University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Robert Vaser

# ALGORITHMS FOR DE NOVO ASSEMBLY OF LARGE GENOMES

DOCTORAL THESIS

Zagreb, 2019

University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Robert Vaser

# ALGORITHMS FOR DE NOVO ASSEMBLY OF LARGE GENOMES

DOCTORAL THESIS

Supervisor:
Professor Mile Šikić, PhD

Zagreb, 2019

Sveučilište u Zagrebu

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Robert Vaser

# ALGORITMI ZA DE NOVO SASTAVLJANJE VELIKIH GENOMA

DOKTORSKI RAD

Mentor:
Prof. dr. sc. Mile Šikić

Zagreb, 2019.

The dissertation was made at the University of Zagreb Faculty of Electrical Engineering and Computing, Department of Electronic Systems and Information Processing.

Supervisor: Professor Mile Šikić, PhD

The dissertation has: 83 pages

The dissertation number: ___________

# About the Supervisor

Mile Šikić was born in Zagreb in 1972. He is a professor at the University of Zagreb Faculty of Electrical Engineering and Computing. He obtained his Bachelor, Master and PhD degrees at the same faculty in 1996, 2002 and 2008, respectively. His PhD thesis was titled “Computational method for prediction of protein-protein interaction sites”. He started working at Faculty of Electrical Engineering and Computing as research associative in 1997. He worked as teaching assistant between 2005 and 2009, as assistant professor between 2009 and 2015, and as associate professor until 2018 when he became a professor. During the period between 2011 and 2012, he joined the Bioinformatics Institute in Singapore, where he was appointed as adjunct scientist in 2013. Currently he is spending his sabbatical year at the Genome Institute of Singapore. He has been working as a consultant and project manager in the fields of computer and mobile networks. His current scientific work is focused on development of new algorithms in the fields of bioinformatics and complex networks, in which his publications have more than 1400 citations and an h-index of 16.

# O mentoru

Mile Šikić rođen je u Zagrebu 1972. godine. Redoviti je profesor na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu, na kojem je 1996., 2002. i 2008. godine stekao titule prvostupnika, magistra odnosno doktora znanosti. Njegov doktorski rad nosio je naslov "Računalna metoda za predviđanje mjesta proteinskih interakcija". Započeo je rad na Fakultetu elektrotehnike i računarstva 1997. godine kao znanstveni suradnik. Između 2005. i 2009. godine radio je kao asistent, između 2009. i 2015. godine kao docent te kao izvanredni profesor od 2015. sve dok nije postao redoviti profesor 2018. godine. Period između 2011. i 2012. godine radio je u Singapuru na institutu Bioinformatics Insitute, gdje je postavljen kao pridruženi znanstvenik od 2013. godine. Trenutno radi na Genome Institute of Singapore, uvažavajući slobodnu godinu na fakultetu. Radio je kao konzultant i voditelj projekata u području računalnih i mobilnih mreža. Milino trenutno područje istraživanja usmjereno je na razvoj novih algoritama u polju bioinformatike i kompleksnih mreža, područja u kojima njegove publikacije imaju više od 1400 citata i h-index koji iznosi 16.

# Acknowledgment

The past four years that I have spent on my PhD had its highs and lows, but thanks to the support and love of my wife Marina, everything seemed as a stroll in the park. I will always cherish you as you have me, thank you love!

I would like to thank my family who enabled me to be where I am today, to my mother Lidija and father Stjepan, my sister Sandra, and my grandparents Sonja, Slavica, Karel and Stjepan.

Many thanks to my supervisor Mile Šikić who has guided me from my undergraduate years. Thank you for your guidance, wisdom and patience, but also for being a very good friend.

Thanks to Ivan Sović for all the discussions we shared about various bioinformatics algorithms and for the assistance in implementing the consensus module Racon. Thanks to Stjepan Begušić for introducing me to nifty visualization tools which led to the graph drawing idea implemented in the assembler Raven. Thanks to Krešimir Križanović for all the help with various faculty obligations.

In the end, I would like to thank my colleagues and friends. Especially, Stjepan Begušić, Josip Marić, Vanessa Keranović, Andro Merćep, Lovro Vrček and Josip Žubrinić, for all the hangouts and dank memes that made things funnier and easier. Thanks also to Ivan Blažon, Marko Car, Tomislav Križan and Dino Lukman for all the ridiculous times while playing board and computer games.

# Abstract

The inability of DNA sequencing technologies to interpret entire molecules led to the development of methods that connect the obtained short fragments back together in a puzzle-like process. They are called assemblers and their design is guided with the notion that similar fragments originate from the same region in the genome. That is often annulled due to sequencing errors and repetitive nature of the genome. Short fragments of first two generations of sequencing are incapable of spanning moderately long repetitive regions and thus hinder a complete assembly. The advent of new sequencing approaches, namely Pacific Biosciences and Oxford Nanopore Technologies, pushed the limit on the fragment lengths at a cost of higher error rates, but still facilitated the assembly problem considerably. First assembly attempts used various types of error correction approaches prior the assembly with existing tools at that time. Although, several long read based assemblers have been proposed in the past years, they demand significant amounts of computational resources. The focus of this research is development of memory efficient and scalable algorithms for de novo assembly of large genomes using third generation of sequencing data without error correction of input sequences. In the scope of the thesis we implemented three novel tools for genome assembly: a memory friendly layout module called Rala, which builds the assembly graph from preprocessed sequences and resolves junctions in graph with the help of force directed placement; a fast and accurate consensus module called Racon based on vectorized partial order alignment; and the complete de novo assembler called Raven, which competes with state-of-the-art assemblers both in quality and resource management.

