order to train our model we need to show it putative PWMs and tell it whether they are real or not. We defined real hits those PWM hits that overlap a CHIP-seq peak at the appropriate cell line or tissue. We will use this CHIP-seq based labeling as a point of reference or golden standard, in order to perform the model training but also in order to evaluate the performance of all models and tools. When training a Neural Network, it is important to keep a part of the data out of the training set so that the network doesn't "see" it. We define two such subsets, a validation set to be used at multiple points during the training phase in order to determine when the network has finished learning, and a test set which will only be used once at the end to determine the final competence of the network.

For this, we set aside about 9% of the PWM hits from each of our datasets for the validation subset, and another 9% for a test subset. Our model will train with the remaining data which forms the training subset, use the validation subset to detect convergence, and will be tested on the test subset.

6.1.2 BATCH SIZE

The training of a NN happens in steps and in each step, a batch of PWM hits are used. In each batch/step, the network computes its output values for each of the elements of the batch and then compares those to the real labels of the elements of the batch. Based on the differences, the network calculates the direction towards which it needs to change its internal matrix values. Large batches allow the network to determine this direction more accurately at each batch, which makes the network train faster. Smaller batches on the other hand make the updates on each batch more erratic and as a consequence the training is slower but this allows the network to explore more and as a result smaller batches often lead to better final classification power. In different words, smaller batch sizes are more likely to escape local minima.

6.1.3 LEARNING RATE

While the batch size influences the direction of the model's updates, the learning rate influences the size of the step towards that direction. It is common practice to start the training with a large learning rate to take the first simple steps fast, and to then decrease the rate in order to better explore the search space. We designed our model with a decaying learning rate that offers this effect but the initial learning rate and rate of decay still influence the success of the training phase.

6.1.4 EARLY STOPPING

As was mentioned before, there is no set point at which a NN's training finished. A model typically goes over the entire training set multiple times, called epochs. To determine the end of training, we monitor a NN's progress by regularly testing the network against the validation subset. By regularly testing the model we hope to detect a point at which our model has stopped improving. To do that we need a metric of how well our NN is doing.

6.1.5 EVALUATING A CLASSIFIER

As we mentioned before, our problem is setup as a binary classification problem; "Which of these putative PWM hits are real and which are false?".Binary classifiers are typically evaluated with Receiver Operating Characteristic (ROC) curves [131] where True Positive Rate (TPR) and False Positive Rate (FPR) are plotted against each other. In these plots, a bad classifier [3] will produce a line on the 45 degree diagonal, while a good classifier will produce a downward facing curve. A perfect classifier would produce a vertical line, from 0,0 to 0,1 since the FPR would remain at 0 while TPR increases with increasing number of elements.

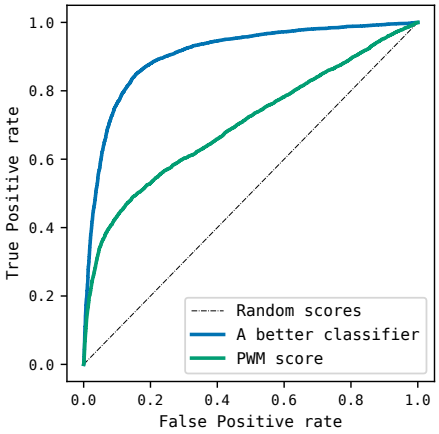


Figure 6.9: ROC curves for three classifiers. The diagonal dotted line is what we would expect from a random classifier. The PWM score can be used itself as a classifier but it is not very strong. In blue, we plot what a theoretically better classifier would look like.

The score of a PWM can be evaluated as a binary classifier. In Fig. 6.9 we plot the ROC curve for the PWMs from one of our datasets [4], as well as the ROC curve for a better classifier of that set, to showcase how these plots are visually intuitive in showing which classifier is better.

The ROC curve can be quantified beyond the visual interpretation with a summary statistic such as the Area Under the Curve (AUC) score which can be interpreted in a variety of ways such as the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one. For the dataset in 6.9, the PWM accomplices an AUC score of 0.66 and the other classifier reaches 0.9.

[3] For example, randomly produced scores

[4] M1957 CTCF hits overlapped with CTCF ChIP-seq in forebrain of mouse

6.2 CTCF AND P63

To design and implement our NN we focused on two TFs, CTCF and p63. We chose these two factors for a number of reasons. First is availability, since as we discussed, we need datasets from biological samples for which both ATAC-seq and ChIP-seq against a protein of interest are available. Another limiting factor, was our goal to make cross-species comparisons, so we needed cases in which we have data for several species.

CTCF is particularly interesting and well suited protein for our endeavor. It is well studied with plenty of data being available, it has good PWMs, and binds strongly to DNA, leaving strong footprints on the ATAC-seq signal. CTCF has 11 zinc-fingers, small protein structural domains that are characterized by the inclusion of one or more zinc ions and is a member of the C2H2 family of factors that was briefly mentioned earlier as one of the most expanded families in human (chapter 1.1).

Evolutionary speaking, CTCF is restricted to bilaterians and is highly conserved across most of the animal evolutionary tree [132, 133].

It binds on often conserved [134] sites of the genome. There is good evidence that the orientation of the binding is important and that distal CTCF proteins come together and with the help of another protein, cohesin, create a strongly bound homodimer which forces the DNA to form a loop [135].

CTCF sites preferentially flank transcription factor genes [134] and have been associated with the orchestration of conserved 3D architecture of the genome [136].

Given its ubiquity in animals and clear connections to transcription regulation, genome compartmentalization and development, CTCF is considered and important part of the evolution of animal form.

For human, we identified three cells lines with available data [5] and for mouse nine embryonic tissues, two of which in two developmental stages for a total of 11 samples (Table 13.1). From this, we compiled two super-sets, a human and a mouse one with which we trained our models.

[5] ATAC-seq and ChIP-seq against CTCF

The tumor protein p63, aka TP63, is a master regulator of epidermal development[137] and another great candidate for our experiments with the neural network.

P63 effects major transcriptional changes and contributes to dynamic long-range chromatin interactions[138]. It has been shown to regulate keratinocyte proliferation and epidermal stratification[139]. We identified two cases that fit our data criteria, one in zebrafish embryo and one in a human keratinocytes, and compiled the datasets listed in table 13.2.

Following, we will describe the architecture of our NN and will then implement the best training conditions using the data for these 2 factors. Furthermore, we will compare our model with similar previously published tools and show how our NN outperforms them and how it can be used across-species.

7

Architecture

7.1 THE FIRST TWO LAYERS

This model will take as input genomic sequence and ATAC-seq signal for a window of 1000 basepairs around PWMhits. The genomic sequence is encoded in the one-hot scheme; for each genomic position we present five values, one for each possible base (plus 'n'). Of these five values, at each position, only the one corresponding to the position's nucleotide is marked as 1, while the other four are marked as 0 (Fig. 7.1).

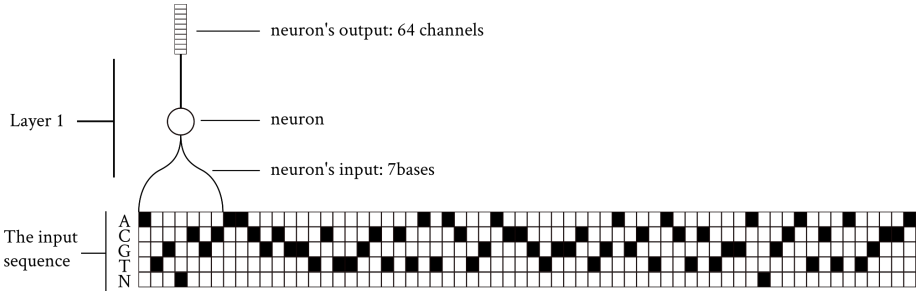


Figure 7.1: The genomic sequence signal as a one-hot sequence with zero shown as white and one shown as black, and a neuron of the first seq-based layer. The neurons of our model's first layer read seven positions of the input sequence and output.

The first layer on the genomic sequence consists of neurons that read 7 basepairs [1] of the raw signal each. In Fig. 7.1 we see one such neuron from layer 1 while in 7.2 we see all the neurons of layer 2 and the output of this first layer. Please note that the colors in the matrix positions (squares) in those figures were an aesthetic choice to show that different positions in those matrices have different values. The colors do not contain any further meaning.

As was mentioned earlier, each neuron outputs a number of values, one for each of the matrices that the layer is designed to contain. These matrices are 7 columns wide (one for each position that the neuron 'reads') and 5 rows tall (one row per possible base, including 'n'), much like a PWM. In our model, each neuron of the first layer concurrently learns 64 [2] different matrices and outputs 64 channels of information, or 64 scores of matrices. The total size of the output of the first layer is 994x64, 994 neurons [3] and 64 channels.

These 64 channels are taken as input by the second layer, which reads the outputs of 25 of the layer 1 neurons. The matrices of this layer are 25 columns wide (for

[1] A choice made empirically

[2] Another empirically made choice, this number should be high enough to allow capturing complexity but not superfluously high because that slows down the model unnecessarily

[3] As many 7bp reading neurons as you can fit in 1000bp sequence

each of the inputs taken) and 64 rows tall, one for each channel that the underlying layer outputs.

This second layer outputs 128 channels of information (Fig. 7.2).

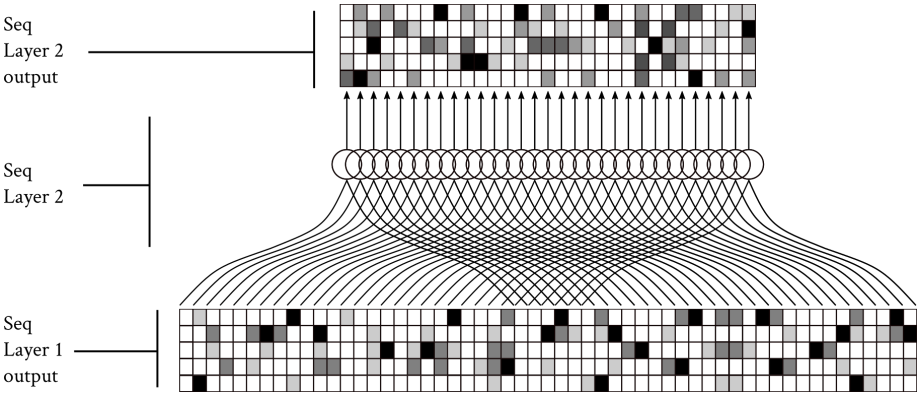


Figure 7.2: The neurons of Seq Layer 2, read 25 of the neurons of the underlying layer each. Each neuron then combines the input values with its inner weight matrices and outputs one value for each for a total of 128 output channels. The inner weight matrices are shared by neurons of the same layer and are learned during the training phase of the model. Please note that the colors in the matrix positions (squares) in this figure and the following were an aesthetic choice to show that different positions in the matrices have different values. The colors do not contain any further meaning.

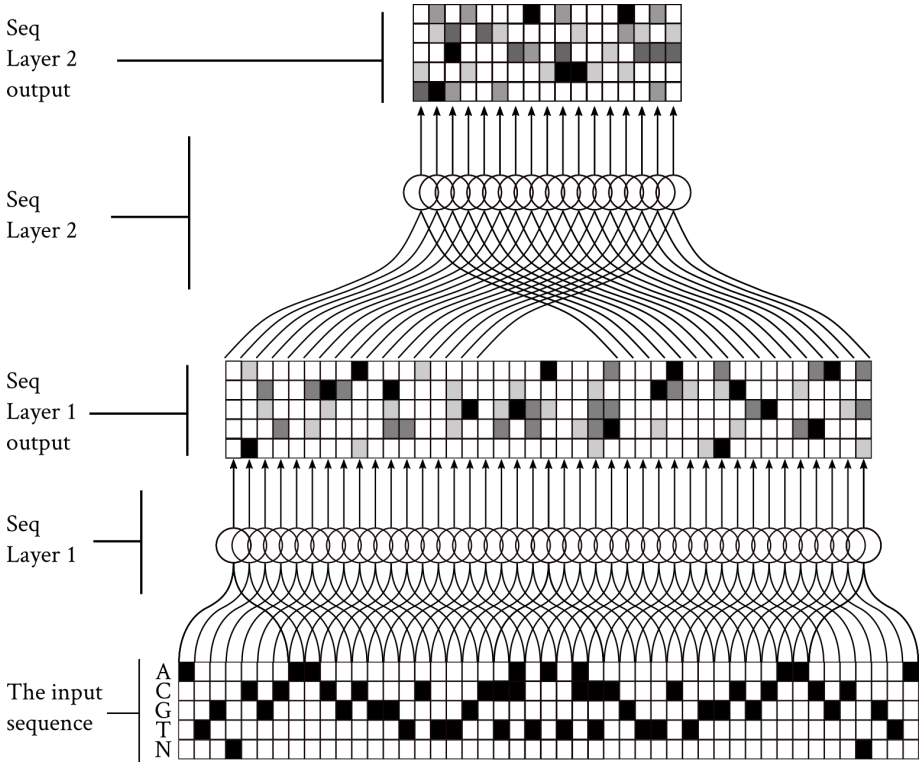


Figure 7.3: AN overview of the first two sequence layers of Nimrod

In Fig. 7.3 we see the first two layers of the model, working on the genomic sequence input. Two more layers with the same configuration [4] are 'built' on top of the ATAC-seq signal (Fig. 7.4).

The ATAC-seq signal is a base-pair accurate count of cutting events on the genome. These cutting events, by nature of paired-end sequencing, have happened in pairs. The distance of each read to its paired read offers additional information to the system so we did not want to completely flatten the signal to a one-dimensional genome-long array.

[4] The neurons of the first layer read 7 positions and the neurons of the second layer read 25 neurons

In that spirit, the ATAC-seq signal is grouped in three layers; small, medium and large fragment size.

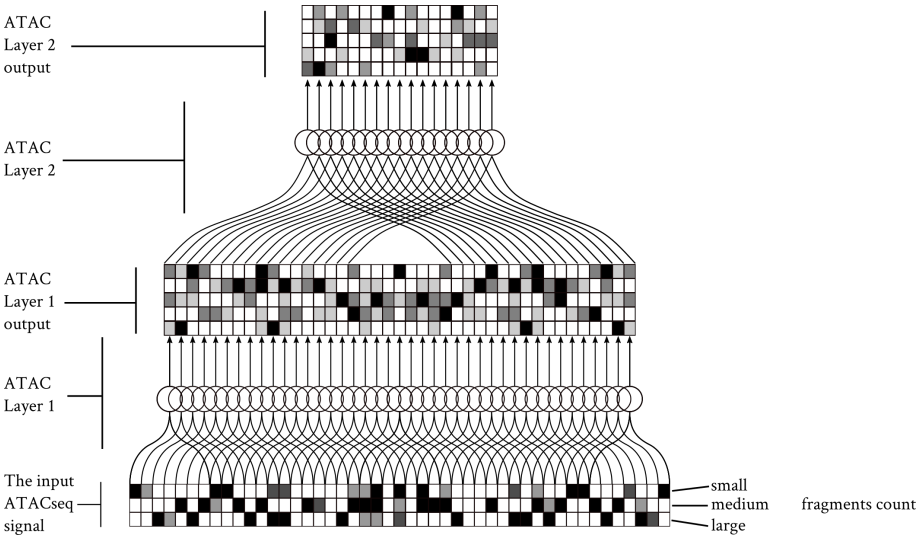


Figure 7.4: The first two layers over the ATAC-seq signal are almost identical to the ones that we designed over the genomic sequence. Notice that the input array has 3 channels instead of the 5 of the sequence-based input array. These layers are almost identical in architecture but will learn their own inner weight matrices, separately from the sequence layers.

7.2 MERGING THE FIRST TWO LAYERS

In the next step, the two outer layers (Seq layer 2 and ATAC layer 2) get 'zipped'/merged into a single layer and the now merged signal gets passed to more layers of the model for even higher order feature discovery. In Fig. 7.5 we see how the two layers are merged into a "zipper" layer.

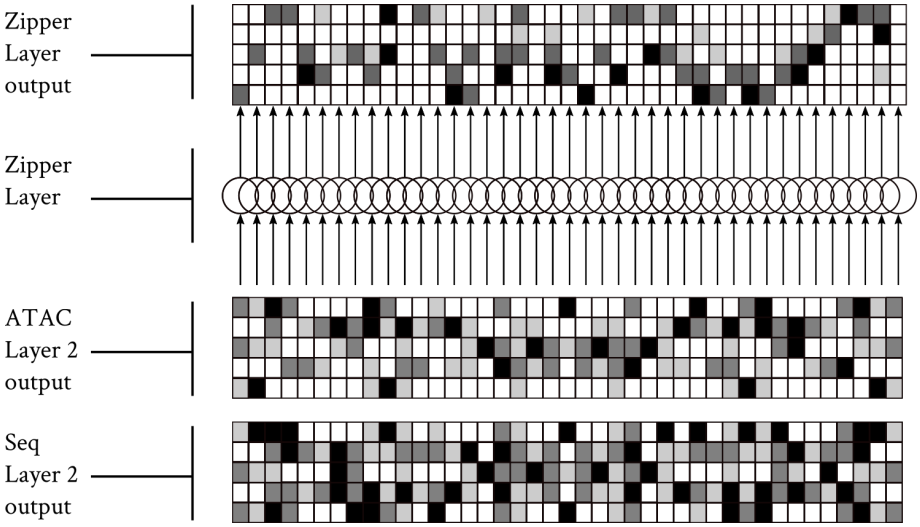


Figure 7.5: Each neuron of the zipper layer reads the output of one neuron from Seq Layer 2 and one from ATAC Layer 2 and outputs 256 channels.

7.3 THE DEEPER LAYERS

The signal gets condensed through a max-pooling layer in the next step. In this layer the neurons do not overlap and so the number of neurons is drastically reduced from 970 to 194 (Fig. 7.6). They output 256 values each.

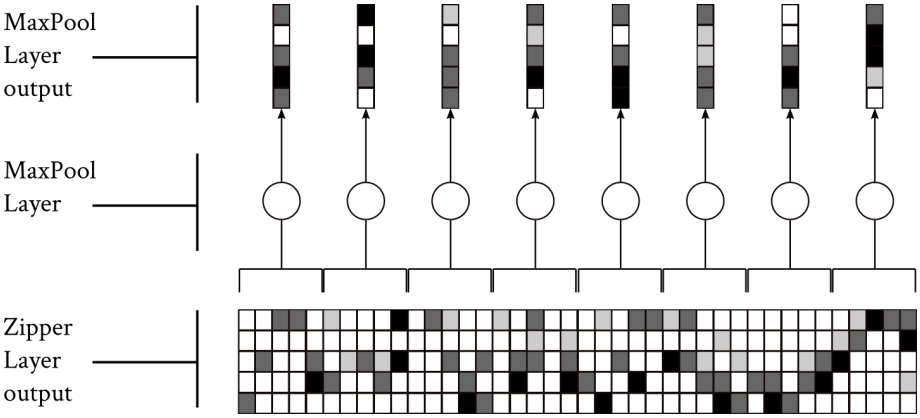


Figure 7.6: The neurons of the maxpool layer don't overlap so their number is reduced dramatically.

The signal finally converges to the output layer of the NN after two more layers, a convolutional one and a fully connected one which collapses the network into a single neuron. The output layer at the end gives us the network's output, two values indicating how much confidence the network has on applying either of the potential labels to the input (Fig. 7.7).

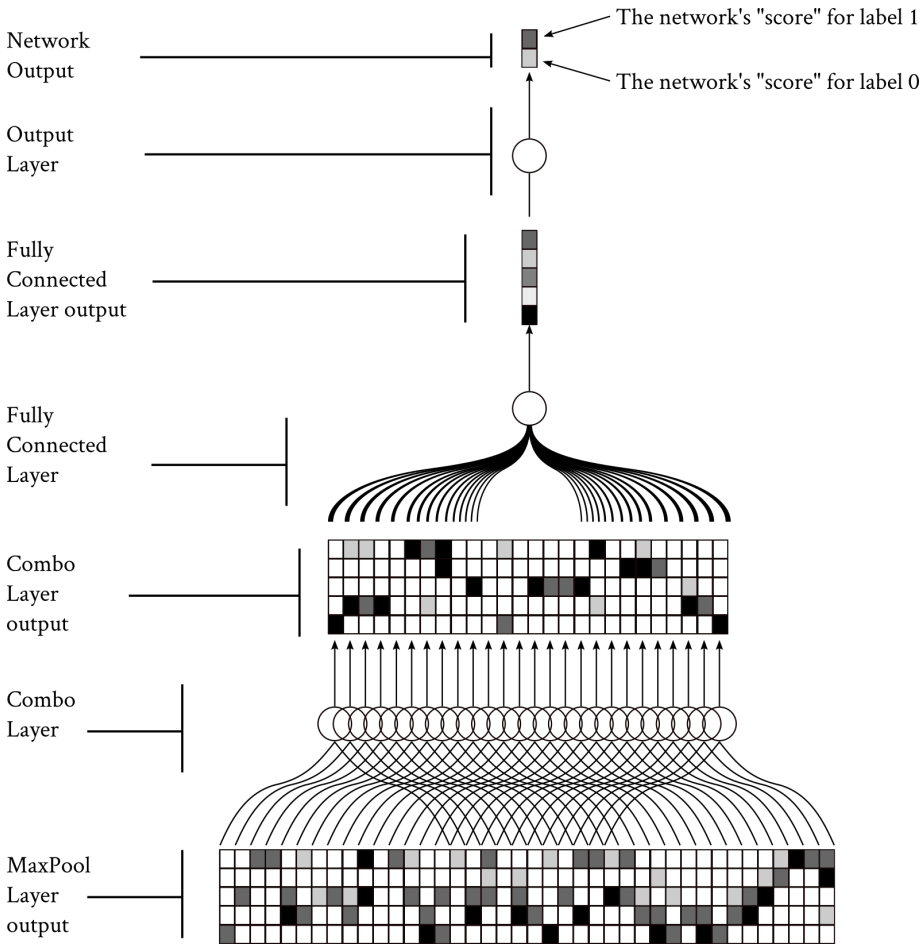


Figure 7.7: The model as it condenses into a single pair of values in the layers after the maxpool step. A final convolution layer (Combo) reads the output of the maxpool layer and is in turn read by a fully connected layer which condenses the model's signal in a single neuron that outputs 256 channels. Those are finally read by the output neuron that outputs 2 values.

In Fig. 7.8 we have a quick overview of our model.

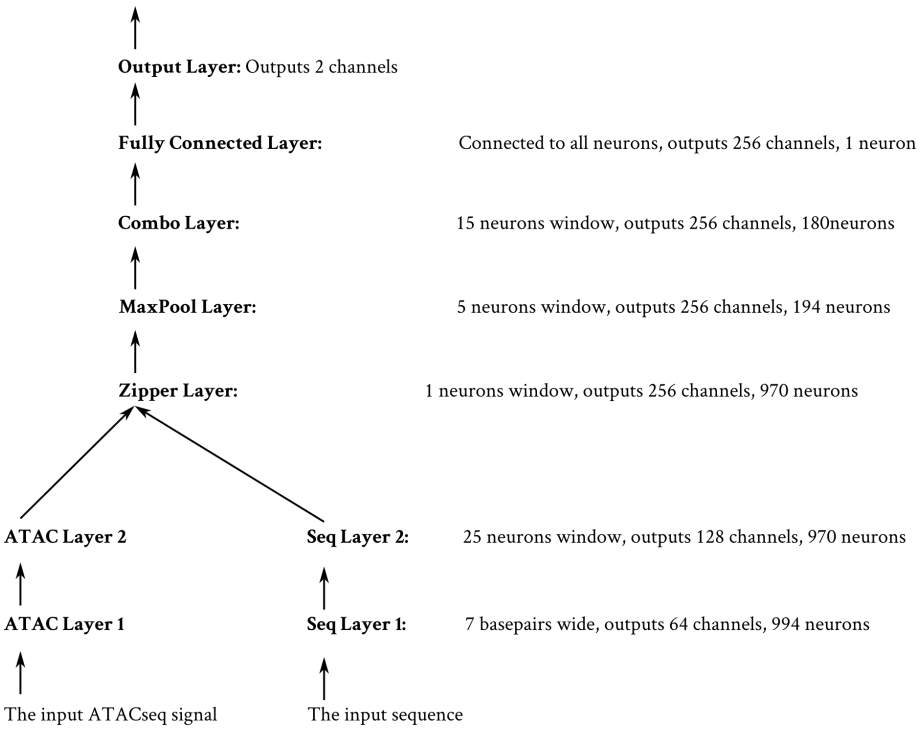


Figure 7.8: The model as it condenses into a single pair of values in the layers after the maxpool step.

Our original input was taken from a 1000bp window around a single PWM hit on the genome. Sequence and ATAC-seq signal are extracted and given to the network as input. These arrays of data are fed to the first two layers of the network and two new arrays are generated, the outputs of Seq layer 1 and ATAC layer 1. The information continues passing from one layer to the next until the final layer's output where the network gives us a score for "this is a true binding site" and a score for "this is a false site".

The internal matrices of the network are initialized with random values. At that stage, the output values of the network are equally random. Our goal is for the network's output values to be as good of a predictor as possible of the "real" label of any PWM site that is tested. To do that, we want to change the network's internal matrices, towards values that make the final output values a good predictor.

The matrices are changed during a training phase. During that phase, for each PWM, we feed the input genomic sequence and ATAC-seq signal to the network, and also provide a "real" label, as determined by overlap with an appropriate ChIP-seq peak. The network uses these labels as guidance. It compares its output values to the real labels and changes its internal matrices accordingly.

8

Training results

8.1 EARLY STOPPING

As we said before, the point at which the training of a NN stops needs to be somehow decided by the user. We monitored the progress of our models by computing many 'accuracy metrics' but relied on the AUC metric to inform our decision about stopping. At fixed steps during training, we would compute the AUC score that our model would get on the validation subset. We kept track of these values and if after 20 validation tests the AUC score hadn't improved, we determined the training to be finished. Here (Fig. 8.1) we plot an example progress of a model's AUC throughout its training, as well as the AUC score of the finished model on the test set.

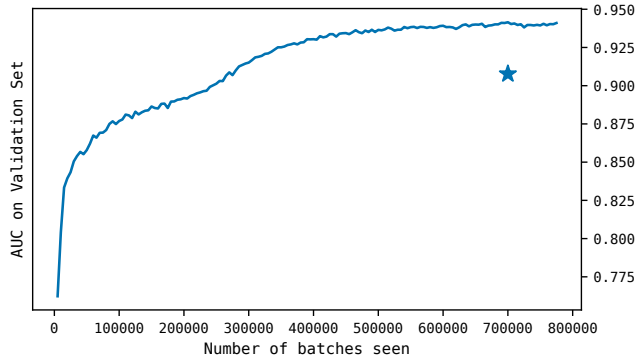


Figure 8.1: After a fixed number of batched have been processed by the network during training, we compute the model's AUC score on the validation subset of our data. Plotting these scores against the number of batches that have been sees, as a proxy to time passed, creates the line seen in this plot. The star marker indicates the point in time when the model's AUC score stopped improving in the X axis, and the AUC score of the model on the test subset in the Y axis.

8.2 BATCH SIZE

In our attempt to optimize the hyper-parameters of the model, we observed that smaller batch size leads to better model training (see Fig.8.2). This is something previously observed in relevant literature [140], with an explanation offered that larger batches allow the model to get stuck in local minima. There is nevertheless some lower limit in the batch size, at least for our model. In some of our efforts, when the batch size was set to a value too low, the model would fail to train, meaning it never manages to escape the randomness of the initial state.

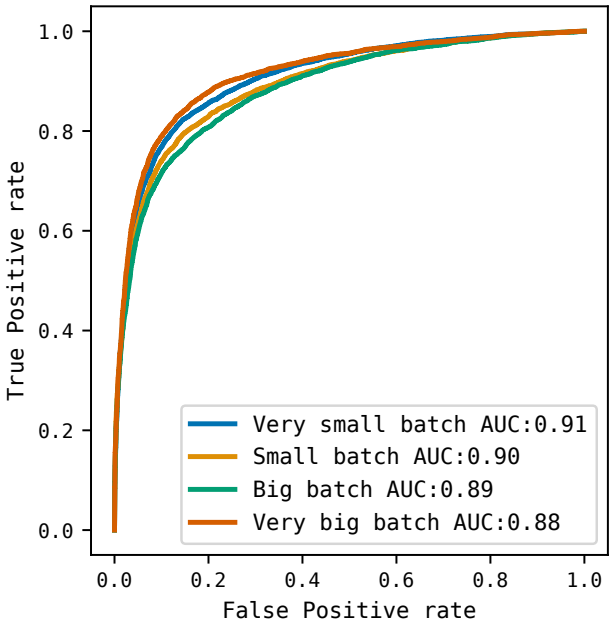


Figure 8.2: The batch size, the number of elements that are given to a NN in each step, is known to be an important factor of training for NNs. Here, we show the ROC curves of our model trained with four different batch sizes, ranging from very small to very big. The small batch sizes provided better training for our model, as expected.

8.3 LEARNING RATE

A second parameter that is known to affect NN training is the learning rate. We implemented a decaying learning rate, meaning that it will start at some initial value and every X batches its value will decay to 0.9*X. As is shown in Fig. 8.3, except in extreme cases where the NN trains too slow, the learning rate didn't influence the final capabilities of the model as much as the batch size did, nevertheless the is some room for optimizing.

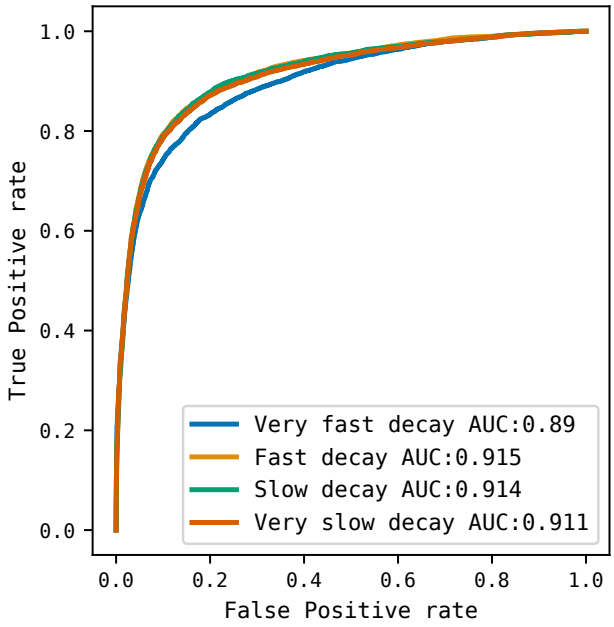


Figure 8.3: The decay rate of the learning rate allows the model to make the first training steps faster and the later ones slower and more carefully. Here we show the ROC curves of our model trained with four different decay rates, ranging from very fast to very slow. The final ability of the model doesn't change too drastically, but a carefully chosen learning rate will help the model achieve slightly better AUC scores.

9

Performance and comparison with other tools

We compared Nimrod to other similar tools that have been designed for the purpose of identifying true binding sites.

TFimpute[46] is another NN that operates only with the genomic sequence and additionally is designed to take into account the tissue of the given input.

Wellington[49] is an algorithm that detects footprints on the ATAC-seq signal, indentations cause by DNA-bound proteins.

PIQ[53] is a computational method that models the magnitude and shape of genome-wide ATAC-seq profiles to facilitate the identification of transcription factor (TF) binding sites.

Our model outperformed all of the other tools in all contexts (Fig. 9.1).

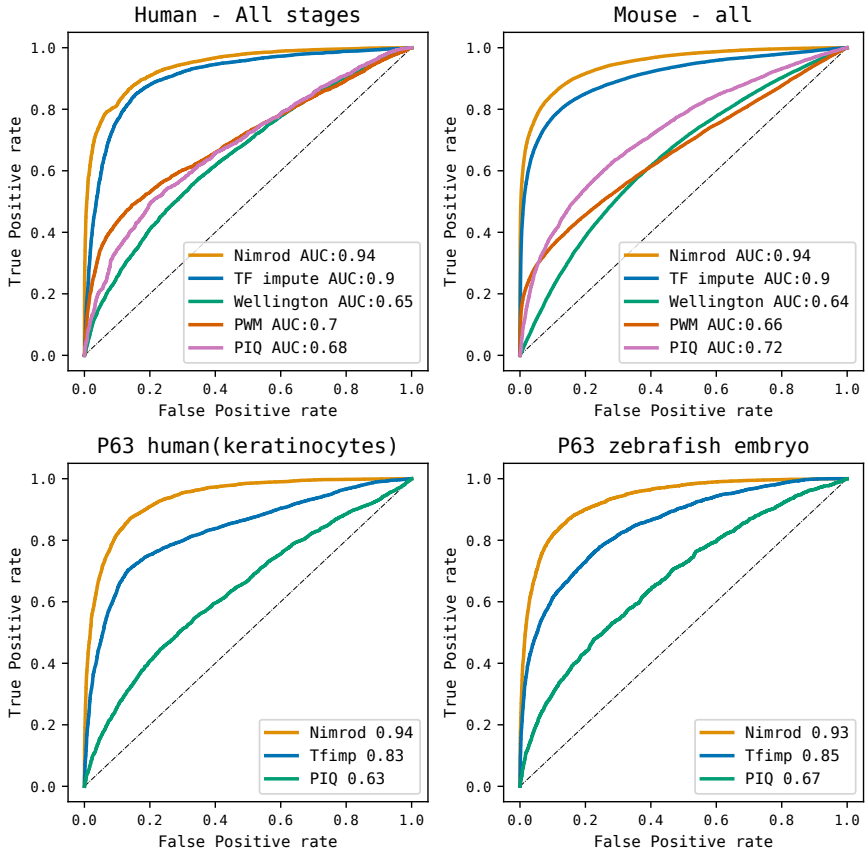


Figure 9.1: The ROC curves of different classification methods/tools on the biggest possible dataset of each protein-organism pair. Upper left: Nimrod and other tools on the human CTCF dataset. Upper right: Nimrod and other tools on the mouse CTCF dataset. Lower left: Nimrod and the two best other models in the p63 keratinocytes dataset. Lower right: Nimrod and the two best other models in the p63 zebrafish dataset.

9.1 CROSS SPECIES

One of our main objectives was to investigate how well our model would generalize in a cross-species context. Given that the TF proteins themselves don't change too radically between organisms, one would reasonably expect that the same features that are learned from one context based on sequence and ATAC-seq should be transferable to a different species.

The model lost, or maintained if you see the glass half-full, a significant amount of its classification ability when used cross-species (Fig. 9.2). In three of the four cases, when the model was tested on the other species than the one it was trained on, it performed worse than the cis-trained model but in many cases still competently when compared to the other tools.

In the case where our model was trained on the mouse ctcf dataset and the tested on the human dataset, the model showed exceptional generalization by achieving the same score on the test set as the cis-trained model. We believe this happened because the mouse ctcf dataset is by far the richest of the four. This must have allowed the model to train much better than the other poorer datasets.

The concept of testing a model such as ours in a cross-species manner, implies the existence of a training step. The model needs to learn in one context and be tested in another. Consequently, most tools cannot be tested in such a way. Of the models we previously compared to Nimrod, TFimpute is the only one that can be tested in this way.

TFimpute retains a good amount of its capabilities when tested cross-species, and very narrowly outperforms Nimrod in one of the four cases, when trained on human and tested on mouse (Fig.9.3). When the training dataset is rich enough for Nimrod though, such as in the case of the mouse dataset, Nimrod achieves much better results, hinting at a higher potential ceiling.

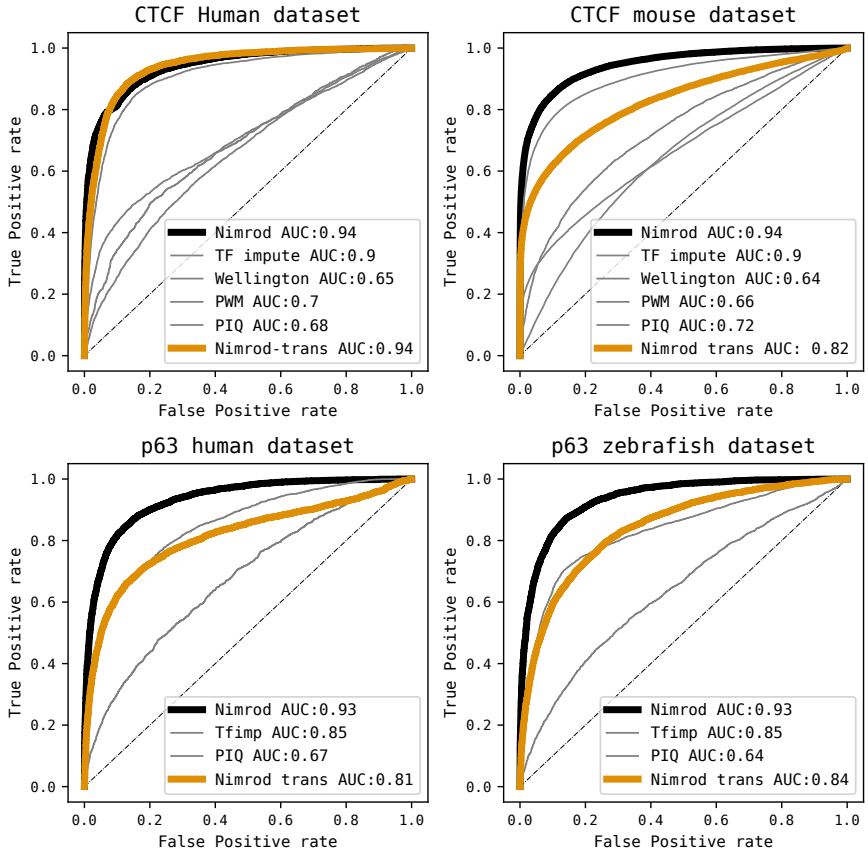


Figure 9.2: Here we plot, with a thick orange line, the performance of Nimrod when tested on an organism other than the one used for training (trans-test). Drawn with a thick black line, is our model evaluated on the same species as the one that it was trained on (cis-test). The cis Nimrod test as well as the other tools which are drawn in thin gray lines are the same ROC curves as in Fig. 9.1 Top left: In orange, our model that was trained with mouse data, is tested on the human dataset. In thick black, the model that was trained with human data. Top right: In orange, the nimrod model that was trained with human data, tested in the mouse test set. Lower left: In orange, the model that was trained with zebrafish data is tested on the keratinocyte data. Lower right: The model that was trained with keratinocyte data is tested on the zebrafish test set.

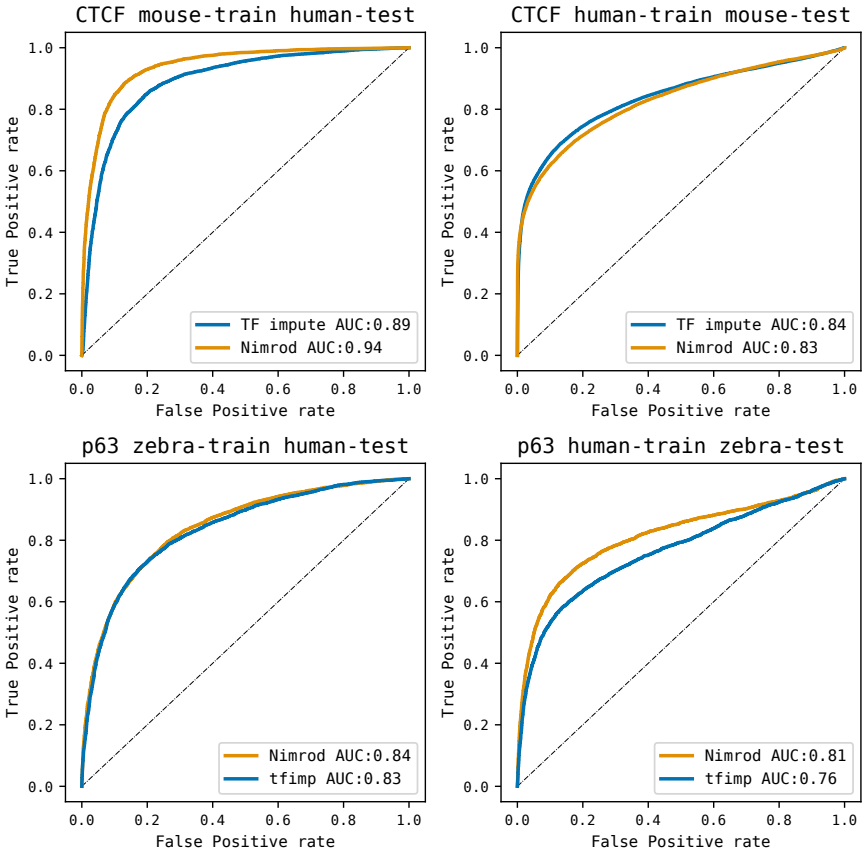


Figure 9.3: Here, we compare the two models when tested in a cross species way. Top left: The two models, trained with mouse data and tested on the human test set. Top right: Trained with human data and tested on the mouse test set. Lower left: Trained with zebrafish data, tested on keratinocyte test set. Lower right: Trained on keratinocyte, tested on zebrafish

Part IV

Discussion

10

Evolution of Cis regulation

Vertebrates are our intimate neighbors on the tree of life. Exploring the other chordates, the organism groups lying just outside the vertebrate subtree, is sure to provide valuable insights.

