Undergraduate Thesis

Hi-C interaction matrix correction using ICE in Rust

Felix Karg

Examiner: Prof. Dr. Backofen
Advisers: Joachim Wolff, Dr. Mehmet Tekman

Albert-Ludwigs-University Freiburg
Faculty of Engineering
Department of Computer Science
Chair for Bioinformatics

July 10th, 2019

Writing Period

10. 04. 2019 – 10. 07. 2019

Examiner

Prof. Dr. Backofen

Advisers

Joachim Wolff, Dr. Mehmet Tekman

Declaration

I hereby declare, that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare, that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Place, Date	Signature

Abstract

foo bar

Zusammenfassung

German version is only needed for an undergraduate thesis.

(TODO: Schreiben!)

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

Hi-C is a method commonly used for getting 3D-information of genomes. Such technologies tend to suffer from technical (e.g. sequencing, mapping) [1] and biological factors (e.g. distinct chromatin states) [2], making them inherently inaccurate.

However, a basic 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 implented in this work, "normalizes" the data we have iteratively. Eigenvector decomposition of the normalized data can then give us new insights on local chromatin states or global patterns of chromosomal interaction [3].

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

1.1 Core setup

This work is part of the HiCExplorer (Section 3.2) tool from the Deeptools (Section 3.1) framework. HiCExplorer is mainly written in Python, which turned out

to currently be a bottleneck for the iterative correction, taking quite some time for currently reasonable-sized data.

As we will see, **(TODO: run experiments!!)**

(EXTEND: Add more details!!)

1.2 Algorithm

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 Operation

(TODO: Change Name!) smb can be run on any Unix-based operating system (tested using ubuntu-18.04) with Python, Rust and common development packages installed (e.g. `libopenssl-dev python3-dev build-essential` ...). For best performance, the size of the matrix should correlate with the number of available cores and the amount of available RAM **(EXTEND: Give rough factors!)**.

1.4 Motivation

1.4.1 Motivation for Hi-C

Within cells, the three-dimensional structure of chromatin can now be analysed using techniques based on chromatin conformation capture, including Hi-C. With Hi-C, we

can construct spatial proximity maps over the whole genome, giving us clues as to which regions are close to each other.

1.4.2 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 other 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.

2 Background

2.1 Chromosome Conformation Capture and Hi-C

2.1.1 Overview

Chromatin usually describes different levels of how DNA organizes itself. The well-known double-helix is only the lowest of several structural layers.

Looking at it from the outside (highest structural layer), DNA looks similar to a big ball of wool. With the help of Hi-C (or other methods) we can visualize the spatial proximity.

Chromosome Conformation Technologies describe several similar methods to compare genomic loci. They all start by:

- creating chromatin cross-links (Section 2.1.2)
- digesting the cross-linked chromatin (Section 2.1.3) and
- ligating the ends (Section 2.1.4)

to get a reversed cross-link sequence.

We will, however focus on Hi-C only, which follows up with:

- shortening the crosslinks by sonication (Section 2.1.6),
- filtering leaving only those with biotin (Section 2.1.7) and
- sequencing (Section 2.1.8).

The full process can be seen in Figure 1.

2.1.2 Cross-linking DNA

The first step is to cross-link DNA strands that are close to each other spatially (see Figure 1 for reference). This is done by adding formaldehyde, which bonds sufficiently close strands together.

A chromatin cross-link is two entirely different parts of the genome held together by a chemical bond with formaldehyde. This process cannot be specifically controlled, so only 'regions near each other' are connected, not necessarily two 'known to be close' regions.

2.1.3 Digestion

The next step is cutting the ball of wool apart in intervals. For this, restriction-enzymes are used (specifically restriction endonuclease). Commonly used enzymes for this are EcoR1 or HindIII, cutting the genome every 4000 base-pairs **(DRAFT: info taken from wikipedia, add some other source)**.

This will result in a lot of cross-linked fragments, as well as not-cross-linked ones.

