Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Probabilistic Models for Genome Assembly with Chromatin Interaction Frequencies

Diploma thesis

2018
Bc. Askar Gafurov

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Probabilistic Models for Genome Assembly with Chromatin Interaction Frequencies

Diploma thesis

Study programme:	Computer Science
Field of study:	Computer Science
Department:	Department of Computer Science
Supervisor:	doc. Mgr. Bronislava Brejová, PhD.

Bratislava, 2018
Bc. Askar Gafurov

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Askar Gafurov
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Probabilistic Models for Genome Assembly with Chromatin Interaction Frequencies
Pravdepodobnostné modely pre zostavovanie genómov s využitím chromatínových interakcií

Anotácia: Zostavovanie genómov je bioinformatický problém, ktorého cieľom je určiť neznámu sekvenciu genómu určitého organizmu z dát produkovaných rôznymi technológiami sekvenovania DNA. Typické sekvenačné dáta pozostávajú z krátkych čítaní z cieľového genómu. Nedávno sa objavili metódy, ktoré na skladanie genómu používajú aj dáta pôvodne určené na objavovanie interakcií medzi časťami chromozómov v jadre bunky. Cieľom diplomovej práce je o takýto typ dát rozšíriť modely a algoritmy programu GAML, ktorý skladá genómy na základe pravdepodobnostných modelov.

Vedúci: doc. Mgr. Bronislava Brejová, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Dátum zadania: 13.12.2016

Dátum schválenia: 14.12.2016 prof. RNDr. Rastislav Kráľovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname: Bc. Askar Gafurov
Study programme: Computer Science (Single degree study, master II. deg., full time form)
Field of Study: Computer Science, Informatics
Type of Thesis: Diploma Thesis
Language of Thesis: English
Secondary language: Slovak

Title: Probabilistic Models for Genome Assembly with Chromatin Interaction Frequencies

Annotation: Genome assembly is a computational problem in bioinformatics aiming to infer an unknown genomic sequence of a given organism from data produced by various DNA sequencing technologies. A typical sequencing data set consists of short reads originating from the target genome. Recently, data originally produced for exploring interaction among parts of chromosomes in a nucleus was also used to improve genome assembly. The goal of the thesis is to extend models and algorithms of the GAML framework for genome assembly based on probabilistic models so that it supports such data sources.

Supervisor: doc. Mgr. Bronislava Brejová, PhD.
Department: FMFI.KI - Department of Computer Science
Head of department: prof. RNDr. Martin Škoviera, PhD.

Assigned: 13.12.2016

Approved: 14.12.2016 prof. RNDr. Rastislav Kráľovič, PhD.
Guarantor of Study Programme

.....
Student

.....
Supervisor

Acknowledgment:

I'd like to thank my family for their endless love and support, my supervisor doc. Brejová for her guidance and patience, and fellow students in M-25 lab for creating a unique working atmosphere.

Abstrakt

V tejto práci sme sa venovali úlohe zostavovania genómov pomocou dát, popisujúcich interakcie jednotlivých častí chromozómov v jadre bunky (nazývané Hi-C). Nadväzujúc na prácu Kaplana a Dekkera [1], vyvinuli sme nový pravdepodobnostný model pre Hi-C dáta, prispôsobený programu GAML [2], ktorý skladá genómy práve na základe pravdepodobnostných metód. Nakoľko pôvodná implementácia GAMLu nebola určená pre modifikácie, museli sme najprv prepísat značnú časť jeho funkcionality do novej implementácie, nazvanej GAML2. Experimentálne výsledky ukazujú, že nami rozšírený GAML2 dokáže skladať genómy lepšie za použitia Hi-C dát.

Kľúčové slová: Zostavovanie genómov, maximálna vierohodnosť, pravdepodobnostné modelovanie, Hi-C, frekvencie chromatínových interakcií, GAML

Abstract

In this paper we have addressed the task of genome assembly using the data of chromatin interactions (Hi-C). Inspired by the work of Kaplan and Dekker [1], we have proposed our own probabilistic model for Hi-C data, suited specifically for the GAML framework for genome assembly [2]. As the original GAML implementation was a prototype not meant to be extended, we have reimplemented most of its features in the second implementation, called GAML2, and then added our model and algorithms to process Hi-C data. We have shown that our implementation of GAML is able to perform better with Hi-C data available.

Keywords: Genome assembly, maximum likelihood, probabilistic modeling, Hi-C, chromatin interaction frequencies, GAML

Contents

List of Figures

Introduction

The discovery of DNA molecules and of their role as transmitters of hereditary information and "blueprints" for every living organism was one of the turning points of the twentieth century. This discovery led, on the one hand, to better under- standing of our world and connections between different species. On the other hand, it allowed to better understand ourselves and in particular how to cure and prevent diseases in better ways.

The discovery of DNA sequencing techniques was the key moment for molecular biology and genetic studies. These techniques were refined over several decades until, in year 2001 [4], the humanity successfully decoded the human genome. This was possible due to our substantial advances in engineering and Chemistry, but also in Computer Science. The amount of calculations, required to assemble long genomes remains gigantic even for today's computers.

Programs that assist in the task of decoding the genome are called sequence assemblers. Their task is similar to the task of a child playing with puzzle. They take small pieces of digitised DNA sequences and try to reconstruct the original picture, the genome, from them.

Although modern sequence assemblers are quite usable, they still have several drawbacks. Firstly, they rely mostly on heuristics approaches (similarly to a child from the metaphor above). Secondly, they are suited for specific types of data and are typically incapable to combine several types of data at the same time.

These issues have been addressed by a probabilistic approach to the task of genome assembly, developed by Boža during his Ph.D. studies at our faculty [2]. Our goal is to enhance his framework GAML and adapt it to new types of sequencing data.

This thesis consists of four chapters. In the first chapter, we will give an overview of genome assembly, current approaches to it, and describe the probabilistic approach to this task, implemented in GAML framework. We will finish this chapter by describing a new type of data, which until recently was not used in this task: the frequencies of chromatin interactions (Hi-C). In the second chapter, we will describe our work on GAML2, the second implementation of the GAML framework. In the third chapter, we will describe a new probabilistic model for Hi-C data, compatible with GAML2. In the fourth chapter, we will show experimental results of genome assembly with GAML2

Chapter 1

Background

In this chapter we will briefly cover basics of genome assembly: what it is, what methods are used and what are main challenges. Then we will describe probabilistic approach to this task and in particular the details of the GAML framework. We will finish this chapter with a description of a new type of data potentially useful for genome assembly: frequencies of chromatin interactions (or *Hi-C* for short).

1.1 Genome assembly

Hereditery information of all living organisms (except some types of viruses) is stored in molecules of *deoxyribonucleic acid* (or *DNA* for short), located in the cells. A DNA molecule is usually formed by two polymer strands, forming a so-called *double helix* (figure 1.1). Each strand of DNA is a chain of monomers, called *nucleotides*. Each nucleotide consists of one of four nitrogen-containing nucleobases (adenine (A), cytosine (C), guanine (G) or thymine (T)), a phosphate group and a deoxyribose (sugar). Nucleotides in a chain are joined together by covalent bonds between the phosphate group of one nucleotide and the sugar of another. The phosphate groups are bounded with the 5' and 3' carbons of the deoxyribose (Figure 1.2). Therefore, one of the ending bases of a DNA strand has a "free" 3' carbon (bound with a $-OH$ group instead), and the other end of the strand is ended with a phosphate group, bounded on the 5' carbon of the last nucleotide. These ends are referred to as *5'-* and *3'-ends*. Nucleotides of two strands are bound together by hydrogen bounds between their nucleobases, according to *base pairing rule*: adenine bounds with thymine, and guanine bounds with cytosine. The

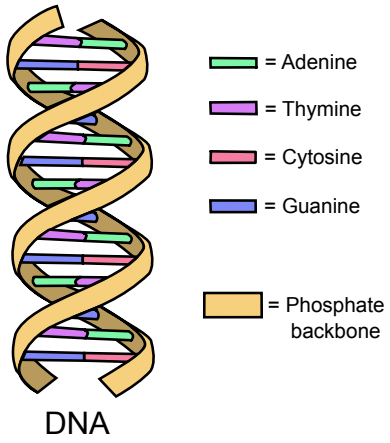


Figure 1.1: DNA structure

orientation of the strands in the molucle is reversed, i.e. a nucleobase of the 5'-end of one strand is bounded to a nucleobase of the 3'-end of the other strand. Each DNA molecule in a cell is called a *chromosome*. A full collection of chromosomes in a cell is called a *genome*.

We represent a DNA molecule as a string of letters A,C,G and T, corresponding to the order of nucleotides of one of its strands starting from the 5'-end. For every DNA molecule consists of two complementary strands, by reversing the order of the string (to preserve the orientation of 5' and 3' ends) and replacing all letters according to the base pairing rule we can obtain another representation of the same molecule (e.g. $ACCGTCA$ and $TGACGGT$ represent the same DNA sequence).

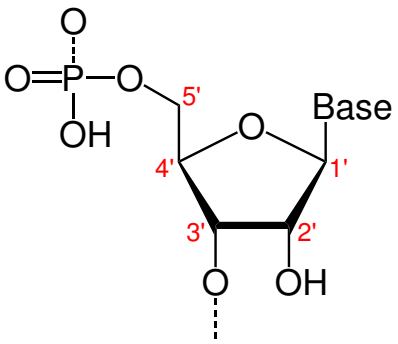


Figure 1.2: Deoxyribose with a phosphate group, attached to its 5' carbon

DNA sequencing is the process of determining the precise order of nucleotides in each chromosome of a genome. The first successful results were obtained in early 1970s. Since then methods of DNA sequencing have been improved, reducing cost and time and automating the whole process. Due to inability of sequencing technologies to capture the whole sequence of long DNA molecules[1], DNA molecules are first fragmented into smaller chains, which are digitised by various methods. The task of reconstructing the sequence of the original molecule from these digitised fragments, called *reads*, is called *sequence assembly* (or, in case of sequencing of an entire genome, *genome assembly*).

More precisely, DNA sequencing process consists of:

1. Extraction of the DNA from cells
2. Fragmentation of extracted DNA molecules
3. Amplification — to create additional copies of DNA fragments to increase chance that every region of DNA will have sufficient coverage by reads
4. Sequencing of DNA fragments, producing reads
5. Computational assembly of reads

To be able to speak about genome assembly, we need to establish several formal definitions:

- *DNA sequence* $S = s_1 s_2 \dots s_n$ is a finite string over an alphabet $\{A, C, G, T, N\}$. $|S| := n$ denotes a length of the string. N stands for "unknown nucleotide".

[1]Some genomes have more than 100 billions basepairs [5]

- *Complementary base* $cb : \{A, C, G, T, N\} \to \{A, C, G, T, N\}$ is a function over a set $\{A, C, G, T, N\}$, defined as follows: $cb(A) = T, cb(T) = A, cb(C) = G, cb(G) = C, cb(N) = N$
- *Reverse complement* $rc : \{A, C, G, T, N\}^* \to \{A, C, G, T, N\}^*$ is a function over DNA sequences, defined as follows: $rc(s_1 \dots s_n) = cb(s_n)cb(s_{n-1}) \dots cb(s_2)cb(s_1)$
- *Chromosome* S is a DNA sequence. We will refer to its two representations S and $rc(S)$ as to *strands* of the chromosome.
- *Assembly* $A \subset_{finite} \{A, C, G, T, N\}^*$ is a finite (multi)set of DNA sequences. Each DNA sequence in the genome assembly may be referred to as *contig*
- *Single read* r is a finite string over an alphabet $\{A, C, G, T, N\}$ (sometimes N is excluded from the alphabet)
- *Paired read* r is a pair of single reads. The individual single reads are mostly reffered to as left read r_1 and right read r_2
- *Global (end-to-end) alignment* of DNA sequence A to DNA sequence B is a sequence of single-letter operations (substitution of a symbol, insertion of a symbol, deletion of a symbol) that transform the string A to the string B. If we assign a penalty for each type of edit operation, we can evaluate the penalty score of a global alignment as a sum of penalties of individual edits. The alignment with minimum possible penalty score of operations is called *best global alignment*. The penalty score of the best global alignment is called *edit distance* between A and B.
- *(Read) alignment* of a read A to a DNA sequence B (where $|B| \geq |A|$) is an global alignment of A to a contiguous subsequence of B.

The goal of the genome assembly is to produce an assembly equal or very close to the original genome, based on an input set of reads. However, this definition cannot be used in case that the given genome has not been yet sequenced. Such task is called *de novo* genome sequencing. In that case, the goal is to produce the most *plausible* assembly with respect to our prior knowledge of biology.

One of the attempts to convert this task to exact computational problem is a so-called problem of *shortest superstring*. The input is a set of strings $R = \{r_1, \dots, r_n\}$, and the demanded output is the shortest possible string A^* that contains every string r_i as its continuous substring. This formulation suffers from the fact that genomes are not evolved as a succinct storage of the information (e.g. some genes are known to have several copies in a genome).

Even in this simplified case, the task of genome assembly from individual reads is NP-hard [6]. Moreover, assembly is complicated by several factors:

- *contamination* — there may be reads from other molecules (e.g. from vector bacteria, used in amplification of DNA fragments)
- *unequal coverage* — some parts of the DNA molecule may have more corresponding reads in the resulting library as others. Some parts could even have no corresponding reads and thus be completely invisible to us.
- *amplification and sequencing errors* — each read (string representation) may contain errors (insertions, deletions, substitutions, etc.)
- *repetitive sequences* — the original DNA molecule may contain the same subsequence in several places
- *tandem repeats* — there may be a short sequence repeating itself many times (e.g. $AAAACAAACAAACAAACAAAC\ldots$)
- *complementarity* — reads are produced from both strands of a DNA molecule. For example, if one strand is $ACCTGCTAA$, the other strand is $TTAGCAGGT$, which is obtained by reverse complementing. We could obtain reads $ACCTGC$ or $TTAGCAG$

In order to overcome problems with repetitive segments, some technologies (e.g. Illumina [7]) offer so-called paired end reads, where both ends of a DNA fragment are sequenced (see figure 1.3). This way we get both precise reads and long-range information about sequence. Other technologies (e.g. PacBio [8]) offer longer reads ($20,000$ bp), but with higher sequencing error rate

1.2 Conventional methods of genome assembly

There are two major approaches to genome assembly: overlap-layout-consensus [9] (or OLC for short) and de Bruijn graphs [10] (DBG for short).