We chose amphioxus, the most basally divergent chordate, for a number of good reasons that were discussed in chapter 2.4.2. By comparing epigenomic data obtained during the development of amphioxus to similar data obtain from zebrafish, we intended to investigate details of the evolution of cis-regulation in chordates.

10.1 CONSERVATION OF CRES

Investigating the evolution of cis-regulatory elements is a challenging task. Conservation in genetics is usually measured in terms of sequence similarities. While coding sequences do change over evolutionary time, they remain similar enough that gene homology families can be constructed and some genes have been observed to be universal in life. Here we worked with homologous gene families in vertebrates which gave us more or less 10-20 thousand genes in each species that we can trace in the tree of vertebrates.

The same approach on non-coding elements yields some conserved elements, but the numbers are significantly lower. A few thousand have been identified in vertebrates and separately in fruitflies [141–145] or nematodes [146], but only a few dozen have been shown to be conserved between human and amphioxus.

In human and mouse, orthologous CREs can be detected very well thanks to the large degree of conservation of sequence and synteny [147–153]. Which is great news since despite the high rates of repurposing and turnover[148, 149, 152, 154], cis-regulation research for therapeutic purposed can be readily investigated in our most important vertebrate model organism[1, 65].

Thee-way comparisons between human, fly and nematodes found little conservation of individual regulatory targets and binding patterns [155] and some conservation of fundamental features of transcription [156]. Fruitflies and nematodes have been shown to be highly derived [157–159] so maybe the lack of CRE conservation should not come as a surprise.

Again, the turnover rates of CREs are much higher than coding sequences, but they can also vary between tissues or cell types [152, 160], developmental stages or types of CREs [144, 148–150, 160].

Our working model of how regulatory elements work suggests that TFs bind on them on specific sites. Those designated binding sites, at least the way in which we understand them and typically model them with PWMs, are always a few base-pairs apart. Consequently the inter-TFbs genomic positions would not be expected to be under evolutionary pressure. On top of this, the requirements of TFs with regards to the sequence are decently plastic. These two dynamics could explain why cis-regulatory sequence might be more receptive to sequence alterations without a change of function.

Yet, striking exceptions do exist as a few mysterious cis-elements appear to be ultra-conserved [161, 162] (over 95% sequence similarity) in chordates without a satisfying explanation. Why would the inter-binding site bases be conserved? Why are they so deeply conserved? Unfortunately for now these questions remain a mystery.

10.1.1 FUNCTIONAL CONSERVATION

Amphioxus shares a lot of morphological homologies with vertebrates, axial notochord, a dorsal neural tube flanked by segmented trunk muscles, pharyngeal gill slits and a ventral heart [163]. If development and tissue formation is defined by gene activity, and two organisms display similar developmental and morphological patterns, it is reasonable to expect homologous genes to be orchestrating the homologous tissue formations and homologous cis-regulatory elements to be orchestrating the gene expression. This is something that we have seen before in more closely related lineages but we cannot be certain that it extends to the root of chordates.

We showed significant amounts of transcriptional conservation in chordates, both in time and space. Gene neighborhoods [1] from amphioxus tend to remain co-expressed in human (NACC analysis, Chapter 5.1). Functional modules of genes, grouped independently in amphioxus and zebrafish, tend to share more homologous genes when they are associated to the same function (Chapter 5.3.1). For example, the brain module of zebrafish is most enriched in orthologous genes when compared to the neural tube modules from amphioxus. Furthermore, amphioxus displays a phylotypic period (Chapter 5.2), a period in early development where the genes of vertebrates, and now as we learned chordates too, are transcribed at the most similar levels between all species.

So the conserved morphology can be seen in conserved gene transcription. Gene transcription is controlled by cis-regulation, so if homologous genes in two species are expressed in similar ways, the function of cis-regulation has been conserved as well. CREs and the putative TF binding sites that they include have often been modeled as a language. The binding sites are words and CREs are short phrases or sets of words that carry some function in cis-regulation. There is some evidence that the 'words' are needed in specific order, but also evidence to the contrary.

Instead of investigating individual elements we looked at the 'words' contained in the entire regulatory landscapes of genes. We showed that modules of genes with many shared homologous genes and similar transcription levels, often also display a similar preference or disinclination for the same TFs. For example, the

[1] Any given gene and its top most co-expressed other genes

cilium-associated modules from both species are enriched in binding sites of RFX TFs, known to regulate ciliogenesis [164].

During development too, we see the cis-regulatory content of gene lanscapes reflecting the conservation of gene transcription levels. We showed that importance of different TFbs in different developmental stages, is maximally similar between the two species at a point in the middle of early development, consistent with the phylotypic period. In fact, the cis-based phylotypic stage was detected right before the gene-based period, making it tempting to think that we caught cause and effect in action, but since we don't have ATAC-seq experiments for some of the intermediate stages, we can't tell for sure.

To recapitulate, we observe an overarching body plan conservation amongst chordates. Amphioxus and vertebrates adhere to the phylotypic developmental pattern and 'use' conserved modules of genes that are controlled by similar TFs. This is most clearly shown for gene modules participating in specialized developmental programs, brain, cilium, skin. It is hard to say if this means something for the programs that we do not detect to be clearly conserved, or if this is a limitation of our approach. Even more detailed transcriptomic datasets during development, perhaps single-cell RNA-seq and ATAC-seq would help shed more light into the rest of the developmental programs.

Being unable to show homology of CREs based on sequence conservation, the origin of the cis-similarities between the species remains open to interpretation. Similar cis activity drives similar gene expression to create similar morphologies. Did CREs get conserved and elaborated during evolution like genes but to a degree that we cannot tell them apart? did they get lost a rediscovered? Maybe we don't even need to consider each CRE as functional unit, and the more vague conservation of the underlying vocabulary in the vicinity of genes is enough.

10.2 COMPLEXITY

We might have assumed, at some point in history, that humans have the biggest and most complex genome in life. We are after all the center of our own universe, the chosen children, our genome would of course be special.

Unfortunately for our pride, it turns out that even the humble onion has a genome

many times bigger than ours. It is to be concluded then that organism complexity is not directly reflected on genome size.

As far as our model organisms are concerned, amphioxus has a smaller genome than the vertebrates (Chapter 4.1). The additional genomic space in vertebrates is not, of course found isolated in some corner of the genome, but rather is distributed between the genes. This is evident by the fact that vertebrate genes have larger genomic spaces around them and their putative CREs are found in greater distances from any TSS (Chapters 4.2 and 4.3). In the microcosm of our work, an increase in genome size comes with an increase in complexity, although no meaningful conclusions could be drawn from such a small scope.

But what is animal complexity?

Complex is defined as something composed of two or more parts, the more parts the higher the complexity. In terms of animals then, complexity could be measured in terms of distinct tissues or organs or cell lines. Vertebrates have evolved plenty of new elaborations on the chordate body plan, and all the necessary new tissues, cell lines and developmental programs that are needed. A head with predatory jaws, complex sensory organs and complex brain, a skeleton are all additions to the ancestral body plan.They are formed during development when the organism consists of a relatively small number of structures, which interact to give rise to more structures, which in turn give rise to more, in a widening cascade (read about developmental-depth in [165]). Our understanding is that novelties in gene regulation can lead to novel cell states, or cell lines. These new cell lines interact with the existing developmental context to produce new structures which lead to morphological novelties.

New cell populations like the ectodermal placodes or the neural crest have been implicated in the evolution of tissue innovations in vertebrates [166]. Additionally the neural crest was linked to the recruitment of ancestral regulatory genes, an observation that connects regulatory diversification to cell type diversification to tissue diversification.

10.2.1 COMPLEXITY AND WGD

The regulatory landscape in an organism that has just underwent a WGD event brims with evolutionary potential. Many gene copies are now redundant and can

be lost, but in the process of being lost, a gene might change enough to be used in a different function, therefore regaining its evolutionary purpose. Genes that are kept in duplicate can themselves evolve and differentiate, presumably at different rates.

We have noticed that different categories of genes are retained at different rates. The general understanding is that genes that are involved in functional modules such as metabolic pathways, metabolic cascades, regulatory networks, etc. tend to be preferentially retained. It is suggested that a non stoichiometric production of the components of these modules will quickly lead to problems for the organism.

Therefore, after a duplication sister genes will be under pressure to maintain a constant level of transcription, to satisfy the needs of the gene module in which they are involved[167]. TFs are among the categories of genes that are known to be preferentially retained, as they are often parts of gene regulatory networks which are such functional modules. Moreover many TFs are pleiotropic, being involved in multiple regulatory networks, increasing the importance of conserving their transcription levels.

Let us consider how a WGD effects a TF and a membrane protein as an counter-example. A TF's "concentration" on the genome will remain the same after a duplication. In other words, there is now double the amount of binding sites where the TF is expected and double the amount of TF protein. The membrane protein's concentration on the other hand doubles since we have double the amount of this protein for the more-or-less same area of membrane. The expected pressure then would be for the TF system to remain as it is, maintaining the TF copies, and for the additional membrane protein copy to be lost, so that things return to status quo.

This implicates the second actor of transcription regulation; cis-regulatory elements. If the transcription of a CRE's target is under pressure to be kept stable, the pressure would extend to the conservation of the CRE itself. Are CREs preferentially retained? The evident high-turnover of CREs, even for relatively close species points to the contrary, but our inability to match CREs across species doesn't mean that the ancestral element wasn't retained after the WGDs. Besides, overprinting of existing CREs is a rampant phenomenon [54].

We showed that vertebrates have more putative regulatory elements in total and per gene (chapter4.7) and those are further away from the gene's transcription start sites (chapter 4.6). A larger number of regulatory elements is to be expected

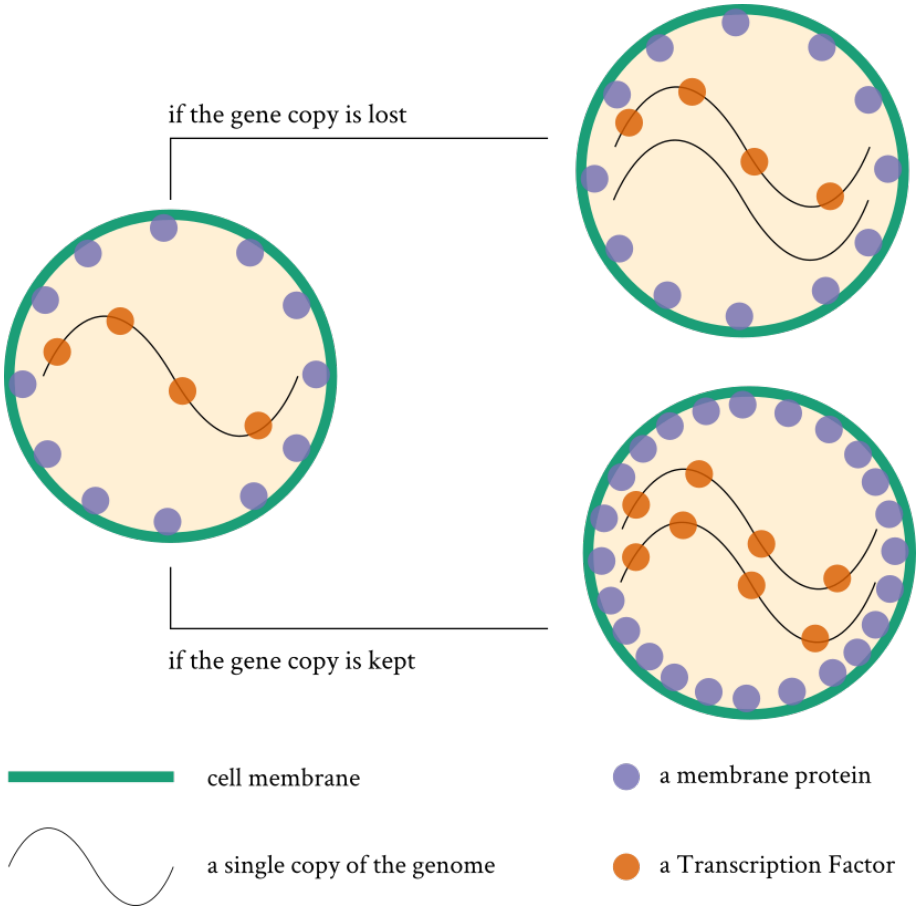


Figure 10.1: The evolutionary pressure on genes copies after a WGD effect might depend on gene function. After a WGD, if the copy of a TF gene is lost, the dosage of the TF protein in the system will be halved since the binding sites where the TF is expected have doubled while the amount of the TF protein won't. This has a multi-pronged effect on cis-regulation since multiple target genes will be affected. Consequently there is strong pressure for the TF to be kept in duplicate. In contrast, other genes like a cell membrane protein, will be pressed to lose the redundant copy since that will return the protein to its expected levels.

in the larger genomic landscapes of vertebrates, but we demonstrated that even

if we group genes in buckets of similarly sized landscapes, zebrafish genes contain more CREs in their landscapes than amphioxus genes (chapter 4.7), indicating that cis regulatory complexity increase in vertebrates can be observed both in terms of number of elements per gene, but also in terms of density of elements in similarly sized regions.

Crucially to our previous arguments, 'trans-dev' labeled genes (TD), genes that are implicated in transcription and development (many of them TFs), tend to have more CREs in their landscapes that housekeeping genes. Similarly so for genes that are retained in multiple copies (many of them TD) versus genes that are kept in a single copy. This fits with the idea that CREs of genes that are preferentially retained in several copies would themselves be preferentially retained. Alternatively, maybe CRE-rich genes are like that because for some reason they tend to create new CREs more often. Interestingly, the homologous genes in amphioxus, both of the TD category and generally of genes that are kept in multiple copies, are also richer in CREs than other amphioxus genes, again fitting with the model of preferential retention and diversification for CREs, specially so for the CREs of TD genes.

This preferential retention of developmentally crucial elements, both genes and CREs, pushes the ceiling of complexity for organisms. Before the WGD, at the node of each regulatory network resided the single copy of a gene. This single copy needs to be transcribed at certain levels and has some evolutionary 'wiggle-room' but there is no safety net. After a WGD, two copies will preferentially be kept, alongside their cis-landscapes and they can now both evolve while relying on their copy to buffer any differences from the ancestral expression pattern.

In the following chapter we discuss the various patterns in which this diversification of gene copies can happen.

10.3 FATE

As we discussed, different functional categories of genes are expected to exhibit different rates of retention after a WGD, an expectation that is supported by a number of biological observations. In Xenopus *tropicalis*, an allotetraploid organism, TFs and components of cell cycle and developmental pathways were preferentially retained while other functional categories like DNA-repair where more

likely to lose one of the copies[168]. This is generally a well established observation in vertebrates[169] and also noticed in the florida amphioxus, a different species than the one we worked with [119]. It should be noted that duplicate genes retained like that can still be lost even after millions of years[169]. We report similar observations when extending the analysis to amphioxus. Genes that are kept in multiple copies are enriched in trans-dev genes (Chapter 6.0.1).

Once a pair of genes is stabilized in an organism, it would be interesting to know how the two copies are transcribed. The DDC model predicts that the two genes will reciprocally lose regulatory units and by extension expression domains and as a result they will share the ancestral expression between them.

Such a dynamic was convincingly shown when considering amphioxus as a proxy to the ancestral state. When multiple genes are retained after WGD, many gene copies restrict their expression domains, but the joint expression of all members recapitulates that of the single amphioxus ortholog, and likely of the ancestral gene (Chapter 6.0.3).

This observation could also be explained by the pattern of 'specialization' where one of the two ohnologues retains the ancestral expression pattern, while the other only retains expression in a small fraction of the ancestral domains. In our comparisons of amphioxus to all three vertebrates, the specialization cases where three times as common as the subfunctionalization cases (Chapter 6.0.3), showing that the proposed reciprocal loss of function for ohnologues is rather hard to accomplish. This recapitulates previous results where asymmetric gene-expression patterns between duplicated genes were found to be the most common pattern[170].

This could fit as a nuanced version of the 'classical model'. Maybe instead of one of the two genes being lost thanks to deleterious mutations, it is lost through loss of transcription activation. The genes that were strongly specialized, meaning they only retained expression in two or less of the ancestral domains, were much more likely to retain expression in the brain than other tissues, an indication of diversification and recruitment perhaps.

With regards to retention of CREs, we observed that genes with more cis-regulatory elements in amphioxus where more likely to be retained in multiple copies in vertebrates 6.0.1, in accordance with the prediction of the the DDC model that regulatory complexity would increase the probability or retention.

Most discussions on the topic of gene fate after a WGD, treat the CRE-gene dynamic as a simple system where one CRE grants one domain of expression to the gene. A loss of a CRE equates a loss of expression in some context. Interestingly, the exact opposite seems to be the case according to our data. Subfunctionalized and specialized genes were found to have significantly more CREs in their landscapes than genes who kept the full ancestral pattern(Chapter 6.0.3). Furthermore, strongly specialized genes had more elements than mildly specialized, and the more expression domains a gene loses, the more CREs it is likely to have.

Coming back to McSheas's argument that we touched upon in the previous chapter, "development [starts with a] relatively small number of structures, which interact to give rise to more structures, which in turn give rise to more, in a widening cascade"[165]. He calls this Developmental Depth and is the developmental manifestation of what he calls Structural Depth, an increase in hierarchical complexity through the emergence of a higher level of nesting of parts within wholes.

Considering the hierarchical nature of the newly elaborated structures after a WGD, we would expect that any gene that is actively transcribed in the paternal structure will be by default also actively transcribed in the new contexts, consequently for further differentiation to happen, more regulatory control needs to be applied to the gene in order to override the default condition of the parental state, thus, an increase of cis-regulation to orchestrate subfunctionalization and specialization should be expected.

11

On artificial Neural Networks and TF binding sites

Artificial Neural Networks show great promise as a framework to model the dynamics of DNA-protein binding. Their hierarchical design allows them to model large numbers of high-order features and as a consequence they concurrently model multiple genomic features.

In image analysis, this is easier to visualize. In [171], Lee et al. visualize the two dimensional features that their model learned by analyzing images. The simple features learned by the first layer (Fig. 11.1 top) are used to created more complex features in the second layer (Fig. 11.1 bottom) and then even more complex features in the deeper layers of their model (Fig. 11.2).

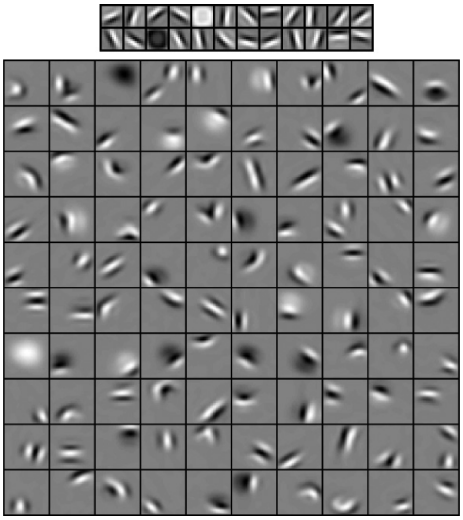


Figure 11.1: The features learned by the first two layers of an NN trained on natural images. The first layer, on the top, learns the most basic of features, while the second learns combinations of the features learned by the first layer.

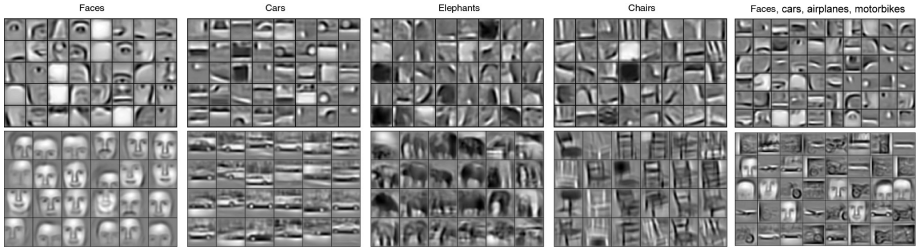


Figure 11.2: Deeper layers of the NN from Fig. 11.1. At these deeper layers, the combinations of features from lower layers start being recognizable features, like eyes, noses, wheels, and eventually faces, cars, animals.

To attempt a parallel in our genomic case, let us consider nucleosomes. They are a big factor of TF occupancy and hard to predict accurately based on sequence only. Nucleosomes though, leave a shadow on the ATAC-seq signal, that is a region of about 140 basepairs that is significantly depleted in sequencing reads

inside another otherwise accessible genomic region. This is something that has been modeled before in order to track nucleosomes on the genome [77] but not something that is modeled in any TFbs tool. One could in theory combine two tools, look for putative TFbs based on some TF-tool and then cross-reference the results of that with the nucleosome positioning model. But then an additional model should be created to connect the influence of nucleosomes to TFbs, after all different TFs have different preferences to nucleosomes.

Our NN should in principle be able to capture and integrate such a complexity seamlessly. The initial layers on the ATAC-seq signal would learn simple patterns like ”accessible area” (many ATAC-seq reads) or ”non-accessible area” (no reads). The inner layers would then learn combinations of those, something like 'two accessible areas separated by a 140bp long non-accessible area' could signal the presence of a nucleosome which the model would then associate positively or negatively with the training TF.

These hypotheses are not trivial to investigate. Attempting to make sense of the inner layers of NNs is often a publication-level amount of work in itself [172, 173] so any assumptions on how exactly our network recognizes binding sites will have to remain unexplored. What is clear, is that the model does achieves great levels of accuracy, especially so when a rich training dataset is available.

High throughput sequencing assays are exploding in popularity and decreasing in cost so we feel that as more and more genomic data becomes available, tools like Nimrod will become more and more important for modern genetic and epigenetic analyses. An ATAC-seq assay, already offers us insight about the active regulatory elements in any biological context. With the help of a well trained model like Nimrod, we would be able to produce highly accurate detections of TFbs even in new systems based on a single, and relatively cheap, biological assay. Furthermore, this approach sidesteps the problem of antibody availability which can be a limiting factor, especially so in new model organisms whose study is necessary for high-quality evolutionary comparisons.

12

Conclusions

1. The tissues, organs and developmental stages that characterize the chordate body plan are associated to conserved modules of co-expressed genes that are also conserved at the cis-regulatory level.

2. This conservation is reflected on the cis-regulatory content of the conserved genes, highlighting the importance of cis-regulation for function and development.

3. Vertebrates have increased the complexity of their gene regulatory landscapes. This increase is particularly intense in genes involved in transcriptional and developmental regulation and genes that have been retained in multiple copies after the vertebrate WGDs, especially those ohnologues that have subfunctionalized or specialized their expression.

4. Contrary to previous expectations, specialization is a much more common fate for WGD gene duplicates than subfunctionalization. Counterintuitively, the ohnologues that restricted their expression increased their cis-regulatory complexity.

5. Nimrod, our newly implemented NN based on both genomic sequence and chromatin accessibility data, predicts TFBSs with high accuracy, outperforming previously existing tools.

6. When trained on rich, high quality data of a given species, Nimrod can be generalized to other species without losing accuracy.

Part V

Methods

13

Notebooks

13.1 PWMS USED

In this work we use a complex set of PWMs which was created for our needs of comparing amphioxus to vertebrates. Very briefly, potential PWMs were selected for amphioxus genes based on genomic similarity to genes for which the binding profile is known. These PWMs were joined to a very thorough dataset of potential PWMs from the CIS-BP database and they were clustered, as a whole, to produce motif clusters. Those are motifs that encapsulate the preferences of a cluster of similar motifs.

For our work, we further clustered these 462 clusters clusters into super-families of regulatory genes. For example, if a number of PWMs were only associated to a single TF, say P63, we would consider all the instances of all these PWM as an instance of P63. This reduced the number of PWMs and consequently the complexity of the system. This approach yielded a list of 242 clusters of motifs assigned to one or more orthologous groups in both amphioxus and zebrafish (Supplementary Dataset 16), which were used for further analyses.

Following is an more in depth explanation of the PWM-cluster creation, in the words of our collaborators that did the analysis. Taken from our work[123]

13.2 TF ANNOTATION AND TF BINDING SPECIFICITY PREDICTION

We identified putative TFs by scanning the amino acid sequences of predicted protein-coding genes for putative DNA binding domains (DBDs) and, when possible, we predicted the DNA binding specificity of each TF using the procedures described in [174]. Briefly, we scanned all protein sequences for putative DBDs using the 81 Pfam[175] models listed in [176] and the HMMER [177] tool, with the recommended detection thresholds of per-sequence E-value < 0.01 and per-domain conditional E-value < 0.01. Each protein was classified into a family based on its DBD and its order in the protein sequence (e.g., bZIPx1, AP2x2, Homeodomain+Pou). We then aligned the resulting DBD sequences within each family using clustalOmega [178], with default settings.

For protein pairs with multiple DBDs, each DBD was aligned separately. From these alignments, we calculated the sequence identity of all DBD sequence pairs (i.e. the percentage of amino acid residues that are exactly the same across all positions in the alignment). Using previously established sequence identify thresholds for each family [174] , we mapped the predicted DNA binding specificities by simple transfer.

For example, the DBD of BL07183_evm0 (UNCX) is 88% identical to the zebrafish BX908797.1 protein. Since the DNA binding specificity of BX908797.1 has already been experimentally determined, and the cut-off for the Homeodomain family of TFs is 70%, we can infer that BL07183_evm0 will have the same binding specificity as BX908797.1. This approach produced a table with putative TFs and associated binding motifs (when possible) for amphioxus (Supplementary Dataset 16 [1].

Next, we created a set of related motifs using the following procedure. We first downloaded all motif and TF information from CIS-BP version 1.02 [174] . For all motifs in Supplementary Dataset 16, we selected similar motifs from CIS-BP based on the Jaccard Index (intersection divided by union) of associated TFs; Jaccard Index ≥ 0.95 was used as the threshold. In addition, based on NCBI taxonomy,

[1] Note, this can be accessed from our published work for [123]

we selected all motifs from the CIS-BP database that were associated with vertebrate proteins. From this set of motifs, we selected all motifs where the sum of the Information Content (IC) of all individual positions was at least 5. We clustered the motifs using GimmeMotifs [179] with a threshold of 0.9999 ($p \leq 0.0001$). All clustered motifs with a total IC ≤ 5 were discarded. The motif clusters were then associated with TFs based on the annotation of the individual motifs in CIS-BP and (Supplementary Dataset 16).

13.3 TF MOTIF MAPPING ONTO ATAC-SEQ PEAKS

To identify TFs that potentially bind to ATAC-seq peaks, we first used gimme threshold, from GimmeMotifs [179] , to determine the detection threshold (1% false positive rate; FPR) for each TF motif, based on randomly selected genomic sequences with a similar GC content distribution as the ATAC-seq peaks. With these thresholds, we scanned all ATAC-seq peaks for our consensus motif clusters using gimme scan from GimmeMotifs version 0.11.0 [179].

Brief Genome Exploration

A simple analysis of reference genome sizes. To start we load the genomes, chromosome info files and masks, which are set in preamble.py

To create the data that we'll plot, we need the raw genomic files which are too big to share here. The following is the code that was used to calculate the values of the plot.

```
>>> # # chrom_info files, two columns with the  chromosome names / sizes
... # genome_info_fps = {
... # "zebrafish" : "/Thesis_shallow/data/genomes/chromInfo_danRer10.txt",
... # "amphioxus" : "/Thesis_shallow/data/genomes/chromInfo_braLan71.txt",
... # "medaka" : "/Thesis_shallow/data/genomes/chromInfo_oryLat2.txt",
... # "mouse" : "/Thesis_shallow/data/genomes/chromInfo_mm10.txt"}
... # # repeat masks, set in the pream
... # repeat_fps = {
... # "zebrafish" : zebrafish_mask,
... # "amphioxus" : amphi_mask,
... # "medaka" : medaka_mask,
... # "mouse" : mouse_mask}
... # fasta_fps = {
... # "zebrafish" : "/scratch/genomes/danRer10.fa",
... # "amphioxus" : "/scratch/genomes/Bl71nemr.fasta",
... # "medaka" : "/scratch/genomes/oryLat2.fa",
... # "mouse" : "/scratch/genomes/mm10.fa"}
... # compl = lambda spec: repeat_fps[spec].complement(g=genome_info_fps[spec])
... # def countNs(fp):
... #     c = Counter()
... #     with open(fp,"r") as fi:
... #         for line in fi:
... #             if line.startswith(">"):
... #                 continue
... #             else:
... #                 c+=Counter(line.rstrip().lower())
... #     return c
...
...
... # def get_counts(spec):
... #     bd = compl(spec)
... #     bd = bd.sequence(fi=fasta_fps[spec])
... #     thed = countNs(bd.seqfn)
... #     Ncount = thed.get('n')
... #     effectiveCount = thed.get('a') + thed.get('c') + thed.get('g')+thed.get('t')
... #     repDF = repeat_fps[spec].to_dataframe()
... #     repeatCount = (repDF['end']-repDF['start']).values.sum()
... #     return repeatCount, Ncount, effectiveCount
... # speciesorder = ['amphioxus','zebrafish','medaka','mouse']
... # # The main loop. It goes through all the fasta files so it takes a while
... # lot =[ ]
... # for spec in speciesorder:
... #     lot.append( (spec, *get_counts(spec)) )
... # df = pd.DataFrame(lot)
... # df.columns = ["species","Repeats",'"N"s',"Effective Genome"]
```

Here we will just plot the final values which we hardcode in the following cell

```
>>>
... df = pd.DataFrame([["amphioxus",152452412, 20252512,327521540],
... ["zebrafish",756790655,1964425,654384100],
... ["medaka",23221380,168573794,677265199],
... ["mouse",1201953154,78082998,1456094557]])
... df.columns = ["species","Repeats","N's","Effective Genome"]
... df
```

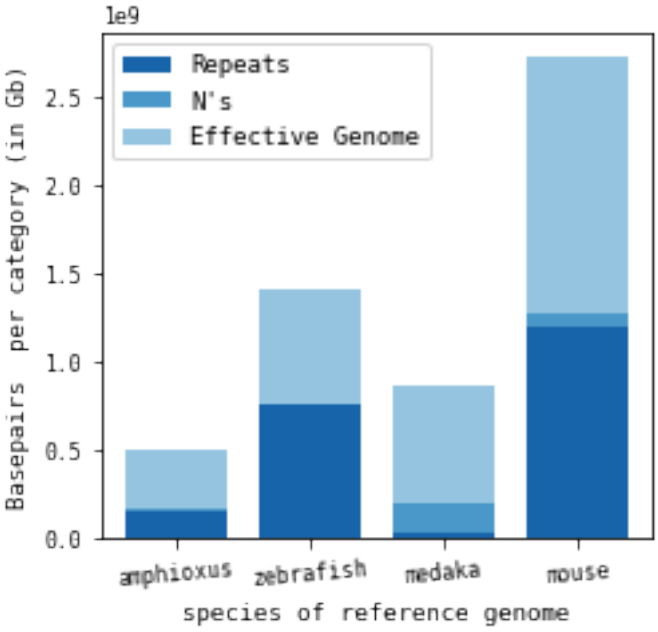
```
     species     Repeats        N's  Effective Genome
0  amphioxus   152452412   20252512         327521540
1  zebrafish   756790655    1964425         654384100
2     medaka    23221380  168573794         677265199
3      mouse  1201953154   78082998        1456094557
```

```
>>> df
```

```
     species     Repeats        N's  Effective Genome
0  amphioxus   152452412   20252512         327521540
1  zebrafish   756790655    1964425         654384100
2     medaka    23221380  168573794         677265199
3      mouse  1201953154   78082998        1456094557
```

```
>>> fig, ax = plt.subplots()
... fig.subplots_adjust(left=.15, bottom=.18, right=.99, top=.95)
... Fwidth = THESIS_PAGEWIDTH/1.5
... Fheight = Fwidth
... # The stacked barplot (it's a bit convoluted...)
... categories = df.columns[1:]
... lo_cases = df.species.values
... N = len(lo_cases) #how many bars in total are we plotting
... palette = sns.color_palette("Blues_r",N)
... ind = np.arange(N)    # the x locations for the groups
... width=0.75 #how wide each column will be (if 1 they touch eachother)
...
... c = np.zeros(N)
... cou = 0
... lot=[]
... for cat,vals in df.iloc[:,1:].iteritems():
...     p = ax.bar(ind, vals.values, width, bottom=c , color=palette[cou], alpha=1)
...     c+=vals.values
...     cou+=1
...     lot.append(cat)
...
... plt.xticks(ind, lo_cases,rotation=5)
... plt.legend(lot, loc='upper left')
...
... #>>> Name your Axes
... ax.set_ylabel('Basepairs  per category (in Gb)')
... ax.set_xlabel('species of reference genome')
```

```
... fig.set_size_inches (Fwidth, Fheight)
... #>>> OUTPUT NAME
... fig.savefig('../Figures/from_notebooks/tfigure_TheGenomes.pdf')
```



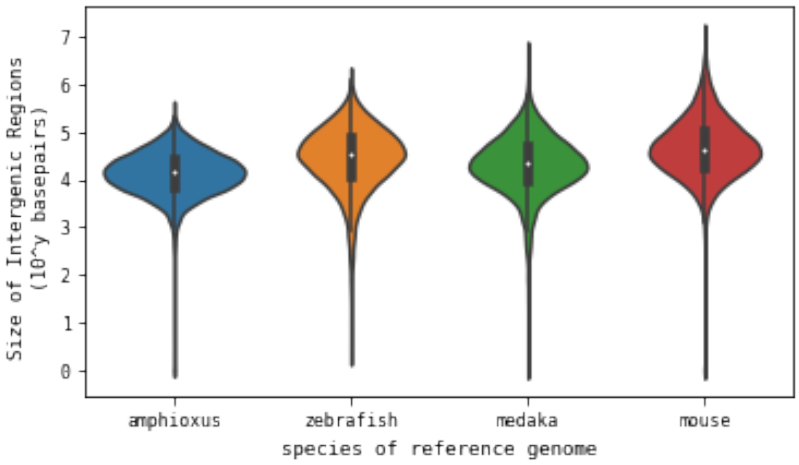
Intergenic and GREAT distance distributions

In this notebook, we plot the distributions of intergenic distances,
the distances between consecutive syntenic transcription start sites, for our 4 reference genomes.

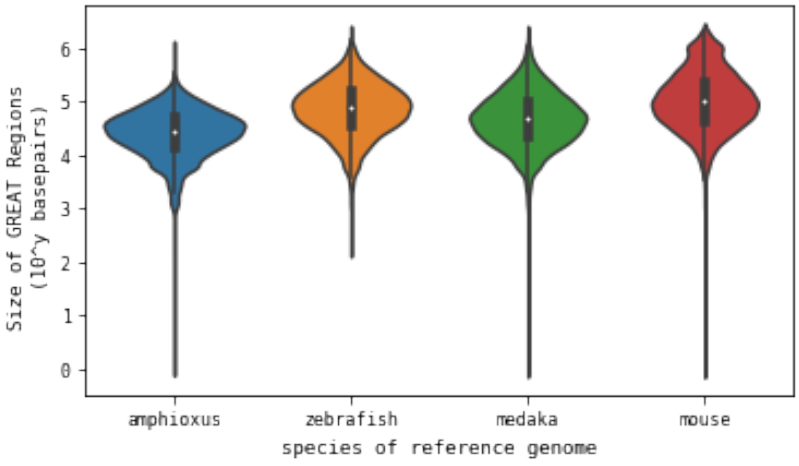
We similarly plot the distributions of GREAT distances, genomice regions assigned to each gene. For more details look into the "make_great_files" notebook

The intergenic and GREAT regions are computed in notebooks that are made available in the thesis appendix and repository.

```
>>> # We load the precomputed distances
... speciesorder = ['amphioxus','zebrafish','medaka','mouse']
... intergenic_regions_D = {
... "zebrafish" : "/Thesis_shallow/data/genomic_regions/intergenics_Dre.tsv.gz",
... "amphioxus" : "/Thesis_shallow/data/genomic_regions/intergenics_Bla.tsv.gz",
... "medaka" : "/Thesis_shallow/data/genomic_regions/intergenics_Ola.tsv.gz",
... "mouse" : "/Thesis_shallow/data/genomic_regions/intergenics_Mmu.tsv.gz"}
... great_regions_D = {
... "zebrafish" : "/Thesis_shallow/data/genomic_regions/GREAT_dre.bed.gz",
... "amphioxus" : "/Thesis_shallow/data/genomic_regions/GREAT_bla.bed.gz",
... "medaka" : "/Thesis_shallow/data/genomic_regions/GREAT_ola.bed.gz",
... "mouse" : "/Thesis_shallow/data/genomic_regions/GREAT_mmu.bed.gz"}


>>> def loadthing(k,v):
...     df = pd.read_csv(v, sep='\t',header=None)
...     df['w'] = df[2] - df[1]
...     df['species'] = k
...     return df[['species','w']].copy()


>>> big = pd.concat([loadthing(k,v) for k,v in intergenic_regions_D.items()])
... big['w'] = big['w'].apply(lambda x: log(x,10))
... gbig = pd.concat([loadthing(k,v) for k,v in great_regions_D.items()])
... gbig['w'] = gbig['w'].apply(lambda x: log(x,10))


>>> fig, ax = plt.subplots()
... fig.subplots_adjust(left=.11, bottom=.13, right=.99, top=.99)
... Fwidth = THESIS_PAGEWIDTH
... Fheight = Fwidth*(9/16.)
...
... sns.violinplot(data=big, x='species', order=speciesorder, y='w', ax=ax)
...
... ax.set_ylabel('Size of Intergenic Regions \n (10^y basepairs)')
... ax.set_xlabel('species of reference genome')
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_IntergenicSizes.pdf')
```




```
>>> fig, ax = plt.subplots()
... fig.subplots_adjust(left=.11, bottom=.13, right=.99, top=.99)
...
... sns.violinplot(data=gbig, x='species', order=speciesorder, y='w', ax=ax)
...
... ax.set_ylabel('Size of GREAT Regions \n (10^y basepairs)')
... ax.set_xlabel('species of reference genome')
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig("../Figures/from_notebooks/tfigure_GreatSizes.pdf")
```



Creating TSS files

This notebook takes genome anotation files "gtf, gff", extracts Transctiption start sites,and outputs them as a bed file with the following columns:

`chromosome start end geneid 0 strand`

The process is finicky thanks to the differences between the various anotation files,so details have to be smoothed out in a case by case way.