**Keywords**: de novo, assembly, long reads, PacBio, Oxford Nanopore, pile-o-gram, partial order alignment, force directed layout, vectorization

# Algoritmi za *de novo* sastavljanje velikih genoma

Glavni nedostatak metoda za sekvenciranje jest duljina dobivenih očitanja koja je znatno kraća od DNA molekula. Očitanja se zbog toga često promatraju kao dijelovi velike slagalice koje je potrebno međusobno spojiti kako bi se rekonstruirala originalna slika, što je u ovom slučaju niz nukleotida. Da bi se očitanja preklapala, nužan je veliki broj identičnih DNA molekula nasumično fragmentirati te svaki dio zasebno sekvencirati. Rekonstrukcija sekvencirane DNA spajanjem očitanja zadatak je alata koje nazivamo asembleri. Oni su dizajnirani s pretpostavkom da slična očitanja potječu iz istog dijela genoma, ali ta pretpostavka je često narušena zbog grešaka u podacima te ponavljajućih regija u genomu. Problem sastavljanja formalno se definira pomoću usmjerenih grafova, za koje postoje egzaktna rješenja, ali ih je teško pronaći postojećim računalnim resursima. Umjesto toga upotrebljavaju se metode pojednostavljenja, kako bi se graf linearizirao i pronašli što duži dijelovi sekvenciranog genoma. Postoje dva različita pristupa sastavljanju, jedan baziran na paradigmi preklapanje-razmještaj-konsenzus koja opisuje odnose između očitanja, te drugi koji gradi de Bruijn graf od kratkih podnizova dobivenih iz svih očitanja. U prvom pristupu najprije se traže približna sufiks-prefiks preklapanja između svih parova očitanja koja su potrebna za izgradnju nekog oblika grafa preklapanja. Nedvosmisleni dijelovi pojednostavljenog grafa proglase se dijelovima sekvenciranog genoma u kojima se ispravljaju pogreške nastale prilikom sekvenciranja pomoću gomile preklapanja cijelog skupa očitanja. Ovaj pristup razvijen je za podatke dobivene Sangerovim sekvenciranjem i teoretski je prikladan za bilo kakve podatke. Pojava druge generacije sekvenciranja s puno većim prinosom očitanja koja su bila nesto kraća, dovela je do dizajna de Bruijn metode zbog dugog izvođenja faze preklapanja. Pristup de Bruijn grafova zaobilazi kvadratnu složenost faze preklapanja režući očitanja na preklapajuće kratke podnizove koji postaju bridovi navedenog grafa. Ovakva formulacija problema zamjenjuje pronalazak Hamiltonovog puta u grafu preklapanja s pronalasakom Eulerovog puta u de Bruijn grafu.

Značajni napredak u sekvenciranju genoma doprinos je tvrtki Pacific Biosciences i Oxford Nanopore Technologies, čije tehnologije predstavljaju treću generaciju sekvenciranja. One su omogućile višestruko povećanje duljine očitanja, ali uz cijenu osjetno veće pogreske i manje propusnosti. Usprkos tome, obje tehnologije pripomogle su u smanjenju fragmentiranosti sastavljenih genoma koji su bili sekvencirani prijašnjim generacijama te su pokrenule razvoj velikog broja novih algoritama. U početku je bio fokus na ispravljanju pogrešaka u podacima prije sastavljanja s već postojećim alatima, a kasnije je istraživanje promijenilo smjer prema algoritmima koji direktno mogu baratati s toliko greškovitim podacima. Posljedica je velik odabir metoda za sastavljanje genoma, od kojih mnoge zahtjevaju značajnu količinu računalnih resursa. To je pogotovo vidljivo kod većih eukariotskih organizama. Stoga je ovo istraživanje fokusirano na dizajnu i implementaciji novih algoritama za sastavljanje genoma obazirući se na računalni trošak. Naglasak je na visokoj točnosti i maloj fragmentiranosti rekonstruiranih DNA molekula, umjerenim memorijskim zahtjevima te paralelnoj efikasnosti paradigme preklapanje-razmještaj-konsenzus, oslanjajući se na dugačka i greškovita očitanja treće generacije sekvenciranja. Doprinos ove disertacije sastoji se od:

- memorijski učinkovitog algoritma za fazu razmještaja paradigme preklapanje-razmještaj-konsenzus koji postiže malen broj fragmenata pri sastavljanju velikih genoma,
- brzog algoritma za fazu konsenzusa paradigme preklapanje-razmještaj-konsenzus koji osigurava visoku točnost sastavljanja velikih genoma, te
- sustava za *de novo* sastavljanje velikih genoma iz podataka dobivenih tehnologijama treće generacije sekvenciranja.

Poglavlje 1 ("Uvod") disertacije daje pregled tehnologija za sekvenciranje genoma zajedno s njihovim prednostima i nedostacima. Objašnjava motivaciju za provedeno istraživanje, iznosi cilj istraživanja te prikazuje ispunjeni doprinos koji se sastoji od tri dijela.

Poglavlje 2 ("Teorijska podloga") ukratko predstavlja teorijsku pozadinu za *de novo* sastavljanje genoma. Prezentirani su koncepti i određeni alati za sastavljanje koji su obilježili područje. Krenuvši od prvih alata koji su na razne načine ispravljali pogreške u očitanjima prije sastavljanja, sve do alata koji koriste sažetu reprezentaciju očitanja za brzu izgradnju neke vrste usmjerenog grafa.