1.2.1 Overlap-layout-consensus method

This method is based on the following idea: if an end of one read is nearly identical to a beginning of another (i.e. they *overlap*), then they probably represent the same segment of the original sequence. Such overlapping segments are merged into continuous segments (called *contigs*). These contigs are then joined further using long-range information from paired or long reads (this sequences of contigs are called *scaffolds* and this process is often being referred to as *scaffolding*). In this step, it is sometimes allowed to leave some bases undefined (often denoted as "N"), if we are convinced that

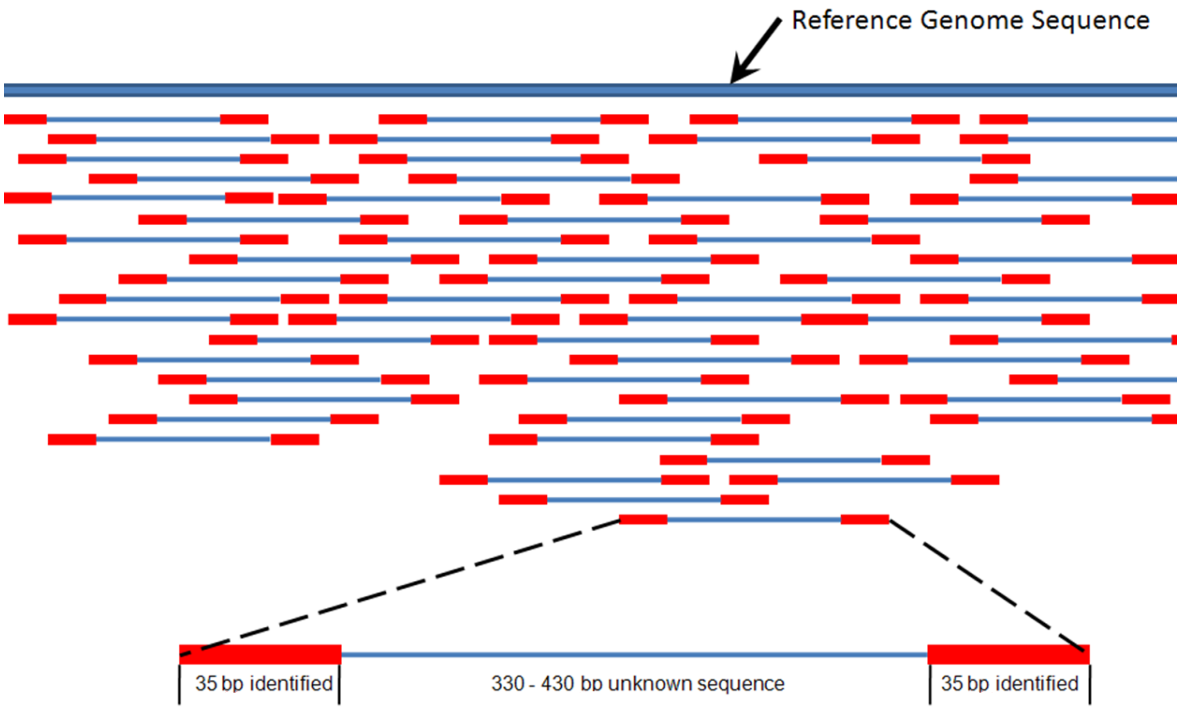


Figure 1.3: Paired end reads

two particular contigs lie in the same chromosome and have the predicted distance, but we have little information about the basepairs between them.

The overlap-layout-consensus method is a standard method for sequencing from long reads (e.g. PacBio, Sanger). The main issue with this approach is that it is computationally heavy.

This method is used e.g. in Arachne [11], Canu [12] and Celera [9] assemblers.

1.2.2 De Bruijn graph method

This approach works on the assumption that the sequencing process generates large amount of data, and therefore we can afford to lose some of the information.

For simplicity of explanation, we assume that the genome consists of only one chromosome, which is fully covered by reads. Moreover, we assume that every read comes from the same strand of the chromosome, so we may ingore the complementarity of the data.

At the beginning, each read is split into overlapping fragments of length k (referred to as k-mers). We then build *de Bruijn graph* from these k-mers by the following rules:

- Let K be a set of all k-mers, produced from the data
- Nodes of de Bruijn graph is a set of all $(k-1)$-mers, produced from the data
- Nodes $a_1a_2\dots a_{k-1}$ and $b_1b_2\dots b_{k-1}$ are connected by an edge iff $a_2\dots a_{k-1} =$

$b_1 \dots b_{k-2}$ and $a_1 \dots a_{k-1} b_{k-1} \in K$

In an ideal case, the solution to the assembly problem then would be an Eulerian path of the graph. However, even then it may differ from the original sequence (due to information loss). In practice, de Bruijn graph could have several Eulerian paths or none at all. Therefore, many heuristics are used, for example:

- Removal of nodes and edges, that likely correspond to sequencing errors
- Creation of multiple edges between two nodes, if there is evidence that the corresponding k-mer has multiple occurences in the original sequence
- Merging of nodes, that correspond to non-branching paths in the graph (these nodes are then considered contigs in the output)

This method is employed e.g. in SPAdes [13] and Velvet [10] assemblers.

1.2.3 Measuring the quality of genome assembly

Total assembly length

The total length of an assembly serves as an estimation of the genome size. Because the genome size could be estimated with non-sequencing methods [5], this metric could be used even in de novo assembly.

Nx and Lx statistics

Nx statistics targets the contiguity of an assembly, i.e. whether contigs or scaffolds are sufficiently long. This metric does not depend on a real genome, and therefore can be used in de novo assembly. Assume we have an assembly $A = (S_1, \dots, S_n)$. Let M denote the total length of scaffolds. Nx statistics of an assembly A is then defined as a maximum possible length l such that scaffolds with length at least l have total length at least $x\%$ of the total assembly size. Formally,

$$Nx(A, G) := \max \left\{ l \in \mathbb{N} : \sum_{S \in A} |S| \, [|S| \geq l] \geq \frac{x}{100} M \right\}$$

The x is often set to 50 or 90, so the statistics is then called $N50$ or $N90$.

We will show the example of an evaluation of $N50$ statistics. Let our assembly have scaffolds with lengths $(50, 60, 70, 80, 100, 120)$. The total assembly length is therefore equal to 480. Then the $N50$ is equal to 80, because $80 + 100 + 120 = 300 \geq 240 = 50\% \cdot 480$.

The Lx statistics is defined as a smallest number of contigs that have total length is at least $x\%$ of a genome size. In other words, Lx is equal to the number of contigs of length greater or equal to Nx. For the example above, the $L50$ would be equal to 3.

NAx and LAx statistics

This statistics are very similar to Nx and Lx. The only difference is that all scaffolds are first aligned to the real genome and misaligned scaffolds are split into smaller blocks at the misassembly events. The resulting assembly is then passed as an input for the Nx (Lx) statistics, described above. Note that this metric could not be used in de novo assembly, as the real genome sequence is used to identify misassemblies.

However, if we know the genome, this metric is more useful than Nx and Lx, because Nx and Lx can be easily fooled by concatenating all contigs into one scaffold.

1.3 Probabilistic approach to genome assembly

Due to the stochastic character of genome sequencing, it is only natural to define the task of genome assembly in the language of theory of probability. This approach has several potential advantages: first, we can incorporate our knowledge of molecular biology into our model; second, instead of devising custom heuristics for each type of data, we can devise a precise model and then find the best solution using general well-established heuristic frameworks; third, we can use several types of data without discarding any information.

Ghodzi et al. in [14] devised a generative statistical model to evaluate the quality of a genome assembly. Boža et al. [2] later adapted their model to directly find a high-quality assembly by finding a maximum a posteriori probability estimate. Their work resulted in an assembler called GAML. The main challenge of this approach are computational requirements.

The general idea is to define a conditional probability $P(X|\theta)$ of observations X (in our case, reads) with respect to parameters of the model θ (in our case, a genome assembly) and a prior probability of parameters $P(\theta)$. Then, we can find the optimal parameters by maximisation:

$$\theta^*(X) = \arg\max_\theta P(\theta|X) = \arg\max_\theta \frac{P(X|\theta)P(\theta)}{P(X)} = \arg\max_\theta P(X|\theta)P(\theta)$$

The rest of this section discusses the original GAML framework in detail.

1.3.1 Probabilistic model for a genome assembly in GAML framework

To better explain the model, we will first establish a simplified model for single reads with several assumptions. First assumption is that a genome consists of only one chromosome. Second, there are no sequencing errors, i.e. each observed (single) read is

a substring of a chromosome, either from a direct or a reverse strand. Third, every read is sampled independently from every position of the chromosome with equal probability. Fourth, every (single) read has the same length l.

With these assumptions in mind, the probability of a single read r to be observed from a chromosome S is simply:

$$P(r|S) = \frac{a_{r,S} + a_{r,rc(S)}}{2(|S| - l + 1)},$$

where $a_{r,S}$ stands for the number of occurences of the read r as a (contiguous) substring of strand S. Formally,

$$a_{r,S} := |\{i \in \{1, \dots (|S| - l + 1)\} \mid S[i : i + l - 1] = r\}|$$

Then the probability of observing a read set $R = \{r_1, \dots, r_m\}$ from the chromosome S is simply the product of individual probabilities:

$$P(R|S) = \prod_{i=1}^{m} P(r_i|S) = \prod_{i=1}^{m} \frac{a_{r_i,S} + a_{r_i,rc(S)}}{2(|S| - l + 1)}$$

Term $a_{r,S}$ could be also be written as a sum over Iverson brackets (Iverson bracket $[A]$ is equal to 1 if condition A is true, and is equal to 0 otherwise):

$$a_{r,S} := \sum_{i=1}^{n-l+1} \Big[r = S[i : i + l - 1]\Big]$$

We can interpret i-th term of the sum as a level of our belief that, if we start sequencing from the i-th position of the strand S, we would generate the read r. Due to our second and fourth assumptions, this term could only be either 0 or 1.

But, if we abandon the second assumption and allow sequencing errors, the read could potentially be obtained from any position in the genome. Ghodzi et al. modelled this belief as $\varepsilon^q(1 - \varepsilon)^m$, where m and q are the numbers of matches and errors (insertions, deletions and substitutions) respectively in a best alignment of read r to the respective subsequence of the chromosome[2], and ε is a chance of error. This implies that the probability of each type of error is the same. We will denote this belief as a $b_i(r, S)$. Using this notation, the probability of observing a read r from a chromosome S is equal to:

$$P(r|S) = \sum_{i=1}^{|S|} \frac{b_i(r, S) + b_i(r, rc(S))}{2|S|}$$

Note that now every base of the chromosome may be a starting position, although not very believable. We have also simplified $2(|S| - l + 1)$ to $2|S|$ as $l - 1$ is typically negligible compared to the chromosome size $|S|$.

[2]the q is essentially a Levenshtein distance between the read and a respective subsequence

We are now ready to abandon our first assumption, and examine the situation where the genome A consists of several chromosomes $S_1, S_2, \ldots, S_n$. Due to our third assumption, each read could be generated from any chromosome with the probability, proportional to its length:

$$P(r|A) = \sum_{k=1}^{n} P(r|S_k) \cdot \frac{|S_k|}{\sum_{j=1}^{n} |S_j|} = \frac{\sum_{k=1}^{n} P(r|S_k) \cdot |S_k|}{\sum_{j=1}^{n} |S_j|} =$$

$$= \frac{1}{2\sum_{j=1}^{n} |S_j|} \sum_{k=1}^{n} \sum_{i=1}^{|S_k|} b_i(r, S_k) + b_i(r, rc(S_k))$$

Notice that term $|S_k|$ has disappeared from the inner summation part of the equation.

Now we are ready to extend this model to the paired reads. We assume that a paired read is a pair of a prefix and a reverse complement of a suffix of an unknown single read. We further assume that both parts of a paired read have the same fixed length l, and that the length of the original unseen single read (we will refer to it as *insert length*) has the distribution p_{insert}. In the GAML framework, p_{insert} is set to be a normal distribution and the parameters μ and σ^2 are estimated from the data.

The probability of obtaining a paired read $r = (r_1, r_2)$ from a chromosome S is therefore:

$$P(r|S) = \sum_{i=1}^{|S|} \sum_{j=1}^{|S|} \left(\frac{b_i(r_1, S)}{2|S|} b_j(r_2, rc(S)) \tilde{p}_{orient}(i, j) + \frac{b_i(r_1, rc(S))}{2|S|} b_j(r_2, S) \tilde{p}_{orient}(j, i) \right),$$

where $\tilde{p}_{orient}$ is equal to p_{insert} iff r_1 and r_2 are well-aligned (i.e. the prefix comes before the suffix), and 0 otherwise. Formally,

$$\tilde{p}_{orient}(i, j) := \begin{cases} p_{insert}(j + l - 1 - i) & \text{if } i \leq j + l - 1 \\ 0 & \text{otherwise} \end{cases}$$

In cases of several reads and several chromosomes, paired reads behave in the same way as single reads.

As was told earlier, one of the advantages of this approach is that we can use several datasets at the same time. Assuming read sets R_1, R_2,... R_k are conditionally independent with respect to an assembly A, the joint probability is a simple product:

$$P(A|R_1, \ldots R_k) = \frac{P(R_1, \ldots R_k|A) \cdot P(A)}{P(R_1, \ldots R_k)} = \frac{P(R_1|A) \ldots P(R_k|A) \cdot P(A)}{P(R_1, \ldots R_k)}$$

We can omit $P(R_1, \ldots R_k)$ as it remains constant during optimisation process.