Zebrafish

```
>>> known_chroms = pd.read_csv("chromInfo_danRer10.txt",
...
...                             sep='\t', header=None)[0].values
... with gzip.open('danRer10.gtf.gz','r') as fi:
...     lot = []
...     for line in fi:
...         if line.startswith("#"):
...             continue
...
...         chrom,db,taip,start,end,dot,strand,frame,info = (
...             line.rstrip().split('\t'))
...         d = dict([pair.split('"')[:2] for pair in \
...                 info.replace(' ','').split(";")[:-1]])
...         d['chrom'] = chrom
...         d['db'] = db
...         d['taip'] = taip
...         d['start'] = start
...         d['end'] = end
...         d['strand'] = strand
...         d['frame'] = frame
...         lot.append(d)


>>> df = pd.DataFrame(lot)
...
... # Fix the chromosome names
... df['chrom'] = (df['chrom']
...                .apply(lambda x:'chr{}'.format(x) if x.isdigit() else x))
... mask1 = df['chrom'].str.endswith('.1')
... mask2 = df['chrom'].str.startswith('KN')
... df.loc[mask1,'chrom']= (df['chrom'][mask1]
...                         .apply(lambda x: x[:-2]))
... df.loc[mask2,'chrom']= (df['chrom'][mask2]
...                         .apply(lambda x: "chrUn_{}v1".format(x)))
... df.loc[df.chrom=='MT','chrom'] = 'chrM'
...
... # #filter unnecessary ones
... df = df[df['transcript_biotype']=='protein_coding']
... df = df[df['taip']=='transcript']
...
... df = (df
```

```
...       .sort_values(by='transcript_name')
...       .drop_duplicates('gene_id', keep='first'))


>>> # # get the tss ranges i want
... df.loc[df['strand']=='-', 'tss'] = df.loc[df['strand']=='-', 'end']
... df.loc[df['strand']=='+', 'tss'] = df.loc[df['strand']=='+', 'start']
... df['start'] = df['tss'].astype(int)
... df['score'] = 0
... # # correct out of bounds stuff
... D = dict(pd.read_csv("chromInfo_danRer10.txt",
...     sep='\t', header=None).iloc[:,:2].to_records(index=False))
...
... mask1 = df['start'] >= df['chrom'].map(D)
... df.loc[mask1, 'start'] = df.loc[mask1, 'start'] - 2
... df['start'] = df['start'].clip(lower=0)
... df['end'] = df['start'] + 1
... mask2 = df['end'] >= df['chrom'].map(D)
... df.loc[mask2, 'end'] = df.loc[mask2, 'chrom'].map(D) -1
...
... # # write to disk
... (df[~(df['start']>df['end'])] \
...          [['chrom','start','end','gene_id','score','strand']]
...     .sort_values(['chrom','start','end'])
...     .to_csv("TSS_dre.bed", sep='\t', header=None, index=False))
```

Amphioxus

```
>>> with gzip.open("bl71nemr.gtf.gz",'r') as fi:
...     lot = []
...     line = fi.next()
...     while line.startswith("#"):
...         line = fi.next()
...     # i know this skips one line, fuck this fucking fucknoise
...     for line in fi:
...         (chrom,db,taip,start,end,
...          dot,strand,frame,info) = line.rstrip().split('\t')
...         d = dict([pair.split('"')[:2] \
...             for pair in  info.replace(' ','').split(";")[:-1]])
...         d['chrom'] = chrom
...         d['db'] = db
...         d['taip'] = taip
...         d['start'] = start
...         d['end'] = end
...         d['strand'] = strand
...         d['frame'] = frame
...         lot.append(d)
... df = pd.DataFrame(lot)
... f2 = df['db'] == 'protein_coding'
... f3 = df['taip'] == 'exon'


>>> # select the TSS with the most evidence per gene:
... f = lambda g,x: int(sorted(
```

```
...     Counter(g.loc[g.exon_number=='1', x].values).items(),
...     key=lambda x:x[1], reverse=True)[0][0])
... lot = []
... for gn,g in df[f2 & f3].groupby("gene_id"):
...
...     gstr = g['strand'].value_counts().index.values[0]
...     if gstr == '-':
...         tss = f(g,'end')
...     else:
...         tss = f(g,'start')
...     lot.append( (g.chrom.iloc[0], tss, tss+1, gn, gstr) )
...
... dftss = pd.DataFrame(lot)
... dftss.columns = ['chrom','start','end','gene_id','strand']
... dftss.head()
```

	chrom	start	end	gene_id	strand
0	Sc0000095	792197	792198	BL00000	-
1	Sc0000095	478485	478486	BL00001	-
2	Sc0000095	93743	93744	BL00002	+
3	Sc0000095	352372	352373	BL00003	+
4	Sc0000095	825900	825901	BL00004	+

```
>>> (dftss.sort_values(by=['chrom','start','end'])
...   .to_csv("TSS_bla.bed",sep='\t',index=False))
```

MEDAKA

```
>>> with gzip.open('oryLat2.gtf.gz','r') as fi:
...     lot = []
...     line = fi.next()
...     while line.startswith("#"):
...         line = fi.next()
...     for line in fi:
...         (chrom,db,taip,start,end,
...          dot,strand,frame,info) = line.rstrip().split('\t')
...         d = dict([pair.split('"')[:2] for \
...             pair in  info.replace(' ','').split(";")[:-1]])
...         d['chrom'] = chrom
...         d['db'] = db
...         d['taip'] = taip
...         d['start'] = start
...         d['end'] = end
...         d['strand'] = strand
...         d['frame'] = frame
...         lot.append(d)
... df = pd.DataFrame(lot)

>>> df['chrom'] = (df['chrom']
...     .apply(lambda x: 'chr{}'.format(x) if x.isdigit() else x))
```

```
...
... mask1=df['chrom'].str.endswith('.1')
... df.loc[mask1,'chrom']= df['chrom'][mask1].apply(lambda x: x[:-2])
... df.loc[df.chrom=='MT','chrom'] = 'chrM'
...
... f2 = df['gene_biotype'] == 'protein_coding'
... f3 = df['taip'] == 'transcript'
...
...
... df = (df.sort_values(by='transcript_name')
...     .drop_duplicates('gene_id', keep='first'))
...
... # # get the tss ranges i want
... df.loc[df['strand']=='-', 'tss'] = df.loc[df['strand']=='-', 'end']
... df.loc[df['strand']=='+', 'tss'] = df.loc[df['strand']=='+', 'start']
... df['start'] = df['tss'].astype(int)
... df['score'] = 0
... # # correct out of bounds stuff


>>> D = dict(pd.read_csv("chromInfo_orylat2.txt",
...             sep='\t', header=None).iloc[:,:2].to_records(index=False))


>>> mask1 = df['start'] >= df['chrom'].map(D)
... mask2 = df['end'] >= df['chrom'].map(D)
... df.loc[mask1, 'start'] = df.loc[mask1, 'start'] - 2
... df['start'] = df['start'].clip(lower=0)
... df['end'] = df['start'] + 1
... df.loc[mask2, 'end'] = df.loc[mask2, 'chrom'].map(D) -1


>>> # # write to disk
... (df[~(df['start']>df['end'])]\
...     [['chrom','start','end','gene_id','score','strand']]
...     .sort_values(['chrom','start','end'])
...     .to_csv("TSS_ola", sep='\t', header=None, index=False))
```

Mouse

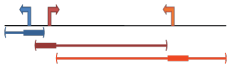
```
>>> with gzip.open("mm10.gtf.gz", "rt") as fi:
...     lot = []
...     line = fi.next()
...     while line.startswith("#"):
...         line = fi.next()
...     # i know this skips one line, fuck this fucking fucknoise
...     for line in fi:
...         (chrom,db,taip,start,end,
...          dot,strand,frame,info) = line.rstrip().split('\t')
...         d = dict([pair.split('"')[:2] for\
...             pair in  info.replace(' ','').split(";")[:-1]])
...         d['chrom'] = chrom
...         d['db'] = db
...         d['taip'] = taip
...         d['start'] = start
```

```
...         d['end'] = end
... #         d['dot'] = dot
...         d['strand'] = strand
...         d['frame'] = frame
... #         d['info'] = info
...         lot.append(d)


>>> df = pd.DataFrame(lot)


>>> f = lambda g,x: int(sorted(
...     Counter(g.loc[g.exon_number=='1', x].values).items(),
...     key=lambda x:x[1], reverse=True)[0][0])
... mgb = df.gene_biotype == 'protein_coding'
... mtb = df.transcript_biotype == 'protein_coding'
...
... lot = []
... for gn,g in df[mgb & mtb].groupby("gene_id"):
...     gstr = g['strand'].value_counts().index.values[0]
...     if gstr == '-':
...         tss = f(g,'end')
...     else:
...         tss = f(g,'start')
...     lot.append( (g.chrom.iloc[0], tss, tss+1, gn, gstr) )


>>> dftss = pd.DataFrame(lot)
... dftss.columns = ['chrom','start','end','gene_id','strand']
... dftss['chrom'] = (dftss['chrom']
...     .apply(lambda x: 'chr{}'.format(x) if x.isdigit() else x))
... dftss['chrom'] = (dftss['chrom']
...     .apply(lambda x: 'chr{}'.format(x) if x in ['X','Y'] else x))
... mask1 = dftss['chrom'].str.endswith('.1')
... dftss.loc[mask1,'chrom']= dftss['chrom'][mask1].apply(lambda x: x[:-2])
... dftss.loc[dftss.chrom=='MT','chrom'] = 'chrM'
...
... (dftss
...     .sort_values(by=['chrom','start','end'])
...     .to_csv("TSS_mm10.bed", sep='\t',index=False))


>>> D = dict(pd.read_csv("chromInfo_Mm10.txt", sep='\t',
...                      header=None).iloc[:,:2].to_records(index=False))
```

Applying the GREAT method on our genomes

We wanted to assign cis regulatory elements to genes in a better way than "assign to closest gene", so we used the method described in (http://great.stanford.edu/public/html/).

Briefly, the method assigns a basal area to each gene (-5kb,+5kb) from its Transcription Start Site, and then extends it up to 1Mb each way until another basal region is met.



The following is code that allows the greation of great-like genomic region files (.bed) based on a TSS bed file and a chrominfo file. For the needs of our analyses, we only keep genes that were found in at least one homologue in another vertebrate species.

```
>>> PROXIMAL_DOWNSTREAM = 1000
... PROXIMAL_UPSTREAM = 5000
... DISTAL = 1000000


>>> # Example for amphioxus, these
... # need to be changed accordingly
... # for each species
... TSS_FP = "TSS_bla.bed"
... CHROMINFO_FP = "chromInfo_braLan71.txt"
... OUT_FP = "GREAT_bla.bed"
... BASAL_OUT_FP = "BASAL_bla.bed"
... SPECIES_COLUMN = 'BraLan'


>>> def gimme_genes(col):
...     log = []
...     for thing in genefams['Dre']:
...         if thing == thing:
...             log += thing
...     return log


>>> D = dict(pd.
...           read_csv(CHROMINFO_FP, sep='\t', header=None)
...           .iloc[:,:2].to_records(index=False))
...
... list_of_genes = gimme_genes(SPECIES_COLUMN)
...
... tss = pd.read_csv(TSS_FP, sep='\t',header=None)
... assert len(tss.columns == 6)
... tss.columns = ['chrom','start','end','name','score','strand']
... tss = tss[tss['name'].isin(list_of_genes)]
...
```

```
... _p = tss['strand']=='+'
... _n = tss['strand']=='-'


>>> basal = tss.copy()
... # Extend from the TSS to the basal regions
... # + genes
... basal.loc[_p,'start'] -= PROXIMAL_UPSTREAM
... basal.loc[_p,'end'] += PROXIMAL_DOWNSTREAM
... # - genes
... basal.loc[_n,'start'] -= PROXIMAL_DOWNSTREAM
... basal.loc[_n,'end'] += PROXIMAL_UPSTREAM


>>> # We have probably gone under 0 and over the chromosome
... # limit in some cases, so we need to correct:
... # We shouldn't really have any case where the end is
... # under 0 or the start over the chromosome limit though,
... # so let's make sure this doesn't happen without us knowing
... assert len(basal[basal.end <=0])==0
... assert len(basal.loc[basal['start'] >= basal['chrom'].map(D)]) ==0
... # now to correct
... basal['start'] = basal['start'].clip(lower = 0)
... basal['end'] = basal['end'].clip(upper = basal['chrom'].map(D))
... basal = basal.reset_index(drop=True)
...
... (basal[['chrom','start','end','name','score','strand']]
...     .sort_values(by=['chrom','start'])
...     .to_csv(BASAL_OUT_FP, sep='\t',header=None,index=False))


>>> basal['index'] = basal.index.values
...
... basal_ends_df = basal[['chrom','end','end',
...                        'name','score','strand','index']]
... basal_ends_df.columns = ['chrom','start','end',
...                          'name','score','strand','index']
... basal_ends_df['start'] -= 1
...
... basal_starts_df = basal[['chrom','start','start',
...                          'name','score','strand','index']]
... basal_starts_df.columns = ['chrom','start','end',
...                            'name','score','strand','index']
... basal_starts_df['end'] += 1


>>> bed_basal = BT().from_dataframe(basal).sort()
... bed_ends = BT().from_dataframe(basal_ends_df).sort()
... bed_starts = BT().from_dataframe(basal_starts_df).sort()
...
... extend_downstream_df = (bed_ends.closest( bed_basal, D='ref',
...                          iu=True, N=True, t='first')
...                          .to_dataframe(names=range(15)))
... extend_downstream_df.set_index(6, drop=True,inplace=True)
... extend_downstream_df.sort_index(inplace=True)
... # cases where no closest was found, probably the rightmost
... # guys in each chromosome
... # we will try to extend by DISTAL and trim it to chrom size later
```

```
... extend_downstream_df.loc[extend_downstream_df[8]==-1 , 14] = DISTAL
...
... extend_downstream = ((extend_downstream_df
...                         .iloc[:,-1] -1 )
...                         .clip(lower=0, upper=DISTAL))


>>> extend_upstream_df = (bed_starts
...                .closest( bed_basal, D='ref', id=True, N=True, t='first')
...                .to_dataframe(names=range(15)))
...
... extend_upstream_df.set_index(6, drop=True,inplace=True)
... extend_upstream_df.sort_index(inplace=True)
...
... # bedtools gives us negative values here
... # we look for the cases where no closest was found
... extend_upstream_df.loc[extend_upstream_df[8]==-1 , 14] = -DISTAL
...
... extend_upstream = ((extend_upstream_df
...                         .iloc[:,-1] + 1)
...                         .clip(lower=-DISTAL, upper=0))
... final = basal.copy()
...
... final['start'] = (final['start'] + extend_upstream ).clip(lower = 0)
... final['end'] = ((final['end'] + extend_downstream )
...                         .clip(upper = final['chrom'].map(D)))
...
... (final[['chrom','start','end','name','score','strand']]
...     .sort_values(by=['chrom','start'])
...     .to_csv(OUT_FP, sep='\t',header=None,index=False))
```

Make the intergenic regions

We define intergenic regions as those between any two consecutive genes. To extract them, we manipulate a bedfile containing the TSS. We only keep a subset of genes since the tss file contaings too many ""fake"" genes, models etc. Out selected genes are all homologous in vertebrates.

```
>>> # Here amphioxus as an example, repeat for the other species
... # (i don't shoe the other species here to minimize the
... # printing of the thesis)
... bla_tss  =pd.read_csv("TSS_bla.bed", sep='\t', header=None)
... lot = []
...
... for gn,g in bla_tss[bla_tss[3].isin(bla_greg.geneID.values)].groupby(0):
...     iterat = g.iterrows()
...     _,curr = next(iterat)
...     while 1:
...         try:
...             _,next_ = next(iterat)
...             lot.append((gn,curr[1], next_[2], curr[3]))
...             lot.append((gn,curr[1], next_[2], next_[3]))
...
...             curr = next_
...         except StopIteration:
...             break
... bla_intergenics = pd.DataFrame(lot)


>>> bla_intergenics.to_csv("intergenics_Bla.tsv",
...                        sep='\t',header=None, index=False)
```

CHIPseq

(Chromatin immunoprecipitation sequencing)

The methods for material handling and the wet lab parts of these molecular assays are detailed in previous publications [WHICH ONES?]. We conducted some additional experiments on zebrafish to complement the available dataset. These include replicates for dome, 80% epiboly and 24hpf, as well as H3k27ac assays in ducplicate for the shield and 8 somites stages.

In total, we have 5 stages in zebrafish: some, shield, 80% epiboly, 8 somites, 24hpf (these are the words i have typed the most in my phd) and three in amphixus: 8hpf, 15hpf, 36hpf.
After the material is sequences we obtain a .fq file, which contains the raw sequencing reads, a collection of about 49bp long sequences and their sequencing quality scores. To use these we must map them to a reference genome, converting each sequence to its absolute coordinates in the genome. We typically only keep reads that can only be mapped to a single position in the genome in order to avoid conflicts.

---

1. Fuentes, M. et al. Preliminary observations on the spawning conditions of the European amphioxus (Branchiostoma lanceolatum) in captivity. J Exp Zool B Mol Dev Evol 302, 384-391 (2004).
2. Fuentes, M. et al. Insights into spawning behavior and development of the European amphioxus (Branchiostoma lanceolatum). J Exp Zool B Mol Dev Evol 308, 484-493 (2007).

Data processing

We used the following commands to map our .fq files

```
# map to a temporary bam file
>> bowtie2 -p 12 -x ${GenomeIndex} -U ${fqfile}  --very-sensitive-local |\
    samtools view -Shu - > ${temp_bam}

# Sort, remove duplicates, index
>> samtools sort -@ 11 ${temp_bam}  |  samtools rmdup -s - - > ${final_bam}
>> samtools index ${final_bam}
```

The resulting .bam files contain the reads, that were succesfully mapped according to our settings, as positions on the genome. This is in essence the raw signal of each experiment.
We used these files to detect peaks with the following command

```
>> macs2 callpeak -f BAM -g ${genome_size} -p  0.005 -t ${fqfile} \
    -n ${outmputname} --outdir ${outputfolder}
```

The macs2 program takes care of estimating an apropriate amount of basepairs to extend each reach, an important factor in this process.

As a result we obtain a bed file, a set of rows, each describing a genomic region and some scores regarding how sure we are that this is a statistically important peak of the signal.

As a final step, for each developmental level, we merge the peaks from the two respective replicates into one combined file.

Further analysis

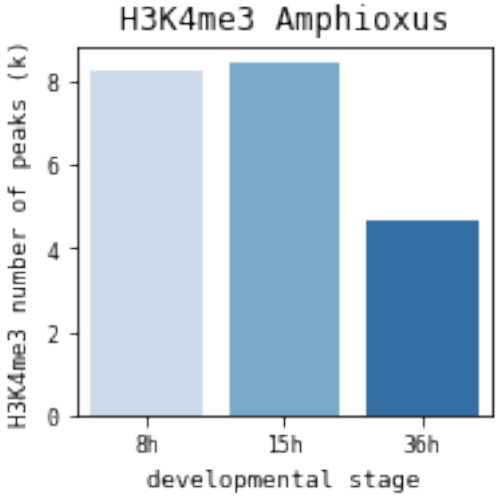
To better analyze our peaks, we will need some more data and some code:

```
>>> load_NP = lambda fp: pd.read_csv( fp, sep='\t',header=None)
...
... speciesorder = ['amphioxus','zebrafish']
...
... amphioxus_stages = ['8h','15h','36h']
... zebrafish_stages = ['dome','shield','80epi','8som','24h','48h']


>>> def get_widths(f, stage):
...     df = load_NP(f(stage))
...     return pd.DataFrame((df[2]-df[1]))
...
... def get_masked_widths(f, stage,mask):
...     df = load_NP(f(stage))
...     ndf = BT().from_dataframe( df.iloc[:,:3] ).subtract(mask).to_dataframe()
...     return pd.DataFrame((ndf.end - ndf.start))
...
... def get_masked_genome_coverage(f, stage,mask, effective):
...     df = load_NP(f(stage))
...     ndf = BT().from_dataframe( df.iloc[:,:3] ).subtract(mask).to_dataframe()
...
...     return (ndf.end-ndf.start).sum()*100 / effective
...
... def get_numbers(f, stage):
...     return len(load_NP(f(stage)))
```

Plots

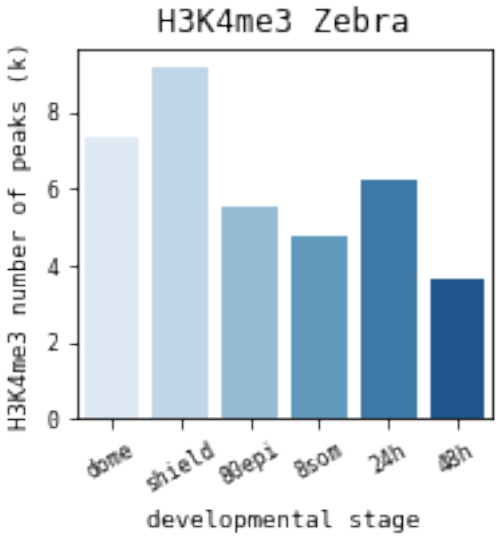
```
>>> Fwidth = THESIS_PAGEWIDTH/2
... Fheight = THESIS_PAGEWIDTH/2
```

```
>>> amphi_ef = amphi_k4_005combo
... amphi_lot = []
... for st in amphioxus_stages:
...     ldf = get_numbers(amphi_ef,st)
...     amphi_lot.append([st, ldf])
... amphi_tp = pd.DataFrame(amphi_lot)
...
... amphi_tp[1] = amphi_tp[1]/10000
...
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.15, bottom=.15, right=.99, top=.9)
... ax.set_title('H3K4me3 Amphioxus')
... sns.barplot(data=amphi_tp, x=0,y=1, ax=ax, palette='Blues', )
... ax.set_ylabel('H3K4me3 number of peaks (k)')
... ax.set_xlabel('developmental stage')
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_amphiK4peaks.pdf')
```

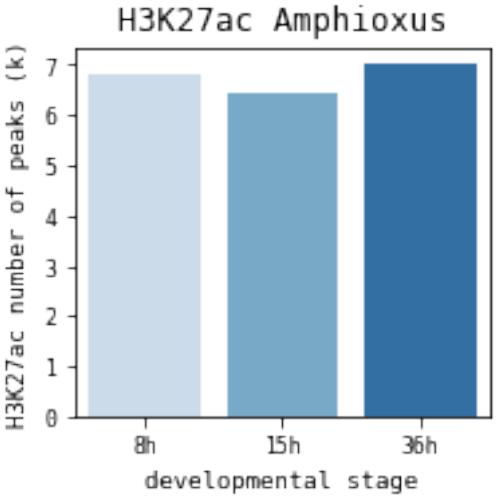



```
>>> zebra_ef = zebra_k4_005combo
... zebra_lot = []
... for st in zebrafish_stages:
...     ldf = get_numbers(zebra_ef,st)
...     zebra_lot.append([st, ldf])
... zebra_tp = pd.DataFrame(zebra_lot)
... zebra_tp[1] = zebra_tp[1]/10000
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.15, bottom=.15, right=.99, top=.9)
...
... sns.barplot(data=zebra_tp, x=0,y=1, ax=ax, palette='Blues', )
... ax.set_title('H3K4me3 Zebra')
... ax.set_ylabel('H3K4me3 number of peaks (k)')
```

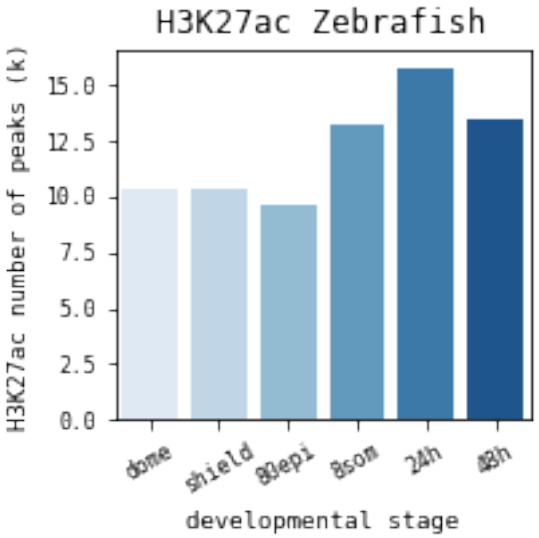
```
... ax.set_xlabel('developmental stage')
... plt.xticks(rotation=30)
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_zebraK4peaks.pdf')
```



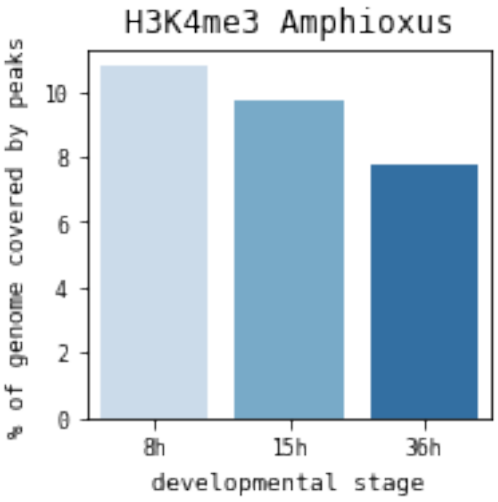

```
>>> amphi_ef = amphi_k27_005combo
... amphi_lot = []
... for st in amphioxus_stages:
...     ldf = get_numbers(amphi_ef,st)
...     amphi_lot.append([st, ldf])
... amphi_tp = pd.DataFrame(amphi_lot)
...
... amphi_tp[1] = amphi_tp[1]/10000
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.15, bottom=.15, right=.99, top=.9)
... sns.barplot(data=amphi_tp, x=0,y=1, ax=ax, palette='Blues', )
... ax.set_title('H3K27ac Amphioxus')
... ax.set_ylabel('H3K27ac number of peaks (k)')
... ax.set_xlabel('developmental stage')
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_amphiK27peaks.pdf')
```



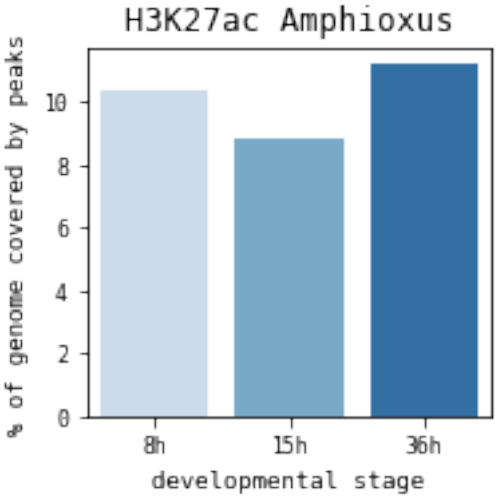

```
>>> zebra_ef = zebra_k27_005combo
... zebra_lot = []
... for st in zebrafish_stages:
...     ldf = get_numbers(zebra_ef,st)
...     zebra_lot.append([st, ldf])
... zebra_tp = pd.DataFrame(zebra_lot)
... zebra_tp[1] = zebra_tp[1]/10000
...
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.15, bottom=.15, right=.99, top=.9)
... sns.barplot(data=zebra_tp, x=0,y=1, ax=ax, palette='Blues', )
... ax.set_title('H3K27ac Zebrafish')
... ax.set_ylabel('H3K27ac number of peaks (k)')
... ax.set_xlabel('developmental stage')
... plt.xticks(rotation=30)
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_zebraK27peaks.pdf')
```



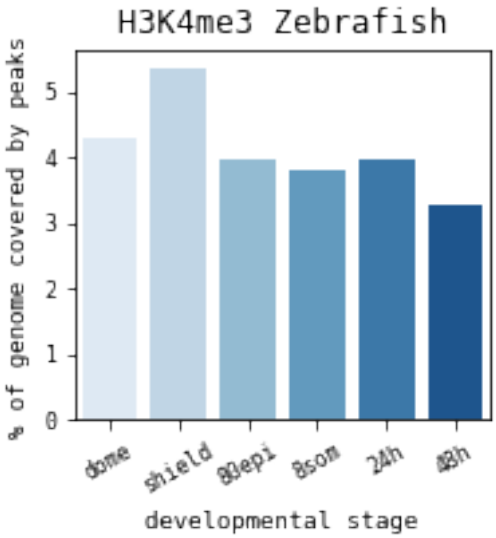

```
>>> amphi_ef = amphi_k4_005combo
... amphi_lot = []
... for st in amphioxus_stages:
...     ldf = get_masked_genome_coverage(amphi_ef,st, amphi_mask, amphi_effective)
...     amphi_lot.append([st, ldf])
... amphi_tp = pd.DataFrame(amphi_lot)
...
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.17, bottom=.15, right=.99, top=.9)
... sns.barplot(data=amphi_tp, x=0,y=1, ax=ax, palette='Blues', )
... ax.set_title('H3K4me3 Amphioxus')
... ax.set_ylabel('% of genome covered by peaks')
... ax.set_xlabel('developmental stage')
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_amphiK4genomeCoverage.pdf')
```



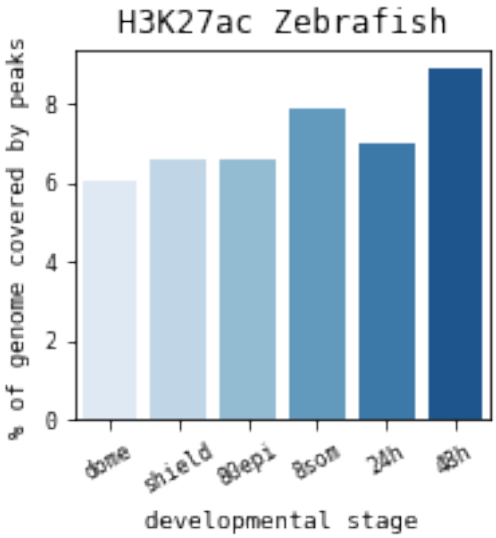

```
>>> amphi_ef = amphi_k27_005combo
... amphi_lot = []
... for st in amphioxus_stages:
...     ldf = get_masked_genome_coverage(amphi_ef,st, amphi_mask, amphi_effective)
...     amphi_lot.append([st, ldf])
... amphi_tp = pd.DataFrame(amphi_lot)
...
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.17, bottom=.15, right=.99, top=.9)
... sns.barplot(data=amphi_tp, x=0,y=1, ax=ax, palette='Blues', )
... ax.set_title('H3K27ac Amphioxus')
... ax.set_ylabel('% of genome covered by peaks')
... ax.set_xlabel('developmental stage')
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_amphiK27genomeCoverage.pdf')
```



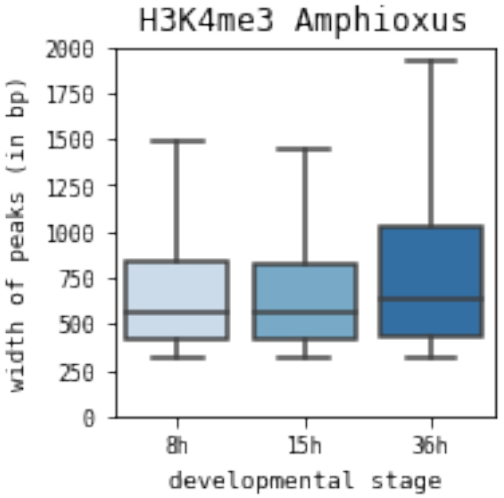

```
>>> zebra_ef = zebra_k4_005combo
... zebra_lot = []
... for st in zebrafish_stages:
...     ldf = get_masked_genome_coverage(zebra_ef,st,
...                                      zebrafish_mask, zebrafish_effective)
...     zebra_lot.append([st, ldf])
... zebra_tp = pd.DataFrame(zebra_lot)
...
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.15, bottom=.15, right=.99, top=.9)
...
... sns.barplot(data=zebra_tp, x=0,y=1, ax=ax, palette='Blues', )
... ax.set_title('H3K4me3 Zebrafish')
... ax.set_ylabel('% of genome covered by peaks')
... ax.set_xlabel('developmental stage')
... plt.xticks(rotation=30)
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_zebraK4genomeCoverage.pdf')
```



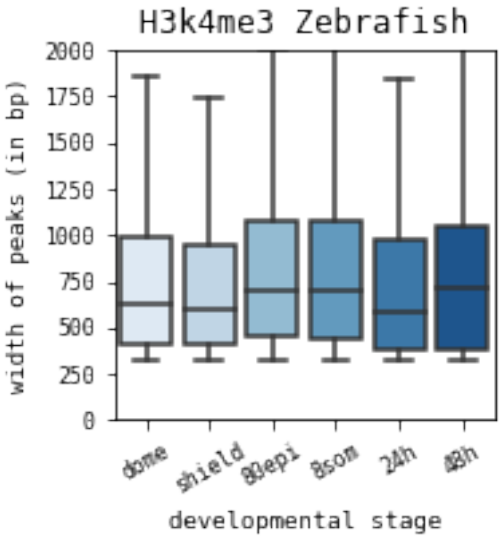

```
>>> zebra_ef = zebra_k27_005combo
... zebra_lot = []
... for st in zebrafish_stages:
...     ldf = get_masked_genome_coverage(zebra_ef,st, zebrafish_mask,
...                                      zebrafish_effective)
...     zebra_lot.append([st, ldf])
... zebra_tp = pd.DataFrame(zebra_lot)
...
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.15, bottom=.15, right=.99, top=.9)
... sns.barplot(data=zebra_tp, x=0,y=1, ax=ax, palette='Blues', )
... ax.set_title('H3K27ac Zebrafish')
... ax.set_ylabel('% of genome covered by peaks')
... ax.set_xlabel('developmental stage')
... plt.xticks(rotation=30)
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_zebraK27genomeCoverage.pdf')
```



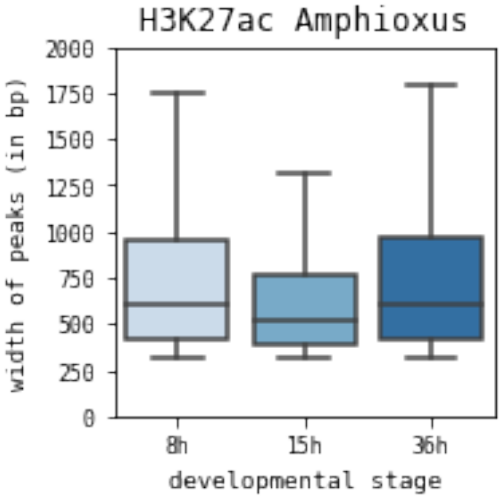

```
>>> amphi_ef = amphi_k4_005combo
... amphi_lot = []
... for st in amphioxus_stages:
...     ldf = get_widths(amphi_ef,st)
...     ldf['stage'] = st
...     amphi_lot.append(ldf)
... amphi_tp = pd.concat(amphi_lot)
...
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.22, bottom=.15, right=.99, top=.9)
... sns.boxplot(data=amphi_tp, x='stage',y=0, ax=ax, palette='Blues',
...                                                  fliersize=0)
... plt.ylim((0,2000))
... ax.set_title('H3K4me3 Amphioxus')
... ax.set_ylabel('width of peaks (in bp)')
... ax.set_xlabel('developmental stage')
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_amphiK4peakwidth.pdf')
```



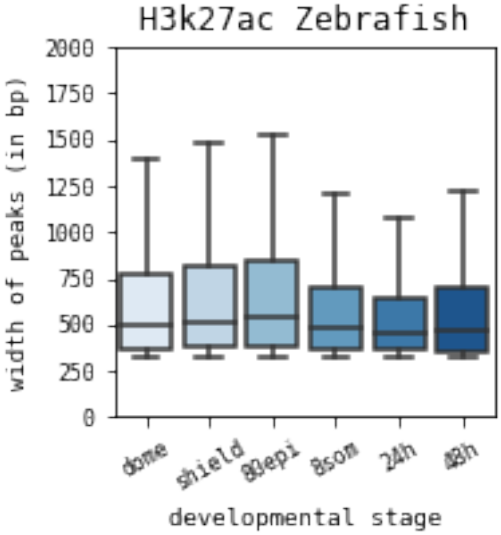

```
>>> zebra_ef = zebra_k4_005combo
... zebra_lot = []
... for st in zebrafish_stages:
...     ldf = get_widths(zebra_ef,st)
...     ldf['stage'] = st
...     zebra_lot.append(ldf)
... zebra_tp = pd.concat(zebra_lot)
...
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.22, bottom=.15, right=.99, top=.9)
... sns.boxplot(data=zebra_tp, x='stage',y=0, ax=ax, palette='Blues',
...                                                    fliersize=0)
... plt.ylim((0,2000))
... ax.set_title('H3k4me3 Zebrafish')
... ax.set_ylabel('width of peaks (in bp)')
... ax.set_xlabel('developmental stage')
... plt.xticks(rotation=30)
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_zebraK4peakwidth.pdf')
```




```
>>> amphi_ef = amphi_k27_005combo
... amphi_lot = []
... for st in amphioxus_stages:
...     ldf = get_widths(amphi_ef,st)
...     ldf['stage'] = st
...     amphi_lot.append(ldf)
... amphi_tp = pd.concat(amphi_lot)
...
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.22, bottom=.15, right=.99, top=.90)
... sns.boxplot(data=amphi_tp, x='stage',y=0, ax=ax, palette='Blues', fliersize=0)
... plt.ylim((0,2000))
... ax.set_title('H3K27ac Amphioxus')
... ax.set_ylabel('width of peaks (in bp)')
... ax.set_xlabel('developmental stage')
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_amphiK27peakwidth.pdf')
```




```
>>> zebra_ef = zebra_k27_005combo
... zebra_lot = []
... for st in zebrafish_stages:
...     ldf = get_widths(zebra_ef,st)
...     ldf['stage'] = st
...     zebra_lot.append(ldf)
... zebra_tp = pd.concat(zebra_lot)
...
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.22, bottom=.15, right=.99, top=.9)
... sns.boxplot(data=zebra_tp, x='stage',y=0, ax=ax, palette='Blues', fliersize=0)
... plt.ylim((0,2000))
... ax.set_title('H3k27ac Zebrafish')
... ax.set_ylabel('width of peaks (in bp)')
... ax.set_xlabel('developmental stage')
... plt.xticks(rotation=30)
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_zebraK27peakwidth.pdf')
```


H3k27ac Zebrafish
width of peaks (in bp)
2000
1750
1500
1250
1000
750
500
250
0
dome
shield
80epi
8som
24h
48h
developmental stage

ATACseq

The methods for material handling and the wet lab parts of these molecular assays are detailed in previous publications [WHICH ONES?].

In total, we have 6 stages in zebrafish: some, shield, 80% epiboly, 8 somites, 24hpf, 48hpf and four in amphixus: 8hpf, 15hpf, 36hpf, 60hpf. After the material is sequences we obtain a .fq file, which contains the raw sequencing reads, a collection of about 49bp long sequences and their sequencing quality scores. To use these we must map them to a reference genome, converting each sequence to its absolute coordinates in the genome. We typically only keep reads that can only be mapped to a single position in the genome in order to avoid conflicts.

---

1. Fuentes, M. et al. Preliminary observations on the spawning conditions of the European amphioxus (Branchiostoma lanceolatum) in captivity. J Exp Zool B Mol Dev Evol 302, 384-391 (2004).
2. Fuentes, M. et al. Insights into spawning behavior and development of the European amphioxus (Branchiostoma lanceolatum). J Exp Zool B Mol Dev Evol 308, 484-493 (2007).

Data processing

We used the following commands to map our .fq files

```
# map to a temporary bam file
>> bowtie2 -p 12 -x ${Genomeindex} -1 ${fqfile_1} -2 ${fqfile_2} \
      --dovetail -I 0 -X 1200 --very-sensitive  |  \
      samtools view -Shu - > ${temp_bam}

>> samtools sort -@ 11 -T ${temp_bam} | \
        samtools rmdup -s - - > ${final_bam}

>>samtools index ${final_bam
```

In comparison to the CHIPseq experiments, we need to do some preprocessing before looking for peaks. The most important difference is that we will filter reads by their fragment size and keep only those under a certain threshold. Since our data is pair-ended sequenced, we can deduce the fragment size of each of dna fragments that were sequenced, that's the distance between each two successive reads. Fragments over 146bp long are much more likely to originate from transposase events that happened on two sides of a nucleosome. A nucleosome typically has 146bp wrapped around it, so but taking

fragments under 146bp long, we make sure to enrich our signal in nucleosome free regions which are prime susspects for cis-regulatory elements.

To filter the reads, we used a home made script found in the Appendinx which we used like this:

```
# We filter for under 120bp to be extra strict
>> Thesis/scripts/get_nf_reads.py ${input_bam_file} 120 24 \
                                      "./results/chunk_{}.bed.gz"
```

As a result, we obtain the approved sequencing reads in a .bed file format which we can use to look for peaks. We apply the idr method to detect peaks in ATACseq and take advantage of having replicates and a more appropriate type of signal (many shorter peaks).

The analysis between this point and the final peaks file is a bit too long to include here so it can be found in the appendix :

`Appendices / Scripts / ATACseq IDR Peakcalling`

Briefly, we apply the IDR method from [1] with no modifications. See [2] for the code repository of their python package that we used.