Poglavlje 3 ("Algoritmi za fazu razmještaja paradigme preklapanje-razmještaj-konsenzus") ove disertacije opisuje fazu razmještaja koja je bazirana na grafu sastavljanja s pripadajućim metodama za pojednostavljenje. Graf sastavljanja je modifikacija grafa nizova korištenog u prvom preklapanje-razmještaj-konsenzus asembleru te se gradi iz skupa očitanja koja nisu međusobno sadržana u cijelosti. Svako očitanje i pripadajući preokrenuti komplement postaju vrhovi grafa sastavljanja, dok su međusobna prefiks-sufiks preklapanja bridovi. Skup preklapanja dobiven je najbržom poznatom knjižicom za greškovite podatke treće generacije sekvenciranja. Graf sastavljanja pojednostavljuje se izbacivanjem tranzitivnih bridova, kratkih ogranaka i mjehurića. Tranzitivni bridovi traže se kod svih međusobno povezanih trojki vrhova te se izbacuju svi odjednom zbog mogućnosti gubitka tranzitivnosti. Kratki ogranci u grafu lako se pronađu pretraživanjem u dubinu od svih vrhova koji nemaju prefiks preklapanja te se izbace ako nisu predugački. Složenije strukture koje tvore više puteva između dvaju vrhova (mjehurići) pronalaze se pretraživanjem u širinu. Svaki mjehurić koji se sastoji od dva puta provjerava se ako sadrži otprilike jednak niz nakon čega se traže bridovi u jednom od puteva, koji ako bi bili odstranjeni nebi narušavali povezanost ostatka grafa. Nakon iterativnog pojednostavljenja, nedvosmisleni putevi grafa sastavljanja transformiraju se u nizove koji naposljetku obuhvaćaju cijele kromosome ili žavrsavaju na ponavljajućim regijama.

Pogreške sekvenciranja i ponavljajuće regije genoma proizvode lažne bridove u grafu preklapanja koja često onemogućuju potpunu rekonstrukciju sekvenciranog genoma. Kako bi se zaobišao ovaj problem, implementirane su dodatne metode prije konstrukcije grafa sastavjanja te nakon pojednostavljenja. Predobrada podataka koristi gomile preklapanja, jednodimenzionalne signale dobivene naslagivanjem svih preklapanja nekog očitanja te sumiranjem njihovog broja nad svakim nukleotidom. Oblik signala pomaže u anotaciji očitanja zbog naglih promjena u amplitudi, koje su karakteristične za pojedinu klasu. Kimerna očitanja sastavljena su od više dijelova, čiji raspored nije prisutan u genomu te njihove gomile imaju prekide u amplitudi, dok očitanja koja preklapaju ponavljajuće regije u genomu imaju povišenu amplitudu u tom dijelu gomile. Proces anotacije koristi gradijente i median kako bi identificario navedene oblike te odrezao problematične dijelove kimernih očitanja i odstranio prefiks-sufiks preklapanja, koja se u cijelosti nalaze u ponavljajućoj regiji. Obrada grafa sastavljanja nakon pojednostavljenja koristi udaljenosti bridova u dvodimenzionalnom prikazu grafa dobivenim simulacijom djelovanja sila. Algoritam nastoji nacrtati dobro povezane vrhove blizu, što produljuje bridove koji slabo povezuju udaljenje dijelove grafa. Zbog toga se u pojednostavljeni graf vraćaju tranzitivni bridovi, kako bi se pojačale privlačne sile te se dobio željeni oblik grafa. Bridovi koji povezuju vrhove s više sufiks preklapanja su izbačeni iz skupa bridova ako su puno duži od ostalih pripadajućih bridova. Kako ovaj algoritam ima kvadratnu složenost, graf sastavljanja se smanjuje zamjenom nedvosmislenih puteva jednim vrhom, ali izostavljaju se vrhovi koji su blizu čvorišta. Dodatno se koristi i loglinearna Barnes-Hut aproksimacija koja točno opisuje sile između susjednih vrhova, a aproksimira sile između udaljenih vrhova.

Poglavlje 4 ("Algoritmi za fazu konsenzusa paradigme preklapanje-razmještaj-konsenzus") predstavlja fazu konsenzusa koja se temelji na poravnanju parcijalnog uređaja proširenog vektorizacijom i rezanjem nizova, kako bi se ostvarila linearna složenost izvođenja s obzirom na veličinu sekvenciranog genoma. Poravnanje parcijalnog uređaja je metoda koja se koristi za višestruko poravnanje nizova, kako bi se iz gomile poravnanja ispravile pogreške nastale prilikom sekvenciranja. Višestruko poravnanje spremljeno je kao usmjeren aciklički graf, koji je moguće poravnati s nizom pomoću proširenog algoritma poravnaja dvaju nizova. Graf parcijalnog uređaja sprema znakove kao vrhove, koji su međusobno povezani bridovima ako su uzastopni u bilo kojem nizu koji je dodan u graf. Težine bridova označuju broj nizova koji sadrži par znakova, ali mogu se ukomponirati i vrijednosti kvalitete dobivene sekvenciranjem. Poravnanje je prošireno s obzirom na mogućnost višestrukih prethodnika svakog vrha u grafu te se njihove vrijednosti moraju uzeti u obzir kod procedure dinamičkog programiranja. Kada su svi željeni nizovi iterativno poravnati i dodani u graf parcijalnog uređaja, finali konsenzusni niz dobije se pronalaskom najtežeg puta u grafu. Ovakav način višestrukog poravnanja odbacuje sklonost nizova da se slažu s referentnim, pojava koja je prisutna u ostalim progresivnim metodama kod kojih se nizovi poravnavaju s reprezentativnim nizom trenutnog višestrukog poravnanja. Kako je kvadratna složenost memorije i izvođenja algoritma za poravnanje neupotrebiva za dulja očitanja treće generacije, svaki fragment dobiven iz faze razmještaja, narezan je na kratke nepreklapajuće prozore. Sva očitanja prvo se poravnaju na cijele fragmente kako bi se distribuirala na pripadajuće prozore. Dodatno, korištenjem SIMD instrukcija vektoriziran je algoritam poravnanja parcijalnog uređaja i to za tri različita tipa poravnanja i tri modela procjepa. Zbog novouvedenih zavisnosti u poravnanju parcijalnog uređaja, najjednostavniji način za paralelizaciju je postaviti vektore paralelno s očitanjem, ali su vodoravne zavisnosti riješene algoritmom prefix-max. Kako su podaci treće generacije sekvencirana vrlo greškoviti, faza konsenzusa dodatno miče očitanja koja imaju loše poravnanje na fragmente faze razmještaja te dijelove očitanja s lošom prosječnom kvalitetom.