We did not specify apriori distribution of an assembly $P(A)$ in this section, because in the original GAML framework it is set to be a constant, essentialy reducing the problem to maximising a likelihood function (hence the name of the framework: "Genome Assembly by Maximum Likelihood").

1.3.2 Approximation of the probabilistic model

The obvious problem of this model is that in order to evaluate it we have to align every read of a dataset to each position of each chromosome, which is immensely time-consuming. It is also obvious, that each read has only a few places, where it could align well, therefore for most of positions the belief score would be negligible. For example, a read of length 300 bp is expected to have $300 - \frac{300}{4} = 225$ substitution errors if aligned with a random part of a DNA sequence. For the error rate ε set to 0.01 it would gave a belief score of $0.01^{225} \cdot 0.99^{75} \approx 0.47 \cdot 10^{-450}$. Even if we will sum this score among 1 billion positions, it would give us the total score equal to 10^{-441} (which is way below the limit of double-precision floating point variables). In comparison with the belief score of a single perfect matching $0.99^{300} = 0.049$ it is truly negligible.

Therefore, the total probability is evaluated not for every position, but only where a good alignment exists. A good alignment for a single read is defined as an alignment with fewer than `max_err` errors of any type. For purposes of this section, we define an alignment of a single read r to a strand S as a tuple $a = (p, m, q)$, where p stands for the starting position on the strand, m stands for the number of matches and q stands for the number of errors in the alignment. We may refer to the components of an alignment a as a_p, a_m and a_q respectively. We denote a set of good alignments of a single read r to a strand S as $AL(r, S)$. We will also denote our model for observing a single read from a given position as $p_{single}(a) := \varepsilon^{a_q}(1-\varepsilon)^{a_m}$.

Using this notation, we are now ready to write down an approximation of a probability of a single read r being observed from a chromosome S:

$$P(r|S) \approx \frac{1}{2}\left(\sum_{a \in AL(r,S)} \frac{p_{single}(a)}{|S|} + \sum_{a \in AL(r,rc(S))} \frac{p_{single}(a)}{|S|}\right) = \\ = \frac{1}{2|S|} \sum_{a \in AL(r,S) \cup AL(r,rc(S))} p_{single}(a)$$

To reduce the bulkiness of formulas, we will introduce some additional notations. We denote a set of alignments of a single read r to both strands of a chromosome S as $BAL(r, S)$. Formally,

$$BAL(r, S) := AL(r, S) \times \{0\} \cup \{(|S| - p + 1, m, q) | (p, m, q) \in AL(r, rc(S))\} \times \{1\},$$

where $\times$ denotes the Cartesian product of two sets.

Informally, we added the information about whether the read was aligned to the direct or reverse strand as the fourth value in the tuple (we will refer to it as a_o), and unified the position of the alignment. This notation allows to evaluate the insert size and mutual orientation of reads more simply.

The approximation of a probability of a paired read $r = (r_1, r_2)$ being observed from a chromosome S is then:

$$P(r|S) \approx \frac{1}{2|S|} \sum_{\substack{a_1 \in BAL(r_1,S) \\ a_2 \in BAL(r_2,S)}} p_{single}(a_1) p_{single}(a_2) p_{orient}(a_1, a_2),$$

where $p_{orient}(a_1, a_2)$ is equal to p_{insert} if the mutual orientation of the two alignments is correct, and zero otherwise. Formally,

$$p_{orient}(x, y) := \begin{cases} 0 & \text{if } x_o + y_o \neq 1 \vee y_p - x_p < l \\ p_{insert}(y_p - x_p) & \text{otherwise} \end{cases}$$

In case, that a read has no good alignments, this formula would raise zero, thus reducing the total result to zero too. To prevent that, we restrict the probability $P(r|A)$ of each read to be at least e^{c+kl} for a single read and $e^{c+k(l_1+l_2)}$, where l is the length of a single read, l_1 and l_2 are the lengths of left and right parts of a paired read, and k and c are the scaling constants. Ghodzi et al. [14] proposed this scoring system as an analogy to the case that the read would be added as a new contig to an assembly.

1.3.3 Simulated annealing

The simulated annealing [15] is a metaheuristic method for finding a global optimum of a given target function (in our case, the posterior probability of the assembly). This approach is an extension of a simpler method, called *hill climbing* or *local neighbourhood search.*

The idea of hill climbing is simple: First, we define a relation called *neighbour* (hence the name of the method) between elements of the space of all possible solutions (in our case, all genome assemblies). All points that are neighbouring with a given point, are called *neighbourhood* of the given point. Second, we choose a starting (initial) point in the searched space. Third, we iteratively look at the neighbouring points of the current point, evaluate their target function and choose the best one. If no point in the neighbourhood is better than the current point, we stop and return the current point as the result. If the defined neighbourhood is too big to evaluate at each iteration, one can pick a point from the neighbourhood at random and see if it is better. In that case, the algorithm is usually terminated after a certain number of iterations.

The drawback of the hill climbing method is that there is no guarantee that the result is a global optimum (not just a local one). The tendency of hill climbing to end in a local minimum near the starting point is addressed (although still with no guarantees) by simulated annealing. The difference between the two methods is that in the latter, it is allowed to choose a worse solution if no better is available. The probability of transition to the worse solution depends on the difference of the target function

values — bigger loss means lower chance of the transition. The probability of picking a worse solution is also governed by a parameter called *temperature* — higher the temperature means greater chance of transition to a worse solution. The temperature is usually set to be very high at the beginning, and then gradually decreases until it eventually reaches zero (this process is therefore being referred to as *a cooling schedule*). This method is based on an analogy with annealing in metallurgy, a technique to eliminate defects in a material by heating it up and then cooling in a controlled manner.

There are many ways to choose the transitional probability and the cooling schedule. In GAML framework, the temperature of n-th iteration is chosen as

$$T(n) := \frac{T_0}{\log\left(1 + \frac{n}{d}\right)},$$

where d is a constant called divisor, which governs the pace of cooling, and T_0 is an initial temperature. The transitional probability from a point x to a point x' (for a maximisation) is then defined as

$$P_{transit}(x, x') := \begin{cases} 1 & \text{if } f(x') > f(x) \\ \exp\left(\frac{f(x')-f(x)}{T}\right) & \text{otherwise} \end{cases}$$

1.3.4 GAML implementation details

In GAML framework, the searched space for simulated annealing is defined as a multiset of all finite walks (i.e. sequences of coincident[3] nodes) in a de Bruijn graph, constructed from input reads by Velvet assembler. Velvet removes nodes that likely correspond to sequencing errors and merge nodes that are likely consequent in the genome. Therefore, nodes of the graph may correspond to longer DNA sequences. GAML obtains both reads and the de Bruijn graph as its input. The initial assembly is chosen as a set of walks of length 1, consisting of nodes with corresponding DNA sequences that are longer than 500 bp. Such nodes are called *big* or *long* (the rest of nodes are called *small* or *short*).

We will use terms *paths* and *walks* interchangeably. Both terms would correspond to the common meaning of a graph walk, i.e. a sequence of coincident nodes, that are allowed to have multiple occurrences in the sequence.

The neighbourhood of a multiset of paths (walks) is defined by the allowed set of changes on them, called moves (e.g. split one path into two, join two paths, etc). These moves are described thoroughly in chapter 2.

The target function (or score) of an assembly is defined as a logarithm of posterior probability of the assembly with respect to input read sets, divided by the total length

[3]Nodes are coincident iff they are connected by an edge in a graph

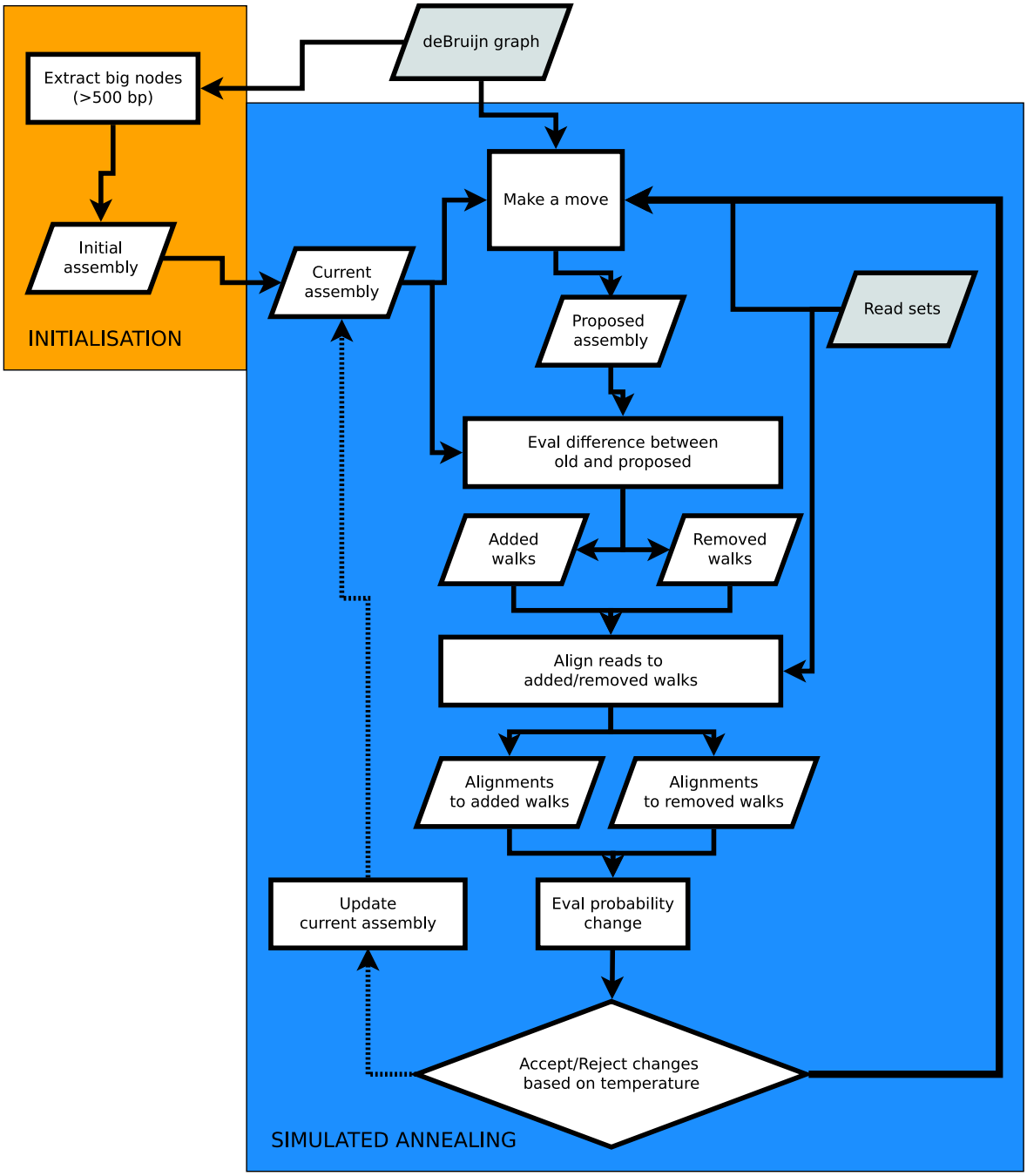


Figure 1.4: GAML program flow

of the reads. Formally,

$$f_{score}(A, R_1, \ldots, R_k) := \frac{\log\Big(P(R_1|A)P(R_2|A)\ldots P(R_k|A)P(A)\Big)}{\sum_{i=1}^{k}\sum_{r\in R_i}|r_i|}$$

The normalisation by the total length of input reads has no impact on the topology of the searched space, but it is convenient for tuning of the cooling schedule parameters.

We will call the contribution of individual paths (walks) of an assembly to the resulting total score as to *path's score*.

The computation flow of GAML is depicted in figure 1.4. As we have mentioned earlier, the initial assembly is created from sufficiently big nodes of an input de Bruijn graph. In the beginning of each iteration, a random type of change (move) is picked and applied to the current assembly, producing the new assembly. If the picked type of change fails to produce a viable move, then new changes are generated until we would obtain a viable one (without the change of the temperature, as it is still the same iteration).

It is crucial not to evaluate the total score of the new assembly from scratch in each iteration, as the aligning of reads to the paths is the most time-consuming operation. Instead, we compare the old and the new assemblies and determine the difference between them, i.e. added and removed paths. Then we evaluate the alignments for these paths. Based on the obtained alignments, we evaluate the score difference between the old and new assemblies.

If the score difference is positive (i.e. the new assembly is better than the old), we accept the change, store the new assembly as the current and move on to the next iteration. If the score difference is negative (i.e. the new assembly is worse than the old), we decide whether to accept it based on the temperature of this iteration (as it has been described above in subsection 1.3.3) and then move on to the next iteration.

1.4 Chromatin interactions in genome

DNA molecules in a nucleus are not stored as linear molecule, but in a very compact structure, called *chromatin*. In order to study this structure, scientists evolved several methods ([3, 16–19]) to capture fragments of DNA that are at close physical proximity in a nucleus. We will briefly describe the Hi-C method by Lieberman-Aiden et al. [3] (figure 1.5).