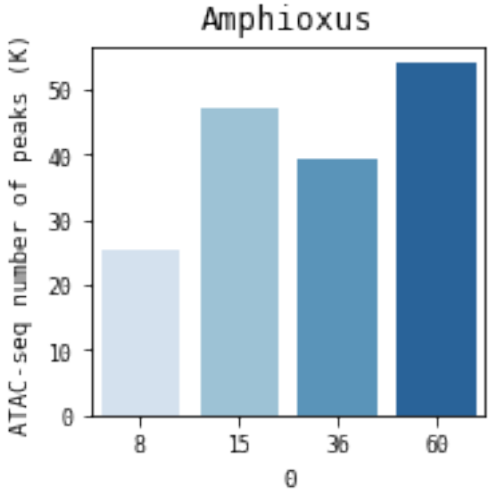
---

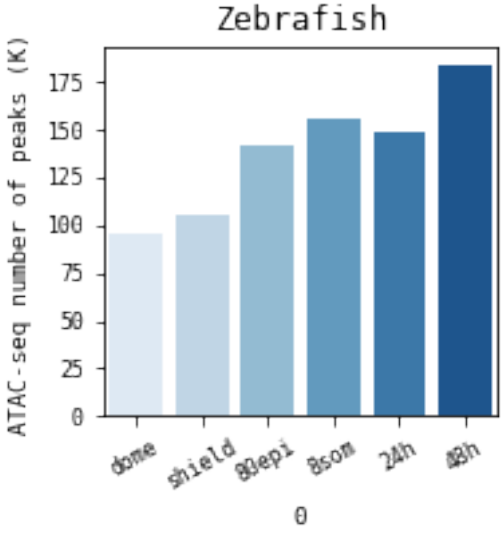
1. "Measuring reproducibility of high-throughput experiments" (2011), Annals of Applied Statistics, Vol. 5, No. 3, 1752-1779, by Li, Brown, Huang, and Bickel
2. https://github.com/nboley/idr

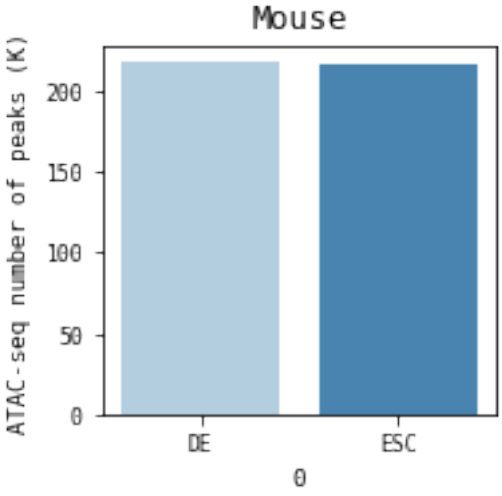
```
>>> load_NP = lambda fp: pd.read_csv( fp, sep='\t',header=None)
...
... speciesorder = ['amphioxus','zebrafish','medaka','mouse']
... amphioxus_stages = ['8','15','36','60']
... zebrafish_stages = ['dome','shield','80epi','8som','24h','48h']
... medaka_stages = ['dome','shield','8som','24h','48h']
... mouse_stages = ['DE','ESC']


>>> def get_widths(f, stage):
...     df = load_NP(f(stage))
...     return pd.DataFrame((df[2]-df[1]))
... def get_masked_widths(f, stage,mask):
...     df = load_NP(f(stage))
...     ndf = (BT().
...            from_dataframe( df.iloc[:,:3] )
...            .subtract(mask, nonamecheck=True)
...            .to_dataframe())
...     return pd.DataFrame((ndf.end - ndf.start))
... def get_masked_genome_coverage(f, stage,mask, effective):
...     df = load_NP(f(stage))
...     ndf = (BT().
...            from_dataframe( df.iloc[:,:3] )
...            .subtract(mask, nonamecheck=True)
...            .to_dataframe())
```

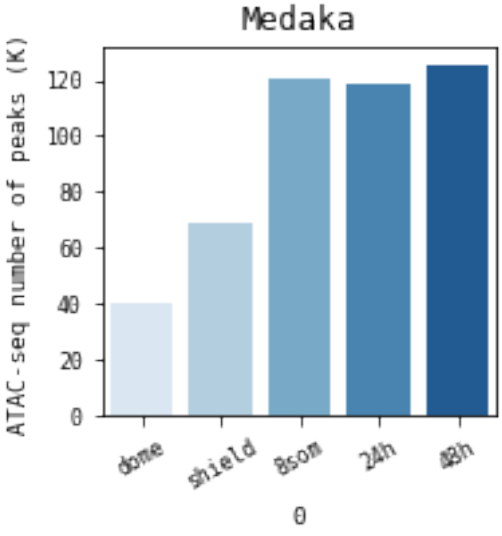
```
...     return (ndf.end-ndf.start).sum()*100 / effective
... def get_numbers(f, stage):
...     return len(load_NP(f(stage)))


>>> # load the data for each species
... amphi_ef = amphi_idr
... zebra_ef = zebra_idr
... mouse_ef = mouse_idr
... medaka_ef = medaka_idr
...
... amphi_lot = []
... for st in amphioxus_stages:
...     ldf = get_numbers(amphi_ef,st)
...     amphi_lot.append([st, ldf])
... amphi_tp = pd.DataFrame(amphi_lot)
...
... zebra_lot = []
... for st in zebrafish_stages:
...     ldf = get_numbers(zebra_ef,st)
...     zebra_lot.append([st, ldf])
... zebra_tp = pd.DataFrame(zebra_lot)
...
... mouse_lot = []
... for st in mouse_stages:
...     ldf = get_numbers(mouse_ef,st)
...     mouse_lot.append([st, ldf])
... mouse_tp = pd.DataFrame(mouse_lot)
...
... medaka_lot = []
... for st in medaka_stages:
...     ldf = get_numbers(medaka_ef,st)
...     medaka_lot.append([st, ldf])
... medaka_tp = pd.DataFrame(medaka_lot)


>>> #the _tp dataframes look like this:
... amphi_tp.head(2)


    0      1
0   8  25455
1  15  47002


>>> # divide by 1000 just to make the y axis labels on the plot smaller
... amphi_tpc = amphi_tp.copy()
... amphi_tpc[1] = amphi_tpc[1]/1000
... zebra_tpc = zebra_tp.copy()
... zebra_tpc[1] = zebra_tpc[1]/1000
... mouse_tpc = mouse_tp.copy()
... mouse_tpc[1] = mouse_tpc[1]/1000
... medaka_tpc = medaka_tp.copy()
... medaka_tpc[1] = medaka_tpc[1]/1000


>>>
```

```
>>> Fwidth = THESIS_PAGEWIDTH/2
... Fheight = THESIS_PAGEWIDTH/2
...
... fig, ax = plt.subplots(1, 1)
... fig.subplots_adjust(left=.19, bottom=.15, right=.99, top=.9)
... sns.barplot(data=amphi_tpc, x=0,y=1, ax=ax, palette='Blues',order=['8','15','36','60'] )
... # ax.set_xlabel('amphioxus (dev stages)')
... ax.set_ylabel('ATAC-seq number of peaks (K)')
... fig.set_size_inches(Fwidth, Fheight)
... plt.title('Amphioxus')
... fig.savefig('../Figures/from_notebooks/tfigure_amphiK4peaks_0.pdf')
...
... fig, ax = plt.subplots(1, 1)
... fig.subplots_adjust(left=.19, bottom=.15, right=.99, top=.9)
... sns.barplot(data=zebra_tpc, x=0,y=1, ax=ax, palette='Blues', )
... # ax.set_xlabel('zebrfish (dev stages)')
... ax.set_ylabel('ATAC-seq number of peaks (K)')
... plt.xticks(rotation=30)
... fig.set_size_inches(Fwidth, Fheight)
... plt.title('Zebrafish')
... fig.savefig('../Figures/from_notebooks/tfigure_amphiK4peaks_1.pdf')
...
... fig, ax = plt.subplots(1, 1)
... fig.subplots_adjust(left=.19, bottom=.15, right=.99, top=.9)
... sns.barplot(data=mouse_tpc, x=0,y=1, ax=ax, palette='Blues', )
... # ax.set_xlabel('mouse (dev stages)')
... ax.set_ylabel('ATAC-seq number of peaks (K)')
... fig.set_size_inches(Fwidth, Fheight)
... plt.title('Mouse')
... fig.savefig('../Figures/from_notebooks/tfigure_amphiK4peaks_2.pdf')
...
... fig, ax = plt.subplots(1, 1)
... fig.subplots_adjust(left=.19, bottom=.15, right=.99, top=.9)
... sns.barplot(data=medaka_tpc, x=0,y=1, ax=ax, palette='Blues', )
... # ax.set_xlabel('medaka (dev stages)')
... ax.set_ylabel('ATAC-seq number of peaks (K)')
... plt.xticks(rotation=30)
... fig.set_size_inches(Fwidth, Fheight)
... plt.title('Medaka')
... fig.savefig('../Figures/from_notebooks/tfigure_amphiK4peaks_3.pdf')
```


Amphioxus
ATAC-seq number of peaks (K)
0
10
20
30
40
50
8
15
36
60
0


Zebrafish
ATAC-seq number of peaks (K)
0
25
50
75
100
125
150
175
dome
shield
80epi
8som
24h
48h
0


Mouse
ATAC-seq number of peaks (K)
200
150
100
50
0
DE
ESC
0


Medaka
ATAC-seq number of peaks (K)
120
100
80
60
40
20
0
dome
shield
8som
24h
48h
0

>>>

TSS-CRE distances

In this notebook, we plot the cumulative distributions of the distances of ATACseq peaks from TranscriptionStartSites. We also plot the values after normalizing them based on the average intergenic distance of each genome.

```
>>> # We extract all homologous genes for each species
... # (genefams is loaded in the preamble.py)
... good_amphi_genes = [x for y in genefams['Bla'].dropna() for x in y]
... good_zebra_genes = [x for y in genefams['Dre'].dropna() for x in y]
... good_medaka_genes = [x for y in genefams['Ola'].dropna() for x in y]
... good_mouse_genes = [x for y in genefams['Mmu'].dropna() for x in y]


>>> (len(good_amphi_genes),len(good_zebra_genes),
...     len(good_medaka_genes),len(good_mouse_genes))


(20569, 20082, 15978, 19429)


>>> # for a notebook on how to make the TSS file see:
... # myphdthesis/other_notebooks/make_tss.ipynb
... amphi_tss_df = pd.read_csv("/Thesis_shallow/data/genomic_regions/TSS_bla.bed.gz",
...                            sep='\t')
... amphi_tss_df['score'] = 0
... amphi_tss_df = amphi_tss_df[['chrom','start','end','gene_id','score','strand']]
... amphi_tss_df = amphi_tss_df[amphi_tss_df.gene_id.isin(good_amphi_genes)]
... amphi_tss_bed = BT().from_dataframe(amphi_tss_df)
...
... zebra_tss_df = pd.read_csv("/Thesis_shallow/data/genomic_regions/TSS_dre.bed.gz",
...                            sep='\t', header=None)
... zebra_tss_df.columns = ['chrom','start','end','gene_id','score','strand']
... zebra_tss_df = (zebra_tss_df[zebra_tss_df.gene_id.isin(good_zebra_genes)]
...                 .copy())
... zebra_tss_bed = BT().from_dataframe(zebra_tss_df)
...
... medaka_tss_df = pd.read_csv("/Thesis_shallow/data/genomic_regions/TSS_ola.bed.gz",
...                             sep='\t', header=None)
... medaka_tss_df.columns = ['chrom','start','end','gene_id','score','strand']
... medaka_tss_df =( medaka_tss_df[medaka_tss_df.gene_id.isin(good_medaka_genes)]
...                 .copy())
... medaka_tss_bed = BT().from_dataframe(medaka_tss_df)
...
... mouse_tss_df = pd.read_csv("/Thesis_shallow/data/genomic_regions/TSS_mmu.bed.gz",
...                            sep='\t', header=None)
... mouse_tss_df.columns = ['chrom','start','end','gene_id','score','strand']
... mouse_tss_df = mouse_tss_df[['chrom','start','end','gene_id','score','strand']]
... mouse_tss_df = (mouse_tss_df[mouse_tss_df.gene_id.isin(good_mouse_genes)]
...                 .copy())
... mouse_tss_bed = BT().from_dataframe(mouse_tss_df)
...
... mouse_tss_df.head()
```

	chrom	start	end	gene_id	score	strand

```
0  chr1  3671498  3671499  ENSMUSG00000051951      0      -
1  chr1  4360314  4360315  ENSMUSG00000025900      0      -
2  chr1  4496413  4496414  ENSMUSG00000025902      0      -
3  chr1  4785710  4785711  ENSMUSG00000033845      0      -
4  chr1  4807823  4807824  ENSMUSG00000025903      0      +
```

To get the average intergenic regions:

```
>>> # for a notebook on how to make the intergenics file see:
... # myphdthesis/other_notebooks/make_intergenic_regions.ipynb
...
... amphi_intergenics = (
...     pd.read_csv("/Thesis_shallow/data/genomic_regions/intergenics_Bla.tsv.gz",
...     sep='\t', header=None))
... amphi_aid = (amphi_intergenics[2]-amphi_intergenics[1]).mean()
...
... zebra_intergenics = (
...     pd.read_csv("/Thesis_shallow/data/genomic_regions/intergenics_Dre.tsv.gz",
...     sep='\t', header=None))
... zebra_aid = (zebra_intergenics[2]-zebra_intergenics[1]).mean()
...
... medaka_intergenics = (
...     pd.read_csv("/Thesis_shallow/data/genomic_regions/intergenics_Ola.tsv.gz",
...     sep='\t', header=None))
... medaka_aid = (medaka_intergenics[2]-medaka_intergenics[1]).mean()
...
... mouse_intergenics = (
...     pd.read_csv("/Thesis_shallow/data/genomic_regions/intergenics_Mmu.tsv.gz",
...     sep='\t', header=None))
... mouse_aid = (mouse_intergenics[2]-mouse_intergenics[1]).mean()


>>> # the average intergenic regions, we'll use these later to normalize
... amphi_aid, zebra_aid, medaka_aid, mouse_aid


(22280.674919495326,
 67502.639512022986,
 48814.903121566596,
 136213.51419155949)


>>> amphi_tss_df.shape, zebra_tss_df.shape, medaka_tss_df.shape, mouse_tss_df.shape


((20569, 6), (20053, 6), (15978, 6), (18842, 6))
```

In the following cells:

The species_APs_ii are ATAC peaks found only inside the intergenic regions, This way we can investigate their distance to a gene and make sure that those distances are not affected by the fragmenttion of the genome.

species_closest then is a

```
>>> amphi_APs = BT(amphi_idr('merged'))
... amphi_APs_ii = (amphi_APs
...     .intersect(BT()
...     .from_dataframe(amphi_intergenics), u=True))
... amphi_closest = (amphi_APs
...     .closest(b=amphi_tss_bed, D='b', io=True, t='first')
...     .to_dataframe())
... amphi_ii_closest = (amphi_APs_ii
...     .closest(b=amphi_tss_bed, D='b', io=True, t='first')
...     .to_dataframe())
...
... zebra_APs = BT(zebra_idr('merged'))
... zebra_closest = (zebra_APs
...     .closest(b=zebra_tss_bed, D='b', io=True, t='first', nonamecheck=True)
...     .to_dataframe())
... zebra_APs_ii = (zebra_APs
...     .intersect(BT()
...     .from_dataframe(zebra_intergenics), u=True, nonamecheck=True))
... zebra_ii_closest = (zebra_APs_ii
...     .closest(b=zebra_tss_bed, D='b', io=True, t='first', nonamecheck=True)
...     .to_dataframe())
...
... medaka_APs = BT(medaka_idr('merged'))
... medaka_closest = (medaka_APs
...     .closest(b=medaka_tss_bed, D='b', io=True, t='first',nonamecheck=True)
...     .to_dataframe())
... medaka_APs_ii = (medaka_APs
...     .intersect(BT()
...     .from_dataframe(medaka_intergenics), u=True, nonamecheck=True))
... medaka_ii_closest = (medaka_APs_ii
...     .closest(b=medaka_tss_bed, D='b', io=True, t='first', nonamecheck=True)
...     .to_dataframe())
...
... mouse_APs = BT(mouse_idr('merged'))
... mouse_closest = (mouse_APs
...     .closest(b=mouse_tss_bed, D='b', io=True, t='first')
...     .to_dataframe())
... mouse_APs_ii = (mouse_APs
...     .intersect(BT()
...     .from_dataframe(mouse_intergenics), u=True, nonamecheck=True))
... mouse_ii_closest = (mouse_APs_ii
...     .closest(b=mouse_tss_bed, D='b', io=True, t='first', nonamecheck=True)
...     .to_dataframe())
```

Some peaks will not be assigned a "closest" TSS, for example if there no TSS on the scaffold where the peak was found. We want to get rid of those and we

do it by filtering the rows where 'thickStart'=='.' (those are the rows with no assignment)

```
>>> amphi_closest = amphi_closest[amphi_closest.thickStart!='.']
... amphi_ii_closest = amphi_ii_closest[amphi_ii_closest.thickStart!='.']
... zebra_closest = zebra_closest[zebra_closest.thickStart!='.']
... zebra_ii_closest = zebra_ii_closest[zebra_ii_closest.thickStart!='.']


>>> medaka_closest = medaka_closest[medaka_closest.thickStart!='.']
... medaka_ii_closest = medaka_ii_closest[medaka_ii_closest.thickStart!='.']
... mouse_closest = mouse_closest[mouse_closest.thickStart!='.']
... mouse_ii_closest = mouse_ii_closest[mouse_ii_closest.thickStart!='.']


>>> # Make the distances absolute:
... amphi_closest['absdist'] = amphi_closest.blockCount.abs()
... zebra_closest['absdist'] = zebra_closest.blockCount.abs()
... medaka_closest['absdist'] = medaka_closest.blockCount.abs()
... mouse_closest['absdist'] = mouse_closest.blockCount.abs()
... amphi_ii_closest['absdist'] = amphi_ii_closest.blockCount.abs()
... zebra_ii_closest['absdist'] = zebra_ii_closest.blockCount.abs()
... medaka_ii_closest['absdist'] = medaka_ii_closest.blockCount.abs()
... mouse_ii_closest['absdist'] = mouse_ii_closest.blockCount.abs()
```

Now we manipulate a bit in order to get the data in plotting order

```
>>> # we order our peaks by distance
... ac = amphi_closest.sort_values(by='absdist').copy()
... # Then we divide the rank of each peak by the number of peaks
... # this value then becomes: "what % of peaks are above me in the table?"
... ac['EDCV'] = np.arange(len(ac))/len(ac)
... # make a normalized absolute distance:
... ac['absdist_normed'] = ac.absdist/amphi_aid
... ac[['thickStart','EDCV','absdist_normed']].head()
... # to-plot object:
... # we drop the duplicates of 'absdist_normed' and keep only the last instance
... # of each the 'EDCV' value will be our Y acis values with the absdist_normed
... # value on the X axis
... tp_ac = (ac
...   .drop_duplicates('absdist_normed',
...                    keep='last')[['absdist_normed', 'EDCV']]
...   .to_records( index=False))


>>> # repeat for the other species
... zc = zebra_closest.sort_values(by='absdist').copy()
... zc['EDCV'] = np.arange(len(zc))/len(zc)
... zc['absdist_normed'] = zc.absdist/zebra_aid
... tp_zc = (zc
...     .drop_duplicates('absdist_normed',
...                      keep='last')[['absdist_normed', 'EDCV']]
...     .to_records( index=False))
```

```
...
... medc = medaka_closest.sort_values(by='absdist').copy()
... medc['EDCV'] = np.arange(len(medc))/len(medc)
... medc['absdist_normed'] = medc.absdist/medaka_aid
... tp_medc = (medc
...     .drop_duplicates('absdist_normed',
...                      keep='last')[['absdist_normed', 'EDCV']]
...     .to_records( index=False))
...
... mmuc = mouse_closest.sort_values(by='absdist').copy()
... mmuc['EDCV'] = np.arange(len(mmuc))/len(mmuc)
... mmuc['absdist_normed'] = mmuc.absdist/mouse_aid
... tp_mmuc = (mmuc[mmuc.chrom !='chrY']
...     .drop_duplicates('absdist_normed',
...                      keep='last')[['absdist_normed', 'EDCV']]
...     .to_records( index=False))
```

```
>>> len(ac),len(zc),len(medc),len(mmuc)
```

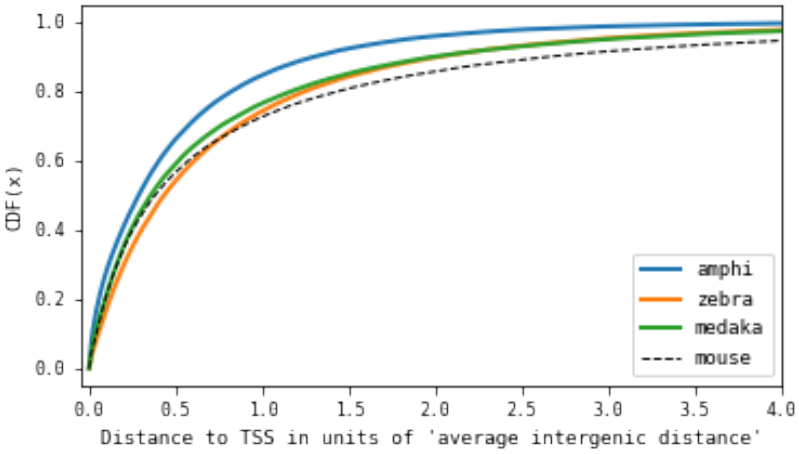
```
(83471, 252774, 174139, 326486)
```

```
>>> len(tp_ac),len(tp_zc),len(tp_medc),len(tp_mmuc)
```

```
(29738, 111926, 75342, 176963)
```

```
>>> Fwidth = THESIS_PAGEWIDTH
... Fheight = (9/16.)*Fwidth
...
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.11, bottom=.14, right=.99, top=.99)
...
... x,y = list(zip(*tp_ac))
... ax.plot(x,y, label='amphi', linewidth=2)
... x,y = list(zip(*tp_zc))
... ax.plot(x,y, label='zebra', linewidth=2, alpha=1, )
... x,y = list(zip(*tp_medc))
... ax.plot(x,y, label='medaka', linewidth=2)
... x,y = list(zip(*tp_mmuc))
... ax.plot(x,y, label='mouse', linewidth=1, alpha=1,
...         linestyle='--' , color='black')
...
... plt.legend(loc='lower right')
... plt.xlim((-0.05,4))
...
... #>>> Name your Axes
... ax.set_ylabel(("CDF(x)"))
... ax.set_xlabel(("Distance to TSS in units of 'average intergenic distance'"))
...
... # ax.yaxis.tick_right()
... # ax.yaxis.set_label_position("right")
... fig.set_size_inches (Fwidth, Fheight)
... #>>> OUTPUT NAME
```

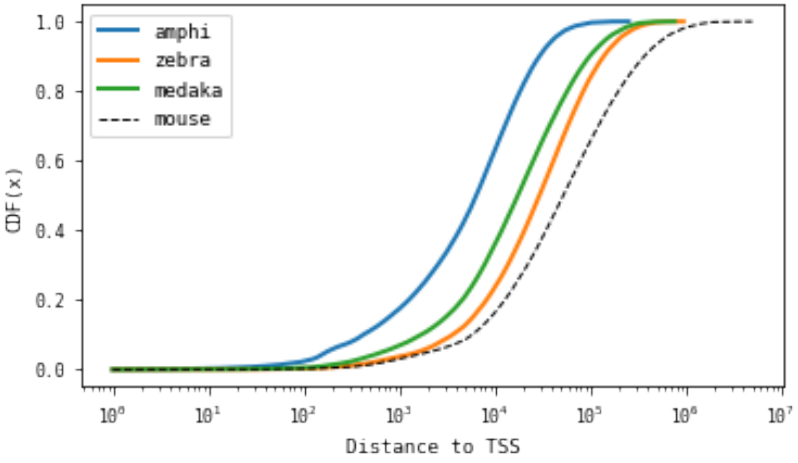
```
... fig.savefig('../Figures/from_notebooks/tfigure_TSSdistanceNorm.pdf')
```



Now we'll do the same but without normalizing the distances

```
>>> # we order our peaks by distance
... ac = amphi_closest.sort_values(by='absdist').copy()
... # Then we divide the rank of each peak by the number of peaks
... # this value then becomes: "what % of peaks are above me in the table?"
... ac['EDCV'] = np.arange(len(ac))/len(ac)
...
... # to-plot object:
... # we drop the duplicates of 'absdist'
... tp_ac = (ac
...          .drop_duplicates('absdist',
...                           keep='last')[['absdist', 'EDCV']]
...          .to_records( index=False))
...
... # repeat for the other species
... zc = zebra_closest.sort_values(by='absdist').copy()
... zc['EDCV'] = np.arange(len(zc))/len(zc)
... tp_zc = zc.drop_duplicates('absdist',
...                            keep='last')[['absdist', 'EDCV']].to_records( index=False)
...
... medc = medaka_closest.sort_values(by='absdist').copy()
... medc['EDCV'] = np.arange(len(medc))/len(medc)
...
... tp_medc = medc.drop_duplicates('absdist',
...                            keep='last')[['absdist', 'EDCV']].to_records( index=False)
...
... mmuc = mouse_closest.sort_values(by='absdist').copy()
... mmuc['EDCV'] = np.arange(len(mmuc))/len(mmuc)
```

```
...
... tp_mmuc = mmuc[mmuc.chrom !='chrY'].drop_duplicates('absdist',
...                 keep='last')[['absdist', 'EDCV']].to_records( index=False)


>>> Fwidth = THESIS_PAGEWIDTH
... Fheight = (9/16.)*Fwidth
...
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.11, bottom=.14, right=.99, top=.99)
...
...
... # ax.set_title('ATACseq peak numbers overview')
...
... x,y = list(zip(*tp_ac))
... ax.plot(x,y, label='amphi', linewidth=2)
... x,y = list(zip(*tp_zc))
... ax.plot(x,y, label='zebra', linewidth=2, alpha=1, )
... x,y = list(zip(*tp_medc))
... ax.plot(x,y, label='medaka', linewidth=2)
... x,y = list(zip(*tp_mmuc))
...
... ax.plot(x,y, label='mouse', linewidth=1, alpha=1,
...         linestyle='--' , color='black')
...
...
... plt.legend()
... # plt.xscale('log')
... # plt.xlim((10**-3,20))
...
... #>>> Name your Axes
... ax.set_ylabel(("CDF(x)"))
... ax.set_xlabel(("Distance to TSS"))
...
...
... plt.xscale('log')
... # ax.yaxis.tick_right()
... # ax.yaxis.set_label_position("right")
... fig.set_size_inches (Fwidth, Fheight)
... #>>> OUTPUT NAME
... fig.savefig('../Figures/from_notebooks/tfigure_tssDistCDF.pdf')
```


amphi
zebra
medaka
mouse
CDF(x)
1.0
0.8
0.6
0.4
0.2
0.0
10^0
10^1
10^2
10^3
10^4
10^5
10^6
10^7
Distance to TSS

The *dalton* plots

In this notebook we conduct some analyses regarding the distibutions of counts of ATACseq peaks inside genomic regions assigned to genes. These are most often GREAT regions (see relevant notebook)

We separate the data in various meaningful ways and plot the distributions with boxplots.

Definition of trans-dev and housekeeping genes

Orthologous clusters of trans-dev genes (i.e. genes implicated in transcriptional regulation or development were defined based on the GO annotations for the mouse orthologs. We downloaded GO annotations for mouse from Ensembl Biomart (release 80) and defined trans-dev genes as those with: (i) GO:0009790 (embryo development) and/or GO:0030154 (cell differentiation) Biological Process annotations; and (ii) GO:0043565 (sequence-specific DNA binding), GO:0007267 (cell-cell signaling) and/or GO:0008380 (RNA splicing) Molecular Function annotations. In contrast, mouse housekeeping genes were defined as those with a 1-to-1 ortholog with yeast (Saccharomyces cerevisiae) based on Ensembl Biomart, and that did not have GO:0009790 (embryo development) or GO:0030154 (cell differentiation) Biological Process annotations. After assigning these categories to the mouse genes, the annotations were transferred to all genes from the same orthologous group, including homologs from other species and their paralogs. This resulted in a total of 654/817, 809/916, 680/805 and 362/865 trans-dev/housekeeping genes in mouse, zebrafish, medaka and amphioxus, respectively (Supplementary Dataset 10).

Data loading and handling

```
>>> # This is a dictionary containing dataframes with the GREAT regions of each
... # species
... greg = gr_great
... #how many genes in each species:
... print( [(k,len(v)) for k,v in greg.items()])
```

```
[('Dre', 20053), ('Bla', 20569), ('Ola', 15978), ('Mmu', 18842)]
```

for each species we have a dataframe in there, each gene is assigned a genomic region for GREAT regions those might overlap. Using this in the later steps we will count the number of ATACseq peaks in the region of each gene

```
>>> greg['Mmu'].head(3)
```

```
  chrom    start      end               geneID    score strand
0  chr1  2670503  4359310  ENSMUSG00000051951  1688807      -
1  chr1  3676503  4495409  ENSMUSG00000025900   818906      -
2  chr1  4365319  4784706  ENSMUSG00000025902   419387      -
```

```
>>> # busywork
... stages = {}
... stages['Bla'] = ['8','15','36','60']
... stages['Dre'] = ["dome","shield","80epi","8som","24h","48h"]
... stages['Ola'] = ["dome","shield","8som","24h","48h"]
... stages['Mmu'] = ['DE','ESC']
```

```
>>> #load the ATACseq peaks
... peak_beds = {}
... peak_beds['Dre'] = [ (BT(zebra_idr(x))
...                       .sort()
...                       )for x in stages['Dre']]
... peak_beds['Bla'] = [(BT(amphi_idr(x))
...                        .sort()
...                        ) for x in stages['Bla']]
... peak_beds['Ola'] = [ (BT(medaka_idr(x))
...                       .sort()
...                       )for x in stages['Ola']]
... peak_beds['Mmu'] = [ (BT(mouse_idr(x))
...                       .sort()
...                       )for x in stages['Mmu']]
```

```
>>> stagespecorder = ['bla_8','bla_15','bla_36','bla_60',' ',
...  'dre_dome','dre_shield','dre_80epi','dre_8som','dre_24h','dre_48h',' ',
...  'ola_dome','ola_shield','ola_8som','ola_24h','ola_48h',' ',
...  'mmu_ESC','mmu_DE'
... ]
```

We will use the following to split genes in genomic categories according to how many gene copies are found in mouse by making some masks for the dataframe

```
>>> mask_oto = (genefamsC['Bla']==1) & (genefamsC['Mmu']==1)  # 1-1
... mask_ottw = (genefamsC['Bla']==1) & (genefamsC['Mmu']==2) # 1-2
... mask_otth = (genefamsC['Bla']==1) & (genefamsC['Mmu']==3) # 1-3
... mask_otfo = (genefamsC['Bla']==1) & (genefamsC['Mmu']==4) # 1-4
...
... masks = [mask_oto, mask_ottw,mask_otth,mask_otfo]
... titles = ['1-1','1-2','1-3','1-4']
... # then some sets
... oto_genes = genefams.loc[mask_oto,['Bla','Dre','Mmu','Ola']]
... oto_genes = set([x for y in oto_genes.values.flatten() if y==y for x in y])
... ottw_genes = genefams.loc[mask_ottw,['Bla','Dre','Mmu','Ola']]
... ottw_genes = set([x for y in ottw_genes.values.flatten() if y==y for x in y])
... otth_genes = genefams.loc[mask_otth,['Bla','Dre','Mmu','Ola']]
```

```
... otth_genes = set([x for y in otth_genes.values.flatten() if y==y for x in y])
... otfo_genes = genefams.loc[mask_otfo,['Bla','Dre','Mmu','Ola']]
... otfo_genes = set([x for y in otfo_genes.values.flatten() if y==y for x in y])
... # and use the sets to categorize the genes inside the dataframe
... def categorize(x):
...     if x in oto_genes:
...         return '1-1'
...     elif x in ottw_genes:
...         return '1-2'
...     elif x in otth_genes:
...         return '1-3'
...     elif x in otfo_genes:
...         return '1-4'
...     else:
...         return 'nop'
```

We intersect the GREAT file with the ATACpeaks files
to count peaks inside the region of each gene
This is essentially a bedtools intersect (using pybedtools)

```
>>> bedfields = ['chrom','start','end','name','score','strand']
... big = {}
... big['Dre'] = BT().from_dataframe(greg['Dre']).sort()
... for bee in peak_beds['Dre']:
...     big['Dre'] = big['Dre'].intersect(b = bee, c=True,
...             sorted=True, nonamecheck=True)
... big['Dre'] = big['Dre'].to_dataframe()
... big['Dre'].columns = bedfields + stages['Dre']
... # the other species:
... big['Bla'] = BT().from_dataframe(greg['Bla']).sort()
... for bee in peak_beds['Bla']:
...     big['Bla'] = big['Bla'].intersect(b = bee, c=True,
...             sorted=True, nonamecheck=True)
... big['Bla'] = big['Bla'].to_dataframe()
... big['Bla'].columns = bedfields + stages['Bla']
... big['Ola'] = BT().from_dataframe(greg['Ola']).sort()
... for bee in peak_beds['Ola']:
...     big['Ola'] = big['Ola'].intersect(b = bee, c=True,
...             sorted=True, nonamecheck=True)
... big['Ola'] = big['Ola'].to_dataframe()
... big['Ola'].columns = bedfields + stages['Ola']
... big['Mmu'] = BT().from_dataframe(greg['Mmu']).sort()
... for bee in peak_beds['Mmu']:
...     big['Mmu'] = big['Mmu'].intersect(b = bee, c=True,
...             sorted=True, nonamecheck=True)
... big['Mmu'] = big['Mmu'].to_dataframe()
... big['Mmu'].columns = bedfields + stages['Mmu']


>>> # These dataframe now look like this
... # For each gene, we have its regulatory region
... # in the first 6 columns
... # and then for each ATACseq stage, one column
... # with a count of peaks per gene
... big['Dre'].sample(5)
```

```
       chrom     start       end                 name   score strand  dome  \
10300  chr21  21871040  22084672  ENSDARG00000062056  213632      -    10
13537   chr3  18431140  18559265  ENSDARG00000019932  128125      -     9
10417  chr21  26691246  26881202  ENSDARG00000098766  189956      +     5
10746  chr22    573349    612992  ENSDARG00000059360   39643      -     7
7513   chr19   3231985   3298703  ENSDARG00000014222   66718      +    12

       shield  80epi  8som  24h  48h
10300      13     15    23   17   38
13537       8     15    11    9    7
10417       9     14    12   15   23
10746       4      7     6    5    7
7513        6      9     5    8   10
```

```
>>> # set the gene ID as index in all dataframes of 'big'
... big_ind = {}
... for k,v in big.items():
...     big_ind[k] = v.set_index('name')
...     big_ind[k].columns = [str(x) for x in big_ind[k].columns]
```

```
>>> # some more dataframe manipulations
... dd_dre = big_ind['Dre'].copy()
... dd_dre['category'] = dd_dre.index.to_series().map(categorize)
... dd_dre['species'] = 'dre'
... dd_dre = dd_dre[stages['Dre']+['category','species','score']]
... dd_dre.columns = stages['Dre']+['category','species','score']
...
... dd_ola = big_ind['Ola'].copy()
... dd_ola['category'] = dd_ola.index.to_series().map(categorize)
... dd_ola['species'] = 'ola'
... dd_ola = dd_ola[stages['Ola']+['category','species','score']]
... dd_ola.columns = stages['Ola']+['category','species','score']
...
... dd_bla = big_ind['Bla'].copy()
... dd_bla['category'] = dd_bla.index.to_series().map(categorize)
... dd_bla['species'] = 'bla'
... dd_bla = dd_bla[stages['Bla']+['category','species','score']]
... dd_bla.columns = stages['Bla']+['category','species','score']
...
... dd_mmu = big_ind['Mmu'].copy()
... dd_mmu['category'] = dd_mmu.index.to_series().map(categorize)
... dd_mmu['species'] = 'mmu'
... dd_mmu = dd_mmu[stages['Mmu']+['category','species','score']]
... dd_mmu.columns = stages['Mmu']+['category','species','score']
```

```
>>> # We merge the dataframes of all species into a single one
... # to plot everything together
... # Melting brings the data in the tidy data format which is
... # expected from the seaborn library that we use for plotting
... TOPLOT = pd.concat([
...     pd.melt(dd_dre, id_vars=['category','species','score']),
```

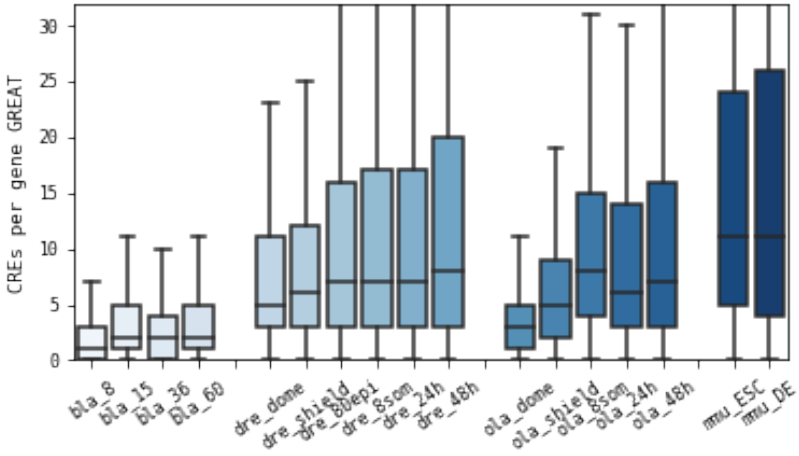
```
...     pd.melt(dd_bla, id_vars=['category','species','score']),
...     pd.melt(dd_ola, id_vars=['category','species','score']),
...     pd.melt(dd_mmu, id_vars=['category','species','score'])
...             ])
... TOPLOT.columns = ['category','species','score','stage','count']
... TOPLOT['specstage'] = TOPLOT.species + '_' + TOPLOT.stage
... TOPLOT.head(2)
```

```
  category species  score stage  count specstage
0      nop     dre   4804  dome      0  dre_dome
1      nop     dre   2189  dome      0  dre_dome
```

The plots

All species, per stage

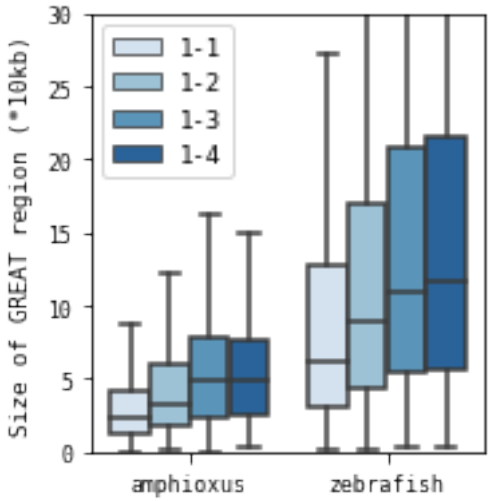
```
>>> Fwidth = THESIS_PAGEWIDTH
... Fheight = Fwidth*(9/16.)
...
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.09, bottom=.19, right=.99, top=.99)
...
... # ax.set_title('ATACseq peak numbers overview')
... sns.boxplot(data = TOPLOT,
...             x='specstage',
...             order=stagespecorder,
...             y='count',
...            fliersize=0, palette='Blues',
...             ax=ax
...            )
...
... ax.set_ylim((0,32))
... for label in ax.get_xticklabels():
...         label.set_rotation(35)
... ax.set_ylabel('CREs per gene GREAT')
... ax.set_xlabel('')
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_dalton2.pdf')
```



Amphi-Zebra 1-X gene categories

```
>>> TP = pd.concat(
... [TOPLOT[ (TOPLOT.species=='dre')
...          & (TOPLOT.stage=='8som')
...          & (TOPLOT.category!='nop')],
... TOPLOT[ (TOPLOT.species=='bla')
...         & (TOPLOT.stage=='15')
...         & (TOPLOT.category!='nop')]
...                 ])
... TP['score'] = TP['score']/10000.
... Fwidth = THESIS_PAGEWIDTH/2
... Fheight = Fwidth
...
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.18, bottom=.09, right=.98, top=.98)
... # ax.set_title('ATACseq peak numbers overview')
... sns.boxplot(data = TP,
...             x='species',
...             order=['bla','dre'],
...             y='score',
...             hue='category',
...             hue_order=['1-1','1-2','1-3','1-4'],
...            fliersize=0, palette='Blues')
...
... ax.set_ylim((0,30))
... plt.legend(loc='upper left')
... ax.set_xticklabels(['amphioxus','zebrafish'],
...                    rotation = 0, ha="center")
... ax.set_ylabel('Size of GREAT region (*10kb)')
... ax.set_xlabel('')
```

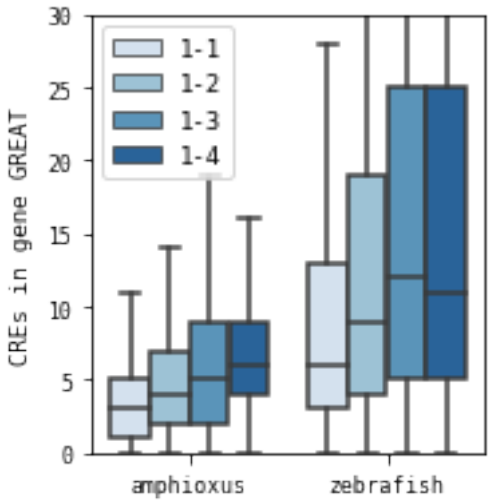
```
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_daltonGR.pdf')
```



Amphi-Zebra 1-X gene categories (GREAT size)

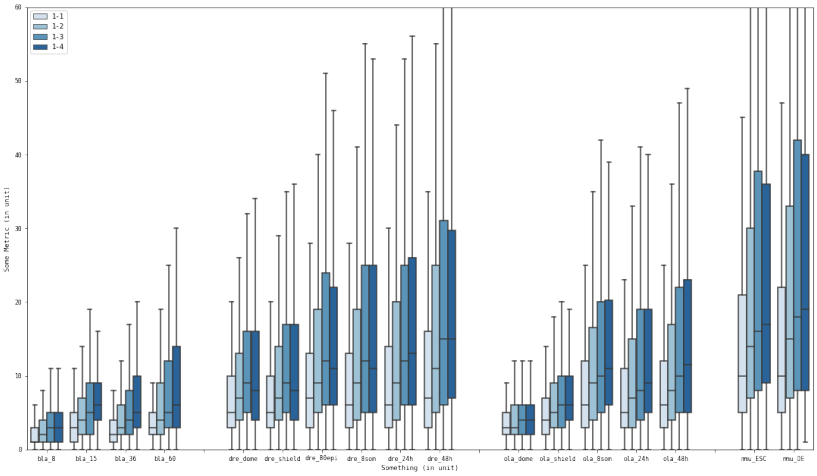
```
>>> TP = pd.concat(
... [TOPLOT[ (TOPLOT.species=='dre')
...          & (TOPLOT.stage=='8som')
...          & (TOPLOT.category!='nop')],
... TOPLOT[ (TOPLOT.species=='bla')
...         & (TOPLOT.stage=='15')
...         & (TOPLOT.category!='nop')]
...                 ])
... Fwidth = THESIS_PAGEWIDTH/2
... Fheight = Fwidth
...
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.18, bottom=.09, right=.98, top=.98)
...
... # ax.set_title('ATACseq peak numbers overview')
... sns.boxplot(data = TP,
...             x='species',
...             order=['bla','dre'],
...             y='count',
...             hue='category',
...             hue_order=['1-1','1-2','1-3','1-4'],
...            fliersize=0, palette='Blues')
...
... ax.set_ylim((0,30))
```

```
... plt.legend(loc='upper left')
... ax.set_xticklabels(['amphioxus','zebrafish'],
...                    rotation = 0, ha="center")
... ax.set_ylabel('CREs in gene GREAT')
... ax.set_xlabel('')
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_dalton3.pdf')
```



All species 1-X gene categories (GREAT size)