U poglavlju 5 ("Integracija i evaluacija") opisana je integracija spomenutih algoritama u samostojeći alat za *de novo* sastavljanje genoma te evaluacija rada s najefikasnijim poznatim metodama namijenjenih za treću generaciju sekvenciranja. Svaki alat koji je proizašao iz ove disertacije, implementiran je u programskom jeziku C++ i javno je dostupan na servisu GitHub pod MIT licencom. Algoritmi za fazu razmještaja implementirani su u samostojeći alat nazvan Rala. Vektorizacija poravnanja parcijalnog uređaja samostojeća je knjižica zvana Spoa, koja se koristi kao jezgreni dio modula za fazu konsenzusa zvanog Racon. Alati Rala i Racon s C++ implementacijom postojećeg algoritma za fazu preklapanja integrirani su u samostojeći *de novo* asembler zvan Raven. Alat Raven kao ulaz prima skup greškovitih očitanja treće generacije sekvenciranja te vraća rekonstruirani genom visoke točnosti. Kronološki gledano, prvo su razvijeni alati Spoa i Racon koji su objavljeni u časopisu s visokim faktorom odjeka, potvrđujući tezu da točne rekonstrukcije sekvenciranih genoma su moguće bez prethodnog ispravljanja grešaka u očitanjima. Nakon toga istraživane su različite heuristike za fazu razmještaja koje su povezane zajedno skriptnim jezikom u program radnog imena Ra te prezentirane na konferenciji. Naposljetku, alat Raven je izrađen s ciljem da se zaobiđe spremanje podataka na tvrdi disk između faza sastavljanja genoma te da se smanji memorijsko zauzeće cijelog postupka pomoću blokovskog načina rada.

Evaluacija alata Raven provedena je na desetak skupova podataka koji su javno dostupni, obuhvačajući obje tehnologije sekvenciranja treće generacije te raspon organizama od malih prokariotskih do većih eukariotskih. Četiri najefikasnije poznate metode za *de novo* sastavljanje genoma uključene su u evaluaciju s alatom Raven. Ocjenjivala se točnost sastavljanja u odnosu na referentni genom, NG50 vrijednost, broj dobivenih fragmenata po kromosomu, broj krivo sastavljenih fragmenata, vrijeme izvođenja te memorijsko zauzeće. Asembler Raven rangiran je treći u pogledu vremena izvođenja te drugi za potrebnu memoriju. Rekonstrukcije alata Raven dosljedno imaju najmanji broj fragmenata, ali vrijednosti NG50 su nešto manje. Točnost i broj krivo sastavljenih fragmenata usporedivi su za sve asemblere kroz sve korištene skupove podataka. Detaljniji pogled u vrijeme izvođenja alata Raven upućuje da je faza preklapanja najkritičniji dio kod većih eukariotskih genoma što otvara prostor za unaprijeđenje.

Prezentirani algoritmi integrirani u samostojeći alat Raven uspješno su potvrdili hipotezu

da je moguće ostvariti kvalitetne rekonstrukcije velikih genoma u malom broju fragmenata, i to bez prethodnog ispravljanja grešaka u podacima te pritom koristiti razumne količine računalnih resursa. Algoritmi su objavljeni u znanstvenom časopisu i prezentirani na međunarodnoj konferenciji. Alati Spoa, Racon, Rala, Ra i Raven dostupni su na sljedećim poveznicama: `https://github.com/rvaser/spoa`, `https://github.com/lbcb-sci/racon`, `https://github.com/rvaser/rala`, `https://github.com/lbcb-sci/ra` i `https://github.com/lbcb-sci/raven`.

**Ključne riječi:** *de novo*, sastavljanje, dugačka očitanja, PacBio, Oxford Nanopore, gomila preklapanja, razmještaj simulacijom djelovanja sila, poravnanje parcijalnog uređaja, vektorizacija

# Contents

**1. Introduction** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
  1.1. Research objectives . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
  1.2. Organization of the thesis . . . . . . . . . . . . . . . . . . . . . . . . 3

**2. Background** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
  2.1. Approaches to de novo genome assembly . . . . . . . . . . . . . . . . . . 5

**3. Algorithms for layout phase of the OLC paradigm** . . . . . . . . . . . . . . . 11
  3.1. Assembly graph . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
    3.1.1. Simplification methods . . . . . . . . . . . . . . . . . . . . . . . 12
    3.1.2. Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
  3.2. Preprocessing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
    3.2.1. Sequence annotation . . . . . . . . . . . . . . . . . . . . . . . . 21
    3.2.2. Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
  3.3. Postprocessing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
    3.3.1. Force-directed placement . . . . . . . . . . . . . . . . . . . . . . 27
    3.3.2. Approximation techniques . . . . . . . . . . . . . . . . . . . . . . 32
    3.3.3. Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37

**4. Algorithms for consensus phase of the OLC paradigm** . . . . . . . . . . . . . 40
  4.1. Partial order alignment . . . . . . . . . . . . . . . . . . . . . . . . . 40
    4.1.1. Vectorization . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46
    4.1.2. Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
  4.2. Unitig polishing . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
    4.2.1. Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 49