First, the DNA is treated by formaldehyde to crosslink fragments of DNA that are spatially close. Second, the segments of DNA, connected by formaldehyde, are cut out from the rest of the DNA structure. Third, the ends of these two segments are "glued" (ligated) together to form a single double-stranded DNA molecule. Fourth, the regions

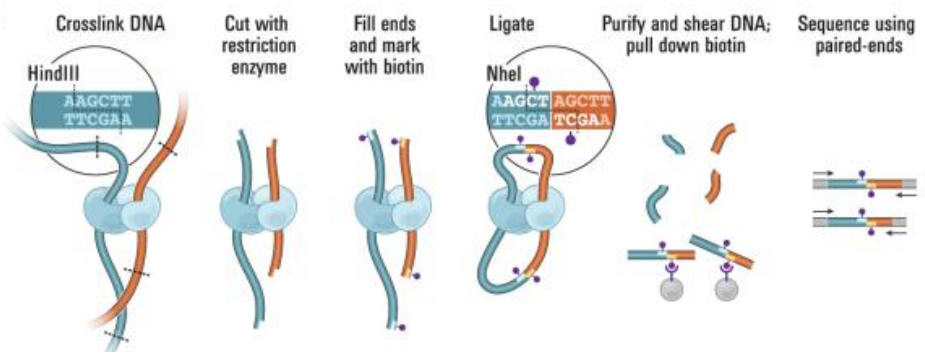


Figure 1.5: Chromatin interaction capturing

near the ligation joints are cut out and both ends of resulting segments are sequenced by Illumina technology, producing paired reads.

The resulting parts of the paired reads come from spacially close regions of the DNA. By aligning these reads to the genome, one can evaluate mutual interaction rate (i.e. the number of aligned reads) between regions (e.g. 100 kb long segments) of the genome. The interaction rates closely correlate with physical proximity, and therefore are often called *contact maps*. These contact maps are essential for devising and verifying models of macrolevel structure of DNA molecules in nuclei.

Later, Kaplan and Dekker in [1] adapted this type of data for genome scaffolding. They established two fundamental patterns that Hi-C reads exhibit: *cis-trans-ratio* (CTR) and *distance-dependent decay* (DDD). The CTR pattern means that interaction rate between segments that are on the same chromosome (*cis*-segments) is much higher than between two segments from different chromosomes (*trans*-segments). The DDD pattern means that the interaction rate between two segments on the same chromosome is roughly exponentially decreasing with the genomic distance between them.

In their article [1], the authors used Hi-C data in two tasks: adding a small amount of contigs to the mostly completed assembly, and scaffolding of contigs during de novo assembly. We will discuss their approach to the latter task.

The input for a task is a set of contigs, and a dataset of Hi-C paired reads. The scaffolding runs in three phases: evaluation of interaction rates, clusterisation to obtain putative chromosomes and ordering contigs in the individual clusters.

Evaluation of interaction rates First, Hi-C reads are aligned to the contigs, each part of the paired reads separately. Second, the interaction rate is evaluated between every pair of contigs. We will denote the number of interactions (the number of well-aligned trans-reads) between two contigs S_i and S_j as $h(S_i, S_j)$.

Clustering The goal of this phase is to determine the number of chromosomes and also assign individual contigs to them. This goal is achieved by employing hierarchical

agglomerative clustering [20], utilising the CTR pattern, i.e. contigs that belong to the same chromosome should have higher interaction rates. The resulting clusters, formed of the contigs, correspond to the chromosomes.

The clustering method requires a definition of a distance between two clusters to work. The distance between two contigs S_i and S_j is chosen as a logarithm of the number of trans-chromosomal reads between them, flipped from similarity measure to distance by subtracting the data from the maximum value among all pairs of contigs:

$$D(S_i, S_j) := \log(1 + max_{p \neq q} h(S_p, S_q)) - \log(1 + h(S_i, S_j))$$

Constant 1 is added to avoid undefined $\log 0$.

The distance $d(C_1, C_2)$ between the two clusters C_1 and C_2 is defined as an average distance between pairs of contigs in them:

$$dist(C_1, C_2) := \frac{1}{|C_1| \cdot |C_2|} \sum_{S_1 \in C_1, S_2 \in C_2} D(S_1, S_2)$$

Ordering of the contigs in the clusters The goal of this phase is to order the contigs in each cluster to create the resulting chromosomes. This phase utilises the DDD pattern, i.e. the interaction rate between the contigs should be proportional (not necessary linearly) to their distance in the chromosome. The goal is achieved by multidimensional scaling [21].

The method works in the following way. First, we evaluate a so-called *dissimilarity matrix* D, where $D_{i,j}$ denotes the dissimilarity (or distance) between i-th and j-th contig in the cluster, based on their interaction rates. Second, we find a set of points in a one-dimensional Euclidean space (i.e. a line), which satisfies the dissimilarity matrix the best (i.e. their distance matrix is as close as possible to the dissimilarity matrix of the contigs). Third, we use the resulting positions of points on the line as the ordering for the contigs on the chromosome.

Kaplan and Dekker defined the distance (dissimilarity) and the measure of quality of their resulting solution in a rather convoluted way, based on their probability model. They tested this methods on data from a human genome, and the experiments were very promising.

Chapter 2

Enhancements of GAML2 framework

The original GAML implementation [2] was a prototype not designed for extendability. Boža had started the work of rewriting GAML, naming the new implementation GAML2 [22]. Unfortunately, his work was not finished, and GAML2 has been missing several key features. Nevertheless, we have decided to use this incomplete implementation as our basis.

In this chapter, we will describe our work on reimplementing and enhancing GAML2. Because of the incomplete documentation of GAML, some features were redesigned almost from scratch. It is important to mention, that most of design decisions were resolved in favor of time and memory efficiency.

2.1 Better probability model for single reads

As we have described in section 1.3, Ghodzi et al. [14], as well as Boža [2] used $\varepsilon^q(1-\varepsilon)^m$ for modelling the probability of an alignment with m matches and q errors. This formula raises from the generative representation of the sequencing process. At each step, the sequencer either will read the current basepair correctly with the probability equal to $(1-\varepsilon)$, or will commit an error (substitution, insertion or deletion) with probability equal to ε. We have two objections to this formula.

First, if the chance of commiting each type of error is the same (which both articles assumed), then the chance of producing individual type of error is equal to $\frac{\varepsilon}{3}$. Therefore, the resulting belief should be equal to $\left(\frac{\varepsilon}{3}\right)^q(1-\varepsilon)^m$.

Second, reads produced by Illumina technologies have a much lower chance of insertion and deletion errors than substitutions. Therefore it makes sense to assign individual error rates ε_i, ε_d, and ε_s for each respective type of error. The resulting probability will then have following form:

$$b_{new}(a) := \varepsilon_i^{a_i}\varepsilon_d^{a_d}\varepsilon_s^{a_s}(1-4\varepsilon_i-\varepsilon_d-3\varepsilon_s)^{a_m},$$

where a_i denotes the number of insertions, a_d denotes the number of deletions, a_s denotes the number of substitutions and a_m denotes the number of matches in the alignment a.

To accommodate this extension, we had to extend the aligner algorithm to be able to report this data. Read alignment is done in two phases. First, at the beginning of the program, all reads are processed by storing each k-mer (default k is set to 13) into a hash table. Second, during the search of alignments for a given contig, each k-mer of the contig is looked up in the hash table and used as a seed for aligning reads containing this k-mer to the respective position of the contig. This corresponds to finding exactly matching k bases between a read and the contig.

Alignment is done by greedy 0-1 BFS in both directions from the position of the seed. We augmented the output with the number of insert, deletion and substitution edges along the best found alignment.

The drawback of this approach is that aligner does not prefer one type of error over other (because each edge has the weight equal to either 0 or 1). We decided to keep the algorithm instead of expanding it to the full-blown Dijkstra algorithm because of speed requirements.

2.2 Adding apriori probability of an assembly

In our early experiments, the assembler had often returned assemblies that are significantly longer than the target genomes. We have realised that if we would take an assembly $A = (S_1, \ldots, S_n)$ and add a copy of each contig $A^2 := (S_1, S_1, S_2, S_2, \ldots, S_n, S_n)$, the conditional probability for reads would not change, i.e $P(R|A) = P(R|A^2)$:

$$P(R|A^2) = \prod_{r \in R} P(r|A^2) = \prod_{r \in R} \sum_{S_i \in A^2} P(r|S_i) \cdot \frac{|S_i|}{\sum_{S_j \in A^2} |S_j|} =$$

$$= \prod_{r \in R} \left(2 \sum_{S_i \in A} P(r|S_i) \cdot \frac{|S_i|}{2 \sum_{S_j \in A} |S_j|} \right) =$$

$$= \prod_{r \in R} \sum_{S_i \in A} P(r|S_i) \cdot \frac{|S_i|}{\sum_{S_j \in A} |S_j|} = \prod_{r \in R} P(r|A) = P(R|A)$$

That means we have to control the length of assemblies in other way. If we assume that every chromosome of length k has equal probability, then the a priori probability of such a chromosome is equal to 4^{-k} (ignoring the dual representation of two strands DNA sequences). We have used this rudimentary formula for constraining the total

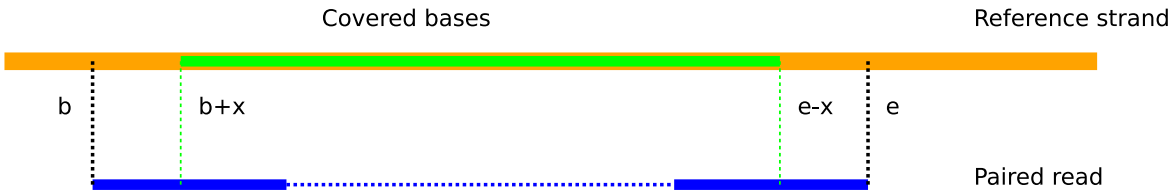


Figure 2.1: Detection of poorly covered bases. Green bases are sufficiently covered.

length of an assembly:

$$P(A) :\approx \prod_{S \in A} 4^{-|S|} = 4^{-\sum_{S \in A} |S|}$$

We acknowledge that this formula does not satisfy the definition of a probability density function, because we do not model the number and lengths of chromosomes. Thus, the sum over all possibilities is not even finite. However, we did not want to add any additional skewness toward any concrete length without any knowledge. It is possible to incorporate the karyotype (i.e. the number of chromosomes) and relative lengths of chromosomes to obtain better solutions.

2.3 Penalty for spuriously joined contigs

Boža in the original GAML framework devised a method for preventing joins of contigs with little support from the data. The method is based on adding a penalty for bases, that are not covered by any reads, so there is no evidence of connection. Because every node of the graph is produced from the reads, and therefore is covered by at least one, we define *poorly covered bases* as bases that are not covered by an *inner part* of any read alignments. Inner part of an alignment with beginning position b and ending position e is defined as an interval $[b + x, e - x]$, where x is a threshold (defined by the user) (figure 2.1).

Formally, assume that we have a contig $S = (s_1, \ldots, s_n)$, a set of begins and ends of (paired) read alignments $AL = \{(b_1, e_1), (b_2, e_2), \ldots, (b_m, e_m)\}$ and a threshold x. Then we define poorly covered base as such base s_i that there is no alignment, such that its inner part contains (covers) the given base s_i:

$$\text{is\_poorly\_covered}(i) := \neg\exists (b_k, e_k) \in AL : b_k + x \leq i \leq e_k - x$$

The enumeration of poorly covered bases for the chromosome S of length n and a read set R with m alignments can be done trivially in $O(n \cdot m)$ time complexity. However, this could be improved to $O(n + m)$ time complexity by employing the idea of prefix sums.

We start with an array A of length $n + 1$, filled with zeros. Then, for each interval (b_k, e_k) we increase $A[b_k + x]$ by one and decrease $A[e_k - x + 1]$ by one. Finally, we

evaluate the prefix sums P for the array A, formally $P[i] := \sum_{j=1}^{i} A[j]$. This is achieved in linear time, thanks to this simple recurrence: $P[i] = P[i-1] + A[i]$. The array P then holds the amount of covering reads for each base.

It is expected of bases on the ends of contigs to be poorly covered, as their covering reads are not yet aligned due to contig split. We therefore ignore the first and last y bases of every contig (the threshold y is set by the user).

The resulting probability $P(R|A)$ is then penalised by a factor α^{-d}, where d is the number of poorly covered bases (by the dataset R) and α is the penalty constant.

2.4 Support of paired end reads

The support of paired reads was implemented in the original GAML, but GAML2 lacked it. Due to architecture changes in GAML2, we have decided to implement this feature in a different way.

We treat the paired read set as a pair of single read sets (referred to as *left* and *right read sets*) with additional information about insert size distribution and expected orientation of left and right reads. The mathematical foundation was described in the section 1.3.

The alignment of the paired reads to individual contigs worked in the following way. First, we aligned both sets of single reads separately to a contig (we refer to this alignments as to *partial* alignments). Second, we sorted the alignments by the read identificator. This allowed us to efficiently "merge" both arrays of partial alignments by the read identificators and check for correct mutual orientation of merged pairs (see section 1.3).

2.5 Existing moves in simulated annealing

We will start by briefly describing the two types of moves that were already implemented in GAML2.

2.5.1 "Break" move

A random path from the current assembly is chosen. This path is then split at a randomly chosen node. The resulting paths are also stripped of short nodes (nodes representing shorter than 500 bp long sequences, see section 1.3.3 for details) to preserve the invariant that every path starts and ends with a big node. This move was left unchanged.

2.5.2 "Random extend" move

A random path from the current assembly is chosen. Then we start a random walk from one of its ends, until we hit a big node or the upper limit for extension length is exceeded. Random walk works in the following way: from the starting position, we uniformly randomly choose an outgoing edge and repeat the same process. If we have hit a big node that is an ending node of a path, we join the chosen path and the found one. If the found big node is not the end of any path, it is appended to the extended path with probability v (the probability v is set by the user) and with probability $(1 - v)$ we declare the move unsuccessful. We have added the parameter v because the old version led to unsupported overgrowing of the resulting assembly, as the same node was reused in many paths.

2.6 "Join with advice" move in the simulated annealing

This move was implemented in the original GAML and was considered the main move for scaffolding using long-range data (e.g. paired reads or PacBio reads). However, this move was missing in GAML2.

We start by randomly picking any path from the current assembly as a starting path. Next, we exclude paths that have intersecting ends with the starting path (i.e. last h bases of the starting path is the same as first h bases of another path) from our set of candidate paths to join. This is made mainly to simplify the process of paths' joining (paths that have common nodes are handled separately by "untangle crossed paths" move). We then align (or rather load the alignments from the cache, see section 2.8) all left and right parts of paired reads separately to the starting and candidate paths.

