1 Introduction

(TODO: include feedback!)

(TODO: Add Glossary?)

Even though much is known about the one-dimensional structure of our DNA, it also organizes itself in three-dimensional ways. Recently, with the advent of chromosome conformation capture (3C) technologies, it has become possible to gain insights in the three-dimensional structure as well. Gene-regulators have been intensively looked at in the one-dimensional case, but enhancer and promoter also have influence on their three-dimensional surroundings, making knowledge about the spatial organization relevant.

In this work, data gotten through Hi-C [?] will be used. Hi-C is a method for getting 3D-information of genomes. This is done by 'strapping' together parts of the genome that are close by, cutting apart the genome with restriction enzymes, combining the ends of strapped-together fragments and sequencing them. This will be explained in detail in Section **??**.

However, such technologies tend to suffer from technical (e.g. sequencing, mapping) [?] and biological factors (e.g. distinct chromatin states) [?], making them inherently inaccurate.

Biases are unavoidable, in particular, as some regions are more sensible for biotin labeling enrichments (See Section **??** why this is relevant) they will be measured more often when compared to others. PCR artifacts cannot be excluded [?]. Mapping locations may be unclear or not unique, introducing even more sources for possible biases. Certain sequencing methods have certain biases themselves. Some of the measured interactions are questionable, it is unclear if these are actual, spatially close points, or if it simply is a technical error or a randomly happened interaction.

(TODO: cite the papers for KR and ICE at the first introduction)

However, a basic but strong assumption about the structure of the genome can be made, which is that every location has the same amount of interactions (with other locations) as every other location. The data does not show this due to the several aforementioned inaccuracies. Algorithms such as ICE [?] (Iterative Correction and Eigenvector decomposition, Section **??**) or KR [?] (Knight-Ruiz, Section **??**) can be applied to normalize the matrix nonetheless.

ICE, the algorithm implemented in this work, normalizes the data iteratively. Eigenvector decomposition of the normalized data can then give new insights on local chromatin states or global patterns of chromosomal interaction, as was done here [?].

We will not do the Eigenvector decomposition, but the iterative correction ("normalization") before that.

1.1 Core setup

(TODO: remove deepTools entirely)

(TODO: include general setup; HiCExplorer running on Linux/Mac, Python, numpy, scipy, KR in C++, Missing interface from python to C but missing to RUST)

This work is part of the HiCExplorer (Section **??**) tool from the Deeptools (Section **??**) framework. HiCExplorer is mainly written in Python, being limited by the high resource requirements of the iteration process.

(TODO: run experiments!!)

(EXTEND: Add more details!!)

1.2 Algorithm

(TODO: move this section to storytelliing in introduction)

Fundamentally, every part in our genome has the same amount of summed up interactions with other parts of the genome. The algorithm takes severe advantage of this property, downright enforcing it. A full description can be found in Section **??**.

1.3 Motivation

1.3.1 Motivation for Hi-C

(TODO: include in storytelling)

Within cells, the three-dimensional structure of chromatin can now be analyzed using techniques based on chromatin conformation capture (3C), including Hi-C. With Hi-C **(DRAFT: what is the long name?)**, we can construct spatial proximity maps over the whole genome, giving us clues as to which regions are close to each other. With C3 it can be tested if two genomic locations are close to each other, but with Hi-C we get (biased but still) all spatial connections of the genome at once.

1.3.2 Motivation for Rust

There is several reasons for using Rust, including faster development than C++ and better tooling, while still having the same if not slightly better performance. A massive advantage is the easiness of adding parallelism, and the modularity of Rust code. Libraries in Rust can easily be stacked, and the compiler will complain if the memory is not handled the way it should be (in C++ it is hard to correctly manage all memory, especially from libraries, easily resulting in segfaults and similar issues hard to debug). The benefits to using Rust instead of Python are only apparent if performance or safety (of execution, at runtime) is needed - which is the case here. **(TODO: explain this more: thread safety, modularity, stackability, hard to debug C++)**

(TODO: more formal language)

1.4 Concrete Task

(TODO: Concrete task definition as 1.1)

This work is mainly testing the integration between Rust and Python, intending to answer the queston if this is a better implementation.

In the three-dimensional space of a cell the DNA forms a structure that looks close to that of a ball of wool. Obviously, many points of contacts of the DNA wire with itself, called DNA interactions, exist in this “ball of wool” and form structures including

DNA loops. However, many of these contacts are random contacts or measurement errors that need to be corrected. A Python implementation exists but is limited for high resolution data due to high memory usage. This [..] project aims to reimplement a more memory efficient method in C++ (which ended up being Rust).

(TODO: around half a page to one)

ToDo Counters

To Dos: 12;   1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

Parts to extend: 1;   1

Draft parts: 1;   1