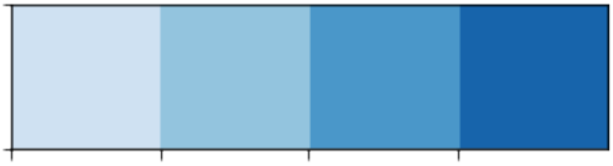
```
>>> fig, ax = plt.subplots()
... fig.subplots_adjust(left=.05, bottom=.05, right=.95, top=.95)
... Fwidth = 16
... Fheight = 9
... sns.boxplot(data = TOPLOT,
...              x='specstage',
...              order=stagespecorder,
...              hue='category',
...              hue_order = ['1-1','1-2','1-3','1-4'],
...              y='count',
...             fliersize=0, palette='Blues',
...              ax=ax
...             )
... ax.set_ylim((0,60))
... plt.legend(loc='upper left')
... ax.set_ylabel('Some Metric (in unit)')
... ax.set_xlabel('Something (in unit)')
... fig.set_size_inches(Fwidth, Fheight)
```

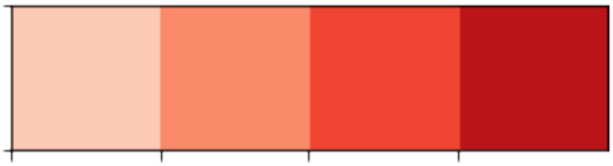


MinMax in each gene family

```
>>> # The following goes through our data
... # and extracts the smallest and biggest
... # count of peaks in each genomic family
... lot = {}
... lot['Dre'] = []
... lot['Bla'] = []
... # for each category ('1-1','1-2' etc)
... for mask,title in zip(masks, titles):
...     # go through the gene families table
...     # and for each family
...     for i,row in genefams.loc[mask,['Bla','Dre']].iterrows():
...         # for each species
...         for spec in ['Bla','Dre']:
...             try:
...                 lgl = row[spec]
...                 #if there's genes in this family for this species
...                 if lgl == lgl:
...                     slaice = (big_ind[spec]
...                               .loc[lgl, stages[spec]])
...                     slaice = pd.concat(
...                         [slaice.min(),slaice.max()], axis=1)
...                     slaice.columns = ['min','max']
...                     slaice['stage'] = (
...                         spec + "_" + slaice.index.values)
...                     slaice = pd.melt(
...                         slaice, id_vars=['stage'])
...                     slaice['title'] = title
...
...                     lot[spec].append( slaice )
...             except:
```

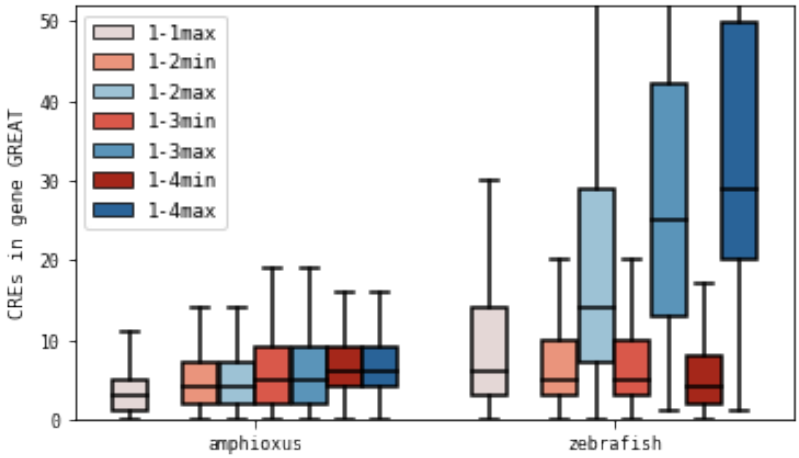
```
...                 pass
... lot['Dre'] = pd.concat(lot['Dre'])
... lot['Bla'] = pd.concat(lot['Bla'])
... bigmelt3 = pd.concat([lot['Dre']
...                       , lot['Bla']])


>>> # a hand-made palette
... bcp = sns.color_palette('Blues',4)
... sns.palplot(bcp)
... rcp = sns.color_palette('Reds',4)
... sns.palplot(rcp)
... mycp = {
... '1-1max':(0.90274 ,  0.83764,  0.82901),
... '1-1min':(0.99137, 0.79137, 0.70823),
... '1-2max':(0.57960, 0.77019, 0.87372),
... '1-2min':(0.98745, 0.54117, 0.41568),
... '1-3max':(0.29098, 0.59450, 0.78901),
... '1-3min':(0.94666, 0.26823, 0.19607),
... '1-4max':(0.09019, 0.39294, 0.67058),
... '1-4min':(0.73647, 0.08, 0.010117),
... '_':(0,0,0)}
```





```
>>> foo = bigmelt3.copy()
... foo['nf'] = foo.title + foo.variable
... foo['norm'] = 0
... toplot = foo.copy()
... toplot['hew'] = toplot.title  + toplot.variable
...
... Fwidth = THESIS_PAGEWIDTH
... Fheight = Fwidth*(9/16.)
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.10, bottom=.08, right=.99, top=.99)
```

```
...
... g = sns.boxplot(
...     x="stage",
...     y="value",
...     hue="hew",
...     fliersize=0,
...     hue_order=['1-1max','_','1-2min','1-2max',
...                '1-3min','1-3max','1-4min','1-4max'],
...     palette = mycp,
...     order=['Bla_15','Dre_8som'],
...     data=toplot,
...     ax=ax
...             )
...
... ax.set_ylim((0,52))
... ax.set_xticklabels(['amphioxus','zebrafish'])
... ax.set_ylabel('CREs in gene GREAT')
... ax.set_xlabel('')
... plt.legend(loc='upper left')
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_dalton4.pdf')
```



TransDev / Housekeeping Genes

```
>>> hkg = pd.read_csv("/Thesis_shallow/data/gene_categories/HouseKeepGenes.tab.gz",
...             header=None,sep='\t',usecols=[1,2])
...
... hkg = hkg[hkg[1].isin(['Bla','Dre'])]
... hkgenes = hkg[2].values
```

```
>>> # We load the housekeeping and later the TD genes
... hkg.head(3)


      1                    2
1   Bla              BL24571
3   Dre  ENSDARG00000002720
12  Bla              BL05749


>>> tdg = pd.read_csv("/Thesis_shallow/data/gene_categories/TransDevGenes.tab.gz",
...             header=None,sep='\t',usecols=[1,2])
...
... tdg = tdg[tdg[1].isin(['Bla','Dre'])]
... tdgenes = tdg[2].values


>>> blac = dd_bla.copy()
... drec = dd_dre.copy()


>>> # apply our TD and HK labels to the rest of the data
... blac['class'] = 'nop'
... drec['class'] = 'nop'
... blac['class'] = blac.apply(
...     lambda x: 'TD' if (x.name in tdgenes) else x['class'],
...     axis=1  )
... blac['class'] = blac.apply(
...     lambda x: 'HK' if (x.name in hkgenes) else x['class'],
...     axis=1  )
... drec['class'] = drec.apply(
...     lambda x: 'TD' if (x.name in tdgenes) else x['class'],
...     axis=1  )
... drec['class'] = drec.apply(
...     lambda x: 'HK' if (x.name in hkgenes) else x['class'],
...     axis=1  )


>>> # To get the pvalues for enrichment of TD/HK in
... # each gene category:
... from scipy.stats import hypergeom
... statf = hypergeom.sf
...
... foo = drec
...
... td_in_population = foo['class'].value_counts()['TD']
... hk_in_population = foo['class'].value_counts()['HK']
...
... pop_size = len(foo)
... for gn,g in foo.groupby('category'):
...
...     ss = len(g)
...     td_is = g['class'].value_counts().get('TD',0)
...     hk_is = g['class'].value_counts().get('HK',0)
...     try:
...         print('td',gn,
...               statf(td_is,  pop_size,  ss, td_in_population ),
```

```
...              sep='\t')
...         print('hk',gn,
...               statf(hk_is,  pop_size,  ss, hk_in_population ),
...              sep='\t')
...     except:
...         continue
```

```
td         1-1         1.0
hk         1-1         0.0
td         1-2         1.92058095378e-11
hk         1-2         1.0
td         1-3         4.41908946369e-45
hk         1-3         1.0
td         1-4         4.18209713364e-27
hk         1-4         1.0
td         nop         1.0
hk         nop         0.999999999991
```

```
>>> # Bit of a lazy compilation of the data we need:
... a = dd_bla.loc[hkg.loc[hkg[1]=='Bla',2].values]
... a['class'] = 'House Keeping'
... b = dd_bla.loc[tdg.loc[tdg[1]=='Bla',2].values]
... b['class'] = 'Trans Dev'
... c1 = pd.concat([a,b])
...
... a = dd_dre.loc[hkg.loc[hkg[1]=='Dre',2].values]
... a['class'] = 'House Keeping'
... b = dd_dre.loc[tdg.loc[tdg[1]=='Dre',2].values]
... b['class'] = 'Trans Dev'
... c2 = pd.concat([a,b])
... c = pd.concat([pd.melt(c1, id_vars=['category','class','species']),
...         pd.melt(c2, id_vars=['category','class','species'])])
```

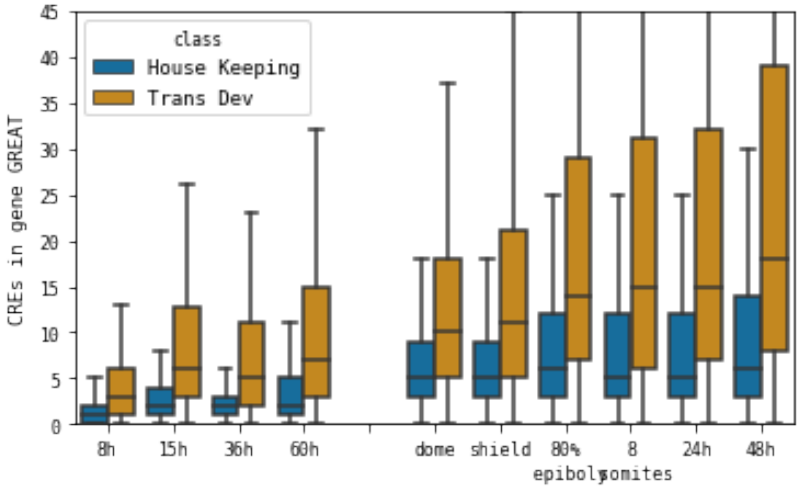
```
>>> c.head(3)
```

```
  category          class species variable  value
0      1-1  House Keeping     bla        8      1
1      1-1  House Keeping     bla        8      1
2      1-1  House Keeping     bla        8      2
```

```
>>> c['catstage'] = c['species'] + '_' + c.variable
```

```
>>> Fwidth = THESIS_PAGEWIDTH
... Fheight = Fwidth*(9/16.)
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.10, bottom=.08, right=.99, top=.99)
...
... _ = sns.boxplot(data=c,
...            x='catstage',
```

```
...              order=stagespecorder[:11],
...             y='value',
...             hue='class',
...             fliersize=0,
...              palette=sns.color_palette("colorblind", 8)
...             )
... ax.set_ylim((0,45))
... ax.set_xticklabels(['8h','15h','36h','60h','',
...     'dome','shield','80%\n epiboly','8\n somites','24h','48h'],
...    rotation=0, ha='center')
...
... ax.set_ylabel('CREs in gene GREAT')
... ax.set_xlabel('')
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_TDHK.pdf')
```

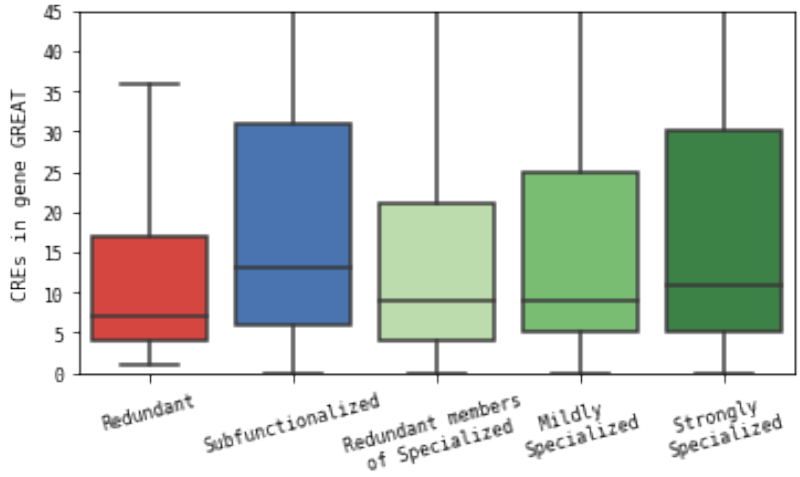


Genes by fate

Here we plot genes split based on how the ohnologues 'behaved' in zebrafish. Redundant genes are genes that both copies retain the full ancestral expression. Subfunctionalized genes lose domains reciprocally, specialized genes lose domains but at least one ohnologue maintains the ancestrall expression (spec_equal). Spec strong genes have lost more domains than spec mild

```
>>> fate_dict_zebra = dict(
...     pd.read_csv("/Thesis_shallow/data/gene_fates/Gene_types-Dre-v3.txt.gz",
...     sep='\t',header=None).set_index(0)[1])
... fate_dict_mouse = dict(
...     pd.read_csv("/Thesis_shallow/data/gene_fates/Gene_types-Mmu-v3.txt.gz",
```

```
...     sep='\t',header=None).set_index(0)[1])
...
... FATE_ORDER = ['REDUNDANT','SUBFUNCT','SPEC_EQUAL',
...               'SPEC_MILD','SPEC_STRONG']
...
... dd_dre['fate'] = dd_dre.index.to_series().map(fate_dict_zebra)
... NTP = pd.melt(dd_dre, id_vars=['category','fate','species'])
...
... NTP.columns = ['category','fate','species','stage','count']
... NTP['specstage'] = NTP.species + '_' + NTP.stage


>>> Fwidth = THESIS_PAGEWIDTH
... Fheight = Fwidth*(9/16.)
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.10, bottom=.20, right=.98, top=.99)
...
... _ = sns.boxplot( data=NTP.dropna(),
...            y="count",
...            x="fate",
...         fliersize=0,
...             order=FATE_ORDER,
...
...             palette=["#ef2f28ff",
...                      "#3871c1ff",
...                       "#b9e4a5ff",
...                       "#6fca65ff",
...                       "#308d3eff"
...                      ]
...            )
... ax.set_ylim((0,45))
... ax.set_xticklabels(['Redundant','Subfunctionalized',
...                     'Redundant members\nof Specialized',
...                     'Mildly \nSpecialized',
...                     'Strongly \nSpecialized'])
...
...
... ax.set_ylabel('CREs in gene GREAT')
... ax.set_xlabel('')
... plt.xticks(rotation=15)
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_fates.pdf')
```

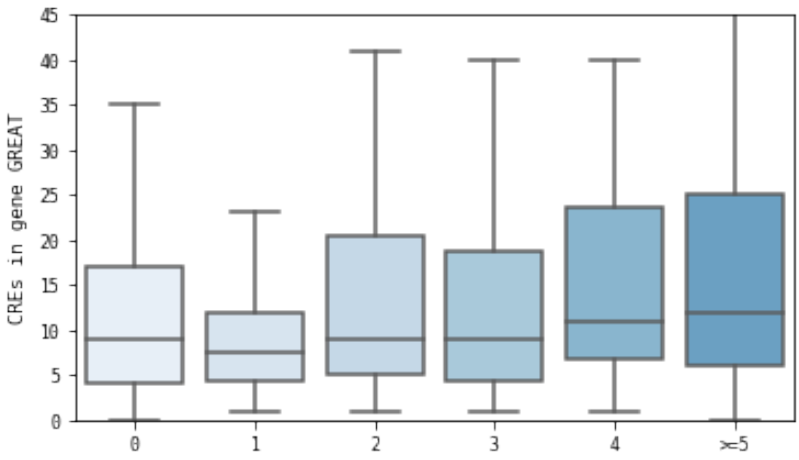


Genes by domain loss

Here we plot genes split based on how many domains of expression were lost in comparison to an amphioxus orthologue. This was computed by our collaborators so we just load the gene lists:

```
>>> fate_dict_zebra = dict(
...     pd.read_csv("/Thesis_shallow/data/gene_fates/Spec_genes_byLost-Dre-v3.txt.gz",
...              sep='\t',header=None).set_index(0)[1])
... dd_dre['fate'] = dd_dre.index.to_series().map(fate_dict_zebra)
... NTP = pd.melt(dd_dre, id_vars=['category','fate','species'])
... NTP.columns = ['category','fate','species','stage','count']
... NTP['specstage'] = NTP.species + '_' + NTP.stage


>>> NTP.loc[NTP['fate']>=5,['fate']] = '>=5'


>>> Fwidth = THESIS_PAGEWIDTH
... Fheight = Fwidth*(9/16.)
...
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.1, bottom=.1, right=.99, top=.99)
... _ = sns.boxplot( data=NTP[NTP.stage=='48h'].dropna(),
...             y="count",
...             x="fate",
...          fliersize=0,
...              order=[0,1,2,3,4,'>=5'],
...              palette=sns.color_palette("Blues", 10)
...             )
... ax.set_ylim((0,45))
```

```
... ax.set_ylabel('CREs in gene GREAT')
... ax.set_xlabel('')
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_domlost.pdf')
```



Data loading and handling

```
>>> # This is a dictionary containing dataframes with the GREAT regions of
... # each species
... greg = gr_great
... #how many genes in each species:
... print( [(k,len(v)) for k,v in greg.items()])
```

```
[('Dre', 20053), ('Bla', 20569), ('Ola', 15978), ('Mmu', 18842)]
```

for each species we have a dataframe
in there, each gene is assigned a genomic region
for GREAT regions those might overlap
Using this in the later steps we will count the number of ATACseq peaks in the region of each gene

```
>>> # busywork
... stages = {}
... stages['Bla'] = ['8','15','36','60']
... stages['Dre'] = ["dome","shield","80epi","8som","24h","48h"]
... stages['Ola'] = ["dome","shield","8som","24h","48h"]
... stages['Mmu'] = ['DE','ESC']


>>> #load the ATACseq peaks
... peak_beds = {}
... peak_beds['Dre'] = [ (BT(zebra_idr(x))
...                       .sort()
...                       )for x in stages['Dre']]
... peak_beds['Bla'] = [(BT(amphi_idr(x))
...                       .sort()
...                       ) for x in stages['Bla']]


>>> stagespecorder = ['bla_8','bla_15','bla_36','bla_60',' ',
...               'dre_dome','dre_shield','dre_80epi','dre_8som',
...                    'dre_24h','dre_48h'
...             ]


>>> bedfields = ['chrom','start','end','name','score','strand']
... big = {}
... big['Dre'] = BT().from_dataframe(greg['Dre']).sort()
... for bee in peak_beds['Dre']:
...     big['Dre'] = big['Dre'].intersect(b = bee,
...                 c=True, sorted=True, nonamecheck=True)
... big['Dre'] = big['Dre'].to_dataframe()
... big['Dre'].columns = bedfields + stages['Dre']
... # the other species:
... big['Bla'] = BT().from_dataframe(greg['Bla']).sort()
... for bee in peak_beds['Bla']:
...     big['Bla'] = big['Bla'].intersect(b = bee,
...                 c=True, sorted=True, nonamecheck=True)
... big['Bla'] = big['Bla'].to_dataframe()
```

```
... big['Bla'].columns = bedfields + stages['Bla']


>>> # These dataframe now have
... big['Dre'].sample(3)
```

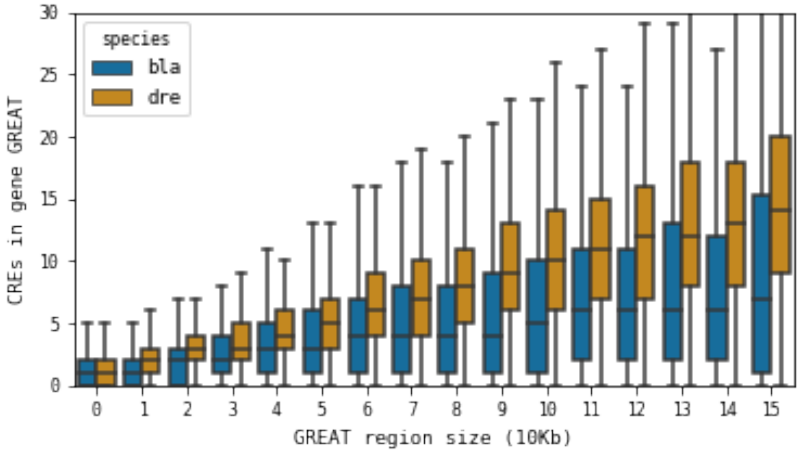
```
       chrom     start       end                 name   score strand  dome  \
11503  chr23  14173274  14336427  ENSDARG00000061445  163153      -     7
15364   chr5  11443577  11529953  ENSDARG00000053019   86376      -     6
12326  chr24  23870711  23914384  ENSDARG00000057789   43673      -     3

       shield  80epi  8som  24h  48h
11503      14     19    10   13   13
15364       5      8     9    8    4
12326       1      3     1    2    5
```

```
>>> buz = pd.melt( big['Dre'][['score'] + stages['Dre']],
...              id_vars=['score'] )
... buz['species'] = 'dre'
...
... biz = pd.melt( big['Bla'][['score'] + stages['Bla']],
...              id_vars=['score'] )
... biz['species'] = 'bla'
...
... buz['score'] = buz['score'].apply(lambda x: int(x/10000) )
... biz['score'] = biz['score'].apply(lambda x: int(x/10000) )
...
... comb = pd.concat([biz,buz])
... comb.head()
```

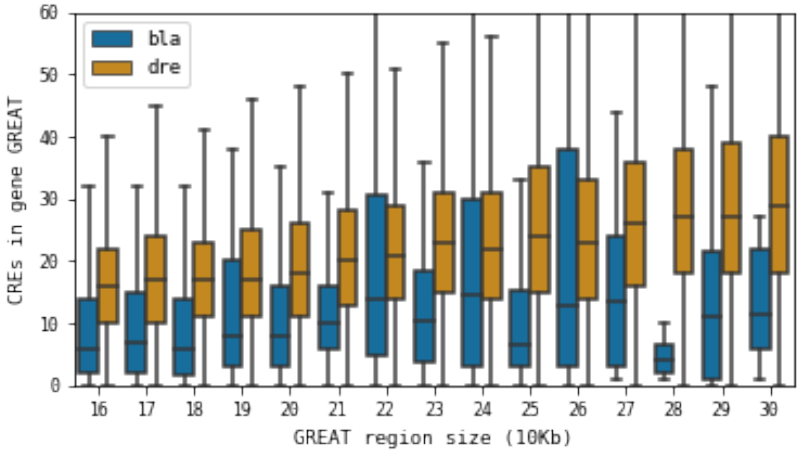
```
   score variable  value species
0     16        8      6     bla
1     34        8     11     bla
2     21        8      6     bla
3      8        8      4     bla
4     11        8      8     bla
```

```
>>> Fwidth = THESIS_PAGEWIDTH
... Fheight = Fwidth*(9/16.)
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.09, bottom=.15, right=.99, top=.98)
...
... sns.boxplot(data=comb[comb['score']<=15],
...             x='score',hue='species',y='value',
...             fliersize=0,
...             palette=sns.color_palette("colorblind", 8))
...
... plt.ylim((0,30))
... ax.set_ylabel('CREs in gene GREAT')
... ax.set_xlabel('GREAT region size (10Kb)')
```

```
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_stratified1.pdf')
```




```
>>> Fwidth = THESIS_PAGEWIDTH
... Fheight = Fwidth*(9/16.)
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.09, bottom=.15, right=.99, top=.98)
...
... sns.boxplot(data=comb[(comb['score']>15) & (comb['score']<=30)],
...             x='score',hue='species',y='value',
...             fliersize=0,
...             palette=sns.color_palette("colorblind", 8))
...
... plt.ylim((0,60))
... ax.set_ylabel('CREs in gene GREAT')
... ax.set_xlabel('GREAT region size (10Kb)')
... plt.legend(loc='upper left')
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_stratified2.pdf')
```

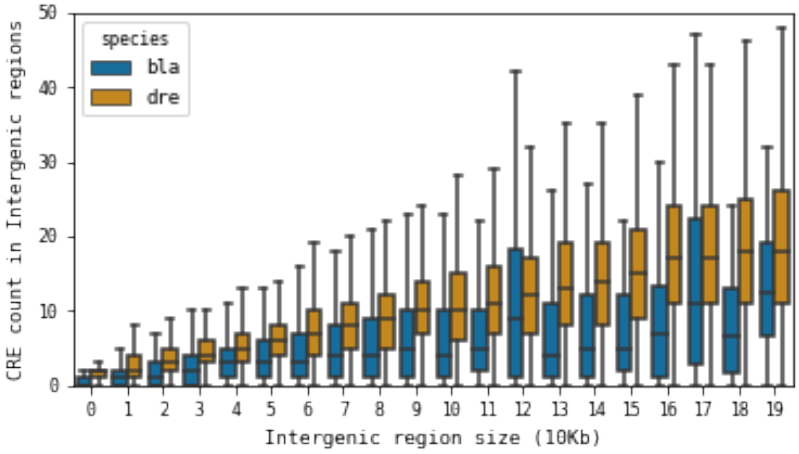



```
>>> gr_interg = {}
... gr_interg['Dre'] = pd.read_csv("/Thesis_shallow/data/genomic_regions/intergenics_Dre.tsv.gz",
...                         sep='\t', header=None)
... gr_interg['Dre'].columns = ['chrom','start','end',
...                             'geneID1','geneID2']
... gr_interg['Dre']['score'] = (gr_interg['Dre']['end']
...                              - gr_interg['Dre']['start'])
... gr_interg['Bla'] = pd.read_csv("/Thesis_shallow/data/genomic_regions/intergenics_Bla.tsv.gz",
...                         sep='\t', header=None)
... gr_interg['Bla'].columns = ['chrom','start','end',
...                             'geneID1','geneID2']
... gr_interg['Bla']['score'] = (gr_interg['Bla']['end']
...                              - gr_interg['Bla']['start'])


>>> greg = gr_interg.copy()
... bedfields = ['chrom','start','end','name','score','strand']
... big = {}
... big['Dre'] = BT().from_dataframe(greg['Dre']).sort()
... for bee in peak_beds['Dre']:
...     big['Dre'] = big['Dre'].intersect(b = bee,
...                               c=True, sorted=True, nonamecheck=True)
... big['Dre'] = big['Dre'].to_dataframe()
... big['Dre'].columns = ['c','st','e','id1','id2','score'] + stages['Dre']
... # the other species:
... big['Bla'] = BT().from_dataframe(greg['Bla']).sort()
... for bee in peak_beds['Bla']:
...     big['Bla'] = big['Bla'].intersect(b = bee,
...                               c=True, sorted=True, nonamecheck=True)
... big['Bla'] = big['Bla'].to_dataframe()
... big['Bla'].columns = ['c','st','e','id1','id2','score'] + stages['Bla']


>>> buz = pd.melt( big['Dre'][['score'] + stages['Dre']], id_vars=['score'] )
... buz['species'] = 'dre'
```

```
...
... biz = pd.melt( big['Bla'][['score'] + stages['Bla']], id_vars=['score'] )
... biz['species'] = 'bla'
...
... buz['score'] = buz['score'].apply(lambda x: int(x/10000) )
... biz['score'] = biz['score'].apply(lambda x: int(x/10000) )
...
... comb = pd.concat([biz,buz])
... # comb['score'] = comb['score'].clip(upper=19)
... comb.head()
```

	score	variable	value	species
0	28	8	5	bla
1	5	8	4	bla
2	1	8	1	bla
3	10	8	5	bla
4	2	8	3	bla

```
>>> Fwidth = THESIS_PAGEWIDTH
... Fheight = Fwidth*(9/16.)
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.09, bottom=.15, right=.99, top=.98)
...
... sns.boxplot(data=comb[comb.score<=19],
...             x='score',hue='species',
...             y='value',
...             fliersize=0,
...             palette=sns.color_palette("colorblind", 8))
...
... plt.ylim((0,50))
... ax.set_ylabel('CRE count in Intergenic regions')
... ax.set_xlabel('Intergenic region size (10Kb)')
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_stratified_intergenics.pdf')
```


species
bla
dre
CRE count in Intergenic regions
0
10
20
30
40
50
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Intergenic region size (10Kb)

Downsampling

In this analysis, we have called peaks (macs2 in each replicate, then idr peaks on the results of the macs2 peaks) in all stages, with increasingly lower number of reads.

For example, the file in

`data/atac_peaks/downsampling/zebra_danRer10_8som_15600000_idr01Peaks.bed.gz`

is idr atac peaks called with 15600000 reads (in each replicate) of the reads from the 8somite zebrafish experiments which means that its coverage (15600000/avail_genome['zebra']) is 0.89 (we'll call it 90%) of the amphioxus coverage (9200000/ avail_genome['amphi'])

The number of reads was chosen so as to have 100%, 90%, 80% etc of the amphioxus number.

The question is, do we still have more peaks in the vertebrates with worse (less reads) experiments?

```
>>> # assign genomic regions to the genes
... greg = gr_great
...
... print( [(k,len(v)) for k,v in greg.items()])
```

```
[('Dre', 20053), ('Bla', 20569), ('Ola', 15978), ('Mmu', 18842)]
```

```
>>> stages = {}
... stages['Bla'] = ['15']
... stages['Dre'] = ["8som"]
... stages['Ola'] = ["8som"]
```

```
>>> # some hardcoded numbers
... df = pd.DataFrame([['zebra', 1371719383, 1369631918, 756790655, 756666441,
...         0.5517095292076951, 0.0015217872006989541, 612841263, 653072511],
...        ['amphi', 495353434, 474881800, 152452412, 152231682,
...         0.3077649240642995, 0.041327328317259604, 322429388, 347547588],
...        ['medaka', 869000216, 700386597, 23221380, 23181494,
...         0.026721949629526905, 0.19403173427979903, 677165217, 846778620]])
... df.columns = ['species','genome','genome_notN','repMask','repnotN',
...       '% genome in repM','% genome is N','effective','previous_effective']
```

```
>>> df
```

```
  species      genome  genome_notN    repMask    repnotN  % genome in repM  \
0   zebra  1371719383   1369631918  756790655  756666441          0.551710
1   amphi   495353434    474881800  152452412  152231682          0.307765
2  medaka   869000216    700386597   23221380   23181494          0.026722
```

```
   % genome is N  effective  previous_effective
0       0.001522  612841263           653072511
1       0.041327  322429388           347547588
2       0.194032  677165217           846778620
```

We will normalize the number of reads based on the amount of available genome in each species.
That is the total genome minus the regions covered by the repeat masker.

Unintuitively, medaka ends up with more available genome that zebrafish because of the very large number of blacklisted regions in zebrafish.

```
>>> avail_genome = dict(df[['species','effective']].to_records(index=False))
... avail_genome
```

```
{'zebra': 612841263, 'amphi': 322429388, 'medaka': 677165217}
```

This is how many reads the bla sample has. It was much easier to hard code this number than to count in this notebook:

```
>>> # We calculate coverage as the
... # #ofReads per kilobase of available genome:
... Bla_coverage = 9200000*1000/ avail_genome['amphi']
... # the percentages are not exactly 70 or 80 etc,
... #so we will force-smooth them
... # a tiny bit to make the graphs better.
... fixcols =[1,10,20,30,40,50,60,70,80,90,100]
```

We have the downsample series in the data subfolder, so lets load them up:

```
>>> greg_ = BT().from_dataframe(greg['Dre']).sort()
... pre_cols = ['chrom','start','end','gene','width','strand']
... cols = []
...
... for thing in glob(
...     "../data/atac_peaks/downsampling/zebra_danRer10_8som_*_idr01Peaks.bed.gz"):
...     reads = int(re.findall(r"[0-9]+", thing)[2])
...
...     #     The coverage in this experiment:
...     cov = reads*1000/avail_genome['zebra']
...     #    The coverage in relation to the Bla one
...     cov = round(cov*100 / Bla_coverage,2)
...     cov = int(round(cov))
...     cols.append( cov )
...     greg_ = greg_.intersect(b=BT( thing ), nonamecheck=True, c=True)
...
... counts_Dre = greg_.to_dataframe(names=pre_cols+cols)
... counts_Dre = counts_Dre[pre_cols+ sorted(cols)]
... counts_Dre.columns = pre_cols + fixcols
... dre_melt = pd.melt(counts_Dre[['gene']+fixcols], id_vars='gene')
... dre_melt.columns = ['gene','pc','count']
```

Same for medaka:

```
>>> greg_ = BT().from_dataframe(greg['Ola']).sort()
... pre_cols = ['chrom','start','end','gene','width','strand']
... cols = []
...
... for thing in glob(
...     "../data/atac_peaks/downsampling/medaka_8som_*_idr01Peaks.bed.gz"):
...     reads = int(re.findall(r"[0-9]+", thing)[1])
...
...     cov = reads*1000/avail_genome['medaka']
...     cov = round(cov*100 / Bla_coverage,2)
...     cov = int(round(cov))
...
...     cols.append( cov )
...     greg_ = greg_.intersect(b=BT( thing ), nonamecheck=True, c=True)
...
... counts_Ola = greg_.to_dataframe(names=pre_cols+cols)
... counts_Ola = counts_Ola[pre_cols+ sorted(cols)]
... counts_Ola.columns = pre_cols + fixcols
... ola_melt = pd.melt(counts_Ola[['gene']+fixcols], id_vars='gene')
... ola_melt.columns = ['gene','pc','count']
```

For amphioxus we'll only load two stages, the full non-downsampled one (101%) and the very minimally downsampled one (100%), which has only been downsampled so that the two replicates have the same number of reads.

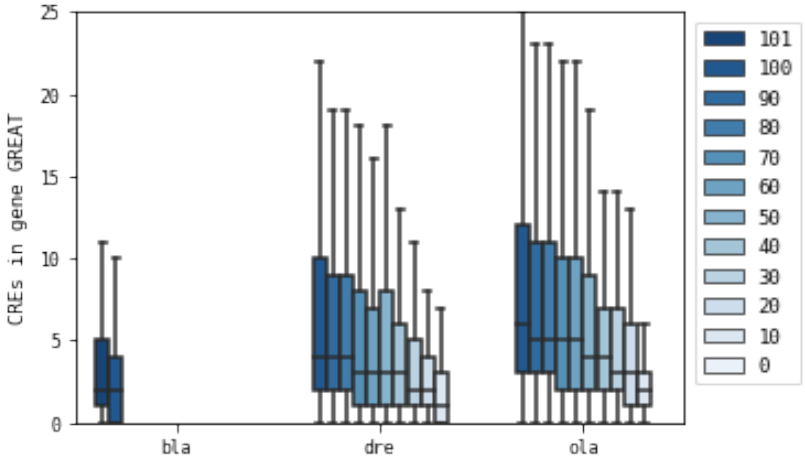
```
>>> org = "bla"
... stage='15'
...
... full_peaks = '../data/atac_peaks/amphi_15_idrpeaks.bed.gz'
... sub_peaks = '../data/atac_peaks/downsampling/amphi_15_9200000_idr01Peaks.bed.gz'
...
...
... greg_ = BT().from_dataframe(greg['Bla']).sort()
... cols = ['chrom','start','end','gene','width','strand']
...
... greg_ = greg_.intersect(b=BT( full_peaks ), nonamecheck=True, c=True)
... greg_ = greg_.intersect(b=BT( sub_peaks ), nonamecheck=True, c=True)
...
... counts_Bla = greg_.to_dataframe(names=cols + [101,100])
...
... bla_melt = pd.melt(counts_Bla[['gene',101,100]], id_vars='gene')
... bla_melt.columns = ['gene','pc','count']


>>> bla_melt['species'] = 'bla'
... ola_melt['species'] = 'ola'
... dre_melt['species'] = 'dre'


>>> bla_melt.head(2)


      gene   pc  count species
0  BL09450  101     10     bla
```

```
1  BL10006  101     17     bla
```

```
>>> toplot = pd.concat([dre_melt, ola_melt, bla_melt])
... toplot.sample(5)
```

```
                       gene   pc  count species
89254   ENSORLG00000016834   50      4     ola
201554  ENSDARG00000040338  100      0     dre
156282  ENSDARG00000092285   70      7     dre
34690   ENSDARG00000007405   10      0     dre
9704    ENSORLG00000002351    1      0     ola
```

```
>>> Fwidth = THESIS_PAGEWIDTH
... Fheight = Fwidth*(9/16.)
...
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.10, bottom=.08, right=.85, top=.98)
...
... # ax.set_title('ATACseq peak numbers overview')
... sns.boxplot(
...     data=toplot,
...
...     x = 'species',
...     order= ['bla','dre','ola'],
...     hue='pc',
...     hue_order=[101,100,90,80,70,60,50,40,30,20,10,0],
...
...     fliersize=0,
...     y='count',
...     palette='Blues_r'
... )
... # ax.set_xlim(0, 3*np.pi)
... ax.set_ylim((0,25))
... # plt.legend(loc='upper left')
...
... # for label in ax.get_xticklabels():
... #         label.set_rotation(45)
...
... plt.legend(loc='upper left')
... #>>> Name your Axes
... ax.set_ylabel('CREs in gene GREAT')
... ax.set_xlabel('')
... # ax.yaxis.tick_right()
... # ax.yaxis.set_label_position("right")
... plt.legend(bbox_to_anchor=(1.0, 1), )
...
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_downsampling.pdf')
```



Let's calculate some PValues:

```
>>> lot = []
... blavals = (toplot
...     .loc[(toplot.species=='bla') & (toplot.pc==101),'count'].values)
... blavals_sub = (toplot
...     .loc[(toplot.species=='bla') & (toplot.pc==100),'count'].values)
...
... for gn,g in toplot.groupby(['species','pc']):
...     species,pc = gn
...     if species=='bla':
...         continue
...     pv = MWU( g['count'].values , blavals,
...               alternative='greater' ).pvalue
...     pv_sub = MWU( g['count'].values , blavals_sub,
...                   alternative='greater' ).pvalue
...
...     lot.append([ species, pc, pv_sub,pv])
...
...
... df = pd.DataFrame(lot)
... df.columns = ['species','% of reads','pVal to downsampled Amphi',
...               'pVal to full Amphi']
... df.sample(3)
```

	species	% of reads	pVal to downsampled Amphi	pVal to full Amphi
4	dre	40	3.859252e-283	1.817653e-94
17	ola	60	0.000000e+00	0.000000e+00
7	dre	70	0.000000e+00	3.523587e-268

Comparison of motif similarity at embryonic stages.

We wanted to investigate if the two species display similarities in their cis-regulatory content. As a first step, we will assess the similarity of developmental stages in the two species based on what TFs are important in them. This notebook contains a few supplementary plots regarding the data and statistical manipulations we apply.

For each experiment (stage), we look into the atac peaks and get counts of "motifs" (PWM significant hits) for a large database of factors. We can then compare two stages based on those counts. We can for example calculate the correlation between stages and thus, we created a heatmap of correlation for the developmental stages of the two species.

We counted motifs (PWMs) in dynamic ATAC peaks of our various developmental stages. We divided each motif's count by the total hits of the motif in all stages and then scaled the values in each stage.

Preparation

Motif families

We have a large set of Position Weight Matrices, which we group together by their protein e.g. AP-2_Average_15 and AP-2_Average_17 are different position weight matrices, but both assigned to the 'Tfap2e' transcription factor so we will consider all instances of these two PWMs as instances of Tfap2e.

Another example, are ARID_BRIGHT_RFX_Average_2,ARID_BRIGHT_RFX_Average_3 and ARID_BRIGHT_RFX_M4343_1.02 :
These 3 PWMs all are assigned to the TFs "Rfx1;Rfx2;Rfx4;Rfx7" so we will consider all instances of those 3 PWMs
to be instances of a "Rfx1;Rfx2;Rfx4;Rfx7" protein family.

To accomplish this, we will make a dictionary that maps PWMs to Protein family names :

The mapped motifs:

```
>>> dan_motif_bed = BT("/myphdthesis/data/PWM/danre_pwm_hits.bed.gz")
... bla_motif_bed = BT("/myphdthesis/data/PWM/bralan_pwm_hits.bed.gz")
... bla_motif_bed.head(2)
```

```
Sc0000000        5593        5598        Homeodomain_Average_626        6.607572
 Sc0000000        5695        5707        C2H2_ZF_Average_154        10.88041492
```

The ATACseq peaks

We have compiled a set of peaks that turn ON or OFF in a reasonable pattern during development, and called these peaks 'dynamic'.

Per stage we will only consider these dynamic peaks.

We load them in pyBedTool objects, BT().

We load the stages from each stage, and keep the ones that have been marked as logicaldynamic

ATAC dynamic peaks

ATACseq peak calling is performed individually in each experiment (developmental stage). Some regions will be open in more than one stage, but due to the nature of the experiments, or because these open regions open slightly more or slightly less in different stages/tissues/cells, we cannot expect these regions to be called with perfectly accurate edges. In order then to track peaks from one stage to another, we considered peaks from different stages to be the "same peak" if they overlap. I.e., if we call a peak in stage 0, from base 10 to base 23 and then another peak in stage 1 from base 5 to base 21, we would consider these two peaks to be the same peak, having remained open for the two stages.

```
stage 0
    chrX 10 23
stage 1
    chrX 5 21
    chrX 50 55
```

We merged the peaks from all experiments to obtain a set of all regions we could consider a peak I.e., the two previously mentioned example peaks would be merged into one:

```
merged peaks
chrX 5 23
chrX 50 55
```

We then consider each of those peaks as active or not active in the various stages depending on whether or not it overlaps with a peak from the stage.

```
merged peaks
chrX 5 23 <--- is active in both stages
chrX 50 55 <--- is only active in stage 1
```

If we mark the activity of a peak with zeros and ones, we can summarize its activity. The "chrX 5 23" peak from our example would have "11" activity, and the other peak would have "01" activity. I.e., we have six developmental stages for zebrafish, so if a peak "turns on" in the third stage and then off again in the sixth stage, it would have activity: "001110".