The caching of partial alignments is essential here, because otherwise we would have to realign each read to each path in the current assembly in each iteration. This would be a waste of computational resources, considering the fact that most of the assembly remain unchanged between consecutive iterations.

The number of reads that have one part aligned to the starting path and another part to the candidate defines a score for this candidate path. We use this score to randomly choose the target path. Formally, if a candidate path p_i has z_i reads, aligned partially to the target path and partially to the candidate path, then the probability of choosing the path p_i as a target is equal to $\frac{z_i}{\sum_j z_j}$.

We then sample random walks between the ends of the starting and target paths (random walks are sampled in the same way as in "Random extend" move). To speed

up the process, we first find all nodes, that could lead to the target path's ends, by the standard BFS algorithm and then restrict the random walks to that nodes.

We finally choose a random walk with the best score among sampled as our result. The original paths, both target and starting, are discarded from the assembly.

2.7 "Untangle crossed paths" move in the simulated annealing

This move was implemented in the original GAML, but not in GAML2. Our implementation is loosely based on the GAML version, as the original move was described in a rather laconic way.

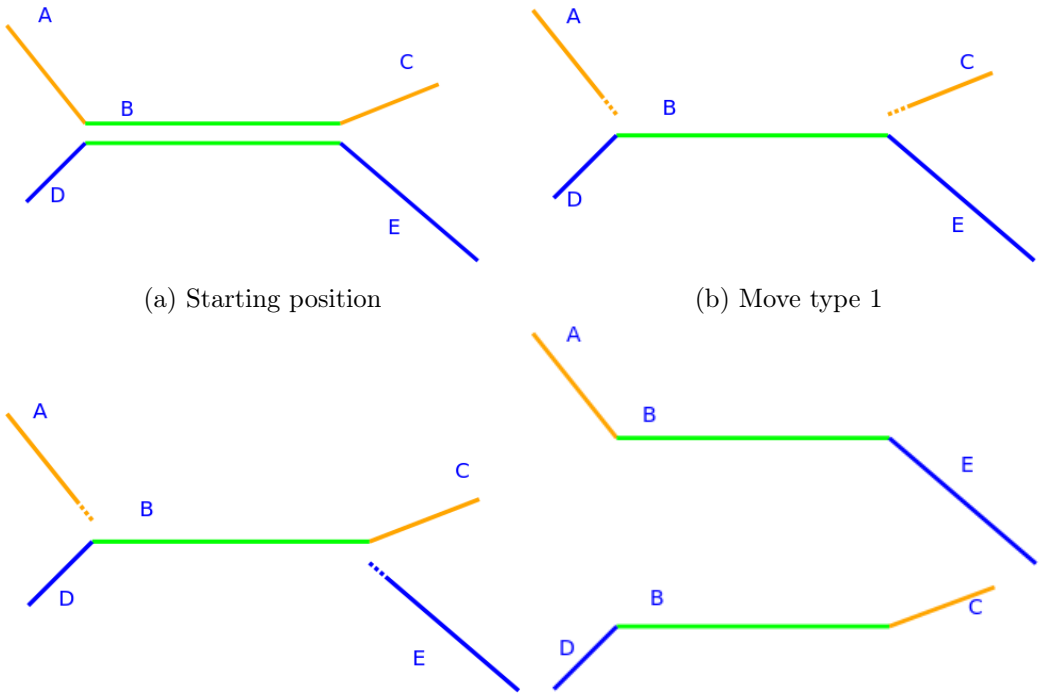
This move targets paths that have common nodes (figure 2.2a). Such situations mostly occur as artifacts from "random extend" move, but can correspond to genuine repeats.

First we randomly pick a path from the current assembly. Second, for every node in the picked path we compute their occurrences in other paths of the current assembly and store them in a hash table. Third, we randomly pick a node that belongs to two or more paths. This way we assure that each joint is picked proportionally to its length in node count (i.e. if two paths have common path of length 30, then there are 30 nodes that could be picked). Fourth, if the node belongs to more than one other path, we randomly pick the second path to untangle.

We then compute the length of the common subpath for these two paths. We will denote the first path as $A - B - C$ and the second as $D - B - E$, where B is the common subpath (the longest common subpath containing the picked node) (figure 2.2a). We now have several possibilities:

- delete the middle part B from one of the paths, trim small nodes from the ends of the residual paths (figure 2.2b), obtaining paths $A-$, $-C$ and $D - B - E$ (or $D-$, $-E$ and $A - B - C$)
- create a new path $A - B - E$ (or $D - B - C$) as a combination of the input paths and keep trimmed residual paths $D-$ and $-C$ ($A-$ and $-E$) (figure 2.2c)
- switch residuals to create paths $A - B - E$ and $D - B - C$ (figure 2.2d).

We evaluate the probability change for each move and then return the best one as a result.



(a) Starting position

(b) Move type 1

(c) Move type 2

(d) Move type 3

Figure 2.2: Change types for "Untangle crossed paths" move.

2.8 Caching of evaluated alignments

One of the key enhancements of the speed in GAML was caching of already evaluated alignments. However, such caching was not implemented in GAML2. The idea is to store the alignments for big nodes (nodes representing at least 500 bp long sequence, see section 1.3.3 for details) and then evaluate alignments for a path by aligning read to the joints and extracting cached alignments for the rest of the path. As the original caching algorithm was considered too complex to implement, we have devised less efficient, but simpler (and therefore less prone to implementation errors) solution.

We store all full and partial (see section 2.4) alignments *for whole paths* in a hash table. This way, we have to reevaluate alignments for a path, created by altering of the old paths, from scratch, but we still save computational time for example during "Join with advice" move (see section 2.6).

We had several possibilities for hash function for paths, e.g. linear hashing (where the sequence of the path's nodes is taken as a sequence of coefficients of a polynomial over a finite field, and the resulting hash is a value of this polynomial for a fixed (prime) element of the given finite field). Hashing functions of this type are usually used for string hashing.

We have in the end settled for the identificator of the first node in the path as the path's hash. Such a hash function is very computationally efficient and at the same time produces almost no collisions, because contigs are not expect to start with the

same node (and if such paths emerge, they are likely to be altered by "untangle crossed paths" move).

2.9 Improved simulated annealing

In this section we describe three improvements of the simulated annealing in GAML2. The last of the three, namely cleaning of an assembly, was implemented in the original GAML, but not in GAML2. Our implementation of this feature is simpler than as the original version. We are not aware of any features in the original GAML similar to the first two modifications.

Setting the temperature to zero at the end Because of the cooling schedule chosen by Boža, the temperature never reaches zero, therefore always leaving a possibility of worsening the resulting assembly. To prevent that and reach a local minimum, we keep the temperature at constant zero at the end to compensate for possible decrease in quality score in moves towards the end of the search, thus effectively reducing the algorithm to *hill climbing* (see subsection 1.3.3).

Adaptive choice of moves At the beginning of each iteration of the simulated annealing, each move type has a predefined probability of execution. The problem is that not every move type can always produce a viable solution's change (e.g. there are no tangled paths). If a move fails to produce a candidate, a new move is sampled in the same iteration (see subsection 1.3.4 for details). As some types of moves are computationally heavy, having them running many times in the same iteration with no expected result is wasteful. Therefore, we decrease the probability of picking a move type after its failure during one iteration. These probabilities are restored back to the original settings for the next iteration.

For example, let $p = (5, 5, 5, 5)$ be the original probability scores for four move types denoted B, E, J and U. We evaluate probability for move type i as $\frac{p_i}{\sum_j p_j}$, therefore the initial probabilities of each move at the beginning of each iteration of simulated annealing are equal to $\frac{5}{5+5+5+5} = \frac{1}{4}$. Assume that J move was picked, but produced no viable change. We then decrease the score for the unsuccessful move type by one, changing the probability scores to $p' = (5, 5, 4, 5)$. Now the probability of choosing J move type is equal to $\frac{4}{19}$ (as opposed to $\frac{5}{19}$ for the other three move types). Assume that U move was picked next, and produced a viable change. We then return the change as a result, and in the next iteration the probability scores will be set back to the original $(5, 5, 5, 5)$.

Final cleaning of an assembly after the simulated annealing After the finish of the simulated annealing, we run several iterations of "break" move with very low temperature (the number of iterations is set by the user). This helps to identify and disconnect contigs that have little evidence for being joined.

Chapter 3

Usage of Hi-C data in GAML framework

In this chapter, we will describe our work on adding Hi-C data processing into GAML2. We build on our extended implementation of GAML2, described in the previous chapter. The probabilistic model of Hi-C data, described in this chapter, is inspired by the model from [1], but is adapted to fulfill the need of the GAML framework.

3.1 The probability model for Hi-C data

Each Hi-C read is a paired read, sequenced by Illumina. Some key assumptions, used in modeling paired reads, still hold: each read is generated independently from the same distribution, and the probability of generating a single part of a Hi-C read is the same as for Illumina reads.

There is a major difference from common Illumina reads though: the left and right parts of a Hi-C read could be generated either from the same chromosome (such Hi-C reads are called *cis*-reads) or from two different ones (such Hi-C reads are called *trans*-reads). Furthermore, there are usually a lot more of cis-reads, but the exact ratio varies between species (see the DDD pattern in section 1.4).

We denote the probability that a Hi-C read is generated from the same chromosome as p_{cis}. We denote the probability of a given Hi-C read r being generated as a cis-chromosomal read from a given assembly A as $P_{cis}(r|A)$. Similarly, $P_{trans}(r|A)$ stands for the probability of being generated as a trans-chromosomal read. The total probability of the Hi-C read r being generated from the assembly A is then equal to:

$$P_{hic}(r|A) = p_{cis} \cdot P_{cis}(r|A) + (1 - p_{cis}) \cdot P_{trans}(r|A)$$

In the rest of the section, we will define $P_{cis}(r|A)$ and $P_{trans}(r|A)$, and then discuss the effective way of updating the probability for a new assembly.

3.1.1 The probability model for cis-reads

The cis-reads are very similar to the common paired reads, described in section 1.3. There are however two differences. First, the insert lengths have exponential distributions, specific for each chromosome (see section 1.4 for details). Second, the orientation of the individual parts of the paired Hi-C reads is less strict: both reads could be sampled from the same strand of a chromosome. Instead of devising a new formula, we will adapt the formula for the paired reads from the section 1.3. We show it here once more:

$$P_{paired}(r|S) \approx \frac{1}{2|S|} \sum_{\substack{a_1 \in BAL(r_1,S) \\ a_2 \in BAL(r_2,S)}} p_{single}(a_1) p_{single}(a_2) p_{orient}(a_1, a_2)$$

Let's start with the former difference. We need to change the p_{insert} distribution to model the exponential distribution instead. The probability density function of an exponential distribution is:

$$f_{exp}(x;\lambda) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

where λ is its only parameter. The parameter λ is at the same time equal to an inverse of the expected value of this distribution, and therefore can be estimated from the mean of the data (in our case, insert sizes of all alignments of reads R to a particular contig S):

$$\hat{\lambda}_{MLE}(R,S) := \frac{\sum_{(r_1,r_2)\in R} |BAL(r_1,S)| \cdot |BAL(r_2,S)|}{\sum_{(r_1,r_2)\in R} \sum_{\substack{a^{(1)} \in BAL(r_1,S) \\ a^{(2)} \in BAL(r_2,S)}} \left| a_p^{(1)} - a_p^{(2)} \right|}$$

The change in read orientation handling is very easy to incorporate. It is enough to omit the respective condition from the definition of the p_{orient} function from the formula for common paired reads.

The resulting approximation of the probability of observing a Hi-C paired read $r = (r_1, r_2)$ from a Hi-C read set R as a cis-read from chromosome S is:

$$P_{cis}(r|S) \approx \frac{1}{2|S|} \sum_{\substack{a^{(1)} \in BAL(r_1,S) \\ a^{(2)} \in BAL(r_2,S)}} p_{single}\left(a^{(1)}\right) p_{single}\left(a^{(2)}\right) f_{exp}\left(|a_p^{(1)} - a_p^{(2)}|; \hat{\lambda}_{MLE}(R,S)\right)$$

3.1.2 The probability model for trans-reads

We denote as $P_{trans}(r|S_1, S_2)$ the probability of observing a Hi-C paired read $r = (r_1, r_2)$ as a trans-read from chromosomes S_1 and S_2, where r_1 is obtained from the S_1 and r_2 is obtained from the S_2. Analogically to single reads, we approximate our belief

that the left read r_1 was generated from the S_1 as $\sum_{a_1 \in BAL(r_1,S_1)} \frac{p_{single}(a_1)}{2|S_1|}$ (the right read r_2 is evaluated in the same way), and therefore the resulting approximation of the probability $P_{trans}(r|S_1, S_2)$ is:

$$P_{trans}(r|S_1, S_2) \approx \sum_{\substack{a_1 \in BAL(r_1,S_1) \\ a_2 \in BAL(r_2,S_2)}} \frac{p_{single}(a_1)}{2|S_1|} \frac{p_{single}(a_2)}{2|S_2|}$$

We assume that the probability for a Hi-C read to be produced from a particular pair of chromosomes among an assembly is proportional to their lengths:

$$\begin{aligned} P_{trans}(r|A) &\approx \sum_{1 \le i < j \le |A|} \frac{|S_i| \cdot |S_j|}{\sum_{1 \le u < v \le |A|} |S_u| \cdot |S_v|} \left(\frac{1}{2} P_{trans}(r|S_i, S_j) + \frac{1}{2} P_{trans}(r|S_j, S_i) \right) = \\ &= \frac{1}{2 \sum\limits_{1 \le u < v \le |A|} |S_u| \cdot |S_v|} \sum_{1 \le i < j \le |A|} (P_{trans}(r|S_i, S_j) + P_{trans}(r|S_j, S_i)) \cdot |S_i| \cdot |S_j| = \\ &= \frac{1}{8 \sum\limits_{1 \le u < v \le |A|} |S_u| \cdot |S_v|} \sum_{1 \le i < j \le |A|} \left(\sum_{\substack{a_1 \in BAL(r_1,S_i) \\ a_2 \in BAL(r_2,S_j)}} p_{single}(a_1) p_{single}(a_2) + \right. \\ &\qquad \left. + \sum_{\substack{a_1 \in BAL(r_1,S_j) \\ a_2 \in BAL(r_2,S_i)}} p_{single}(a_1) p_{single}(a_2) \right) \end{aligned}$$