**5. Integration and evaluation** . . . . . . . . . . . . . . . . . . . . . . . . 51
  5.1. Implementation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 51
  5.2. Datasets . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54
  5.3. Evaluation methods . . . . . . . . . . . . . . . . . . . . . . . . . . . 56

5.4. Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57
5.5. Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61

**6. Conclusion** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 63
6.1. Contribution of the thesis . . . . . . . . . . . . . . . . . . . . . . . . 63
6.2. Future research . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65

**Bibliography** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 66

**Biography** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 81

**Životopis** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 83

# Chapter 1

# Introduction

Determining the order of nucleotides of a *deoxyribonucleic acid* (DNA) is called sequencing, a method that dates back from the mid 1970s. First experimental procedures, proposed by Maxam and Gilbert [1], and Sanger *et al.* [2], were based on gel electrophoresis in which the fragments of a sample were separated by length and visualized by autoradiography. This enabled the interpretation of few hundred nucleotides from the beginning of the sample, forming short sequences called *reads*. In order to reconstruct the much longer DNA molecule that is being sequenced, obtained reads need to be stitched together (*assembled*) what is usually done without a reference (*de novo*). Refinements to the electrophoretic procedures together with the chain-termination method by Sanger *et al.* marked the first generation of sequencing, which lasted for a couple of decades yielding highly accurate reads up to two thousand nucleotides.

Sequencing cost, long sequencing times and intensive labor led to the development of high-throughput sequencing technologies of the second generation, which enabled a broader range of applications, such as gene expression analysis and discovery of genomic variation [3]. Reads produced by Illumina, Roche and Life Technologies were shorter than the previous generation, ranging from 100 to 450 nucleotides, but kept the level of error around 2% [4]. Due to the inability of short reads to fully span repetitive genomic regions, unambiguosly reconstructing even small microbial genomes is near impossible [5]. To circumvent that, special protocols were developed called *paired-end* and *mate-pair* sequencing, which sequence DNA fragments from both sides. This yields a pair of reads with the same length, but the approximate distance between them is known and used as a valuable asset in genome assembly.

The most recent advance in sequencing was pushed by Pacific Biosciences and Oxford Nanopore Technologies, which increased the read lengths manifold at a cost of higher error rates and lower throughput. Together they constitute the third generation of sequencing which facilitated significant progress in contiguity of genome assemblies and removed the need for DNA amplification before sequencing. Pacific Biosciences sequencing captures the DNA during replication by using flourescent-labeled nucleotides that emit distinct spectrums. The obtained light pulses are translated to long reads with average length of $10kb$ and error rates between $11-15\%$ [6]. The polymerase can continue replication on the other DNA strand crossing over hairpin adapters to increase the overall accuracy, but is limited with its lifespan and thus longer reads usually have higher error rates. On the other hand, sequencers devised by Oxford Nanopore Technologies measure the disruption of an ion current that flows through a protein nanopore, which is incorporated into an electrically resistant membrane. Nucleotides of DNA molecules that pass through the pore will disrupt the current both in form and magnitude, yielding a signal that can as well be translated. The accuracy ranged between $65-88\%$ at the beginning with average lenghts around $8kb$ [7], but newer pores and basecallers increased that significantly. Both third generation sequencing giants are continuously improving their methods and throughput, which will mitigate the assembly problem even further. Novel protocol of Pacific Biosciences enables the generation of highly accurate reads with average length of $13.5kb$ and accuracy of $99.8\%$ [8], while the recent ultra-long protocol of Oxford Nanopore Technologies yielded reads with average lenght around $100kb$, with a whopping maximum of $882kb$ [9].

Dawn of long read sequencing started an avalanche of new algorithms aimed to deal with high error rates. Some researchers tried to adapt existing assembly pipelines by employing error correction prior assembly, while others developed algorithms sensitive enough to directly operate on erroneous reads. The result is a vast amount of de novo genome assemblers available to choose from. A lot of them demand significant amounts of computational resources, which is most apparent on large eukaryotic genomes. Therefore, in this thesis we present a novel approach to de novo genome assembly with the focus on low computational cost.

## 1.1 Research objectives

This research is aimed at development of novel algorithms for de novo assembly of large genomes. The emphasis is on assembly accuracy and contiguity, moderate memory requirements and parallel efficiency of the overlap-layout-consensus paradigm, all relying on long error-prone reads of third generation sequencing technologies. Contribution of this thesis includes the following:

- Memory efficient algorithm for layout phase of the overlap-layout-consensus paradigm achieving a low number of fragments in large genome assemblies.
- Fast algorithm for consensus phase of the overlap-layout-consensus paradigm ensuring high accuracy of large genome assemblies.
- System for de novo assembly of large genomes from data produced by third generation of sequencing.

## 1.2 Organization of the thesis

Chapter 2 kicks off with a brief overview of algorithms used for de novo genome assembly and presents the state-of-the-art assemblers.

Chapter 3 describes simplification methods for the assembly graph, utilizing information from pairwise overlaps for preprocessing and vertex distances in graph drawings for postprocessing.

Chapter 4 presents an optimized approach to multiple sequence alignment with the help of partial order graphs.

Chapter 5 displays the complete overlap-layout-consensus based assembler, which incorporates algorithms described in chapters 3 and 4. The assembler is thoroughly evaluated and compared to the state-of-the-art.

Chapter 6 brings the conclusion of the thesis with a short view on future possibilities.

# Chapter 2

# Background

The shortcoming of sequencing methods is that the read length is considerably smaller than the sequenced DNA molecule. Reads are often considered as pieces of a hefty jigsaw puzzle, and need to be interlocked in order to reconstruct the original picture, which is in this case a sequence of nucleotides. For reads to overlap, a large amount of identical DNA molecules are randomly fragmented and then sequenced, forming a process called *shotgun sequencing* [10]. Combining reads back together is done with software called *assemblers*. The main assumption upon which they are build is that similar reads originate from the same genomic region [5], but this is often invalidated due to erroneous data and repetitive nature of the genome. The assembly problem is formally defined through directed graphs for which exact solutions are computationally difficult to find. Instead, various simplification methods are applied before unambiguous paths of the graph are declared as contiguous portions of the sequenced genome.