We considered a subset of peaks as "logically dynamic" if they had one of the following activity profiles:

```
'000011', '000110', '001100', '011000', '110000',
'000111', '001110', '011100', '111000',
'001111', '011110', '111100',
'011111', '111110'
```

Peaks with only one active stage we considered "stage specific", and peaks that we always active were considered "constitutive".

```
>>> # some helper functions to get the dynamic peaks per stage
... a_dynamic = BT(
...     "/myphdthesis/data/atac_peaks/amphi_logicaldynamic_idr.bed.gz")
... def bla_stagepeaks(stage):
...     sd = BT(amphi_idr(stage))
...     sd = sd.intersect(a_dynamic,u=True, nonamecheck=True)
...     return sd.sort()
...
... z_dynamic = BT(
...     "/myphdthesis/data/atac_peaks/zebra_danRer10_logicaldynamic_idr.bed.gz")
... def dan_stagepeaks(stage):
...     sd = BT(zebra_idr(stage))
...     sd = sd.intersect(z_dynamic,u=True, nonamecheck=True)
...     return sd.sort()
...
... amphi_stages = ['8','15','36','60','hep']
... zebra_stages = ['dome','shield','80epi','8som','24h','48h']
...
... bla_peaks = [bla_stagepeaks(stage) for stage in amphi_stages]
... dan_peaks = [dan_stagepeaks(stage) for stage in zebra_stages]


>>> dan_peaks = [BT().from_dataframe(x.to_dataframe().iloc[:,:3])
...              for x in dan_peaks]
... bla_peaks = [BT().from_dataframe(x.to_dataframe().iloc[:,:3])
...              for x in bla_peaks]
... bla_peaks[0].head(3)


Sc0000000         311943         312518
 Sc0000000         319117         319493
 Sc0000000         356649         356918
```

```
>>> TFids = sorted(set(SFDu.values()))


>>> # Here we get a dictionary, mapping the TF unique numbers to a total
... # count per organism
... # We will use this to normalize later on
... _temp = (BT(zebra_idr('merged'))
...          .sort()
...         .intersect(dan_motif_bed, loj=True, sorted=True,
...                    nonamecheck=True)
...         .to_dataframe())
...
... # we then map the SFDu dictionary on the column carrying the PWM names,
... # drop empty rows,and get a count (with Counter, a handy python extension
... # of dictionaries) of each unique ID
... dan_totals = Counter(_temp['thickStart'].map(SFDu).dropna().astype(int))
...
... _temp = (BT(amphi_idr('merged'))
...         .sort()
...         .intersect(bla_motif_bed, loj=True, sorted=True, nonamecheck=True)
...         .to_dataframe())
... bla_totals = Counter(_temp['thickStart'].map(SFDu).dropna().astype(int))
...
... # we cast the total counts in a list with a proper orger:
... dan_normalizer = [dan_totals.get(x,1) for x in TFids]
... bla_normalizer = [bla_totals.get(x,1) for x in TFids]


>>> # the resulting DataFrame after the LOJ operation.
... # for each ATAC peak,we get all its intersections with PWMs
... _temp.head(2)
```

	chrom	start	end	name	score	strand	thickStart	\
0	Sc0000000	5589	5975	Sc0000000	5593	5598	Homeodomain_Average_626	
1	Sc0000000	5589	5975	Sc0000000	5695	5707	C2H2_ZF_Average_154	

	thickEnd	itemRgb
0	6.607572	+
1	10.880415	+

```
>>> #dan_totals is a dictionary mapping the uniqueIds of PWMs to counts
... list(dan_totals.items())[:3]
```

```
[(0, 3653), (1, 16725), (2, 1848)]
```

```
>>> # The "normalizer" lists are derived from the dictionaries and are in
... # order of "TFids"
... bla_normalizer[:3]
```

```
[1188, 3585, 930]
```

Mapping the motifs to peaks:

We did this already to get the normalization counts, now we will do it with the stage dynamic peaks

```
>>> # Here we make a dataframe per stage per organism
... # Each DataFrame corresponds to one stage and maps motifs to the
... # peaks that were active on
... # that stage. We use the LEFT OUTER JOIN function of bedtools to
... # join motifs to peaks:
... bla_lojs = [(stage
...                 .sort()
...                 .intersect(bla_motif_bed, sorted=True, loj=True,
...                            nonamecheck=True)
...                 .to_dataframe()) for stage in bla_peaks]
... dan_lojs = [(stage
...                 .sort()
...                 .intersect(dan_motif_bed, sorted=True, loj=True,
...                            nonamecheck=True)
...                 .to_dataframe()) for stage in dan_peaks]


>>> # For example, the DF for the first stage of amphioxus:
... bla_lojs[0].head(2)


       chrom   start     end       name   score  strand  \
0  Sc0000000  311943  312518  Sc0000000  311942  311946
1  Sc0000000  311943  312518  Sc0000000  312028  312035

              thickStart  thickEnd itemRgb
0    C2H2_ZF_Average_177  5.460094       +
1  Grainyhead_M6529_1.02  7.510921       +


>>> # Finally, we make a DF with counts for each TF in each stage:
... # These are the counts in the dynamic peaks
... ddf = pd.DataFrame(TFids)
... ddf.columns = ['fam']
...
... for en,stage in enumerate(zebra_stages):
...     ddf[stage] = (ddf['fam']
...                 .map(Counter(dan_lojs[en]['thickStart']
...                 .dropna()
...                 .map(SFDu)
...                 .dropna()
...                 .astype(int))))
...
... ddf.set_index('fam', inplace=True, drop=True)
...
... ddf = ddf.T.fillna(1)
... ddf.iloc[:,:4]


fam       0     1    2     3
```

```
dome    161   729   74   389
shield  409  1931  165  1014
80epi   638  2991  228  1657
8som    756  3693  298  2357
24h     785  3911  333  2504
48h     672  3333  335  2185
```

```
>>> # And the same for amphioxus
... bdf = pd.DataFrame(TFids)
... bdf.columns = ['fam']
... for en,stage in enumerate(amphi_stages):
...     bdf[stage] = (bdf['fam']
...                     .map(Counter(bla_lojs[en]['thickStart']
...                                  .dropna()
...                                  .map(SFDu)
...                                  .dropna()
...                                  .astype(int))))
...
... bdf.set_index('fam', inplace=True, drop=True)
...
... bdf = bdf.T.fillna(1)
... bdf.iloc[:,:4]
```

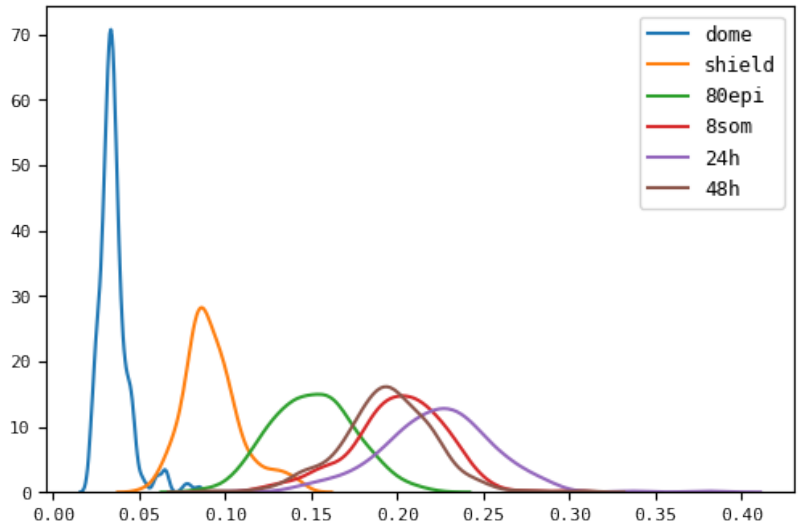
```
fam    0    1    2     3
8     73  257   50   251
15   196  618  172   707
36   219  652  177   942
60   236  761  217  1155
hep  212  608  144   998
```

```
>>> # we divide the count in each stage by the total count
... bdf = bdf/bla_normalizer
... ddf = ddf/dan_normalizer
... bdf.iloc[:,:4]
```

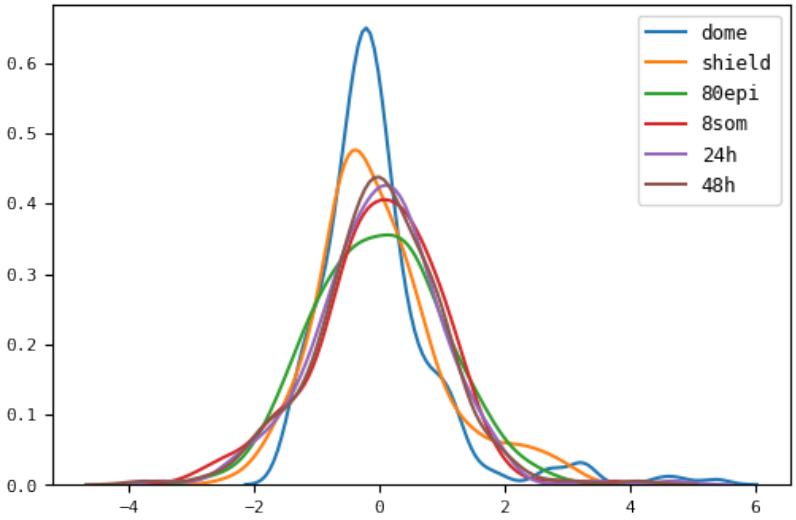
```
fam          0         1         2         3
8     0.061448  0.071688  0.053763  0.052129
15    0.164983  0.172385  0.184946  0.146833
36    0.184343  0.181869  0.190323  0.195639
60    0.198653  0.212273  0.233333  0.239875
hep   0.178451  0.169596  0.154839  0.207269
```

```
>>> # At this point, some TFs have 0 counts, so lets clean them out:
... btodrop = set(bdf.T[bdf.sum() ==0].index.values)
... dtodrop = set(ddf.T[ddf.sum() ==0].index.values)
...
... todrop = btodrop.union(dtodrop)
...
```

```
... bdf = bdf.drop(todrop, axis=1)
... ddf = ddf.drop(todrop, axis=1)


>>> # the distributions of the various stages at this point:
... for irow,row in ddf.iterrows():
...     sns.kdeplot(row)
```




```
>>> # let's scale the counts in order to make stages comparable
... ddf.loc[:,:] = preprocessing.scale(ddf.values, axis=1)
... bdf.loc[:,:] = preprocessing.scale(bdf.values, axis=1)


>>> # the distributions after scaling:
... for irow,row in ddf.iterrows():
...     sns.kdeplot(row)
```



The final table:

In the end, we have a scaled count per motif for each organism-stage. For example the first 5 TF scaled counts in zebrafish-shield look like this:

```
>>> ddf.loc['shield',:].head(3)
```

```
fam
0    1.143067
1    1.350141
2   -0.201232
Name: shield, dtype: float64
```

We can then get a corellation between any two cases

```
>>> scaled_counts_in_zebra_shield = ddf.loc['shield',:]
... scaled_counts_in_amphi_hep = bdf.loc['hep',:]
... # The first value is the pearson metric and the second is
... # the metric's pvalue,how certain we are for the metric.
... pearsonr(scaled_counts_in_zebra_shield.values,
...          scaled_counts_in_amphi_hep.values)
```

```
(-0.18212360335862862, 0.004478666595009111)
```

```
>>> # We can make a table with pair-wise correlations for all stages
... TABLE = pd.DataFrame()
... for dstage in zebra_stages:
...     for bstage in amphi_stages:
...         TABLE.loc[dstage, bstage] = pearsonr(
...             ddf.loc[dstage].values,bdf.loc[bstage].values)[0]
... TABLE
```

	8	15	36	60	hep
dome	0.472447	0.413344	-0.020947	-0.233493	-0.217931
shield	0.427724	0.620226	0.304749	-0.079934	-0.182124
80epi	0.248738	0.577668	0.509119	0.113895	-0.064038
8som	-0.016492	0.297138	0.525042	0.289232	0.156917
24h	-0.091795	0.182115	0.427101	0.263281	0.264657
48h	-0.225320	0.018198	0.359862	0.292274	0.196003

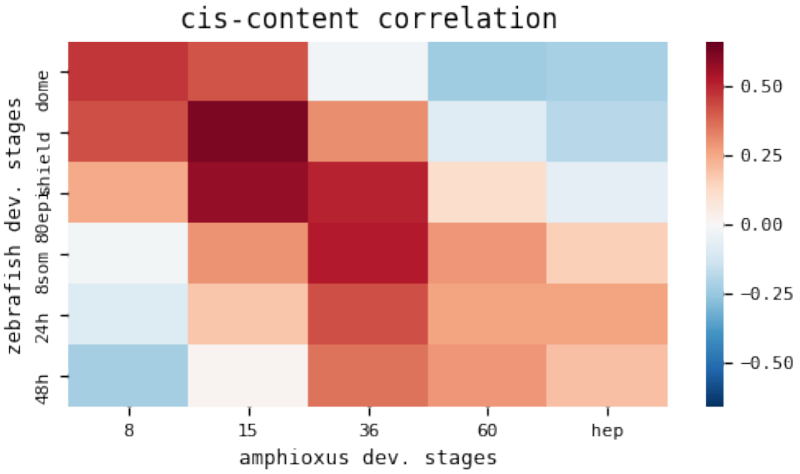
```
>>> # We can make a table with pair-wise correlations for all stages
... TABLE_pvals = pd.DataFrame()
... for dstage in zebra_stages:
...     for bstage in amphi_stages:
...         TABLE_pvals.loc[dstage, bstage] = pearsonr(
...             ddf.loc[dstage].values,bdf.loc[bstage].values)[1]
... TABLE_pvals
```

	8	15	36	60	hep
dome	7.373361e-15	2.101586e-11	7.457777e-01	0.000248	0.000641
shield	3.487620e-12	4.056149e-27	1.354005e-06	0.215337	0.004479
80epi	9.188728e-05	6.117028e-23	2.309811e-17	0.076998	0.321167
8som	7.985353e-01	2.536864e-06	1.510216e-18	0.000005	0.014542
24h	1.545583e-01	4.480531e-03	3.776619e-12	0.000034	0.000030
48h	4.115778e-04	7.782047e-01	8.243826e-09	0.000004	0.002191

We can visualize the table in a heatmap:

```
>>> Fwidth = THESIS_PAGEWIDTH
... Fheight = Fwidth*(9/16.)
...
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.11, bottom=.15, right=1, top=.92)
...
... # ax.set_title('ATACseq peak numbers overview')
... sns.heatmap(TABLE,
...             annot=False,
...             cmap="RdBu_r",
...             vmax=0.66, vmin=-0.66,
...             ax=ax
...            )
... ax.set_title("cis-content correlation")
... #>>> Name your Axes
... ax.set_ylabel('zebrafish dev. stages')
```

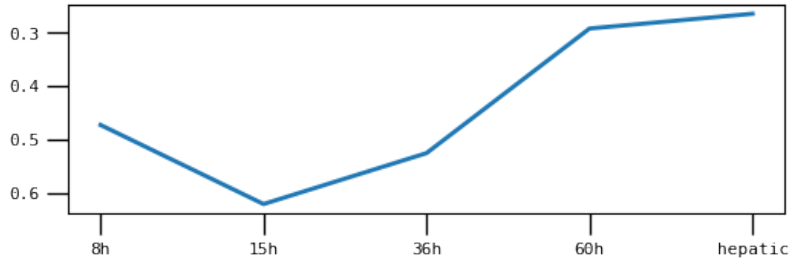
```
... ax.set_xlabel('amphioxus dev. stages')
... # ax.yaxis.tick_right()
... # ax.yaxis.set_label_position("right")
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_phyloheat.pdf')
```



Or we can plot the minimum distance per amphioxus stage, to show the hourglass behaviour

```
>>> x = [ 0, 1, 2, 3, 4]
... labels = ['8h','15h','36h','60h','hepatic']


>>> Fwidth = THESIS_PAGEWIDTH
... Fheight = Fwidth/3
...
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.075, bottom=.17, right=0.99,
...                     top=.97)
...
... # ax.set_title('ATACseq peak numbers overview')
... ax.plot( TABLE.max().values, linewidth=2)
... ax.set_xticks(x)
... ax.set_xticklabels(labels)
... ax.tick_params(direction ='out',
...                length=10, width=1, colors='black')
... ax.invert_yaxis()
...
... # ax.set_title("cis-content correlation")
... #>>> Name your Axes
... # ax.set_ylabel('zebrafish dev. stages')
... # ax.set_xlabel('amphioxus dev. stages')
... # ax.yaxis.tick_right()
```

```
... # ax.yaxis.set_label_position("right")
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_phyloline.pdf')
```



Gene modules comparisons

Weighted Gene Correlation Network Analysis (WGCNA)

Some more details can be found in our work [1], briefly:

To obtain modules of coexpressed genes across developmental stages and adult tissues, we used WGCNA . We selected 17 amphioxus and 27 zebrafish samples, including replicates of adult tissues when possible. Of 20,569 amphioxus and 20,082 zebrafish genes that had an ortholog in at least one other species used for gene family construction, 16,421 and 18,285 genes, respectively, had enough variance (CV $\geq$ 1), to be considered for further analysis. WGCNA was run with default parameters (softPower settings 9 in amphioxus and 7 in zebrafish; with unsigned networks), resulting in 25 and 23 modules in amphioxus and zebrafish, respectively. Genes in each cluster were assigned positive or negative correlation status. Each cluster was assigned a tissue affinity and/or functional category based on its overall gene expression and GO enrichment profiles (Supplementary File 1). Next, to assess the statistical significance of homolog overlap between each pair of clusters from each species, we counted the number of overlapping homologous groups (i.e. paralogs within a gene family were not counted multiple times), and performed a hypergeometric test taking as background only the number of homologous groups with members in both amphioxus and zebrafish.

---

1. Marlétaz, F., Firbas, P. N., Maeso, I., Tena, J. J., Bogdanovic, O., Perry, M., … Irimia, M. (2018). Amphioxus functional genomics and the origins of vertebrate gene regulation. Nature, 564(7734), 64–70. https://doi.org/10.1038/s41586-018-0734-6
2. Langfelder, P. & Horvath, S. WGCNA: an R package for weighted correlation network analysis. BMC Bioinformatics 9, 559 (2008).

What we will do

-

Assignment of genes to modules (WGCNA / MFUZZ)

We use WGCNA to cluster genes by their RNA activity in various TRNAseq experiments, both in developmental stages and pure tissue samples. This analysis assigns each gene in a single cluster. These clusters have names like "blue". "green" etc.We manually explore the clusters, and by determining at which experiment the genes of the cluster seem to be differentially expressed (but also

by GO analysis on each cluster), we assign more meaningful names to the clusters , like "liver", "muscle" etc. The translating dictionaries are in this notebook.

- **Grouping of genes into homology-families**

- **Assignment of genome regions to genes**

  We assign a BASAL region to each gene, it starts from its TSS and extends 5kb upstream and 1kb downstream, or untill it encounters another TSS.

- **cis-regulatory level comparisons**

  We scan the genome for a large set of motifs (see relevant notebook) and extract the motifs that are found inside each gene's basal region. By summing the counts in each module, we have counts of motifs per module. Now we can compair modules on cis-regulatory level

```
>>> ### translate the uninformative color names to manually made
...                   #informative names for each module
... ### Cleaner versions of these dictionaries at the
...                   #bottom of the notebook
... transA = {'black' : '32 cells','blue' : 'N.tube(neurotrans.)',
...           'brown' : 'Gills','cyan' : 'N.tube(neurogen.)',
...           'darkmagenta' : 'Ovary/ Testis(translation)',
...           'darkorange' : 'Cilium','darkred' : 'Muscle',
...           'darkseagreen4' : '8-36hpf embryo/ Skin/ Cirri(memb. synt.)',
...           'darkslateblue' : '8hpf embryo(transcription, spliceosome)',
...           'darkturquoise' : '36hpf embryo',
...           'green' : 'Eggs/ 32 cells(cell cycle)',
...           'greenyellow' : 'Hepatic(lipid catabolism)',
...           'lavenderblush3' : 'Cirri / PreMet. / Muscle - actomyosin',
...           'lightpink4' : '15hpf embryo','magenta' : 'Skin',
...           'navajowhite2' : 'Immune','palevioletred3' : 'Ovary/ Eggs',
...           'pink' : 'Gut','plum1' : 'Gut/ Hepatic','plum2' : 'Proteasome',
...           'red' : 'PreMet.larvae','salmon' : 'Cirri(memb. synt.)',
...           'sienna3' : 'Hepatic','thistle2' : 'Gills/ PreMet larvae',
... 'turquoise' : 'Mitochondrion'}
... transZ = {'bisque4' : "larvae 7d melanin",'black' : "Ovary/ Sperm",
...           'blue' : "Brain",
...           'brown' : "Cilium",
...           'brown4' : "Liver(carboxi.met.)", #/ lipid trans
...           'coral2' : "Mitochondrion",
...           'darkgreen' : "Intestine",
```

```
...           'darkgrey' : "RNA,ribosome,proteasome",
...           'darkmagenta' : "Translation,ribosome,RNA bin.",
...           'darkorange' : "Muscle",
...           'darkred' : "Liver(oxi-red.proc.", #/ hemostasis)
...           'darkseagreen4' : "12-26hpf embryo/ Skin",
...           'darkslateblue' : "Immune",
...           'green' : "Spliceosome",
...           'honeydew1' : "12-26hpf embryo",
...           'ivory' : "Eye",
...           'lightcyan' : "Heart",'lightgreen' : "Pancreas/ Testis",
...           'magenta' : "Gills",'pink' : "Skin",
...     'salmon' : "20-26hpf embryo",'yellow4' : "Kidney",
... 'yellowgreen' : "Gills/ Skin"}
```

Preprocessing:

```
>>> # Here we load the mapping of genes to modules from csv
... # into dictionaries:
... dan_cl = pd.read_csv(
...     "/Thesis_shallow/data/wgcna_results/wgcna_dre_modulekey.tab.gz", sep='\t')
... dan_cl.columns = ['geneID','clusterID','updown']
... dan_cl_d = dict(dan_cl[['geneID','clusterID']].to_records(index=False))
... bla_cl = pd.read_csv(
...     "/Thesis_shallow/data/wgcna_results/wgcna_bla_modulekey.tab.gz", sep='\t')
... bla_cl.columns = ['geneID','clusterID','updown']
... bla_cl_d = dict(bla_cl[['geneID','clusterID']].to_records(index=False))


>>> buz = pd.DataFrame(bla_cl.clusterID.value_counts())
... buz["alt name"] = [transA[x] for x in buz.index.values]


>>> buz.head(3)
```

	clusterID	alt name
green	1672	Eggs/ 32 cells(cell cycle)
turquoise	1287	Mitochondrion
greenyellow	1273	Hepatic(lipid catabolism)

```
>>> bla_cl.clusterID.value_counts().head()
```

```
green          1672
turquoise      1287
greenyellow    1273
darkmagenta    1168
blue           1105
Name: clusterID, dtype: int64
```

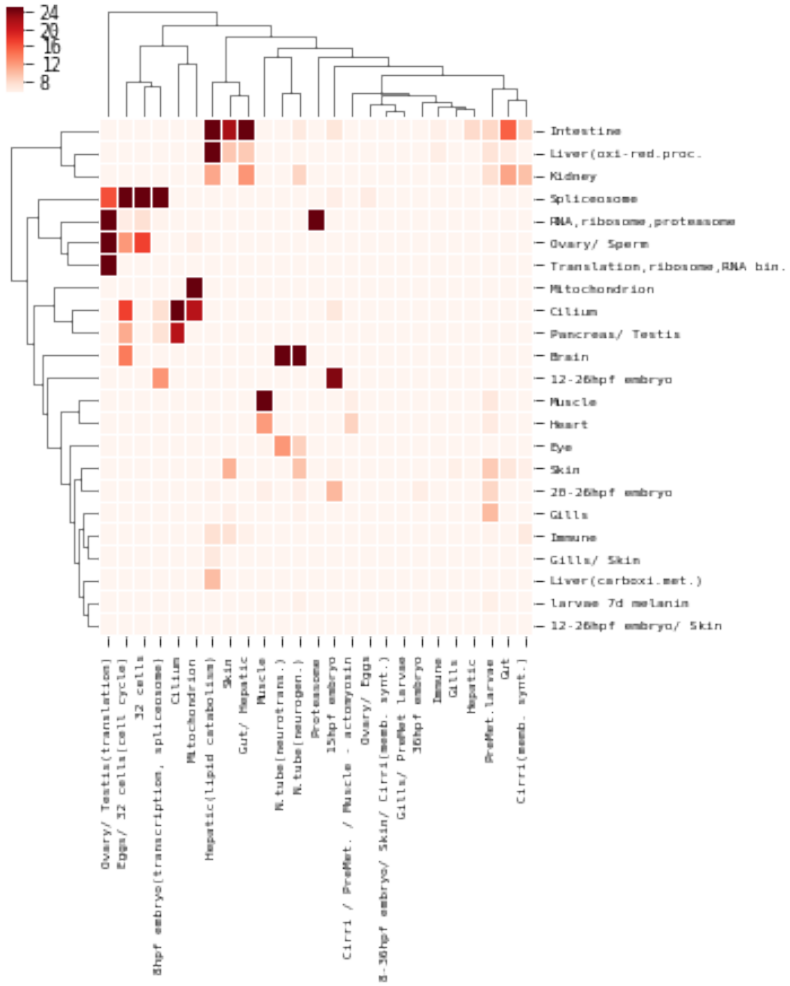
```
>>> # we load the gene family bindings into a dictionary where
... # key = geneID
... # value =  unique gene family number (some integer)
... gfams = genefams
... gfamsC = genefamsC
...
... gfamsD = {}
... for rowi, row in gfams.iterrows():
...     for species in ['Bla','Dre']:
...         thing = row[species]
...         if thing ==thing:
...             for gene in thing:
...                 gfamsD[gene] = int(rowi)


>>> # fam_of maps all genes to their gene family ID
... fam_of = {}
... for rowi,row in gfams[['Bla']].dropna().iterrows():
...     thing = row.Bla
...     if thing ==thing:
...         for gene in thing:
...             fam_of[gene]= rowi
... for rowi,row in gfams[['Dre']].dropna().iterrows():
...     for gene in row.Dre:
...         fam_of[gene]= rowi


>>> # We compute the upper tail of the hypergeometric distribution
... # (survival function) for each pair of modules as a metric of enrichment
... fon = []
... gimme_fams = lambda g: [fam_of.get(x) for x in g.geneID.tolist()]
... # Population size:
... gene_POP = len(gfams)
... PVS = pd.DataFrame()
... # for each module in zebra
... for gn,g  in dan_cl.groupby('clusterID'):
...     # for each module in amphioxus:
...     for bgn,bg  in bla_cl.groupby('clusterID'):
...
...         fams = set(gimme_fams(g))
...         bfams = set(gimme_fams(bg))
...
...         gene_SS = len(fams) # Sample Size
...         gene_SIP = len(bfams) # Success in Population
...         gene_SIS = len(bfams.intersection(fams)) # success in sample
...
...         if gene_SIS>0:
...             #statf is the survival function of the hypergeom. distribution
...             fish = statf(gene_SIS-1,  gene_POP,  gene_SS, gene_SIP)
...             fon.append( (gene_SIS,  gene_SS, gene_SIP,
...                          gene_POP, transZ[gn], transA[bgn]) )
...         else:
...             fish =1
...
...         PVS.loc[transZ[gn], transA[bgn]] = fish
...
... # the translating dictionaries are at the end of the notebook
```

```
>>> fon = pd.DataFrame(fon)
... fon.columns = ["common gene families (Success in Sample)",
...                "zebra families (Sample Size)",
...                "amphi families (Success in Polulation)",
...                "Total Families (population)",
...               "zebra cluster", "amphi cluster"]
```

The gene-based heatmap

```
>>> # we get the -log transformation of the pvalues, clip them to a workable range
... mPVS = PVS.applymap(lambda x: -log(x,10) ).copy().clip(upper=25,lower=0)
...
... # We will compute the clustering outselves in order to be able to re-use it,
... # and to have better control
... # we cluster by row and by column, based on euclidean distance
... # and complete method:
... linkage_HG_rows = hc.linkage(
...     sp.distance.squareform(
...         PWD(  mPVS.values, metric='euclidean'),
...         checks=False), method='complete')
... linkage_HG_cols = hc.linkage(
...     sp.distance.squareform(
...         PWD(  mPVS.T.values, metric='euclidean'),
...         checks=False), method='complete')
...
... # Calculating the clustering like this, allows us to use the
... # same clustering later on


>>> Fwidth = THESIS_PAGEWIDTH
... Fheight = THESIS_PAGEWIDTH*1.2
... ret = sns.clustermap( mPVS,
...                       row_linkage = linkage_HG_rows,
...                       col_linkage = linkage_HG_cols,
...                       linewidths=0.01,
...                       figsize =(12,12),
...                       cmap='Reds',
...                       vmax=25, vmin=5
...                       )
... ax = ret.ax_heatmap
...
... for tick in ax.get_xticklabels():
...     tick.set_rotation(90)
...     tick.set_fontsize(6)
... for tick in ax.get_yticklabels():
...     tick.set_rotation(0)
...     tick.set_fontsize(6)
...
... fig = ret.fig
...
... fig.subplots_adjust(left=.045, bottom=.35, right=.68, top=.98)
...
```

```
... fig.set_size_inches (Fwidth, Fheight)
... #>>> OUTPUT NAME
... fig.savefig('../Figures/from_notebooks/tfigure_WGCNAHEAT.pdf')
```



We now want to map PWMs to modules

To do this, we will use the BASAL region of genes:

```
>>> # gene to "gene region" connection
... # This dataframe connects each gene to a genomic region:
```

```
... dan_greg = gr_basal['Dre']
... dan_greg.columns = ['chrom','start','end','geneID','score','strand']
... # dan_greg = dan_greg[['chrom','start','end','geneID','score','strand']]
...
... bla_greg = gr_basal['Bla']
... bla_greg.columns = ['chrom','start','end','geneID','score','strand']
... # bla_greg = bla_greg[['chrom','start','end','geneID','score','strand']]
```

and of course the PWMs that have been mapped on ATACseq peaks already:

```
>>> # load the PWM hits in a pybedtools object and sort
... dan_motif_bed = BT(
...     "/Thesis_shallow/data/PWM/danre_pwm_hits.bed.gz").sort()
... bla_motif_bed = BT(
...     "/Thesis_shallow/data/PWM/bralan_pwm_hits.bed.gz").sort()
...
... amphi_stages = ['8','15','36','60','hep']
... zebra_stages = ['dome','shield','80epi','8som','24h','48h']
```

We will now compute the LOJ intersection of the PMs to the BASAL regions, For each gene this will give us all PWMs in its "regulatory landscape"

```
>>> # Map the cluster ID to gene IDs:
... dan_greg['cluster'] = dan_greg.geneID.map(dan_cl_d)
... dan_greg = dan_greg[~dan_greg.cluster.isnull()]
...
... bla_greg['cluster'] = bla_greg.geneID.map(bla_cl_d)
... bla_greg = bla_greg[~bla_greg.cluster.isnull()]
...
... # cast the gene region dataframes into pybedtools objects
... # and intersect with motifs (with LEFT OUTER JOIN)
... dan_loj = (BT()
...   .from_dataframe(dan_greg[['chrom','start','end','cluster','score','strand']])
...   .sort()
...   .intersect(dan_motif_bed, loj=True, nonamecheck=True, sorted=True)
...   ).to_dataframe()[['chrom','start','end','name','score','strand','blockCount']]
...
... dan_loj = dan_loj[dan_loj.blockCount != '.']
... dan_loj['fam'] = dan_loj.blockCount.map(SFDu)
... dan_loj = dan_loj[~dan_loj['fam'].isnull()]
... dan_loj.fam = dan_loj.fam.astype(int)
...
... bla_loj = (BT()
...   .from_dataframe(bla_greg[['chrom','start','end','cluster','score','strand']])
...   .sort()
...   .intersect(bla_motif_bed, loj=True, nonamecheck=True, sorted=True)
...   ).to_dataframe()[['chrom','start','end','name','score','strand','blockCount']]
...
... bla_loj = bla_loj[bla_loj.blockCount != '.']
... bla_loj['fam'] = bla_loj.blockCount.map(SFDu)
... bla_loj = bla_loj[~bla_loj['fam'].isnull()]
... dan_loj.fam = dan_loj.fam.astype(int)
... bla_loj.head()
...
... # we had
```

```
... # gene_module --> genes
... # genes --> genomic_regions
... # motifs --> genomic_positions
... # and we managed to connect them:
... # module/cluster --> genes -> genomic regions --> motifs
```

```
       chrom  start    end  name  score strand                  blockCount  \
1  Sc0000000  20509  26510  blue   6001      +         C2H2_ZF_Average_244
5  Sc0000000  20509  26510  blue   6001      +  ARID_BRIGHT_RFX_M4343_1.02
6  Sc0000000  20509  26510  blue   6001      +               RFX_Average_38
7  Sc0000000  20509  26510  blue   6001      +            bZIP_Average_125
8  Sc0000000  20509  26510  blue   6001      +          C2H2_ZF_M6539_1.02

     fam
1   11.0
5    3.0
6    3.0
7  236.0
8   21.0
```

```
>>> # convert the left outer join dataframes into
... # tables that contain a count per TF per stage:
... bla_loC = [Counter(g.fam) for gn,g in bla_loj.groupby("name")]
... bla_temp = pd.DataFrame(bla_loC)
... bla_temp = bla_temp.fillna(0)
... clustorder = [transA[gn] for gn,g in bla_loj.groupby("name")]
...
... bla_temp.index = clustorder
... bla_table = bla_temp.copy()
...
... dan_loC = [Counter(g.fam) for gn,g in dan_loj.groupby("name")]
... dan_temp = pd.DataFrame(dan_loC)
... dan_temp = dan_temp.fillna(0)
... clustorder = [transZ[gn] for gn,g in dan_loj.groupby("name")]
...
... dan_temp.index = clustorder
... dan_table = dan_temp.copy()
... dan_table.iloc[:,:3].head()
```

```
                         0     1      2
larvae 7d melanin      223   537   14.0
Ovary/ Sperm           368  2109   52.0
Brain                 1714  5518  129.0
Cilium                 531  3008   61.0
Liver(carboxi.met.)     29   107    0.0
```

There might be some differences in the columns of the two tables, lets fix that:

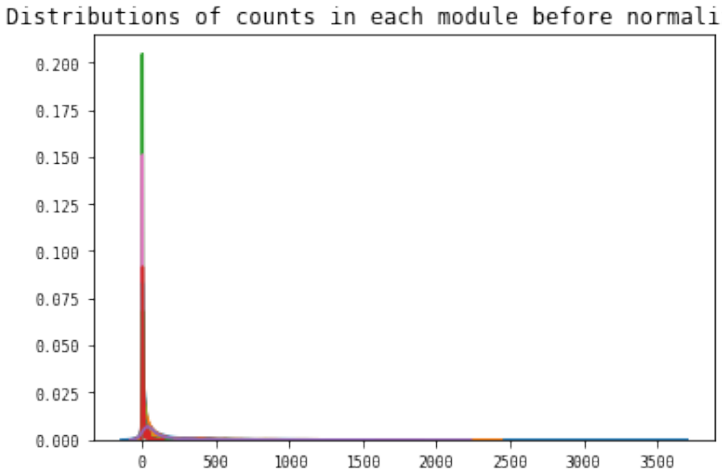
```
>>> # Get a list of the TFs that are found in both species:
```

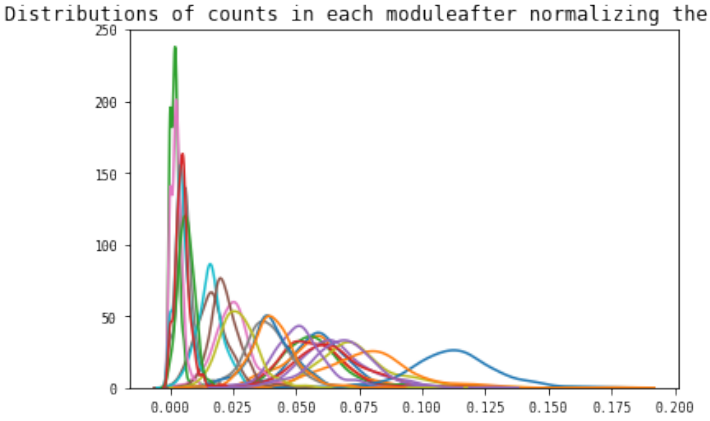
```
... dan_allfams = set(dan_table.columns)
... bla_allfams = set(bla_table.columns)
... allfams = sorted([ int(x) for x in dan_allfams
...                   .intersection(bla_allfams)])

>>> len(dan_allfams), len(bla_allfams),len(allfams)
```

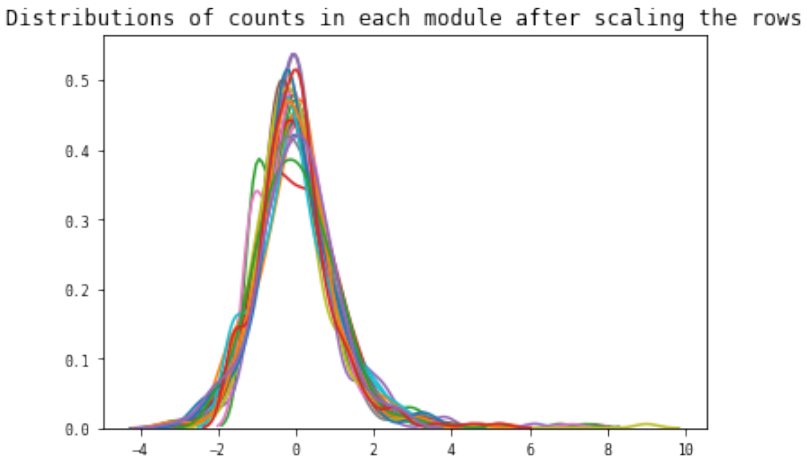
```
(243, 242, 242)
```

Normalization/Scaling

```
>>> plt.figure()
... for rowi,row in bla_table.iterrows():
...     sns.kdeplot(row.values)
... plt.title('Distributions of counts in each module before normalization')
... plt.show()
...
... # We divide each count with the sum for each TF,
... # to normalize for TF promiscuity
... bla_toplot = (bla_table
...               .loc[:,allfams]/bla_table.loc[:,allfams].sum())
...
... plt.figure()
... for rowi,row in bla_toplot.iterrows():
...     sns.kdeplot(row.values)
... plt.title('Distributions of counts in each module\
... after normalizing the columns')
... plt.show()
...
... # then scale the rows to normalize for module size
... bla_toplot.loc[:,:] = scale(bla_toplot, axis=1)
...
... plt.figure()
... for rowi,row in bla_toplot.iterrows():
...     sns.kdeplot(row.values)
... plt.title('Distributions of counts in each module after scaling the rows')
... plt.show()
...
... dan_toplot =  (dan_table
...                .loc[:,allfams]/dan_table.loc[:,allfams].sum())
... dan_toplot.loc[:,:] = scale(dan_toplot, axis=1)
```


Distributions of counts in each module before normalization
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
500
1000
1500
2000
2500
3000
3500


Distributions of counts in each moduleafter normalizing the columns
250
200
150
100
50
0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200