For the purpose of the following subsection, we will establish a few additional notations. First, we will hide the inner parts of these formulas into following symbols:

$$\begin{aligned} \phi(r, S) &:= \sum_{\substack{a^{(1)} \in BAL(r_1,S) \\ a^{(2)} \in BAL(r_2,S)}} p_{single}\left(a^{(1)}\right) p_{single}\left(a^{(2)}\right) f_{exp}\left(|a_p^{(1)} - a_p^{(2)}|; \hat{\lambda}_{MLE}(R, S)\right) \\ \psi(r, S_1, S_2) &:= \sum_{\substack{a_1 \in BAL(r_1,S_1) \\ a_2 \in BAL(r_2,S_2)}} p_{single}(a_1) p_{single}(a_2) \end{aligned}$$

Bu using this notation, we can rewrite our formulas:

$$\begin{aligned} P_{cis}(r|A) &\approx \frac{1}{2 \sum_{S_i \in A} |S_i|} \sum_{S \in A} \phi(r, S) \\ P_{trans}(r|A) &\approx \frac{1}{8 \sum\limits_{1 \le u < v \le |A|} |S_u| \cdot |S_v|} \sum_{1 \le i < j \le |A|} (\psi(r, S_i, S_j) + \psi(r, S_j, S_i)) \end{aligned}$$

As you can see, the notation greatly simplifies the resulting formulas. Notice that both terms $\phi(r, S)$ and $\psi(r, S_1, S_2)$ depend only on their input parameters r, S and r, S_1, S_2 respectively (ϕ also depend on the whole read set R, but the dataset remains constant during the maximisation process).

We now add more notation to simplify the situation:

$$\Phi(r, A) := \sum_{S \in A} \phi(r, S)$$
$$\Psi(r, A) := \sum_{1 \le i < j \le |A|} (\psi(r, S_i, S_j) + \psi(r, S_j, S_i))$$

Now we are finally able to write down the whole formula for the probability of a Hi-C read set R to be observed from the assembly A:

$$P_{hic}(R|A) \approx \prod_{r \in R} \left(\frac{p_{cis}}{2 \sum_{S_i \in A} |S_i|} \Phi(r, A) + \frac{1 - p_{cis}}{8 \sum_{1 \le u < v \le |A|} |S_u| \cdot |S_v|} \Psi(r, A) \right)$$

We will refer to the multiplication factors (i.e. fractions multoplying Φ and Ψ) as to *cis-* and *transconstant*.

3.1.3 Updating the probability

Assume that we have a Hi-C read set R, an old assembly $A = \{S_1, \ldots, S_c\}$ and a new assembly $A' = \{S'_1, \ldots, S'_{c'}\}$. We will denote added contigs as $N := A' - A$, removed contigs as $D := A - A'$ and kept contigs as $K := A \cap A'$. We assume that the changes are small, i.e. $|N| << |A|$ and $|D| << |A|$.

Because the total conditional probability for the read set is a product of the probabilities for the individual reads and the cis- and transconstant are likely to change, we will reevaluate probabilities for all reads from the read set.

$$P_{hic}(r|A') \approx \frac{p_{cis}}{2 \sum_{S'_i \in A'} |S'_i|} \Phi(r, A') + \frac{1 - p_{cis}}{8 \sum_{1 \le u < v \le |A'|} |S'_u| \cdot |S'_v|} \Psi(r, A')$$

As for the new Φ score, it is relatively straightforward. We just need to remove the Φ scores, corresponding to the cis-alignments for the deleted contigs, and add Φ scores, corresponding to the cis-alignments for the added contigs:

$$\Phi(r, A') = \Phi(r, A) - \Phi(r, D) + \Phi(r, N)$$

The situation with the Ψ score is a bit more convoluted. We need to remove all ψ scores, corresponding to the trans-alignments between deleted contigs, trans-alignments between deleted and kept contigs, and add ψ scores, corresponding to the trans-alignments between added contigs, and between added and kept contigs:

$$\Psi(r, A') = \Psi(r, A) - \Psi(r, D) - \sum_{S_1 \in D, S_2 \in K} (\psi(r, S_1, S_2) + \psi(r, S_2, S_1)) +$$
$$+ \Psi(r, N) + \sum_{S_1 \in N, S_2 \in K} (\psi(r, S_1, S_2) + \psi(r, S_2, S_1))$$

3.2 Implementation details of model evaluation for Hi-C data

Probability score evaluation

As we have discussed in subsection 1.3.4, alignment caching is crucial for efficient probability score evaluation. After each update, we need to align our read set to all new paths and add their score to the result and remove all old alignments, belonging to the removed paths, similarly to the common (paired) reads. The difficulty here is that for trans-reads, we need to consider alignments between pairs of paths, whereas for paired reads we only considered paired alignments within a path. We reduce the computational cost by caching alignments for every left and right read (i.e. parts of Hi-C reads) for each path, and check for trans-alignments only where necessary (see subsection 3.1.3). We also store the Φ and Ψ score for each read, because the cis- and trans-constants are reevaluated for each iteration, and therefore the resulting probability has to be reevaluated from scratch.

Implementing "Join with advice" move

This move is the most useful application of Hi-C data in GAML2, because Hi-C reads hold information about extremely distant segments of chromosomes.

We have decided to use the same method for utilising the advice from Hi-C reads as for common paired reads (see section 2.6). One possible modification is to change the scoring scheme of target path by utilising the clustering metric, developed by Kaplan and Dekker (see section 1.4). This way allows to mimic the clustering method by the simulated annealing. However, we do not need to stick strictly to this metric as the final decision for joining also depends on the de Bruijn graph characteristics, i.e. whether there is a sufficiently short path.

(Non)usage of Hi-C data for detection of spuriously joined contigs

We have decided not to use Hi-C data for detecting poorly covered bases, as we did with paired reads (see section 2.3). The reason is that Hi-C reads have a tendency to span over very long distances, hence possibly masking spurious joints.

Chapter 4

Experimental results

In this chapter, we will present our experimental results of using Hi-C data in tandem with paired reads. For our experiments, we have chosen organism *Saccharomyces cerevisiae* (baker's yeast), strain S288c, with 16 chromosomes and one mitochondrial (non-nuclear) circular DNA, and total genome length of $12 \cdot 10^6$ bp (the lengths of the individual chromosomes are shown in the figure 4.1). We have used two read datasets from the NCBI database (`https://www.ncbi.nlm.nih.gov/`):

- **SRR4446972** — paired reads, obtained by Illumina HiSeq 2000 technology. Length of reads is $125 + 125$ bp. Total number of read pairs is $6,162,477$. Mean coverage is $6{,}162{,}477 \cdot 2 \cdot 125/(12 \cdot 10^6) \approx 128$.
- **SRR5077811** — Hi-C reads, obtained by Illumina HiSeq 2000 technology. Length of reads is $102+102$ bp. Total amount of read pairs is 91,776,947. Mean coverage is $91{,}776{,}947 \cdot 2 \cdot 102/(12 \cdot 10^6) \approx 1506$.

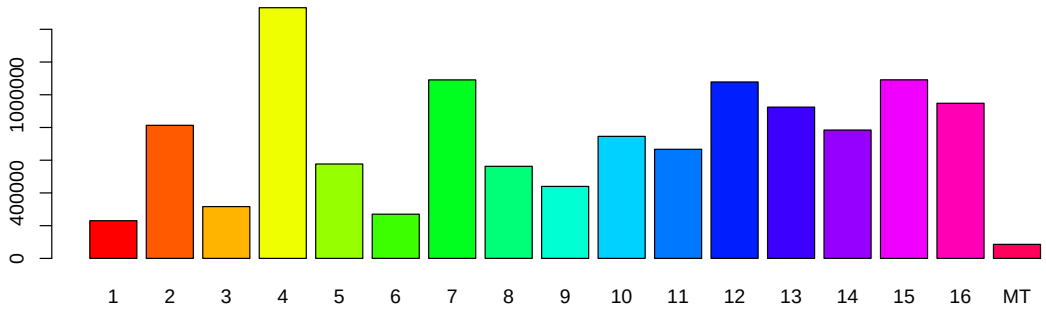


Figure 4.1: Sizes of chromosomes of *Saccharomyces cerevisiae*

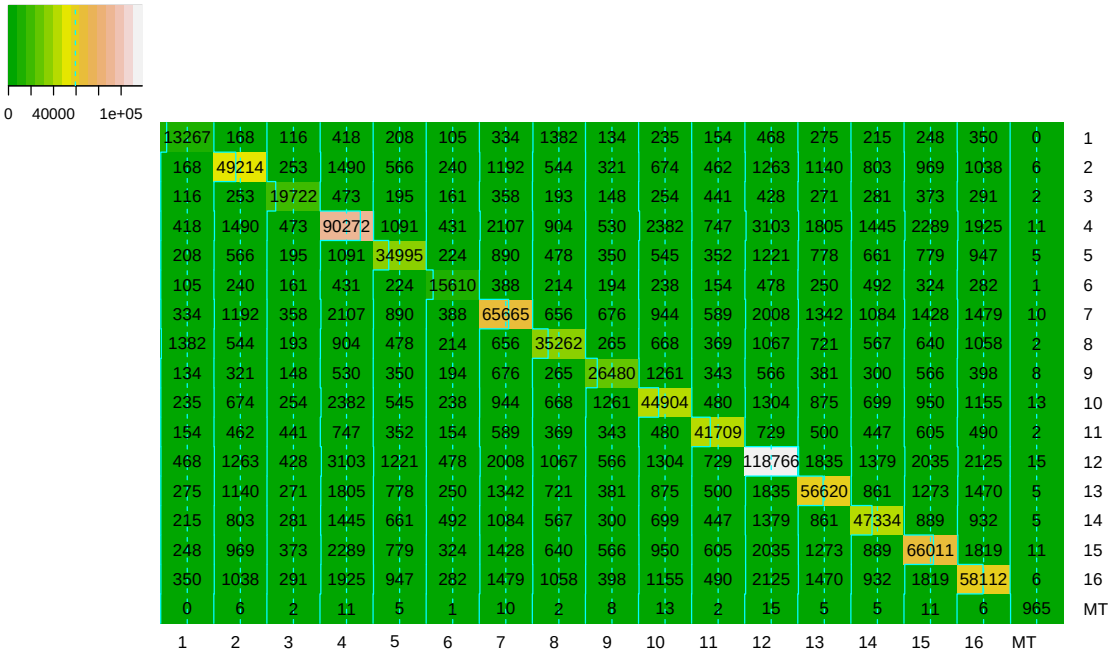


Figure 4.2: Heatmap of the interaction rates between chromosomes. Cells on the diagonal shows the number of cis-alignments for a given chromosome. `MT` stands for the mitochondrial DNA

4.1 Verification of the model for Hi-C reads

Our goal is to verify our model for the CTR pattern, described in the subsection 3.1.2 and the DDD pattern, described in the subsection 3.1.1. For this purposes, we sampled one million Hi-C reads from our dataset and aligned them to the reference genome with Bowtie2 aligner[23], left and right reads separately.

CTR pattern

We counted the cis- and trans- alignments between each pair of the chromosomes. The figure 4.2 shows the interaction rates (cells on the diagonal show the amount of cis-alignments for a given chromosome). Note that the mitochondrial DNA (marked as "MT" in the figure 4.2) has almost no interactions with the nuclear DNA molecules, since they are located in different organelles within the cell. This corresponds to the main goal of the Hi-C method — to uncover the spatial distance between DNA segments. We have excluded the cis- and trans- alignments for the mitochondrial DNA from further analysis. We will denote the interaction rate between i-th and j-th chromosome as $h_{i,j}$.

As we can clearly see, the number of cis-alignments (783,943 after excluding the mitochondrial DNA) is much greater than the number of trans-alignments (90,858).

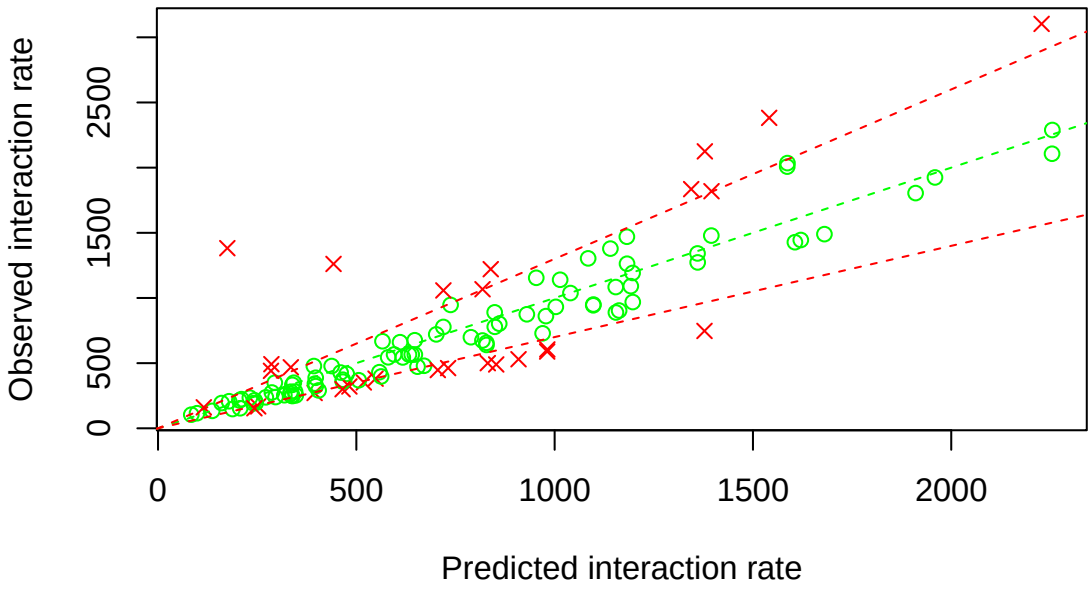


Figure 4.3: Trans interaction rate estimation. Green line shows the prediction of $h_{i,j}$ based on chromosome size, red lines show the boundaries of 70% and 130% relative errors. Overall, 91 out of 120 rates were predicted with relative error in the boundaries (green circle stands for successful estimation and red cross for failure).