There are two distinct approaches to assembly, one is based on the overlap-layout-consensus paradigm (OLC) describing the relationship between reads, while the other builds a de Bruijn graph from short substrings of the read set. Designed for the first generation of sequencing, the modular OLC paradigm builds an overlap graph from approximate overlaps between each pair of reads [11]. Vertices of the overlap graph encapsulate sequences while the suffix-prefix overlaps between them define the weight and orientation of edges. After applying several graph simplification methods, appropriate paths of the graph are extracted and the sequencing errors are amended with multiple sequence alignment of all reads. The huge increase of sequencing yield in form of short and accurate reads present in second generation of sequencing was problematic for the quadratic complexity of the overlap phase. This led to development of an algorithm which omits the need for pairwise overlaps. Reads are sliced into overlapping substrings of length $k$, called *k-mers*, and transformed into edges of a de Bruijn graph. Distinct $k-1$ long prefixes and suffixes of all $k$-mers become vertices of the graph. The optimal value for $k$ depends on factors such as sequencing depth and the error rate. The assembly problem of finding a Hamiltonian path in overlap graphs is replaced with finding a Eulerian path in de

Bruijn graphs [12], for which a linear algorithm is known [13]. The possibility of multiple such paths hinders the discovery of the correct assembly path, so similar simplification methods are applied as well.

Although the OLC paradigm is more suited for longer reads with arbitrary accuracy, some researchers were successful in generalization of de Bruijn graphs, making them more resilient to error-ridden data. Since the appearance of long read technologies, a wide range of options have been proposed for de novo genome assembly, majority of which are presented in the next section.

## 2.1 Approaches to de novo genome assembly

Initial sequencing attempts of Pacific Bioscinces and Oxford Nanopore Technologies had low accuracy, ranging between $82-85\%$ [14] and $58-75\%$ [15], respectively. Researches were sceptical of their usage as existing assembly pipelines could tolerate error rates up to $10\%$. Koren *et al.* introduced a correction algorithm for erroneous Pacific Biosciences reads utilizing high-quality short reads [14]. They were able to increase the contiguity of microbial genome assemblies manifold, using off-the-shelf assembler Celera [16], and proved the power of long reads to resolve repetitive genomic regions. The resulting pipeline, called PBcR, trims and error-corrects individual long reads by computing a multiple sequence alignment of short reads that are mapped to them. Long reads are split if there is a gap in the short read tiling, while setting a threshold for the number of alignments per short read effectively resolves repeats by placing each of them in its highest identity repeat copy. Consensus sequences are infered from the multiple sequence alignment using the AMOS consensus module [17].

The need to sequence the sample with two different sequencing technologies led the research explore assembly approaches that rely solely on the long reads of the third generation of sequencing. Chin *et al.* were able to produce complete high-quality microbial genomes using data only produced by Pacific Biosciences, in a hierarchical genome-assembly process (HGAP) [18]. Only the longest reads summing to a sequencing depth of 20 are error corrected with all other long reads, eliminating the necessity of second generation of sequencing. Using the BLASR mapper [19], long reads are mapped to each other controlling the maximal amount of alignment per read. Using a novel consensus tool PBDAG-Con, pairwise alignments are transformed into a directed acyclic graph from which the consensus sequence is inferred by finding the maximum scoring path of the graph, which is similar to partial order graph approach [20][21]. The accuracy of long reads increases to 99% [18], which is more than suitable for the Celera assembler. The final assembly is polished with a new consensus algorithm called Quiver reaching accuracy of $99.995\%$. Quiver searches for the maximum-likelihood consensus utilizing the full information from raw pulses obtained with sequencing. The reads are mapped to the assembly to create an approximate consensus using partial order alignment on tiling windows. Single base substitutions, insertions and deletions are tested to see if they improve the likelihood of the consensus given the raw data, until no improvements can be made.

The hybrid assembly approach was later proven by Goodwin *et al.* for data obtained with Oxford Nanopore Technologies [15], which had even higher error rates than Pacific Biosciences. They introduced a tool called Nanocorr which follows the concept of PBcR, that is error correction of error-prone long reads with high-fidelity short reads and using them as input for the Celera assembler. For alignment of short reads they used BLAST [22], the most commonly used tool for comparison of biological sequences. Obtained alignments are used to select the optimal set of short sequences that span a long read, and a consensus sequences is constructed with PBDAG-con resulting in long reads with accuracy beyond 97% [15]. All such reads are size selected and assembled with the Celera assembler yielding 10-fold increase in contiguity compared to short read assemblies.

Following the success of HGAP, Loman *et al.* were able to reconstruct a complete bacterial genome by exclusively using data from Oxford Nanopore Technologies, which meanwhile increased the accuracy to $78-85\%$ with newer nanopore chemistry [23]. They presented two novel tools, Nanocorrect and Nanopolish [23]. Nanocorrect employs multiple error correction round by first finding pairwise overlaps with DALIGNER [24], trimming and stacking reads on top of each other based on the alignments, and using the original partial order alignment implementaion [20]. The peak read accuracy of 97.7% is found after two iterations, as more rounds would reduce the number of reads without significant contributions to accuracy. Polished long reads are assembled with the Celera assembler, as all approaches so far. As the elctric signal generated with sequencing has more information than basecaleld reads, Nanopolish utilizes this information to further increase the accuracy of the final assembly. Modifications in form of substitutions, insetions and deletions are introduced to the assembly and evaluted using hidden Markov models. The final assembly of the bacterial genome had accuracy of 99.5% [23].