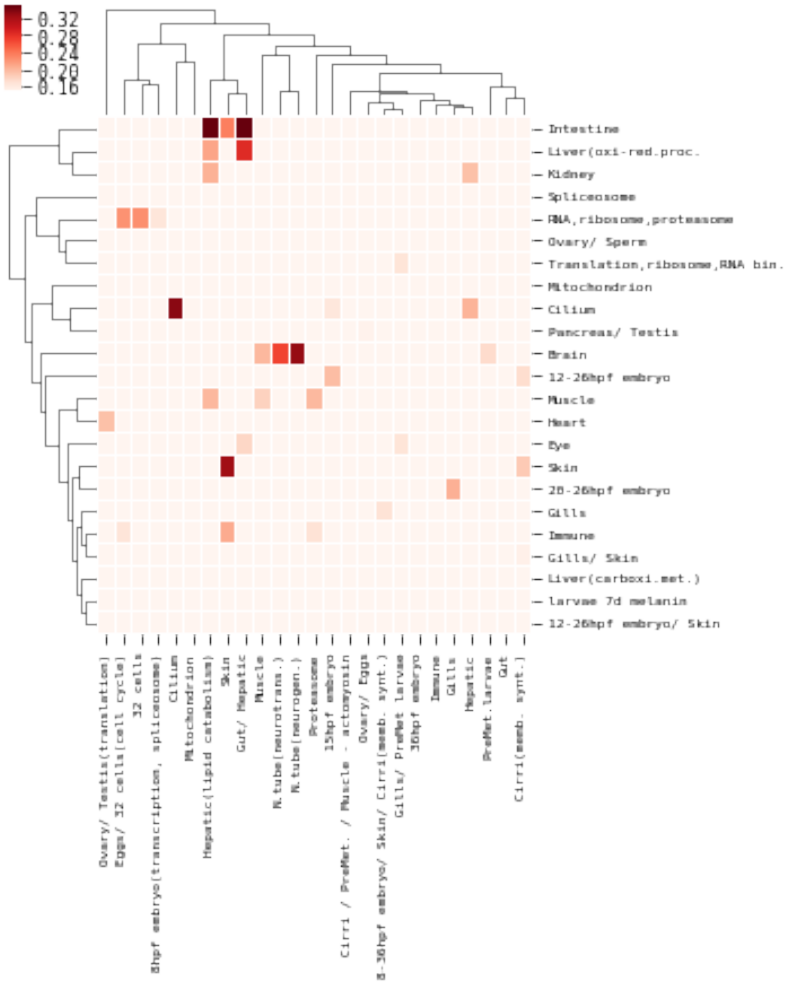
We can now directly compare modules between the two species.
We compute pairwise correlations next:

```
>>> ouf = pd.DataFrame()
...
... for dani,danrow in dan_toplot.iterrows():
...     for blai,blarow in bla_toplot.iterrows():
...         ouf.loc[dani,blai] = pearsonr(danrow.values, blarow.values)[0]


>>> ouf.iloc[:3,:3]

                    32 cells  N.tube(neurotrans.)     Gills
larvae 7d melanin -0.151380             0.052799  0.086495
Ovary/ Sperm       0.062331            -0.057291 -0.042298
Brain             -0.054350             0.270706  0.114507


>>> # Get euclidean distances for each pair of modules in a table:
... dists =  pd.DataFrame( PWD(dan_toplot,bla_toplot, metric="correlation") )
... dists.columns = bla_toplot.index
... dists.index = dan_toplot.index
...
... dists.iloc[:3,:3]

                    32 cells  N.tube(neurotrans.)     Gills
larvae 7d melanin  1.151380             0.947201  0.913505
Ovary/ Sperm       0.937669             1.057291  1.042298
Brain              1.054350             0.729294  0.885493
```

The final plot:

We can now directly visualize the table with the correlations.
We will use the clustering we got for the gene-based table earlier:

```
>>> # we can now visualize the TF-based distances
... # we'll use the clustering from the gene-based comparison
...
...
... Fwidth = THESIS_PAGEWIDTH
... Fheight = THESIS_PAGEWIDTH*1.2
... # fig, axx = plt.subplots()
...
...
... ret = sns.clustermap( 1-dists,
...
...                      # The previously computed clustering
...                      row_linkage=linkage_HG_rows,
...                      col_linkage=linkage_HG_cols,
...
...                      linewidths=0.01,
...                      figsize =(12,12),
...
...                      vmax=0.35,
...                      vmin=0.15,
...                      cmap = 'Reds'
...                     )
...
... ax = ret.ax_heatmap
...
... for tick in ax.get_xticklabels():
...     tick.set_rotation(90)
...     tick.set_fontsize(6)
... for tick in ax.get_yticklabels():
...     tick.set_rotation(0)
...     tick.set_fontsize(6)
...
... fig = ret.fig
... fig.subplots_adjust(left=.045, bottom=.35, right=.68, top=.98)
... fig.set_size_inches (Fwidth, Fheight)
... fig.savefig('../Figures/from_notebooks/tfigure_WGCNAHEAT2.pdf')
```

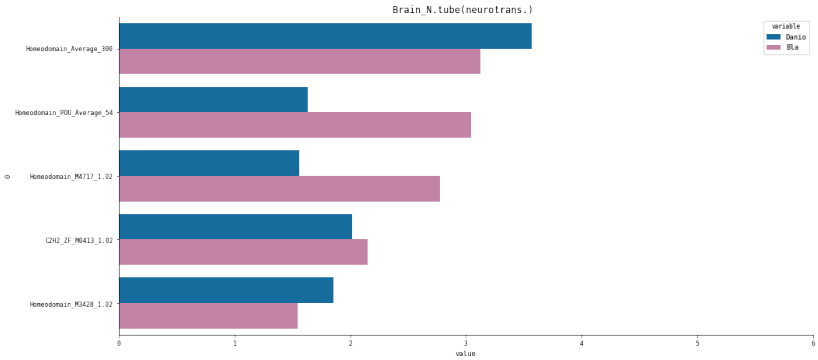


PWM z-scores plots

```
>>> def RBOscore(l1, l2, p = 0.98):
...     # https://github.com/ragrawal/measures/blob/master/measures/rankedlist/RBO.py
...
...     """
...         Calculates Ranked Biased Overlap (RBO) score.
...         l1 -- Ranked List 1
...         l2 -- Ranked List 2
...     """
... #     if l1 == None: l1 = []
```

```
... #     if l2 == None: l2 = []
...
...     sl,ll = sorted([(len(l1), l1),(len(l2),l2)], key=lambda x: x[0])
...     s, S = sl
...     l, L = ll
...     if s == 0: return 0
...
...     # Calculate the overlaps at ranks 1 through l
...     # (the longer of the two lists)
...     ss = set([]) # contains elements from the smaller list till depth i
...     ls = set([]) # contains elements from the longer list till depth i
...     x_d = {0: 0}
...     sum1 = 0.0
...     for i in range(l):
...         x = L[i]
...         y = S[i] if i < s else None
...         d = i + 1
...
...         # if two elements are same then
...         # we don't need to add to either of the set
...         if x == y:
...             x_d[d] = x_d[d-1] + 1.0
...         # else add items to respective list
...         # and calculate overlap
...         else:
...             ls.add(x)
...             if y != None: ss.add(y)
...             x_d[d] = x_d[d-1] + (1.0 if x in ss else 0.0) + (1.0 if y in ls else 0.0)
...         #calculate average overlap
...         sum1 += x_d[d]/d * pow(p, d)
...
...     sum2 = 0.0
...     for i in range(l-s):
...         d = s+i+1
...         sum2 += x_d[d]*(d-s)/(d*s)*pow(p,d)
...
...     sum3 = ((x_d[l]-x_d[s])/l+x_d[s]/s)*pow(p,l)
...
...     # Equation 32
...     rbo_ext = (1-p)/p*(sum1+sum2)+sum3
...     return rbo_ext


>>> superfams_['u'] = superfams_[0].map(SFDu)
... SF = superfams_.set_index('u')


>>> # Select some interesting cases and plot the RBOscore
... # for those TFs in both species:
... flag=0
... for drown, drow in dan_toplot.iterrows():
...     for brown, brow in bla_toplot.iterrows():
...         ld = drow.sort_values(ascending=False).index.values
...         lb = brow.sort_values(ascending=False).index.values
...         rbo = RBOscore(ld,lb, 0.85)
...         if rbo >0.25:
...
```

```
...
...             d = dan_toplot.loc[drown,:].sort_values(ascending=False)
...             dset = set(d[d>1.5].index.values)
...
...             b = bla_toplot.loc[brown,:].sort_values(ascending=False)
...             bset = set(b[b>1.5].index.values)
...
...             gset = dset.intersection(bset)
...
...             gee = SF.loc[gset].drop_duplicates(subset=2).copy()
...
...             gee['Danio'] = gee.index.to_series().map(d.loc[gset])
...             gee['Bla'] = gee.index.to_series().map(b.loc[gset])
...
...
...             tp = gee[[0,'Danio','Bla']]
...
...             plt.figure(figsize=(16,1.5*len(gset)))
...             order = tp.loc[tp.sum( axis=1)
...                            .sort_values(ascending=False)
...                            .index,0].values
...
...             sns.barplot(data=pd.melt(tp,id_vars=0),
...                         x='value',y=0, hue='variable',
...                         order=order,
...                        palette={'Danio':'#0072b2',
...                                 'Bla':'#cc79a7'})
...
...             title = "{}_{} ".format(drown, brown)
...             plt.title(title)
...             sns.despine()
...             plt.xlim((0,6))
...
...             plt.show()
...             flag=1
...             break
...         # use this to limit output when printing
...         if flag:
...             break
```



Quick Overview:

Implementation of the NACC analysis[1]

Consider a test gene *T*, with orthologues in speciesA (*sA*) and speciesB (*sB*).

We define neighborhood of gene T in *sA* (*NA*), the 20 genes with the most similar pattern of expression [2] for a set of RNAseq experiments in species A. Similarly, the neighborhood of the gene in *sB* (*NB*), is calculated based on a range of RNAseq experiments in *sB*. [3]

We can compute the average distance from T to its neighbor genes in *sA* as *DA*, and the distance in *sB* as *DB*. These two distances are computed on different sets of RNA data, so they are not comparable.

To compare the expression of the gene in the two species, we will get the orthologous genes
of *NA*, and of *NB*. This gives us orthologous Neighborhood A (*NoA*) and *NoB*. *NoA* is a list
of genes in *sB* and *NoB* is genes of *sA*.[4]

We can now compute *DoA* and *DoB*, the average distance of T to *NoA* and *NoB*. *oNA* is a list of distances computed in *sB*, so it's comparable to *NB*,
and *oNB* is comparable to *NA*.

The NACC value of *T* is then computed like so:

```
NACC = ((DoA - DB) + (DoB - DA )) / 2
```

We calculated NACC values for a subset of genes from each organism, based on their orthology connection to other species. We only used genes from homology families where there is :

- only one Amphioxus paralogue We wanted to avoid gene families that expanded before the WGDs
- at least one human paralogue but at most 4 human paralogues human was our reference species
- at least one paralogue in species 2 but at most 8 Again, to avoid gene families with too many members

For the control distributions, we applied exactly the same process, with the only difference being that we had shuffled/randomized the orthology connections.

---

1. Yue, F., Cheng, Y., Breschi, A., Vierstra, J., Wu, W., Ryba, T., … Consortium, T. M. E. (2014). A comparative encyclopedia of

DNA elements in the mouse genome. Nature, 515(7527), 355–364. https://doi.org/10.1038/nature13992

2. Based on Euclidean distance

3. To keep thigs simpler, will will only accept one gene (the one with the smallest distance) per orthologous gene family in a neighborhood.

4. Again, for simplicity, from each gene family we will take for orthologous the gene with the smallest distance to T.

```
>>> def get_results(rpkms, set1,set2, metric):
...
...     XB = rpkms.loc[set1,:].copy().fillna(0.0)
...     XA = rpkms.loc[set2,:].copy().fillna(0.0)
...
...     res = pd.DataFrame(cdist(XA,XB, metric))
...
...     res.columns = XB.index
...     res.index = XA.index
...     del XA,XB
...
...     res = res.reindex(index=list(res.columns) +\
...                       list(res.index.difference(res.columns)))
...
...     fillval = res.max().max() + 0.01
...
...     np.fill_diagonal(res.values, fillval)
...     res = res.fillna(fillval)
...
...     return res
... def yield_gene_combos(spec2):
...     # To work only with 1-1-1-1-1 genes:
...     #foo = gfams.loc[(gfamsC.Hsa==1) & (gfamsC.Mmu==1)
... #     & (gfamsC.Dre==1) & (gfamsC.Bla==1) & (gfamsC.Dme==1), ['Hsa',spec2]]
...
...     foo = gfams.loc[(gfamsC.Bla==1)
...                     & (gfamsC.Hsa>0)
...                     & (gfamsC[spec2]>0)
...                     & (gfamsC.Hsa<=4)
...                     & (gfamsC[spec2]<=8),
...                     ['Hsa',spec2]]
...     for irow,row in foo.iterrows():
...         for hsagene in row.Hsa:
...             yield hsagene, row[spec2]
... def get_sets(spec2):
...
...     foo = gfams.loc[(gfamsC.Hsa==1)
...                     & (gfamsC[spec2]==1),['Hsa',spec2]].copy()
...     foo = foo.applymap(lambda x: x[0])
...     foo = foo[~foo[spec2].duplicated(keep=False)]
...
...     assert (len(foo.Hsa)
...             == len(set(foo.Hsa.values))
...             == len(foo[spec2])
...             == len(set(foo[spec2].values)))
```

```
...
...     oto1 = foo.Hsa.values
...     oto2 = foo[spec2].values
...     foo = gfams.loc[(gfamsC.Hsa>0) & (gfamsC[spec2]>0),
...                     ['Hsa',spec2]].copy()
...     multi1 = set(reduce(add, foo.Hsa))
...     multi2 = set(reduce(add, foo[spec2]))
...
...     return oto1,oto2,multi1,multi2
...
... def load_crpkms(fp, css=None):
...     crpkms = pd.read_csv(fp, sep='\t').fillna(0)
...     crpkms.drop("NAME", axis=1, inplace=True)
...     crpkms = crpkms.set_index('ID')
...     crpkms = crpkms.applymap(lambda x: log(x+1,10))
...
...     if css:
...         crpkms = crpkms.loc[:,css]
...
...     crpkms = crpkms[crpkms.std(axis=1)>0]
...
...     return crpkms


>>> def get_naccs(spec2, fp2):
...     fp1 = "/Thesis_shallow/data/cRPKM/crpkm_hsa.tab.gz"
...     metric = 'euclidean'
...
...     acc_gfams = (gfamsC.loc[(gfamsC[spec2]>=1)
...                             & (gfamsC.Hsa>=1), : ]
...                  .index.values)
...     crpkms1 = (load_crpkms(fp1)
...     .loc[reduce(add, gfams
...                         .loc[(gfamsC[spec2]>=1)
...                              & (gfamsC.Hsa>=1), 'Hsa' ]), :]
...     .fillna(0.0))
...     crpkms2 = (load_crpkms(fp2)
...     .loc[reduce(add, gfams
...                         .loc[(gfamsC[spec2]>=1)
...                              & (gfamsC.Hsa>=1), spec2 ]), :]
...     .fillna(0.0))
...
...     print("loaded crpkms")
...     res1 = pd.DataFrame(cdist(crpkms1,crpkms1, 'euclidean'))
...     res2 = pd.DataFrame(cdist(crpkms2,crpkms2, 'euclidean'))
...     res1 = res1.set_index(crpkms1.index.values)
...     res1.columns = crpkms1.index.values
...     res2 = res2.set_index(crpkms2.index.values)
...     res2.columns = crpkms2.index.values
...
...     yol = (res2
...            .reset_index()
...            .drop_duplicates(subset='index', keep='last')
...            .set_index('index'))
...     res2 = (yol.T
...             .reset_index()
...             .drop_duplicates(subset='index', keep='last')
```

```
...             .set_index('index').T)
...
...     print("computed distances")
...
...     lon = []
...     lon_r = []
...
...     for g1,listog in yield_gene_combos(spec2):
...
...         D_sp1 = pd.DataFrame(res1.loc[g1,:])
...         D_sp1['gf'] = D_sp1.index.to_series().map(gfamsD)
...         D_sp1 = D_sp1.dropna()
...         g1fam = gfamsD[g1]
...
...
...         for g2 in listog:
...             D_sp2 = pd.DataFrame(res2.loc[g2,:])
...             D_sp2['gf'] = D_sp2.index.to_series().map(gfamsD)
...             D_sp2 = D_sp2.dropna()
...             g2fam = gfamsD[g2]
...             if g1fam!=g2fam:
...                 continue
...             ourfam = g1fam
...
...             neigb1 = (D_sp1[(D_sp1.gf!=ourfam)
...                             & (D_sp1.gf.isin(acc_gfams))]
...                         .sort_values(by=g1)
...                         .drop_duplicates(subset='gf')
...                         .iloc[:20,:])
...
...             neigb2 = (D_sp2[(D_sp2.gf!=ourfam)
...                             & (D_sp2.gf.isin(acc_gfams))]
...                         .sort_values(by=g2)
...                         .drop_duplicates(subset='gf')
...                         .iloc[:20,:])
...
...             m2 = pd.merge(
...                     neigb2,
...                     (D_sp1
...                         .loc[D_sp1.gf.isin(neigb2.gf.values)]
...                         .sort_values(by=g1)
...                         .drop_duplicates(subset='gf')
...                         .set_index('gf')
...                         .loc[neigb2.gf.values,:]
...                         .reset_index()
...                     ),
...                     on='gf')
...
...             m2R = (neigb1.loc[:,g1] -
...                    D_sp2.sample(20).loc[:,g2].values).abs()
...
...             m1 = pd.merge(
...                     neigb1,
...                     (D_sp2
...                         .loc[D_sp2.gf.isin(neigb1.gf.values)]
...                         .sort_values(by=g2)
...                         .drop_duplicates(subset='gf')
```

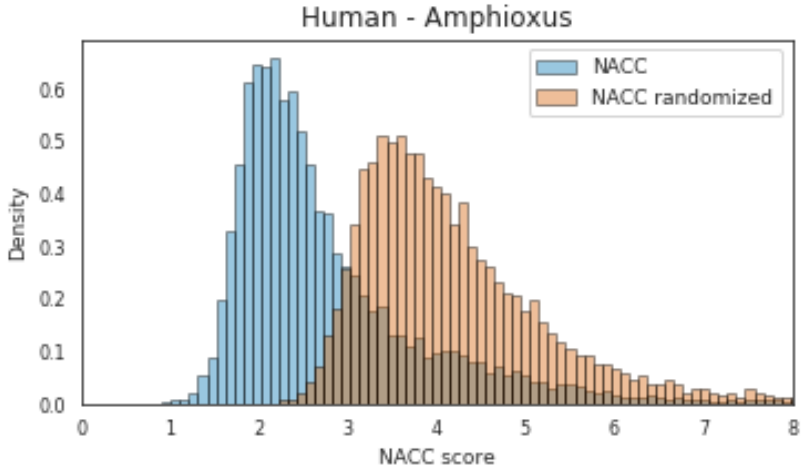
```
...                         .set_index('gf')
...                         .loc[neigb1.gf.values,:]
...                         .reset_index()
...                     ),
...                     on='gf')
...             m1R = (neigb2.loc[:,g2] -
...                    D_sp1.sample(20).loc[:,g1].values).abs()
...
...             DB = m2[g2].mean()
...             DBA = m2[g1].mean()
...             DBAR =  D_sp1.sample(20).loc[:,g1].mean()
...
...             DA = m1[g1].mean()
...             DAB = m1[g2].mean()
...             DABR = D_sp2.sample(20).loc[:,g2].mean()
...
...             NACC = ( (DAB-DA)+(DBA-DB) )/2
...             NACCR = ( (DABR-DA)+(DBAR-DB) )/2
...
...             lon.append(NACC)
...             lon_r.append(NACCR)
...
...     return lon, lon_r


>>> gfams = genefams
... gfamsC = genefamsC
... gfamsD = {}
... for rowi, row in gfams.iterrows():
...     for species in ['Bla','Dre','Mmu','Hsa','Dme']:
...         a = row[species]
...         if a==a:
...             for gene in a:
...                 gfamsD[gene] = int(rowi)


>>> # This is a bit slow:
... # I run it once and stored the results in pickles for quick loading and plotting
... # lon_bla, lonr_bla = get_naccs('Bla', "/Thesis_shallow/data/cRPKM/crpkm_bla.tab.gz")
... # lon_mmu, lonr_mmu = get_naccs('Mmu', "/Thesis_shallow/data/cRPKM/crpkm_mmu.tab.gz")
... # lon_dre, lonr_dre = get_naccs('Dre', "/Thesis_shallow/data/cRPKM/crpkm_dre.tab.gz")
```

Because the above process is a bit slow, I have included the reults in pickled objects:

```
>>> prec_lon_bla, prec_lonr_bla = (pickle
... .load(open("/Thesis_shallow/data/pickles/finalNACC_allGenes_lon_lonr_BLA.pickle",
...             "rb")))
... prec_lon_mmu, prec_lonr_mmu = (pickle
... .load(open("/Thesis_shallow/data/pickles/finalNACC_allGenes_lon_lonr_MMU.pickle",
...             "rb")))
... prec_lon_dre, prec_lonr_dre = (pickle
... .load(open("/Thesis_shallow/data/pickles/finalNACC_allGenes_lon_lonr_DRE.pickle",
...             "rb")))
```

The plots

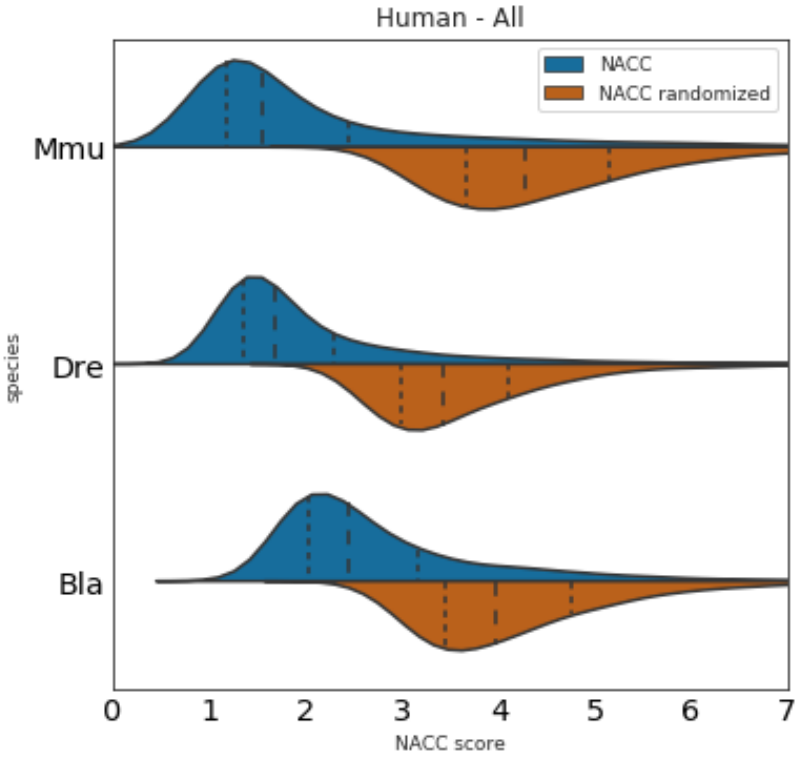
```
>>> # style for the plotting library
... sns.set_style('white')
... # the bins in which we make the histograms
... # we have to set them like this so that they
... # are the same between different datasets
... beans = np.linspace(0,10,100)


>>> # One plot, we can do the other species similarly
... Fwidth = THESIS_PAGEWIDTH
... Fheight = THESIS_PAGEWIDTH*(9/16.)
...
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.10, bottom=.13, right=.97, top=.92)
...
... sns.distplot(prec_lon_bla,
...              kde=False,
...              bins=beans,
...              norm_hist = True,
...              hist_kws={'edgecolor':'black'},
...              label="Orthologous Neighborhood Genes",
...              color=(0.003921, 0.450980, 0.698039,1),
...             ax=ax)
...
... sns.distplot(prec_lonr_bla,
...              kde=False,
...              color=(0.83529, 0.36862, 0.0,1),
...              bins=beans,
...              norm_hist = True,
...              hist_kws={'edgecolor':'black'},
...              label="Random Genes",
...              ax=ax)
...
... handles, labels = ax.get_legend_handles_labels()
... plt.legend(handles=handles, labels=['NACC','NACC randomized'])
...
... plt.xlim((0,8))
... plt.ylabel("Density")
... plt.xlabel("NACC score")
... plt.title("Human - Amphioxus")
... fig.set_size_inches (Fwidth, Fheight)
... #>>> OUTPUT NAME
... fig.savefig('../Figures/from_notebooks/tfigure_NACC1.pdf')
```



Let's merge all the data together for one single plot:

```
>>> df_mmu = pd.concat([pd.melt(pd.DataFrame(prec_lon_mmu,
...                       columns=['nacc'])),
...                     pd.melt(pd.DataFrame(prec_lonr_mmu,
...                       columns=['nacc_r']))])
... df_mmu['species'] = 'mmu'
... df_bla = pd.concat([pd.melt(pd.DataFrame(prec_lon_bla,
...                       columns=['nacc'])),
...                     pd.melt(pd.DataFrame(prec_lonr_bla,
...                       columns=['nacc_r']))])
... df_bla['species'] = 'bla'
... df_dre = pd.concat([pd.melt(pd.DataFrame(prec_lon_dre,
...                       columns=['nacc'])),
...                     pd.melt(pd.DataFrame(prec_lonr_dre,
...                       columns=['nacc_r']))])
... df_dre['species'] = 'dre'
...
... biggy = pd.concat([df_bla, df_mmu, df_dre])
...


>>> # One plot, we can do the other species similarly
... Fwidth = THESIS_PAGEWIDTH
... Fheight = THESIS_PAGEWIDTH
...
... fig, ax = plt.subplots()
... fig.subplots_adjust(left=.11, bottom=.1, right=.98, top=.94)
...
...
... sns.violinplot(data=biggy, y = 'species',
```

```
...                 hue='variable',
...                 x='value', split=True,
...                order = ['mmu','dre','bla'],
...                scale_hue=True, bw=0.2,
...                inner= "quartile",
...                palette={'nacc':(0.00392, 0.45098, 0.69803),
...                         'nacc_r':(0.83529, 0.36862, 0.0)
...                         },
...                 ax=ax
...                )
...
... ax.tick_params(direction ='in',
...                 length=10, width=1.5,
...                 colors='black',labelsize='x-large')
...
... ax.set_yticklabels(['Mmu','Dre','Bla'], rotation=0)
... handles, labels = ax.get_legend_handles_labels()
... plt.legend(handles=handles,
...            labels=['NACC','NACC randomized'])
...
... plt.xlim((0,7))
...
... # plt.ylabel("Density")
... plt.xlabel("NACC score")
... plt.title("Human - All")
... fig.set_size_inches (Fwidth, Fheight)
... #>>> OUTPUT NAME
... fig.savefig('../Figures/from_notebooks/tfigure_NACC2.pdf')
```


Human - All
NACC
NACC randomized
Mmu
Dre
Bla
species
0
1
2
3
4
5
6
7
NACC score

Part VI

Appendices

13.18 TABLES

13.19 NIMROD DATA

CTCF M1957

Species	tissue/line	time	PWMs in ATAC-peaks	False PWMs in ATAC-peaks	True PWMs in ATAC-peaks
human	gm12878	-	116127	74714	41413
human	hl60	-	68766	57015	11751
human	k562	-	28167	20275	7892
human	SUM	-	213060	20275	7892
mouse	forebrain	0 dpf	130990	88583	42407
mouse	heart	0 dpf	80803	55838	24965
mouse	hindbrain	0 dpf	100442	69499	30943
mouse	intestine	0 dpf	69254	45820	23434
mouse	kidney	0 dpf	65319	45825	19494
mouse	liver	0 dpf	111338	79152	32186
mouse	liver	14 dpf	76591	57005	19586
mouse	lung	0 dpf	134393	92074	42319
mouse	lung	14 dpf	97812	75691	22121
mouse	midbrain	0 dpf	62569	41983	20586
mouse	stomach	0 dpf	84259	54525	29734
mouse	SUM	-	1013770	705995	307775

Table 13.1: Overview of the dataset for the M1957 PWM of CTCF

P63 M2321

Species	tissue/line	time	PWMs in ATAC-peaks	False PWMs in ATAC-peaks	True PWMs in ATAC-peaks
human	keratinocytes	-	172317	121763	50554
zebrafish	embryo	24h	219800	190362	29438

Table 13.2: Overview of the dataset for the M2321 PWM of P63

H3K4me3/H3K27ac

species	stage 1	stage 2	stage 3	stage 4	stage 5
amphioxus	8h	15h	36h		60h
zebrafisg	**dome**	**shield**	**80%epiboly**	**8 somites**	**24h**

Table 13.3: The histone modication assays that we analyzed "from scratch"

ATAC-seq

species	stage 1	stage 2	stage 3	stage 4	stage 5	stage 6
amphioxus	8h	15h		36h		60h
zebrafisg	dome	shield	80%epiboly	8 somites	24h	48h
medaka	dome	shield	80%epiboly	8 somites	24h	48h
mouse	Embryonic stem cells			Definite endoderm		

Table 13.4: The ATAC-seq assays that we analyzed "from scratch"

13.20 ATAC-SEQ DATA

13.21 GENOMES

species	genome assembly file	Annotation File
amphioxus	Bl71nemr.fa	Bla_annot-FINAL_v4_names.gtf.gz
zebrafish	Danrer10.fa	Danio_rerio.GRCz10.80.gtf.gz
medaka	Orylat2.fa	Oryzias_latipes.MEDAKA1.85.gtf.gz
mouse	Mm10.fa	Mus_musculus.GRCm38.89.gtf.gz

Table 13.5: The genomes used

13.22 RNA ASSAYS

Tissue/Stage	Species	Samples ID
Eggs_G_a	Amphioxus	Eggs_G_a
Eggs_G_b	Amphioxus	Eggs_G_b
Embr_32cells_G_a	Amphioxus	Embr_32cells_G_a
Embr_32cells_G_b	Amphioxus	Embr_32cells_G_b
Embr_32cells_G_c	Amphioxus	Embr_32cells_G_c
Embr_32cells_a	Amphioxus	Embr_32cells_a
Embr_32cells_b	Amphioxus	Embr_32cells_b
Embr_Blastula_G_a	Amphioxus	Embr_Blastula_G_a
Embr_Blastula_G_b	Amphioxus	Embr_Blastula_G_b
Embr_7h_G_a	Amphioxus	Embr_7h_G_a
Embr_7h_G_b	Amphioxus	Embr_7h_G_b
Embr_8h_a	Amphioxus	Embr_8h_a
Embr_8h_b	Amphioxus	Embr_8h_b
Embr_10h_b	Amphioxus	Embr_10h_b
Embr_10h_o	Amphioxus	Embr_10h_o
Embr_11h_G_a	Amphioxus	Embr_11h_G_a
Embr_11h_G_b	Amphioxus	Embr_11h_G_b
Embr_15h_G_a	Amphioxus	Embr_15h_G_a
Embr_15h_G_b	Amphioxus	Embr_15h_G_b
Embr_15h_a	Amphioxus	Embr_15h_a
Embr_15h_b	Amphioxus	Embr_15h_b
Embr_18h_G_a	Amphioxus	Embr_18h_G_a
Embr_18h_G_b	Amphioxus	Embr_18h_G_b
Embr_24h_o	Amphioxus	Embr_24h_o
Embr_27h_G_a	Amphioxus	Embr_27h_G_a
Embr_27h_G_b	Amphioxus	Embr_27h_G_b
Embr_36h_G	Amphioxus	Embr_36h_G
Embr_36h_d	Amphioxus	Embr_36h_d
Embr_36h_e	Amphioxus	Embr_36h_e
Embr_50h_G_a	Amphioxus	Embr_50h_G_a
Embr_50h_G_b	Amphioxus	Embr_50h_G_b
Embr_60h_G_a	Amphioxus	Embr_60h_G_a
Embr_60h_G_b	Amphioxus	Embr_60h_G_b
PreMetam_G	Amphioxus	PreMetam_G
Cirri_b	Amphioxus	Cirri_b
Epidermis_b	Amphioxus	Epidermis_b
FemGonads_b	Amphioxus	FemGonads_b
Gills_b	Amphioxus	Gills_b
Gills_o	Amphioxus	Gills_o
Gut_b	Amphioxus	Gut_b
Gut_o	Amphioxus	Gut_o
Hepatic_b	Amphioxus	Hepatic_b
Hepatic_o	Amphioxus	Hepatic_o
MaleGonads_b	Amphioxus	MaleGonads_b
MaleGonads_o	Amphioxus	MaleGonads_o
Muscle_b	Amphioxus	Muscle_b
Muscle_o	Amphioxus	Muscle_o
NeuralTube_b	Amphioxus	NeuralTube_b
NeuralTube_o	Amphioxus	NeuralTube_o

Table 13.6: The Amphioxus RNA samples used for the NACC analysis

Tissue/Stage	Species	Samples ID
Eggs	Zebrafish	Egg_a
Eggs	Zebrafish	Egg_b
Eggs	Zebrafish	Egg_c
2 hpf	Zebrafish	Embr_2hpf
12 hpf	Zebrafish	Embr_12hpf
20 hpf	Zebrafish	Embr_20hpf
26 hpf	Zebrafish	Embr_26hpf
7 dpf	Zebrafish	Embr_7dpf_a
7 dpf	Zebrafish	Embr_7dpf_b
7 dpf	Zebrafish	Embr_7dpf_c
Heart	Zebrafish	Heart
Muscle rep 1	Zebrafish	Muscle_a
Muscle rep 2	Zebrafish	Muscle_b
Ovary	Zebrafish	Ovary
Pancreas_CAC	Zebrafish	Pancreas_CAC
Pancreas_nonCAC	Zebrafish	Pancreas_nonCAC
Sperm	Zebrafish	Sperm
Brain rep 1	Zebrafish	Brain_a
Brain rep 2	Zebrafish	Brain_b
Eye	Zebrafish	Eye
Gills rep 1	Zebrafish	Gills_a
Gills rep 2	Zebrafish	Gills_b
Intestine rep 1	Zebrafish	Intestine_a
Intestine rep 2	Zebrafish	Intestine_b
Liver rep 1	Zebrafish	Liver_a
Liver rep 2	Zebrafish	Liver_b
Kidney rep 1	Zebrafish	Kidney_a
Kidney rep 2	Zebrafish	Kidney_b
Skin	Zebrafish	Skin
Testis rep 1	Zebrafish	Testis_a
Testis rep 2	Zebrafish	Testis_b

Table 13.7: The Zebrafish RNA samples used for the NACC analysis

Tissue/Stage	Species	Samples ID
Oocyte	Mouse	Oocyte
Embryo 2 cells	Mouse	Embr_2C
Embryo 4 cells	Mouse	Embr_4C
Embryo 8 cells	Mouse	Embr_8C
ESC_CGR8	Mouse	ESC_CGR8
ESC_D3	Mouse	ESC_D3
ESC_E14	Mouse	ESC_E14
ESC_J1	Mouse	ESC_J1
ESC_OS25_TT2	Mouse	ESC_OS25_TT2
ESC_v65	Mouse	ESC_v65
iPS_a	Mouse	iPS_a
iPS_b	Mouse	iPS_b
PGC_E13_5	Mouse	PGC_E13_5
PGC_E9_5_11_5	Mouse	PGC_E9_5_11_5
Testis rep 1	Mouse	Testis_a
Testis rep 2	Mouse	Testis_b
Embr_Limb_E14_5	Mouse	Embr_Limb_E14_5
Embr_Liver_E14_5	Mouse	Embr_Liver_E14_5
NPC_a	Mouse	NPC_a
NPC_b	Mouse	NPC_b
Ventricular_Zone	Mouse	Ventricular_Zone
Embr_Brain_E11_5	Mouse	Embr_Brain_E11_5
Embr_Brain_E14_5	Mouse	Embr_Brain_E14_5
Embr_Cortex_E17_5	Mouse	Embr_Cortex_E17_5
Whole_Brain rep 1	Mouse	Whole_Brain_a
Whole_Brain rep 2	Mouse	Whole_Brain_b
Cortex	Mouse	Cortex
Frontal_Lobe	Mouse	Frontal_Lobe
Cerebellum	Mouse	Cerebellum
Retina_Eye	Mouse	Retina_Eye
Neurons	Mouse	Neurons_a
Myoblast_0h rep 1	Mouse	Myoblast_0h_a
Myoblast_0h rep 2	Mouse	Myoblast_0h_b
Myoblast_60h	Mouse	Myoblast_60h
Myoblast_120h	Mouse	Myoblast_120h
Myoblast_168h	Mouse	Myoblast_168h
Muscle rep 1	Mouse	Muscle_a
Muscle rep 2	Mouse	Muscle_b
Heart rep 1	Mouse	Heart_a
Heart rep 2	Mouse	Heart_b
proB_T_cells	Mouse	proB_T_cells
Trophoblast_SC	Mouse	Trophoblast_SC
Placenta rep 1	Mouse	Placenta_a
Placenta rep 2	Mouse	Placenta_b
Stomach	Mouse	Stomach
Intestine	Mouse	Intestine
Colon	Mouse	Colon
Kidney rep 1	Mouse	Kidney_a
Kidney rep 2	Mouse	Kidney_b
Liver rep 1	Mouse	Liver_a
Liver rep 2	Mouse	Liver_b
Adrenal	Mouse	Adrenal
Bladder	Mouse	Bladder
FatPad_MammGland	Mouse	FatPad_MammGland
Lung	Mouse	Lung
Ovary	Mouse	Ovary
Spleen	Mouse	Spleen
Thymus	Mouse	Thymus
MEF rep 1	Mouse	MEF_a
MEF rep 2	Mouse	MEF_b
CL_CH12	Mouse	CL_CH12
CL_MEL	Mouse	CL_MEL
CL_3T3	Mouse	CL_3T3
CL_N2A rep 1	Mouse	CL_N2A_a
CL_N2A rep 2	Mouse	CL_N2A_b

Table 13.8: The Mouse RNA samples used for the NACC analysis

Tissue/Stage	Species	Samples ID
Frontal_Gyrus_old	Human	Frontal_Gyrus_old
Frontal_Gyrus_young	Human	Frontal_Gyrus_young
Whole_Brain	Human	Whole_Brain
Sup_Temporal_Gyrus	Human	Sup_Temporal_Gyrus
Cerebellum	Human	Cerebellum
Cortex	Human	Cortex
Heart	Human	Heart
Muscle rep 1	Human	Muscle_a
Muscle rep 2	Human	Muscle_b
ESC_H1 rep 1	Human	ESC_H1_a
ESC_H1 rep 2	Human	ESC_H1_b
ESC_H1 rep 3	Human	ESC_H1_c
ESC_H1 rep 4	Human	ESC_H1_d
ESC_H9 rep 1	Human	ESC_H9_a
ESC_H9 rep 2	Human	ESC_H9_b
ESC_HES2	Human	ESC_HES2
iPS rep 1	Human	iPS_a
iPS rep 2	Human	iPS_b
Placenta_Epith	Human	Placenta_Epith
EpithelialCells	Human	EpithelialCells
HFDPC	Human	HFDPC
CL_K562	Human	CL_K562
CL_HeLa	Human	CL_HeLa
CL_293T	Human	CL_293T
CL_MB231	Human	CL_MB231
CL_PNT2	Human	CL_PNT2
CL_Gm12878	Human	CL_Gm12878
Neuroblastoma	Human	Neuroblastoma
Melanocytes	Human	Melanocytes
Fibroblasts	Human	Fibroblasts
Adipose_Breast	Human	Adipose_Breast
Adrenal	Human	Adrenal
Amnion	Human	Amnion
Chorion	Human	Chorion
Colon	Human	Colon
Decidua	Human	Decidua
Endometrium	Human	Endometrium
Endothelium	Human	Endothelium
GLS_cells	Human	GLS_cells
Kidney	Human	Kidney
Liver	Human	Liver
Lung	Human	Lung
Lymph_node	Human	Lymph_node
Ovary	Human	Ovary
Placenta	Human	Placenta
Prostate	Human	Prostate
Testis	Human	Testis
Thyroid	Human	Thyroid
WBC_MC	Human	WBC_MC
MSC	Human	MSC
NPC rep 1	Human	NPC_a
NPC rep 2	Human	NPC_b

Table 13.9: The Human RNA samples used for the NACC analysis

References

[1] Darío G Lupiáñez et al. “Disruptions of topological chromatin domains cause pathogenic rewiring of gene-enhancer interactions.” In: *Cell* 161.5 (2015), pp. 1012–1025. issn: 1097-4172. doi: 10.1016/j.cell.2015.04.004.

[2] Christophe Benoist and Pierre Chambon. “In vivo sequence requirements of the SV40 early promoter region”. In: *Nature* 290.5804 (1981), p. 304.

[3] P Gruss, R Dhar, and G Khoury. “Simian virus 40 tandem repeated sequences as an element of the early promoter”. In: *Proceedings of the National Academy of Sciences* 78.2 (1981), pp. 943–947. issn: 0027-8424. doi: 10.1073/pnas.78.2.943. url: https://www.pnas.org/content/78/2/943.

[4] Mark Mercola et al. “Transcriptional enhancer elements in the mouse immunoglobulin heavy chain locus”. In: *Science* 221.4611 (1983), pp. 663–665.

[5] Matthew Slattery et al. *Absence of a simple code: How transcription factors read the genome*. 2014. doi: 10.1016/j.tibs.2014.07.002. arXiv: NIHMS150003.

[6] Stephen C Harrison. “A structural taxonomy of DNA-binding domains”. In: *Nature* 353.6346 (1991), pp. 715–719.

[7] Juan M. Vaquerizas et al. “A census of human transcription factors: Function, expression and evolution”. In: *Nature Reviews Genetics* 10.4 (2009), pp. 252–263. issn: 14710056. doi: 10.1038/nrg2538.

[8] A. de Mendoza et al. “Transcription factor evolution in eukaryotes and the assembly of the regulatory toolkit in multicellular lineages”. In: *Proceedings of the National Academy of Sciences* 110.50 (2013), E4858–E4866. issn: 0027-8424. doi: 10.1073/pnas.1311818110.

[9] Rohit Joshi et al. “Functional Specificity of a Hox Protein Mediated by the Recognition of Minor Groove Structure”. In: *Cell* 131.3 (2007), pp. 530–543. doi: 10.1016/J.CELL.2007.09.024.

[10] Giovanni Marsico et al. “Whole genome experimental maps of DNA G-quadruplexes in multiple species”. In: *Nucleic Acids Research* 47.8 (2019), pp. 3862–3874. issn: 0305-1048. doi: 10.1093/nar/gkz179.