2.1.4 Ligation

After reducing the concentration of fragments, DNA ligase is added, to ligate (weld together) dangling fragment ends. The reduction in concentration is done since mostly fragments close together are ligated, and we intend to ligate fragments held together by formaldehyde. Also, Biotin is added to mark the points of ligation. This will let us filter out a lot of fragments that have not been ligated later on.

2.1.5 Reverse Cross-links

Adding a high concentration of salt for some time will reverse the cross-linking through formaldehyde, leaving us with our two originally spatially close fragments ligated and with a biotin-marker.

Our fragments, however, are too long to sequence them effectively (remember that we have now ligated fragments of around 8000 base-pairs, most sequencing methods can only deal with sequence lengths of a few hundred base-pairs).

2.1.6 Sonication

The next step is to put them under influence of ultrasonic waves, breaking them apart in much shorter fragments (due to long sequences not being able to absorb frequent shocks well), short enough to actually enable sequencing.

2.1.7 Filtering and Removal of Biotin

Here we filter all the fragments, leaving only those having a biotin-marker (see Section 2.1.4) and thus have been ligated earlier. Subsequently we remove the marker, since it would get in the way of sequencing.

2.1.8 Sequencing

Sequencing, short for DNA sequencing, describes processes of measuring a DNA sequence. There are several techniques for doing this, most use PCR (Polymerase Chain Reaction) before or while sequencing, which duplicates the fragments several times, to sequence them more accurately.

(TODO: Cite/introduce/... the given papers, and introduce the required concepts)

Required concepts:

- Biology:
- The iterative algorithm (again ?)
- analysis that can be done further
- outlook. Meaning: What can be done, when having the 3C-Data?