This result supports the main idea of the CTR pattern.

Trans-interactions rate

As it was declared in chapter 3, we model the relative trans-interaction rate $g_{i,j}$ between the chromosomes S_i and S_j as

$$g_{i,j} := \frac{|S_i| \cdot |S_j|}{\sum_{1 \leq u < v \leq n} |S_u| \cdot |S_v|},$$

and therefore our estimation for $h_{i,j}$ is simply $g_{i,j}$ times the total number of trans-alignments. As we can see in the figure 4.3, if we allow $\pm 30\%$ error rate, then 91 out of 120 interaction rates are predicted successfully.

It is important to mention that this model is very simple and does not account for the complex fractal structure of the chromatin [3]. This model was chosen because of low computational requirements and surprisingly adequate estimation power.

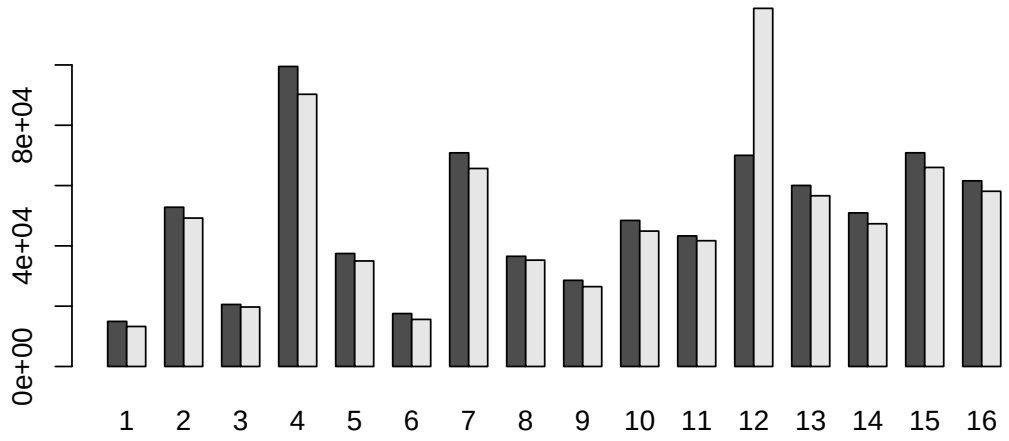


Figure 4.4: Estimation of cis-reads count for chromosomes. Dark grey stands for predicted counts and light grey for the observed counts.

Distribution of cis-alignments among chromosomes

In chapter 3, we have modelled the relative frequency of cis-alignments for individual chromosomes from the genome $G = (S_1, \ldots, S_n)$ as:

$$c_i := \frac{|S_i|}{\sum_{j=1}^{n} |S_j|}$$

The figure 4.4 shows the estimated cis-read counts (dark grey) versus the real cis-read counts (light grey). As you can see, most chromosomes are only slightly overestimated, but chromosome 12 is significantly underestimated by the model

We assume that increased number of cis-alignments in chromosome 12 occurred due to tandem repeats of ribosomal RNA. This is supported by the coverage plot for this chromosome (figure 4.5). There is a 10-fold increase in the coverage (5000 instead of 500) due to fact, that the ribosomal RNA has only two repeats in the reference genome, whereas in reality there could be dozens of such repeats. This would mean that the length of the real chromosome is greater than it is shown in the database (and the reference genome).

The DDD pattern

We have evaluated insert sizes of cis-reads for each chromosome (figure 4.6). Each point represent a bin of size 1000 bp (i.e. each graph is a very dense histogram).

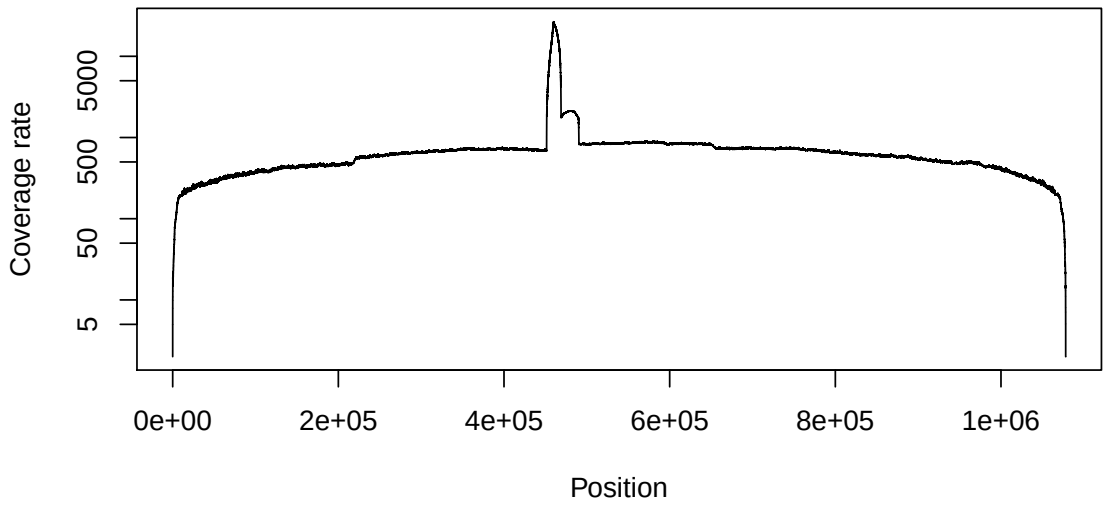


Figure 4.5: Coverage rate for 12-th chromosome. We assume that the spike corresponds to tandem repeats of the ribosomal RNA

The data suggests the exponential decrease of interaction rate, as well as slightly different distributions for the individual chromosomes. The artifacts (e.g. spikes at the longer ranges) were also observed by Kaplan and Dekker [1]. We have decided not to incorporate these abnormalities into our model.

4.2 GAML2 experiments

In this section, we will talk about experimental runs of our implementation of GAML2. First, we will describe the pipeline, then we will describe how we estimated the parameters. We will finish this section with the comparison of results with and without Hi-C data and the discussion of the results.

4.2.1 Pipeline

Our pipeline is built in the following way (figure 4.7). First, we construct de Bruijn graph from the paired reads by Velvet assembler with conservative settings `-cov_cutoff auto`. Second, we subsample both datasets to obtain 1 million reads of each type (mean coverage is then approximately 21 for the paired reads and 16 for the Hi-C reads).

We have been forced to such fierce subsampling by the immense computational requirements. Third, we run GAML2 on the subsampled read sets and the de Bruijn

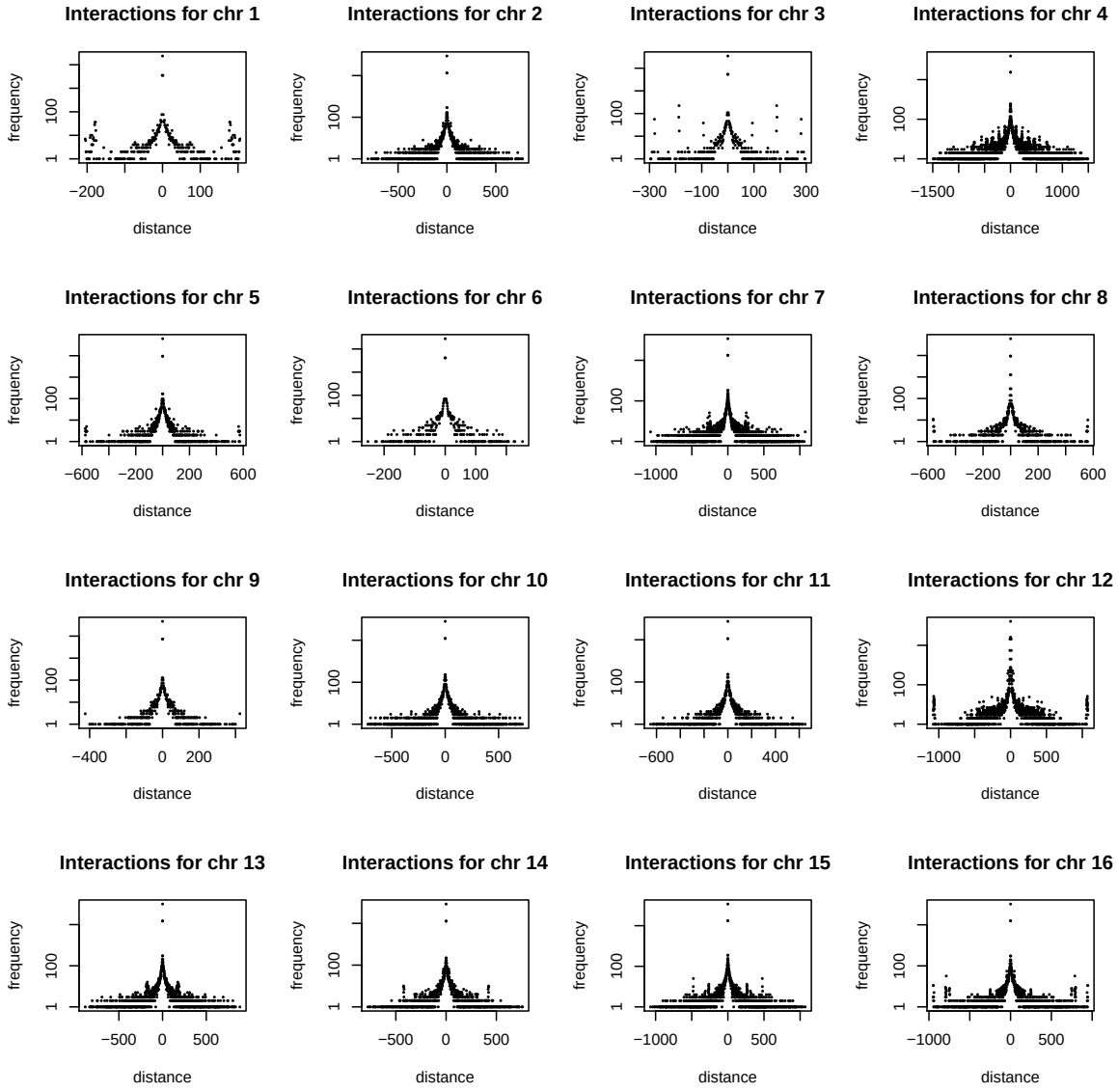


Figure 4.6: Insert sizes for cis-reads on every chromosome (y axis is logarithmic)

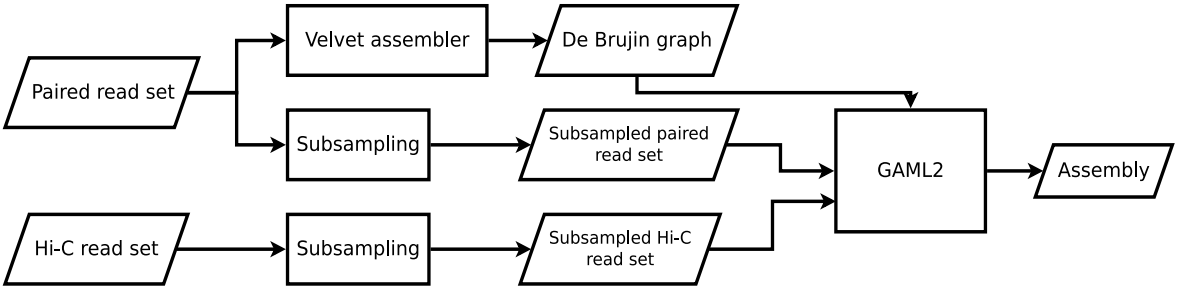


Figure 4.7: Experiment pipeline

graph using a configuration file, containing the estimated parameters for both datasets.

4.2.2 Estimating of parameters for experiments

Our probabilistic model has several parameters, which describe insert size distribution, sequencing error rates, cis-trans-ratio. The estimation of parameters from the input data (i.e. read sets) is beyond the scope of this thesis. Therefore, we have made our estimations based on the reference genome (except for λ parameters for Hi-C cis-reads, which are estimated for each path separately during computations (see subsection 3.1.1)). In future work, initial parameter estimates could be refined within the simulated annealing framework.

Parameters for single read model The individual error probabilities were set to 0.01 for substitution and 0.0001 for indels (because both datasets are produced by Illumina technologies).

Insert length for paired reads We have used the estimation of mean and standard deviation of insert size of the paired reads, computed by BWA-mem aligner. The mean was estimated as 387.2 and the standard deviation as 92.7.