[11] G. D. Stormo. “DNA binding sites: representation and discovery”. In: *Bioinformatics* 16.1 (2000), pp. 16–23. doi: 10.1093/bioinformatics/16.1.16.

[12] Harmen J. Bussemaker, Barrett C. Foat, and Lucas D. Ward. “Predictive Modeling of Genome-Wide mRNA Expression: From Modules to Molecules”. In: *Annual Review of Biophysics and Biomolecular Structure* 36.1 (2007), pp. 329–347. issn: 1056-8700. doi: 10.1146/annurev.biophys.36.040306.132725.

[13] Gwenael Badis et al. “Diversity and complexity in DNA recognition by transcription factors.” In: *Science (New York, N.Y.)* 324.5935 (2009), pp. 1720–3. issn: 1095-9203. doi: 10.1126/science.1162327.

[14] Gary D Stormo and Yue Zhao. “Determining the specificity of protein–DNA interactions”. In: *Nature Reviews Genetics* 11.11 (2010), p. 751.

[15] Matthew T Weirauch et al. “Evaluation of methods for modeling transcription factor sequence specificity.” In: *Nature biotechnology* 31.2 (2013), pp. 126–34. issn: 1546-1696. doi: 10.1038/nbt.2486.

[16] Arttu Jolma et al. “DNA-binding specificities of human transcription factors.” In: *Cell* 152.1-2 (2013), pp. 327–39. issn: 1097-4172. doi: 10.1016/j.cell.2012.12.009.

[17] Michael A White et al. “Massively parallel in vivo enhancer assay reveals that highly local features determine the cis-regulatory function of ChIP-seq peaks.” In: *Proceedings of the National Academy of Sciences of the United States of America* 110.29 (2013), pp. 11952–7. issn: 1091-6490. doi: 10.1073/pnas.1307449110.

[18] Remo Rohs et al. “The role of DNA shape in protein–DNA recognition”. In: *Nature* 461.7268 (2009), pp. 1248–1253. doi: 10.1038/nature08473.

[19] Youngchang Kim et al. "Crystal structure of a yeast TBP/TATA-box complex". In: *Nature* 365.6446 (1993), pp. 512–520. issn: 0028-0836. doi: 10.1038/365512a0.

[20] Joseph L. Kim, Dimitar B. Nikolov, and Stephen K. Burley. "Co-crystal structure of TBP recognizing the minor groove of a TATA element". In: *Nature* 365.6446 (1993), pp. 520–527. issn: 0028-0836. doi: 10.1038/365520a0.

[21] Trevor Siggers et al. "Non-DNA-binding cofactors enhance DNA-binding specificity of a transcriptional regulatory complex." In: *Molecular systems biology* 7 (2011), p. 555. issn: 1744-4292. doi: 10.1038/msb.2011.89.

[22] Matthew Slattery et al. "No Title". In: *Cell* 147.6 (2011). issn: 1097-4172.

[23] Daniel Panne. "The enhanceosome". In: *Current Opinion in Structural Biology* 18.2 (2008), pp. 236–242. issn: 0959-440X. doi: 10.1016/J.SBI.2007.12.002.

[24] T. Siggers and R. Gordan. "Protein-DNA binding: complexities and multi-protein codes". In: *Nucleic Acids Research* 42.4 (2014), pp. 2099–2111. issn: 0305-1048. doi: 10.1093/nar/gkt1112.

[25] Xiao Liu et al. "Whole-genome comparison of Leu3 binding in vitro and in vivo reveals the importance of nucleosome occupancy in target site selection." In: *Genome research* 16.12 (2006), pp. 1517–28. issn: 1088-9051. doi: 10.1101/gr.5655606.

[26] Lu Bai and Alexandre V Morozov. "Gene regulation by nucleosome positioning." In: *Trends in genetics : TIG* 26.11 (2010), pp. 476–83. issn: 0168-9525. doi: 10.1016/j.tig.2010.08.003.

[27] Tommy Kaplan et al. "Quantitative models of the mechanisms that control genome-wide patterns of transcription factor binding during early Drosophila development." In: *PLoS genetics* 7.2 (2011), e1001290. issn: 1553-7404. doi: 10.1371/journal.pgen.1001290.

[28] K.J. Polach and J. Widom. "A Model for the Cooperative Binding of Eukaryotic Regulatory Proteins to Nucleosomal Target Sites". In: *Journal of Molecular Biology* 258.5 (1996), pp. 800–812. issn: 00222836. doi: 10.1006/jmbi.1996.0288.

[29] Iros Barozzi et al. "Coregulation of transcription factor binding and nucleosome occupancy through DNA features of mammalian enhancers." In: *Molecular cell* 54.5 (2014), pp. 844–857. issn: 1097-4164. doi: 10.1016/j.molcel.2014.04.006.

[30] Sebastian Glatt, Claudio Alfieri, and Christoph W Müller. "Recognizing and remodeling the nucleosome". In: *Current Opinion in Structural Biology* 21.3 (2011), pp. 335–341. issn: 0959440X. doi: 10.1016/j.sbi.2011.02.003.

[31] Kenneth S Zaret and Jason S Carroll. "Pioneer transcription factors: establishing competence for gene expression." In: *Genes & development* 25.21 (2011), pp. 2227–41. issn: 1549-5477. doi: 10.1101/gad.176826.111.

[32] Richard I Sherwood et al. "Discovery of directional and nondirectional pioneer transcription factors by modeling DNase profile magnitude and shape". In: *Nature Biotechnology* 32.2 (2014), pp. 171–178. issn: 1087-0156. doi: 10.1038/nbt.2798.

[33] Silvia Domcke et al. "Competition between DNA methylation and transcription factors determines binding of NRF1". In: *Nature* 528.7583 (2015), pp. 575–579. issn: 0028-0836. doi: 10.1038/nature16462.

[34] Jean-Michel Claverie and Stéphane Audic. "The statistical significance of nucleotide position-weight matrix matches". In: *Bioinformatics* 12.5 (1996), pp. 431–439. doi: 10.1093/bioinformatics/12.5.431.

[35] Anthony Mathelier and Wyeth W. Wasserman. "The Next Generation of Transcription Factor Binding Site Prediction". In: *PLoS Computational Biology* 9.9 (2013). Ed. by Ilya Ioshikhes, e1003214. issn: 1553-7358. doi: 10.1371/journal.pcbi.1003214.

[36] Eilon Sharon, Shai Lubliner, and Eran Segal. "A Feature-Based Approach to Modeling Protein–DNA Interactions". In: *PLoS Computational Biology* 4.8 (2008). Ed. by Gary Stormo, e1000154. issn: 1553-7358. doi: 10.1371/journal.pcbi.1000154.

[37] N. I. Gershenzon, Gary D. Stormo, and Ilya P. Ioshikhes. "Computational technique for improvement of the position-weight matrices for the DNA/protein binding sites". In: *Nucleic Acids Research* 33.7 (2005), pp. 2290–2301. issn: 1362-4962. doi: 10.1093/nar/gki519.

[38] Rahul Siddharthan. "Dinucleotide Weight Matrices for Predicting Transcription Factor Binding Sites: Generalizing the Position Weight Matrix". In: *PLoS ONE* 5.3 (2010). Ed. by Raya Khanin, e9722. issn: 1932-6203. doi: 10.1371/journal.pone.0009722.

[39] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*. 2012.

[40] Gary D. Stormo et al. "Use of the 'Perceptron' algorithm to distinguish translational initiation sites in E. coli". In: *Nucleic Acids Research* 10.9 (1982), pp. 2997–3011. issn: 0305-1048. doi: 10.1093/nar/10.9.2997.

[41] Jack Lanchantin et al. "Deep Motif: Visualizing Genomic Sequence Classifications". In: (2016). arXiv: 1605.01133.

[42] Babak Alipanahi et al. "Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning". In: *Nature Biotechnology* 33.8 (2015), pp. 831–838. issn: 1087-0156. doi: 10.1038/nbt.3300.

[43] Alyssa Morrow et al. "Convolutional Kitchen Sinks for Transcription Factor Binding Site Prediction". In: (2017). arXiv: 1706.00125.

[44] Daniel Quang and Xiaohui Xie. "DanQ: a hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences." In: *Nucleic acids research* 44.11 (2016), e107. issn: 1362-4962. doi: 10.1093/nar/gkw226.

[45] Jian Zhou and Olga G Troyanskaya. "Predicting effects of noncoding variants with deep learning–based sequence model". In: *Nature Methods* 12.10 (2015). issn: 1548-7091. doi: 10.1038/nmeth.3547.

[46] Qian Qin and Jianxing Feng. "Imputation for transcription factor binding predictions based on deep learning". In: *PLOS Computational Biology* 13.2 (2017). Ed. by Ilya Ioshikhes, e1005403. issn: 1553-7358. doi: 10.1371/journal.pcbi.1005403.

[47] Alan P Boyle et al. "High-resolution mapping and characterization of open chromatin across the genome." In: *Cell* 132.2 (2008), pp. 311–22. issn: 1097-4172. doi: 10.1016/j.cell.2007.12.014.

[48] Jason D Buenrostro et al. "Transposition of native chromatin for fast and sensitive epigenomic profiling of open chromatin, DNA-binding proteins and nucleosome position". In: *Nature Methods* 10.12 (2013), pp. 1213–1218. issn: 1548-7091. doi: 10.1038/nmeth.2688.

[49] Jason Piper et al. "Wellington: a novel method for the accurate identification of digital genomic footprints from DNase-seq data." In: *Nucleic acids research* 41.21 (2013), e201. issn: 1362-4962. doi: 10.1093/nar/gkt850.

[50] R. Pique-Regi et al. "Accurate inference of transcription factor binding from DNA sequence and chromatin accessibility data". In: *Genome Research* 21.3 (2011), pp. 447–455. issn: 1088-9051. doi: 10.1101/gr.112623.110.

[51] Anil Raj et al. "msCentipede: Modeling Heterogeneity across Genomic Sites and Replicates Improves Accuracy in the Inference of Transcription Factor Binding". In: *PLOS ONE* 10.9 (2015). Ed. by Deyou Zheng, e0138030. issn: 1932-6203. doi: 10.1371/journal.pone.0138030.

[52] Aleksander Jankowski, Jerzy Tiuryn, and Shyam Prabhakar. "Romulus: robust multi-state identification of transcription factor binding sites from DNase-seq data." In: *Bioinformatics (Oxford, England)* 32.16 (2016), pp. 2419–26. issn: 1367-4811. doi: 10.1093/bioinformatics/btw209.

[53] Richard I Sherwood et al. "Discovery of directional and nondirectional pioneer transcription factors by modeling DNase profile magnitude and shape." In: *Nature biotechnology* 32.2 (2014), pp. 171–178. issn: 1546-1696. doi: 10.1038/nbt.2798.

[54] Ignacio Maeso and Juan J. Tena. "Favorable genomic environments for cis- regulatory evolution: A novel theoretical framework". In: *Seminars in Cell & Developmental Biology* 57 (2016), pp. 2–10. issn: 10849521. doi: 10.1016/j.semcdb.2015.12.003.

[55] Sean B. Carroll. "Evo-Devo and an Expanding Evolutionary Synthesis: A Genetic Theory of Morphological Evolution". In: *Cell* 134.1 (2008), pp. 25–36. issn: 00928674. doi: 10.1016/j.cell.2008.06.030.

[56] Hopi E. Hoekstra and Jerry A. Coyne. "The locus of evolution: evo devo and the genetics of adaptation." In: *Evolution* 61.5 (2007), pp. 995–1016. issn: 00143820. doi: 10.1111/j.1558-5646.2007.00105.x.

[57] Menno P Creyghton et al. "Histone H3K27ac separates active from poised enhancers and predicts developmental state." In: *Proceedings of the National Academy of Sciences of the United States of America* 107.50 (2010), pp. 21931–6. issn: 1091-6490. doi: 10.1073/pnas.1016071107.

[58] Alvaro Rada-Iglesias et al. "A unique chromatin signature uncovers early developmental enhancers in humans." In: *Nature* 470.7333 (2011), pp. 279–83. issn: 1476-4687. doi: 10.1038/nature09692.

[59] Tae-Kyung Kim and Ramin Shiekhattar. "Architectural and Functional Commonalities between Enhancers and Promoters". In: *Cell* 162.5 (2015), pp. 948–959. issn: 0092-8674. doi: 10.1016/J.CELL.2015.08.008.

[60] Robin Andersson. "Promoter or enhancer, what's the difference? Deconstruction of established distinctions and presentation of a unifying model". In: *BioEssays* 37.3 (2015), pp. 314–323. issn: 02659247. doi: 10.1002/bies.201400162.

[61] Boris Lenhard, Albin Sandelin, and Piero Carninci. "Metazoan promoters: emerging characteristics and insights into transcriptional regulation". In: *Nature Reviews Genetics* 13.4 (2012), pp. 233–245. issn: 1471-0056. doi: 10.1038/nrg3163.

[62] Yi Jin et al. "The Ground State and Evolution of Promoter Region Directionality." In: *Cell* 170.5 (2017), 889–898.e10. issn: 1097-4172. doi: 10.1016/j.cell.2017.07.006.

[63] Piero Carninci et al. "Genome-wide analysis of mammalian promoter architecture and evolution". In: *Nature Genetics* 38.6 (2006), pp. 626–635. issn: 10614036. doi: 10.1038/ng1789.

[64] Lorenzo Pasquali et al. "Pancreatic islet enhancer clusters enriched in type 2 diabetes risk-associated variants". In: *Nature genetics* 46.2 (2014), p. 136.

[65] Jaret M Karnuta and Peter C Scacheri. "Enhancers: bridging the gap between gene control and human disease". In: *Human Molecular Genetics* 27.R2 (2018), R219–R227. issn: 0964-6906. doi: 10.1093/hmg/ddy167.

[66] Jae-Seok Roe et al. "Enhancer Reprogramming Promotes Pancreatic Cancer Metastasis". In: *Cell* 170.5 (2017), 875–888.e20. issn: 00928674. doi: 10.1016/j.cell.2017.07.007.

[67] Lei Xiong et al. "Aberrant enhancer hypomethylation contributes to hepatic carcinogenesis through global transcriptional reprogramming." In: *Nature communications* 10.1 (2019), p. 335. issn: 2041-1723. doi: 10.1038/s41467-018-08245-z.

[68] Stein Aerts. "Computational Strategies for the Genome-Wide Identification of cis-Regulatory Elements and Transcriptional Targets". In: *Current Topics in Developmental Biology* 98 (2012), pp. 121–145. issn: 0070-2153. doi: 10.1016/B978-0-12-386499-4.00005-7.

[69] Benoit Ballester et al. "Multi-species, multi-transcription factor binding highlights conserved control of tissue-specific biological pathways." In: *eLife* 3 (2014), e02626. issn: 2050-084X. doi: 10.7554/eLife.02626.

[70] Evgeny Z. Kvon. "Using transgenic reporter assays to functionally characterize enhancers in animals". In: *Genomics* 106.3 (2015), pp. 185–192. issn: 0888-7543. doi: 10.1016/J.YGENO.2015.06.007.

[71] Eduardo G Gusmao et al. "Analysis of computational footprinting methods for DNase sequencing experiments". In: *Nature Methods* 13.4 (2016), pp. 303–309. issn: 1548-7091. doi: 10.1038/nmeth.3772.

[72] Jurg Stalder et al. "Tissue-specific DNA cleavages in the globin chromatin domain introduced by DNAase I". In: *Cell* 20.2 (1980), pp. 451–460. issn: 0092-8674. doi: 10.1016/0092-8674(80)90631-5.

[73] Paul G Giresi et al. "FAIRE (Formaldehyde-Assisted Isolation of Regulatory Elements) isolates active regulatory elements from human chromatin." In: *Genome research* 17.6 (2007), pp. 877–85. issn: 1088-9051. doi: 10.1101/gr.5533506.

[74] Ramy K Aziz, Mya Breitbart, and Robert A Edwards. "Transposases are the most abundant, most ubiquitous genes in nature." In: *Nucleic acids research* 38.13 (2010), pp. 4207–17. issn: 1362-4962. doi: 10.1093/nar/gkq140.

[75] Andrew Adey et al. "Rapid, low-input, low-bias construction of shotgun fragment libraries by high-density in vitro transposition". In: *Genome Biology* 11.12 (2010), R119. issn: 1465-6906. doi: 10.1186/gb-2010-11-12-r119.

[76] William S. Reznikoff. "Tn5 as a model for understanding DNA transposition". In: *Molecular Microbiology* 47.5 (2003), pp. 1199–1206. issn: 0950382X. doi: 10.1046/j.1365-2958.2003.03382.x.

[77] Alicia N Schep et al. "Structured nucleosome fingerprints enable high-resolution mapping of chromatin architecture within regulatory regions." In: *Genome research* 25.11 (2015), pp. 1757–70. issn: 1549-5469. doi: 10.1101/gr.192294.115.

[78] Gary Felsenfeld. "A brief history of epigenetics." In: *Cold Spring Harbor perspectives in biology* 6.1 (2014). issn: 1943-0264. doi: 10.1101/cshperspect.a018200.

[79] T. H. Morgan. "An attempt to analyze the constitution of the chromosomes on the basis of sex-limited inheritance in Drosophila". In: *Journal of Experimental Zoology* 11.4 (1911), pp. 365–413. issn: 0022-104X. doi: 10.1002/jez.1400110404.

[80] R A Laskey and J B Gurdon. "Genetic content of adult somatic cells tested by nuclear transplantation from cultured cells." In: *Nature* 228.5278 (1970), pp. 1332–4. issn: 0028-0836.

[81] V.E.A. Russo, R.A. Martienssen, and A.D. Riggs. *Epigenetic Mechanisms of Gene Regulation*. Cold Spring Harbor monograph series. Cold Spring Harbor Laboratory Press, 1996. isbn: 9780879694906. url: https://books.google.es/books?id=clnwAAAAMAAJ.

[82] Bilian Jin, Yajun Li, and Keith D Robertson. "DNA methylation: superior or subordinate in the epigenetic hierarchy?" In: *Genes & cancer* 2.6 (2011), pp. 607–17. issn: 1947-6027. doi: 10.1177/1947601910393957.

[83] Ozren Bogdanović and Gert Jan C. Veenstra. "DNA methylation and methyl-CpG binding proteins: developmental requirements and function". In: *Chromosoma* 118.5 (2009), pp. 549–565. issn: 0009-5915. doi: 10.1007/s00412-009-0221-9.

[84] K Luger et al. "Crystal structure of the nucleosome core particle at 2.8 A resolution". In: *Nature* 389.6648 (1997), pp. 251–260.

[85] Peter N. Cockerill. "Structure and function of active chromatin and DNase I hypersensitive sites". In: *FEBS Journal* 278.13 (2011), pp. 2182–2210. issn: 1742464X. doi: 10.1111/j.1742-4658.2011.08128.x.

[86] V G Allfrey, R Faulkner, and A E Mirsky. "Acetylation and methylation of histones and their possible role in the regulation of RNA synthesis." In: *Proceedings of the National Academy of Sciences of the United States of America* 51 (1964), pp. 786–94. issn: 0027-8424.

[87] Tony Kouzarides. "Chromatin modifications and their function." In: *Cell* 128.4 (2007), pp. 693–705. issn: 0092-8674. doi: 10.1016/j.cell.2007.02.005.

[88] Adam F. Kebede, Robert Schneider, and Sylvain Daujat. "Novel types and sites of histone modifications emerge as players in the transcriptional regulation contest". In: *FEBS Journal* 282.9 (2015), pp. 1658–1674. issn: 1742464X. doi: 10.1111/febs.13047.

[89] Karolin Luger, Mekonnen L Dechassa, and David J Tremethick. "New insights into nucleosome and chromatin structure: an ordered state or a disordered affair?" In: *Nature reviews. Molecular cell biology* 13.7 (2012), pp. 436–47. issn: 1471-0080. doi: 10.1038/nrm3382.

[90] Yupeng Zheng, Paul M. Thomas, and Neil L. Kelleher. "Measurement of acetylation turnover at distinct lysines in human histones identifies long-lived acetylation sites". In: *Nature Communications* 4.1 (2013), p. 2203. issn: 2041-1723. doi: 10.1038/ncomms3203.

[91] Cristina Cruz et al. "Tri-methylation of histone H3 lysine 4 facilitates gene expression in ageing cells". In: *Elife* 7 (2018), e34081.

[92] T. Miller et al. "COMPASS: A complex of proteins associated with a trithorax-related SET domain protein". In: *Proceedings of the National Academy of Sciences* 98.23 (2001), pp. 12902–12907. issn: 0027-8424. doi: 10.1073/pnas.231473398.

[93] Robert Schneider et al. "Histone H3 lysine 4 methylation patterns in higher eukaryotic genes". In: *Nature Cell Biology* 6.1 (2004), pp. 73–77. issn: 1465-7392. doi: 10.1038/ncb1076.

[94] K. Nishioka et al. "Set9, a novel histone H3 methyltransferase that facilitates transcription by precluding histone tail modifications required for heterochromatin formation". In: *Genes & Development* 16.4 (2002), pp. 479–489. issn: 08909369. doi: 10.1101/gad.967202.

[95] Matthew G. Guenther et al. "A Chromatin Landmark and Transcription Initiation at Most Promoters in Human Cells". In: *Cell* 130.1 (2007), pp. 77–88. issn: 0092-8674. doi: 10.1016/J.CELL.2007.05.042.

[96] Aaron W. Aday et al. "Identification of cis regulatory features in the embryonic zebrafish genome through large-scale profiling of H3K4me1 and H3K4me3 binding sites". In: *Developmental Biology* 357.2 (2011), pp. 450–462. issn: 0012-1606. doi: 10.1016/J.YDBIO.2011.03.007.

[97] Ozren Bogdanovic et al. "Dynamics of enhancer chromatin signatures mark the transition from pluripotency to cell specification during embryogenesis." In: *Genome research* 22.10 (2012), pp. 2043–2053. issn: 1549-5469. doi: 10.1101/gr.134833.111.

[98] Noriyuki Suka et al. "Highly Specific Antibodies Determine Histone Acetylation Site Usage in Yeast Heterochromatin and Euchromatin". In: *Molecular Cell* 8.2 (2001), pp. 473–479. issn: 1097-2765. doi: 10.1016/S1097-2765(01)00301-X.

[99] Benjamin A Garcia et al. "Organismal differences in post-translational modifications in histones H3 and H4." In: *The Journal of biological chemistry* 282.10 (2007), pp. 7641–55. issn: 0021-9258. doi: 10.1074/jbc.M607900200.

[100] Zhibin Wang et al. "Combinatorial patterns of histone acetylations and methylations in the human genome." In: *Nature genetics* 40.7 (2008), pp. 897–903. issn: 1546-1718. doi: 10.1038/ng.154.

[101] Stefan Bonn et al. "Tissue-specific analysis of chromatin state identifies temporal signatures of enhancer activity during embryonic development". In: *Nature Genetics* 44.2 (2012), pp. 148–156. issn: 1061-4036. doi: 10.1038/ng.1064.

[102] E. W. Gudger. "The Five Great Naturalists of the Sixteenth Century: Belon, Rondelet, Salviani, Gesner and Aldrovandi: A Chapter in the History of Ichthyology". In: *Isis* 22.1 (1934), p. 21. issn: 0021-1753. doi: 10.1086/346870.

[103] A L Panchen. "Homology–history of a concept." eng. In: *Novartis Foundation symposium* 222 (1999), pp. 5–23. issn: 1528-2511 (Print).

[104] W M Fitch. "Distinguishing homologous from analogous proteins." In: *Systematic zoology* 19.2 (1970), pp. 99–113. issn: 0039-7989.

[105] Frédéric Delsuc et al. "Tunicates and not cephalochordates are the closest living relatives of vertebrates". In: *Nature* 439.7079 (2006), p. 965.

[106] Takeshi Kon et al. "Phylogenetic position of a whale-fall lancelet (Cephalochordata) inferred from whole mitochondrial genome sequences". In: *BMC Evolutionary Biology* 7.1 (2007), p. 127.

[107] Michael Schubert et al. "Amphioxus and tunicates as evolutionary model systems". In: *Trends in Ecology & Evolution* 21.5 (2006), pp. 269–277. issn: 0169-5347. doi: 10.1016/J.TREE.2006.01.009.

[108] Stephanie Bertrand et al. "Evolutionary crossroads in developmental biology: amphioxus." In: *Development (Cambridge, England)* 138.22 (2011), pp. 4819–30. issn: 1477-9129. doi: 10.1242/dev.066720.

[109] Eric S. Lander et al. "Initial sequencing and analysis of the human genome". In: *Nature* 409.6822 (2001), pp. 860–921. issn: 00280836. doi: 10.1038/35057062. arXiv: 11237011.

[110] Craig E Nelson, Bradley M Hersh, and Sean B Carroll. "The regulatory content of intergenic DNA shapes genome architecture". In: *Genome Biology* 5.4 (2004), R25. issn: 1474-760X. doi: 10.1186/gb-2004-5-4-r25.

[111] Ozren Bogdanović et al. "Active DNA demethylation at enhancers during the vertebrate phylotypic period". In: *Nature Genetics* 48.4 (2016), pp. 417–426. issn: 1061-4036. doi: 10.1038/ng.3522.

[112] Linda Z Holland et al. "The amphioxus genome illuminates vertebrate origins and cephalochordate biology." In: *Genome research* 18.7 (2008), pp. 1100–11. issn: 1088-9051. doi: 10.1101/gr.073676.107.

[113] Marie Sémon and Kenneth H. Wolfe. "Consequences of genome duplication". In: *Current Opinion in Genetics and Development* 17.6 (2007), pp. 505–512. issn: 0959437X. doi: 10.1016/j.gde.2007.09.007.

[114] Frédéric Delsuc et al. "Tunicates and not cephalochordates are the closest living relatives of vertebrates". In: *Nature* 439.7079 (2006), pp. 965–968. issn: 0028-0836. doi: 10.1038/nature04336.

[115] Adriana Canapa et al. "Transposons, Genome Size, and Evolutionary Insights in Animals". In: *Cytogenetic and Genome Research* 147.4 (2015), pp. 217–239. issn: 1424-8581. doi: 10.1159/000444429.

[116] Philip C.J. Donoghue and Mark A. Purnell. "Genome duplication, extinction and vertebrate evolution". In: *Trends in Ecology & Evolution* 20.6 (2005), pp. 312–319. issn: 0169-5347. doi: 10.1016/J.TREE.2005.04.008.

[117] John T. Clarke, Graeme T. Lloyd, and Matt Friedman. "Little evidence for enhanced phenotypic evolution in early teleosts relative to their living fossil sister group". In: *Proceedings of the National Academy of Sciences* 113.41 (2016), pp. 11531–11536. issn: 0027-8424. doi: 10.1073/PNAS.1607237113.

[118] Paramvir Dehal and Jeffrey L. Boore. "Two rounds of whole genome duplication in the ancestral vertebrate". In: *PLoS Biology* 3.10 (2005), e314. issn: 15457885. doi: 10.1371/journal.pbio.0030314.

[119] Nicholas H. Putnam et al. "The amphioxus genome and the evolution of the chordate karyotype". In: *Nature* 453.7198 (2008), pp. 1064–1071. issn: 0028-0836. doi: 10.1038/nature06967.

[120] Susumu Ohno. *Evolution by Gene Duplication*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1970. isbn: 978-3-642-86661-6. doi: 10.1007/978-3-642-86659-3.

[121] Allan Force et al. "Preservation of duplicate genes by complementary, degenerative mutations". In: *Genetics* 151.4 (1999), pp. 1531–1545. issn: 00166731. doi: 10101175.

[122] S. Jimenez-Delgado, J. Pascual-Anaya, and J. Garcia-Fernandez. "Implications of duplicated cis-regulatory elements in the evolution of metazoans: the DDI model or how simplicity begets novelty". In: *Briefings in Functional Genomics and Proteomics* 8.4 (2009), pp. 266–275. issn: 1473-9550. doi: 10.1093/bfgp/elp029.

[123] Ferdinand Marlétaz et al. "Amphioxus functional genomics and the origins of vertebrate gene regulation". In: *Nature* 564.7734 (2018), pp. 64–70. issn: 0028-0836. doi: 10.1038/s41586-018-0734-6.

[124] Cory Y McLean et al. "GREAT improves functional interpretation of cis-regulatory regions". In: *Nature Biotechnology* 28.5 (2010), pp. 495–501. doi: 10.1038/nbt.1630.

[125] Feng Yue et al. "A comparative encyclopedia of DNA elements in the mouse genome". In: *Nature* 515.7527 (2014), pp. 355–364. doi: 10.1038/nature13992.

[126] Naoki Irie and Shigeru Kuratani. "Comparative transcriptome analysis reveals vertebrate phylotypic period during organogenesis". In: *Nature Communications* 2.1 (2011), p. 248. doi: 10.1038/ncomms1248.

[127] Itai Yanai. "Development and Evolution through the Lens of Global Gene Regulation". In: *Trends in Genetics* 34.1 (2018), pp. 11–20. doi: 10.1016/J.TIG.2017.09.011.

[128] D Duboule. "Temporal colinearity and the phylotypic progression: a basis for the stability of a vertebrate Bauplan and the evolution of morphologies through heterochrony." In: *Development (Cambridge, England). Supplement* 42 (1994), pp. 135–142. issn: 09501991. doi: 10.1007/SpringerReference_34725.

[129] Haiyang Hu et al. "Constrained vertebrate evolution by pleiotropic genes". In: *Nature Ecology & Evolution* 1.11 (2017), pp. 1722–1730. doi: 10.1038/s41559-017-0318-0.

[130] Martín Abadi et al. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems". In: (2016). arXiv: 1603.04467.

[131] TG Tape. "Using the Receiver Operating Characteristic (ROC) curve to analyze a classification model". In: *Univ. Nebraska Med. Cent* (2000), pp. 1–3.

[132] Matteo Vietri Rudan and Suzana Hadjur. "Genetic Tailors: CTCF and Cohesin Shape the Genome During Evolution". In: *Trends in Genetics* 31.11 (2015), pp. 651–660. issn: 0168-9525. doi: 10.1016/J.TIG.2015.09.004.

[133] Peter Heger et al. "The chromatin insulator CTCF and the emergence of metazoan diversity". In: *Proceedings of the National Academy of Sciences* 109.43 (2012), pp. 17507–17512. issn: 0027-8424. doi: 10.1073/pnas.1111941109. url: https://www.pnas.org/content/109/43/17507.

[134] David Martin et al. "Genome-wide CTCF distribution in vertebrates defines equivalent sites that aid the identification of disease-associated genes". In: *Nature Structural & Molecular Biology* 18.6 (2011), pp. 708–714. issn: 1545-9993. doi: 10.1038/nsmb.2059.

[135] Rodolfo Ghirlando and Gary Felsenfeld. "CTCF: making the right connections." In: *Genes & development* 30.8 (2016), pp. 881–91. issn: 1549-5477. doi: 10.1101/gad.277863.116.

[136] Melisa Gomez-Velazquez et al. "CTCF counter-regulates cardiomyocyte development and maturation programs in the embryonic heart". In: *PLOS Genetics* 13.8 (2017). Ed. by Gregory S. Barsh, e1006985. issn: 1553-7404. doi: 10.1371/journal.pgen.1006985.

[137] Eduardo Soares and Huiqing Zhou. "Master regulatory role of p63 in epidermal development and disease". In: *Cellular and Molecular Life Sciences* 75.7 (2018), pp. 1179–1190. issn: 1420-682X. doi: 10.1007/s00018-017-2701-z.

[138] Jillian M. Pattison et al. "Retinoic acid and BMP4 cooperate with p63 to alter chromatin dynamics during surface epithelial commitment". In: *Nature Genetics* 50.12 (2018), pp. 1658–1665. issn: 15461718. doi: 10.1038/s41588-018-0263-0.

[139] Lingjie Li et al. "TFAP2C- and p63-Dependent Networks Sequentially Rearrange Chromatin Landscapes to Drive Human Epidermal Lineage Commitment". In: *Cell Stem Cell* 24.2 (2019), 271–284.e8. issn: 18759777. doi: 10.1016/j.stem.2018.12.012.

[140] Nitish Shirish Keskar et al. "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima". In: *CoRR* abs/1609.04836 (2016). arXiv: 1609.04836. url: http://arxiv.org/abs/1609.04836.

[141] Pär G Engström et al. "Genomic regulatory blocks underlie extensive microsynteny conservation in insects". In: *Genome research* 17.12 (2007), pp. 1898–1908.

[142] Robert K. Bradley et al. "Binding site turnover produces pervasive quantitative changes in transcription factor binding between closely related Drosophila species." In: 8.3 (2010). Ed. by Gregory A. Wray, e1000343. issn: 1545-7885. doi: 10.1371/journal.pbio.1000343.

[143] Qiye He et al. "High conservation of transcription factor binding and evidence for combinatorial regulation across six Drosophila species". In: *Nature Genetics* 43.5 (2011), pp. 414–420. issn: 1061-4036. doi: 10.1038/ng.808.

[144] Mathilde Paris et al. "Extensive Divergence of Transcription Factor Binding in Drosophila Embryos with Highly Conserved Gene Expression". In: *PLoS Genetics* 9.9 (2013). Ed. by Patricia Wittkopp, e1003748. doi: 10.1371/journal.pgen.1003748.

[145] Pierre Khoueiry et al. "Uncoupling evolutionary changes in DNA sequence, transcription factor occupancy and enhancer activity." In: *eLife* 6 (2017). issn: 2050-084X. doi: 10.7554/eLife.28440.

[146] Tanya Vavouri et al. "Parallel evolution of conserved non-coding elements that target a common set of developmental regulatory genes from worms to humans". In: *Genome biology* 8.2 (2007), R15.

[147] Camille Berthelot et al. "Complexity and conservation of regulatory landscapes underlie evolutionary resilience of mammalian gene expression". In: *Nature Ecology and Evolution* 2.1 (2018), pp. 152–163. issn: 2397334X. doi: 10.1038/s41559-017-0377-2. arXiv: 125435 [10.1101].

[148] Justin Cotney et al. "The evolution of lineage-specific regulatory activities in the human embryonic limb". In: *Cell* 154.1 (2013).

[149] Steven K Reilly et al. "Evolutionary genomics. Evolutionary changes in promoter and enhancer activity during human corticogenesis." In: *Science (New York, N.Y.)* 347.6226 (2015), pp. 1155–9. issn: 1095-9203. doi: 10.1126/science.1260943.

[150] Diego Villar et al. "Enhancer evolution across 20 mammalian species". In: *Cell* 160.3 (2015). issn: 1097-4172.

[151] Andrew B. Stergachis et al. "Conservation of trans-acting circuitry during mammalian regulatory evolution". In: *Nature* 515.7527 (2014). issn: 1476-4687. doi: 10.1038/nature13972.

[152] Jeff Vierstra et al. “Mouse regulatory DNA landscapes reveal global principles of cis-regulatory evolution.” In: *Science (New York, N.Y.)* 346.6212 (2014), pp. 1007–12. issn: 1095-9203. doi: 10.1126/science.1246426.

[153] Xiang Zhou et al. “Epigenetic modifications are associated with interspecies gene expression variation in primates”. In: 15.12 (2014), p. 547. issn: 1474-760X. doi: 10.1186/s13059-014-0547-3.

[154] Duncan T Odom et al. “Tissue-specific transcriptional regulation has diverged significantly between human and mouse.” In: *Nature genetics* 39.6 (2007), pp. 730–2. issn: 1061-4036. doi: 10.1038/ng2047.

[155] Alan P. Boyle et al. “Comparative analysis of regulatory information and circuits across distant species”. In: *Nature* 512.7515 (2014). issn: 1476-4687. doi: 10.1038/nature13668.

[156] Mark B. Gerstein et al. “Comparative analysis of the transcriptome across distant species”. In: *Nature* 512.7515 (2014). doi: 10.1038/nature13424.

[157] Brian Hendrich and Susan Tweedie. “The methyl-CpG binding domain and the evolving role of DNA methylation in animals”. In: *Trends in Genetics* 19.5 (2003), pp. 269–277. issn: 0168-9525. doi: 10.1016/S0168-9525(03)00080-5.

[158] Manuel Irimia et al. “Extensive conservation of ancient microsynteny across metazoans due to cis-regulatory constraints.” In: *Genome research* 22.12 (2012), pp. 2356–2367. issn: 1549-5469. doi: 10.1101/gr.139725.112.

[159] Oleg Simakov et al. “Insights into bilaterian evolution from three spiralian genomes”. In: *Nature* 493.7433 (). issn: 1476-4687. doi: 10.1038/nature11696.

[160] Alex S. S Nord et al. “Comparative analysis of the transcriptome across distant species”. In: *Cell* 155.7 (2013). issn: 1097-4172.

[161] N. Harmston, A. Baresic, and B. Lenhard. “The mystery of extreme non-coding conservation”. In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 368.1632 (2013), pp. 20130021–20130021. issn: 0962-8436. doi: 10.1098/rstb.2013.0021.

[162] I. Maeso et al. “Deep conservation of cis-regulatory elements in metazoans”. In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 368.1632 (2013), pp. 20130020–20130020. issn: 0962-8436. doi: 10.1098/rstb.2013.0020.

[163] Juan R. Martinez-Morales. "Toward understanding the evolution of vertebrate gene regulatory networks: Comparative genomics and epigenomic approaches". In: *Briefings in Functional Genomics* 15.4 (2016), pp. 315–321. issn: 20412657. doi: 10.1093/bfgp/elv032.

[164] Juan Wang, Hillel T. Schwartz, and Maureen M. Barr. "Functional Specialization of Sensory Cilia by an RFX Transcription Factor Isoform". In: *Genetics* 186.4 (2010), pp. 1295–1307. issn: 0016-6731. doi: 10.1534/genetics.110.122879.

[165] Daniel W. McShea. "Possible largest-scale trends in organismal evolution: Eight "Live Hypotheses"". In: *Annual Review of Ecology and Systematics* 29.1 (1998), pp. 293–318. doi: 10.1146/annurev.ecolsys.29.1.293.

[166] Detlev Arendt. "The evolution of cell types in animals: Emerging principles from molecular studies". In: *Nature Reviews Genetics* 9.11 (2008), pp. 868–882. issn: 14710056. doi: 10.1038/nrg2416.

[167] Jean-Francois Gout and Michael Lynch. "Maintenance and Loss of Duplicated Genes by Dosage Subfunctionalization". In: *Molecular Biology and Evolution* 32.8 (2015), pp. 2141–2148. issn: 0737-4038. doi: 10.1093/molbev/msv095.

[168] Adam M. Session et al. "Genome evolution in the allotetraploid frog Xenopus laevis". In: *Nature* 538.7625 (2016), pp. 336–343. issn: 0028-0836. doi: 10.1038/nature19840.

[169] Tine Blomme et al. "The gain and loss of genes during 600 million years of vertebrate evolution". In: *Genome Biology* 7.5 (2006), R43. issn: 14656906. doi: 10.1186/gb-2006-7-5-r43.

[170] Domné Farrè and M. Mar Albà. "Heterogeneous patterns of gene-expression diversification in mammalian gene duplicates". In: *Molecular Biology and Evolution* 27.2 (2010), pp. 325–335. issn: 07374038. doi: 10.1093/molbev/msp242.

[171] Honglak Lee et al. "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations". In: *Proceedings of the 26th annual international conference on machine learning*. ACM. 2009, pp. 609–616.

[172] Luisa M. Zintgraf, Taco S. Cohen, and Max Welling. "A New Method to Visualize Deep Neural Networks". In: *CoRR* abs/1603.02518 (2016). arXiv: 1603.02518. url: http://arxiv.org/abs/1603.02518.

[173] Luisa M. Zintgraf et al. "Visualizing Deep Neural Network Decisions: Prediction Difference Analysis". In: *CoRR* abs/1702.04595 (2017). arXiv: 1702.04595. url: http://arxiv.org/abs/1702.04595.

[174] Mathew G. Lewsey et al. "Determination and Inference of Eukaryotic Transcription Factor Sequence Specificity". In: *Cell* (2014). issn: 00928674. doi: 10.1016/j.cell.2014.08.009.

[175] Robert D. Finn et al. *Pfam: The protein families database*. 2014. doi: 10.1093/nar/gkt1223.

[176] Matthew T. Weirauch and T. R. Hughes. "A catalogue of eukaryotic transcription factor types, their evolutionary origin, and species distribution". In: *Sub-Cellular Biochemistry* (2014). issn: 03060225. doi: 10.1007/978-90-481-9069-0_3.

[177] Sean R Eddy. "A new generation of homology search tools based on probabilistic inference." In: *Genome informatics. International Conference on Genome Informatics* (2009). issn: 0919-9454.

[178] Fabian Sievers et al. "Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega". In: *Molecular Systems Biology* (2011). issn: 17444292. doi: 10.1038/msb.2011.75.

[179] Simon J. van Heeringen and Gert Jan C. Veenstra. "GimmeMotifs: A de novo motif prediction pipeline for ChIP-sequencing experiments". In: *Bioinformatics* (2011). issn: 13674803. doi: 10.1093/bioinformatics/btq636.

List of Abbreviations

TF	Transcription Factor
TFbs	Transcription Factor binding site
CRE	Cis Regulatory Element
TSS	Transcription Start Site
NN	artificial Neural Network
GO	Gene Ontology
WGD	Whole Genome Duplication
ROC	Receiver Operating Characteristic

List of Figures

List of Tables