Due to long running times of proposed assembly algorithms for third generation of sequencing, Berlin *et al.* introduce a new algorithm for overlapping noisy long reads called the MinHash Alignment Process (MHAP), and apply it on larger eukaryotic organisms [25]. Sensitive pairwise alignment between all reads is the most time consuming step of assemblers such as PBcR and HGAP, taking a day to assemble microbial genomes [25]. This is facilitated with a dimensionalty reduction approach in MHAP. Replacing large sequences with a small set of short substrings enables a much faster similarity search. All $k$-mers of a read are converted to integers using a predefined number of randomized hash functions. The minimal value of each such hash function constitutes the sketch of the read and is used to calculated the Jaccard similarity. Using 16-mers and two MinHash filters to find similar sequences, the quadratic time complexity of the overlap step is decreased manifold [25]. The longest reads constituting $40x$ sequencing depth are error corrected using PBDAG-con, and only the longest of them summing to $25x$ sequencing depth are assembled with the Celera assembler. The final assembly is polished with Quiver. All novelties introduced were integrated into the existing PBcR assembler.

Most assemblers fuse haplotypes together loosing allelic and structural variations, which led Chin *et al.* to devise a diploid aware assembler called Falcon [26]. Following the design of HGAP, all raw reads are aligned to each other with DALIGNER and errors are corrected with Falcon-sense, a directed acyclic graph based consensus tool which preserves information from heterozygous sites. The consensus sequences are inferred with a dynamic programming algorithm similar to [21]. Afterwards, the corrected sequences go through a containment removal procedure before builing a string graph [27]. All bubble-like structures in the graph are inspected and alternative paths in them are set aside. Aligning raw reads to the stored primary and associative paths with BLASR enables the division into two haplotypes. The string graph is simplified by removing overlaps between reads of different haplotypes and each unambiguous path is polished with Quiver.

Koren *et al.* developed Canu [28], a much faster version of the PBcR pipeline which made it the successor of the Celera assembler. Canu as PBcR uses MHAP to compare sketches of entire reads, but the k-mers chosen for generation of sketches are adjusted by weighting them in regards to their frequency in the read and inverse frequency in the whole dataset. This helps decrease run time by filtering frequent k-mers out of sketches. They extend the best overlap graph approach [29] and named it Bogart, which is used to filter out repeat-induced overlaps. Usually, the best overlaps are chosen from a pool of longest overlaps having error greater than a fixed threshold, but Canu automatically estimates that threshold from the overlap error distribution. Due to high error rates of third generation of sequences, Canu as well employs error correction prior assembly with Falcon-sense, and the final assembly is polished with PBDAG-con.

Great impact on the field happened with the release of Minimap and Miniasm, tools which enabled genome assembly with erroneous third generation sequencing data without any error correction applied prior assembly [30]. Being orders of magnitude faster than the state-of-the-art, Minimap and Miniasm achieve comparable results regarding assembly contiguity, but the lack of a consensus module leaves the assembly accuracy equal to the accuracy of raw reads and is unusable for many downstream analyses. Influenced by sketches used in MHAP, Minimap chooses minimizers as the sequence representation, which are the smallest $k$-mers in a window of several consecutive ones. Minimizers of multiple sequences are stored in a hash table, from which $k$-mer matches are found and chained by finding the longest increasing subsequence of matches. Such approximate overlaps are then used to build an assembly graph, which is a modification of the overlap graph. Several graph cleaning methods are applied before the contiguous stretches of vertices are declared as the final assembly. Miniasm coupled with Minimap was evaluated on moderate size genomes, but is not well optimized for large repeatrich genomes [30].

The idea to omit error correction of reads was also independently explored by Sović *et al.* with a modified version of Graphmap [31]. Not long after, we developed a fast consensus module which coupled with miniasm enables accurate genomic reconstruction while being an order of magnitude faster than the state-of-the-art [32]. It uses raw reads to polish raw contigs, employing a SIMD* accelerated partial order alignment on small non-overlapping windows. Details about the algorithm are presented later in the thesis.

Another raw read based assembler, called HINGE, was developed by Kamath *et al.* with the aim to build a maximally resolved assembly graph [33]. A variant of the greedy algorithm coupled with methods to identify repeat regions is used to resolve unbridged repeats whenever possible. With information obtained from DALIGNER's pairwise overlaps, Hinge builds a pile-o-gram per read in order to detect peculiar regions. Sharp gradients in pile-o-grams help to annotate beginnings and ends of repeat regions. Annotations are spread to other reads with the contagion algorithm, which propagates the bridging information to reads beginning or ending in a repeat. After repeat annotation, two hinges are placed per unbridged repeat region onto reads that span the most into it. Pile-o-grams are also used to resolve chimeric reads by detecting an abrupt change of the read sets overlapping them. After preprocessing, a greedy algorithm is used to construct the initial assembly graph by picking the longest prefix and longest suffix match for each read, but allowing multiple matches for reads with hinges. Based on the graph layout, some unbridged repeat patterns can be resolved if there is only one possible traversal. The consensus sequence is computed with a variant of Falcon's consensus module.