2.2 Further Processing

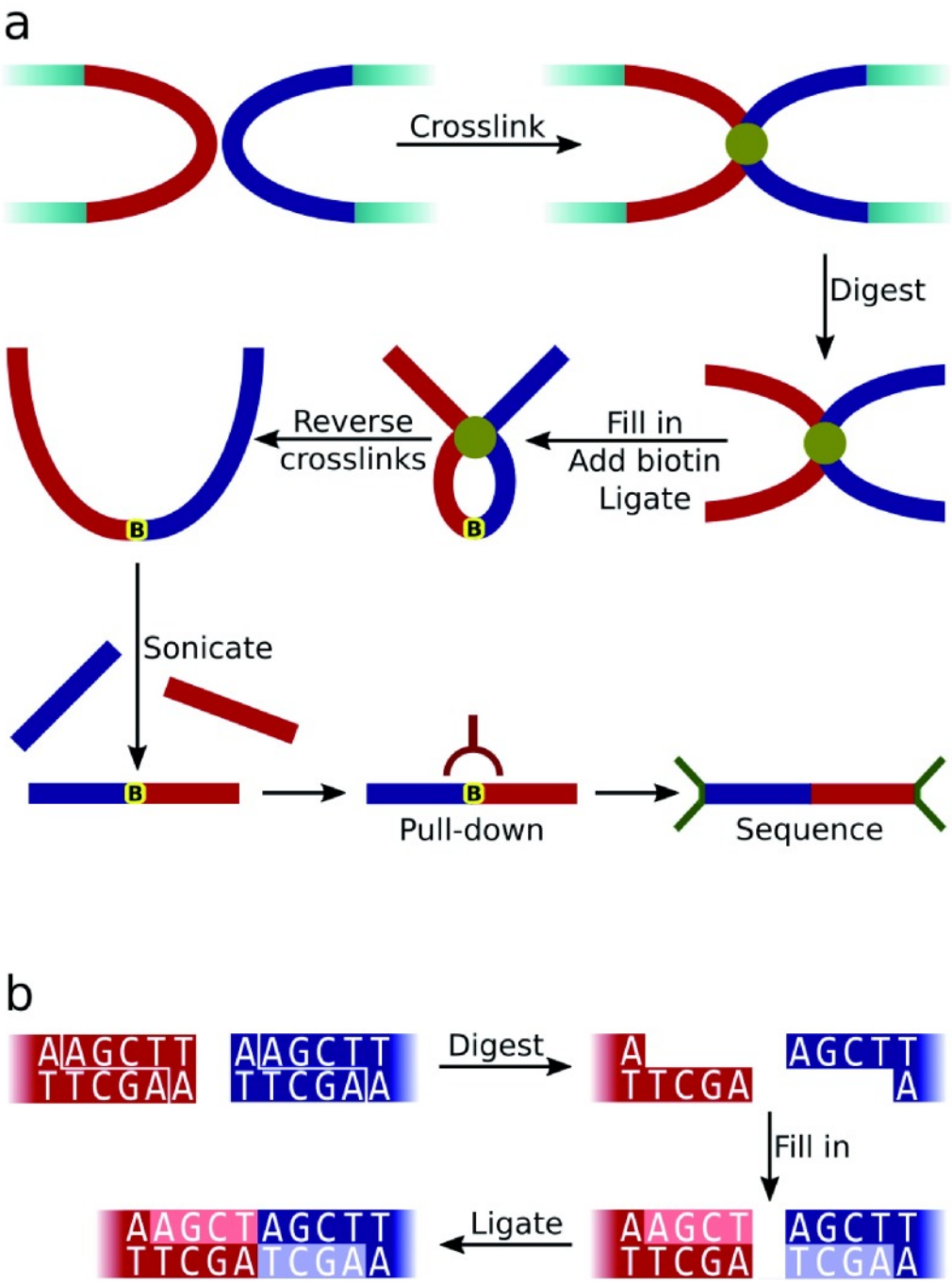

Figure 1: **a)** Diagram summarising the Hi-C experimental protocol. The red and blue rectangles represent cross-linked restriction fragments while the yellow marker shows the position of biotin incorporation. **b)** Generation of the Hi-C ligation junction sequence by successive digestion (with HindIII in this example), fill in and blunt-ended ligation steps. The modified restriction site sequence is not found in the original genomic sequence.

Image and description taken from [4].

3 Related Work

(TODO: introduce original implementation in python and KR-algorithm in C++)

3.1 Deeptools

(TODO: introduce the whole deeptools framework)

3.2 HiCExplorer

(TODO: introduce hicexplorer)

3.3 Python implementation

As the deeptools-framework is mainly written in Python, and this algorithm is a fundamentally important one, it was originally implemented in Python. This implementation however requires too much working memory, limiting the size of usable matrices, which is why a different implementation was asked for.

Rust and Python are two quite different programming languages, which is why a direct “translation” is not even possible. Both are semantically speaking the same,

but details differ. Since Rust has a much finer control about memory and the applying of functions to datastructures, some operations have been seperated while others have been combined.

The biggest difference, however, is that in Rust certain tasks can easily be parallelized - even after originally writing it for only one core. For smaller matrices the overhead can be quite big, for one of our test matrices the optimum was using 'only' three of the available 32 cores (this still gave an improvement of **(TODO: get numbers!)**).

3.4 KR-algorithm

(TODO: read paper + introduce here)

(TODO: maybe also implement in Rust as well)

(TODO: is that really everything related?)

4 Approach

4.1 Problem

(TODO: remove emotional touch (killer feature, weird bugs, ...))

4.2 Choosing the right API to call Rust from Python

(TODO: Add footnotes. Research how this works)

There are three main ways to execute Rust code from Python. In the following, common techniques are investigated.

One common way is rust-cpython. This library requires Rust 1.25 or higher (current versions are 1.33/34/35 for stable/beta/nightly respectively). Rust-cpython grants access to the python gil (global interpreter lock) with which Python code can be evaluated and Python objects modified. The resulting library (directly from compiled rust) can easily be imported into Python (but needs to be renamed). Native Rust code requires some wrapping first, as shown here:

```
#[macro_use] extern crate cpython;
use cpython::{PyResult, Python};
// add bindings to the generated python module
// N.B: names: "librust2py" must be the name of the `.so` or `.pyd` file
```

```
py_module_initializer!(librust2py, initlibrust2py, PyInit_librust2py, |py,
    m.add(py, "__doc__", "This module is implemented in Rust.")?;
    m.add(py, "sum_as_string", py_fn!(py, sum_as_string_py(a: i64, b:i64)))
    Ok(())
});
// logic implemented as a normal rust function
fn sum_as_string(a:i64, b:i64) -> String {
    format!("{}", a + b).to_string()
}
// rust-cpython aware function. All of our python interface could be
// declared in a separate module.
// Note that the py_fn!() macro automatically converts the arguments from
// Python objects to Rust values; and the Rust return value back into a Py
fn sum_as_string_py(_: Python, a:i64, b:i64) -> PyResult<String> {
    let out = sum_as_string(a, b);
    Ok(out)
}
```