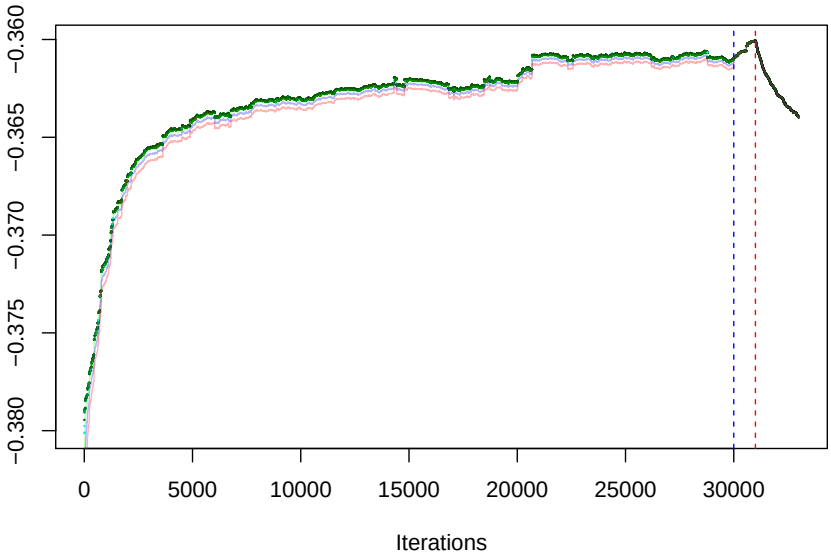
Hi-C data parameters Base on the data, described in section 4.1, we have estimated the cis-trans-ratio, or p_{cis}, using the notation, established in chapter 3, as $\frac{783\,943}{783\,943+90\,858} \approx 0.896$.

The rest of parameters had no clear mathematical foundations (parameters of cooling schedule and penalties) and were chosen based on preliminary experiments with smaller organism, *Escherichia coli*.

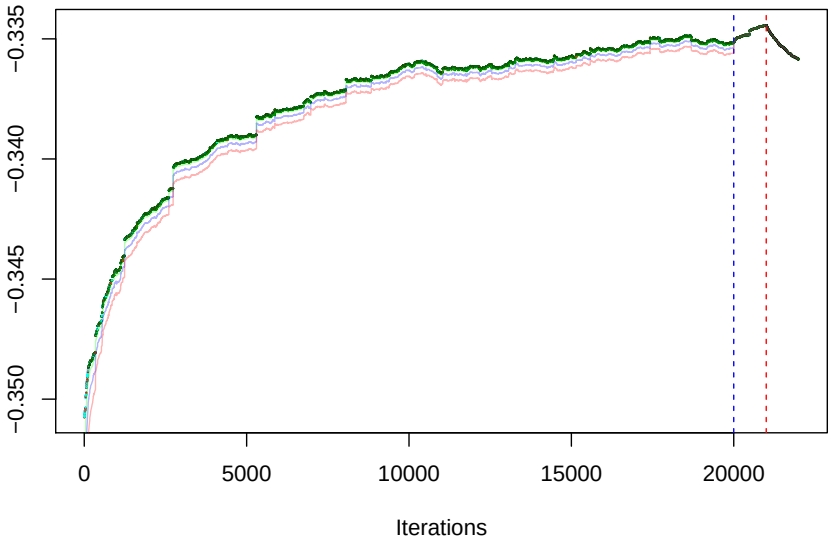
4.2.3 Results

We have run two experiments: the first with only paired reads, and the second with both paired and Hi-C reads. The first experiment's simulated annealing phase lasted for 30 000 iterations and 20 000 for the second experiment. Both experiments then proceeded to the hill climbing phase and finishing cleaning (both described in section 2.9). Both experiments took about three days to finish.

The score evolution during iterations is depicted in the figure 4.8. It is important to mention, that scores are not comparable between different datasets (i.e. the final score -0.337 of the second experiment is not better nor worse than the score -0.365 of the first experiment).



(a) Paired reads only.



(b) Paired and Hi-C reads.

Figure 4.8: Score during the computation of GAML2. Vertical blue line marks the start of hill climbing phase, vertical red line marks the start of cleaning phase. Transparent blue line stands for score decrease, which will be allowed by the temperature with probability 10%. Transparent red line — 1%.

Metric	Velvet output	Only paired	Paired + Hi-C
# contigs ($\geq$ 500 bp)	1450	1098	986
# misassemblies	2	76	102
Total length	11 217 354	11 264 014	11 289 460
N50	14 809	18 720	21 075
L50	241	176	170
NA50	14765	17564	19608
LA50	242	197	181

Table 4.1: Quality metrics for experimental results (by Quast analyser).

Notice that during the cleaning phase of the first experiment (figure 4.8a), the resulting score dropped quite a bit. We assume that this happened due to extremely low coverage, and thus low support for keeping the contigs joined.

In order to compare the results of the experiments, we rely on the common quality metrics, described in subsection 1.2.3. We have used Quast analyser [24] for the quality metrics evaluation. The results are shown in the table 4.1.

As we can see, the second experiment is better by all quantitative metrics, except for misassemblies. The results have shown that in the second experiment we have joined $1450 - 986 = 464$ contigs and we did $464 - 102 = 362$ of them correctly (in comparison with 352 joins and only 276 of them good in the first experiment).

These results are of dual nature. On the one hand, they have proved that the way we have used Hi-C data is viable and it is possible to gain additional information from them. On the other hand, our implementation of GAML still has several challenges to overcome, mainly its computational requirements and non-conservativeness in joining. The latter partially depends on the former, because greater coverage rate may lead to better precision, but also requires more computation.

Summary

In this work, we have described the probabilistic approach used in GAML framework, reimplemented several key features of the original framework in GAML2 and added several new ones in a way that is more open to further extensions. We have also improved the original probability model by adding a distinction between different types of sequencing errors. We have then designed and implemented a probabilistic model for Hi-C data, compatible with GAML framework. The experimental results confirmed that adding Hi-C data improved the results.

Future work

There are several ways to proceed with this research. The first line of works concerns the improvements in the core of the GAML2 engine:

- Reimplementing the rest of moves in the simulated annealing, designed by the original author. These moves are interesting mainly due to their focus on tandem repeats.
- Adding a better caching of the evaluated alignments. Because the solution space is defined by the input de Bruijn graph, it is possible to store alignments for individual nodes of the graph, thus reducing computational cost.
- Change of the core metaheuristic. It may be interesting to abandon simulated annealing method and venture further into a land of discrete optimisations, trying methods like taboo search or memetic algorithms.
- Adding the possibility of parallel evaluations. As aligning of many short reads may be done in parallel, this direction could substantially improve time requirements.
- Improve the scaffolding with the distance measures. Kaplan and Dekker used multidimensional scaling to predict the order of contigs in the scaffolds.

Further improvements could also be done in modeling of Hi-C and other types of data:

- Adding new types of data. There are now many sequencing technologies (such as PacBio, MinION, 10x Genomics, RNA-seq) and adding them would surely prove the universality of the probabilistic approach to the genome assembly.
- Take coverage rate into account. The current model does optimise for the uniform coverage rate, but only implicitly.
- Designing a better model for Hi-C data. We have used a very simple model, and there is still the possibility of improvement.
- Adding a prediction of a genome's karyotype.

Additional materials

Electronic materials (source materials of GAML2 and instructions on how to execute it) are available on the attached CD.

Bibliography

[1] Noam Kaplan and Job Dekker. High-throughput genome scaffolding from in vivo dna interaction frequency. *Nature biotechnology*, 31(12):1143–1147, 2013.

[2] Vladimír Boža, Broňa Brejová, and Tomáš Vinař. Gaml: genome assembly by maximum likelihood. *Algorithms for Molecular Biology*, 10(1):1, 2015.

[3] Erez Lieberman-Aiden, Nynke L van Berkum, Louise Williams, Maxim Imakaev, Tobias Ragoczy, Agnes Telling, Ido Amit, Bryan R Lajoie, Peter J Sabo, Michael O Dorschner, Richard Sandstrom, Bradley Bernstein, M A Bender, Mark Groudine, Andreas Gnirke, John Stamatoyannopoulos, Leonid A Mirny, Eric S Lander, and Job Dekker. Comprehensive mapping of long-range interactions reveals folding principles of the human genome. *Science (New York, N.Y.)*, 326(5950):289–93, oct 2009.

[4] Venter et al. The sequence of the human genome. *Science (New York, N.Y.)*, 291(5507):1304–51, feb 2001.

[5] JAUME PELLICER, MICHAEL F. FAY, and ILIA J. LEITCH. The largest eukaryotic genome of them all? *Botanical Journal of the Linnean Society*, 164(1):10–15, sep 2010.

[6] John Gallant, David Maier, and James Astorer. On finding minimal length superstrings. *Journal of Computer and System Sciences*, 20(1):50–58, feb 1980.

[7] Elaine R. Mardis. Next-Generation DNA Sequencing Methods. *Annual Review of Genomics and Human Genetics*, 9(1):387–402, sep 2008.

[8] Anthony Rhoads and Kin Fai Au. PacBio Sequencing and Its Applications. *Genomics, Proteomics & Bioinformatics*, 13(5):278–289, oct 2015.

[9] Eugene W. Myers, G G Sutton, A L Delcher, I M Dew, D P Fasulo, M J Flanigan, S A Kravitz, C M Mobarry, K H Reinert, K A Remington, E L Anson, R A Bolanos, H H Chou, C M Jordan, A L Halpern, S Lonardi, E M Beasley, R C Brandon, L Chen, P J Dunn, Z Lai, Y Liang, D R Nusskern, M Zhan, Q Zhang,

X Zheng, G M Rubin, M D Adams, and J C Venter. A whole-genome assembly of Drosophila. *Science (New York, N.Y.)*, 287(5461):2196–204, mar 2000.

[10] Daniel R Zerbino and Ewan Birney. Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome research*, 18(5):821–829, 2008.

[11] Serafim Batzoglou, David B Jaffe, Ken Stanley, Jonathan Butler, Sante Gnerre, Evan Mauceli, Bonnie Berger, Jill P Mesirov, and Eric S Lander. ARACHNE: a whole-genome shotgun assembler. *Genome research*, 12(1):177–89, jan 2002.

[12] Sergey Koren, Brian P Walenz, Konstantin Berlin, Jason R Miller, Nicholas H Bergman, and Adam M Phillippy. Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome research*, 27(5):722–736, may 2017.

[13] Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A. Gurevich, Mikhail Dvorkin, Alexander S. Kulikov, Valery M. Lesin, Sergey I. Nikolenko, Son Pham, Andrey D. Prjibelski, Alexey V. Pyshkin, Alexander V. Sirotkin, Nikolay Vyahhi, Glenn Tesler, Max A. Alekseyev, and Pavel A. Pevzner. SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of computational biology : a journal of computational molecular cell biology*, 19(5):455–77, may 2012.

[14] Mohammadreza Ghodsi, Christopher M Hill, Irina Astrovskaya, Henry Lin, Dan D Sommer, Sergey Koren, and Mihai Pop. De novo likelihood-based measures for comparing genome assemblies. *BMC research notes*, 6(1):1, 2013.

[15] V. Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, jan 1985.

[16] Job Dekker, Karsten Rippe, Martijn Dekker, and Nancy Kleckner. Capturing chromosome conformation. *Science (New York, N.Y.)*, 295(5558):1306–11, feb 2002.

[17] J. Dostie, T. A. Richmond, R. A. Arnaout, R. R. Selzer, W. L. Lee, T. A. Honan, E. D. Rubio, A. Krumm, J. Lamb, C. Nusbaum, R. D. Green, and J. Dekker. Chromosome Conformation Capture Carbon Copy (5C): A massively parallel solution for mapping interactions between genomic elements. *Genome Research*, 16(10):1299–1309, oct 2006.

[18] Marieke Simonis, Petra Klous, Erik Splinter, Yuri Moshkin, Rob Willemsen, Elzo de Wit, Bas van Steensel, and Wouter de Laat. Nuclear organization of active and

inactive chromatin domains uncovered by chromosome conformation capture–on-chip (4C). *Nature Genetics*, 38(11):1348–1354, nov 2006.

[19] Zhihu Zhao, Gholamreza Tavoosidana, Mikael Sjölinder, Anita Göndör, Piero Mariano, Sha Wang, Chandrasekhar Kanduri, Magda Lezcano, Kuljeet Singh Sandhu, Umashankar Singh, Vinod Pant, Vijay Tiwari, Sreenivasulu Kurukuti, and Rolf Ohlsson. Circular chromosome conformation capture (4C) uncovers extensive networks of epigenetically regulated intra- and interchromosomal interactions. *Nature Genetics*, 38(11):1341–1347, nov 2006.

[20] Marie Lisandra Zepeda-Mendoza and Osbaldo Resendis-Antonio. Hierarchical Agglomerative Clustering. In *Encyclopedia of Systems Biology*, pages 886–887. Springer New York, New York, NY, 2013.

[21] Jianzhong Wang. Classical Multidimensional Scaling. In *Geometric Structure of High-Dimensional Data and Dimensionality Reduction*, pages 115–129. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[22] Vladimir Boža. Gaml2. `https://github.com/usamec/GAML2`, 2016.

[23] W B Langdon. Performance of genetic programming optimised Bowtie2 on genome comparison and analytic testing (GCAT) benchmarks. *BioData Mining*, 8(1):1, jun 2015.

[24] Alexey Gurevich, Vladislav Saveliev, Nikolay Vyahhi, and Glenn Tesler. Quast: quality assessment tool for genome assemblies. *Bioinformatics*, 29(8):1072–1075, 2013.

[25] Philip N Benfey. *Quickstart Molecular Biology: An Introductory Course for Mathematicians, Physicists, and Computational Scientists*. 2014.

[26] J Craig Venter, Mark D Adams, Eugene W Myers, Peter W Li, Richard J Mural, Granger G Sutton, Hamilton O Smith, Mark Yandell, Cheryl A Evans, Robert A Holt, et al. The sequence of the human genome. *science*, 291(5507):1304–1351, 2001.

[27] Paul Medvedev and Michael Brudno. Maximum likelihood genome assembly. *Journal of computational Biology*, 16(8):1101–1116, 2009.

[28] Scott Clark, Rob Egan, Peter I Frazier, and Zhong Wang. Ale: a generic assembly likelihood evaluation framework for assessing the accuracy of genome and metagenome assemblies. *Bioinformatics*, page bts723, 2013.

[29] Martial Marbouty, Axel Cournac, Jean-François Flot, Hervé Marie-Nelly, Julien Mozziconacci, and Romain Koszul. Metagenomic chromosome conformation capture (meta3c) unveils the diversity of chromosome organization in microorganisms. *Elife*, 3:e03318, 2014.