This kind of wrapping, though quite common and based on the Python C-API makes it hard to write idiomatic Code in Rust. Also, since Python is directly affected, the interactions with Python need to be considered while writing Rust-Code. In computer science one does usually not intentionally strive for complexity.

Another common approach is using the pyO3-library, which started off as a fork of rust-cpython, but has since seen quite drastic changes. For example, its using requires at least Rust version '1.30.0-nightly 2018-08-18'. This has been updated to '1.34.0-nightly 2019-02-06' with the most recently update. This is due to the usage of several unstable features, most of which have recently been able to be promoted to stable. Still missing is Specialisation though, which has at the time of writing still a long way to go. The library would also result in an easily importable (needs to

be renamed first, still) cdylib (same as rust-cpython). The still intermingled way of writing the interface (certainly better but not by much compared to rust-cpython) as well as the dependency on unstable nightly rust versions led to the decision of not using it either.

The third way, that is actually been promoted in the official Rust docs, is to generate a dylib and import that in python. No renaming necessary, but the communication between Rust and Python is a bit more low-level. The main wrapper is on the side of Python, transforming Arguments to Pointers and C-Representations, whilst the Rust part needs to conform to C-practices, which includes receiving a list by getting a pointer and the length of it. Other than that, the Rust code has some additional `\#[no_mangle]` and `\#[repr(C)]` (procedural) macros, which result in these parts actually accessible from e.g. Python or C. Since like this neither language depends on something only internal (or combinatorial), and both just depend upon the 'common, unchanging' C-interface, this seems to be the preferred way.

(TODO: Introduce main approach, problems I came across and more)

5 Experiments

times (precomputed correction factors):

```
542.32user 447.26system 16:26.02elapsed 100\%CPU (0avgtext+0avgdata 116107220maxresident)
14528inputs+8outputs (20major+159388812minor)pagefaults 0swaps
```

But for which setup? single-core? the small_test_matrix?

(TODO: The time-resource-measures done)

(TODO: those were run on a ...)

6 Conclusion

7 Acknowledgments

First and foremost, I would like to thank my primary supervisor, Joachim Wolff, for all the advice given.

ToDo Counters

To Dos: 16; 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16

Parts to extend: 2; 1, 2

Draft parts: 1; 1

Bibliography

[1] M. S. Cheung, T. A. Down, I. Latorre, and J. Ahringer, "Systematic bias in high-throughput sequencing data and its correction by BEADS," *Nucleic Acids Res.*, vol. 39, p. e103, Aug 2011. [PubMed Central:PMC3159482] [DOI:10.1093/nar/gkr425] [PubMed:20824077].

[2] L. Teytelman, B. Ozaydin, O. Zill, P. Lefrancois, M. Snyder, J. Rine, and M. B. Eisen, "Impact of chromatin structures on DNA processing for genomic analyses," *PLoS ONE*, vol. 4, p. e6700, Aug 2009. [PubMed Central:PMC2725323] [DOI:10.1371/journal.pone.0006700] [PubMed:12067662].

[3] M. Imakaev, G. Fudenberg, R. P. McCord, N. Naumova, A. Goloborodko, B. R. Lajoie, J. Dekker, and L. A. Mirny, "Iterative correction of hi-c data reveals hallmarks of chromosome organization," *Nature methods*, vol. 9, no. 10, p. 999, 2012.

[4] S. Wingett, P. Ewels, M. Furlan-Magaril, T. Nagano, S. Schoenfelder, P. Fraser, and S. Andrews, "Hicup: pipeline for mapping and processing hi-c data," *F1000Research*, vol. 4, 2015.