Researchers also explored the use of de Bruijn graphs in assembly of long error-ridden reads. Lin *et al.* build A-Bruijn graphs, a generalization of the de Bruijn graphs. Instead of using all *k*-mers, a containtment-free set of solid sequences is used to construct the graph instead [34]. They are selected by observing *k*-mer frequencies and those that appear at least *t* times with cummulative sum exceeding the estiminated genome length are picked. The A-Bruijn graph is transformed into an assembly graph by removing bubbles and tips, and the genomic path is found with the help of the path extension paradigm used in short read assemblers. Paths ending at junctions are extended if reads traversing an out edge provide enough confidence to do so. Errors in the draft assembly are corrected by aligning all reads using BLASR and combining pairwise alignments into a series of small multiple alignments which are corrected separately by constructing A-Bruijn graphs again. This approach was refined in the Flye assembler by Kolmogorov *et al.* [35]. Fyle constructs arbitrary paths in an unknown assembly graph, randomly walking through the graph and choosing a random read in each junction. Obtained paths are called disjointigs and are concatenated in an arbitrary fashion. The concatenation is aligned

---

*SIMD is an abbreviation for single instruction multiple data, a paradigm which describes central processing units that perform the same operation on multiple values concurrently.

to itself to identify repetitive regions which are represented by high scoring local alignments. From this representation a repeat graph is constructed. All reads are mapped to the repeat graph and bridged repeats are resolved naturally. For unbridged repeats, Flye identifies small differences in nearly exact repeats, groups reads to each copy and then constructs separate consensus sequences.

In a similar fashion, Fuzzy de Bruijn graphs were introduced by Ruan and Li in their Wtdbg2 assembler [36]. It is much faster than the state-of-the-art assemblers Canu, Falcon and Flye, while producing assemblies of similar quality. The authors build a hash table of prefiltered *k*-mers and use it to find sequences that share them. Sequences are binned into 256bp blocks and pairwise aligned by penalizing gaps and bins that do not share *k*-mers. A subset of all distinct *k*-bins, *k* consecutive bins of any read, are used to create the vertex set of the Fuzzy de Bruijn graph, as opposed to de Bruijn graphs in which *k*-mers are used. Multiple paths between vertices are merged and their number is kept in edges, while long range information is kept by storing bin identifiers in vertices. The resulting Fuzzy de Bruijn graph is simplified with pruning of dead ends and bubble popping, and a consensus based on partial order graphs is applied across edges.

The appearance of high-fidelity reads produced by Pacific Biosciences newest sequencing protocol led Chin and Khalak to investigate faster approaches for the most time consuming part of the overlap-layout-consensus paradigm, the overlap step. They implemented a hierarchical approach using minimizers in their assembler called Peregrine [37]. Iteratively reducing the set of minimizers found from sequences with 1% of error, enabled the reduction of the similarity search space by an order of magnitude [37]. They called the whole approach SHIMMER, which is short for sparse hierarchical minimizers. Found pairwise overlaps are afterwards given to a Falcon module which creates the string graph and constructs contigs. The same approach is used to map sequences to the assembly for the consensus phase which is done with Falcon-sense. The Peregrine assembler was tested on various human datasets achieving the same results as Falcon but at a fraction of execution time [37]. They have yet to explore the use of sparse minimizers on more erroneous data.

The most recent assembler aimed for large genomes sequenced with Oxford Nanopore Technologies is called Shasta [38], which outperforms Flye and Wtdbg2 while producing similar assembly results. Shasta stores all reads in homopolymer-compressed form to decrease the noise of errors and represents them as the sequences of predetermined, fixed subset of short *k*-mers, which are chosen at random. Afterwards, they are aligned to each other in their marker forms using banded alignment, discarding most frequent marker *k*-mers. To decrease the number of alignments, a modified MinHash is used to find similar sequences in marker form, by using *m* consecutive markers. They create the marker graph by joining equal markers in overlapping reads, while undirected edges are the repercussion of marker alignments. Approximate transitive reduction, pruning of short branches and removal of bubbles are applied to simplify the graph. The draft assembly is polished with MarginPolish, a tool that builds partial order graphs based on pairwise alignment statistics obtained with hidden Markov models. It iteratively refines the assembly and outputs the graph summary to HELEN, which is a recurrent neural network based polished, to further increase the accuracy.

In this thesis we present a new assembler called Raven, which extends the Minimap-Miniasm pipeline with a standalone consensus module Racon, employs ideas from the HINGE assembler for sequences preprocessing prior assembly graph construction, and applies a novel postprocessing method for graph untangling. The result is an order of magnitude faster assembler than older state-of-the-art assemblers Canu and Falcon. Most recent advances in genome assembly have led to much faster algorithms thanks to shorter representations of reads in the overlap phase. Therefore, we evaluated Raven with the recent assemblers Flye, Wtdbg2 and Shasta.

# Chapter 3

# Algorithms for layout phase of the OLC paradigm

Given the set of nucleotide bases $\Sigma = \{A, C, G, T\}$, a set of DNA sequences is defined as $S = \{s : s = c_1c_2...c_n, c_i \in \Sigma\}$. Assemblers that follow the OLC paradigm initially find pairwise overlaps of $S$ and use them for construction of a directed weighted graph $G = (V, E)$, called the overlap graph [11]. Suffix-prefix overlaps become edges in the overlap graph, while vertices represent sequences of $S$. Depending on the vertex representation we distinguish two types of overlap graphs, the string graph [27] and the assembly graph [30]. In string graphs vertices are either the sequence begining or the sequence end [27], while vertices of assembly graphs represent whole sequences and their Watson-Crick complements* [30]. Nevertheless, all overlap graphs follow the same path both in construction and simplification. Due to its simplicity and resemblance to the double helix, we chose the assembly graph for our implementation, following the concepts of Miniasm [30] and introducing few modifications.

## 3.1 Assembly graph

Overlap graphs without multi-edges that are both containment-free and Watson-Creek complete are also called assembly graphs [30]. The former property imposes that not a single vertex is contained in any other vertex, or formally $\forall v \in V,\ \nexists w \in V \setminus v, v \subseteq w$. This can be achieved directly from pairwise overlaps by discovering which of the sequences completely overlap others. The latter property dictates that both vertices and edges have complementary pairs in the graph [30], formally $\forall v \in V, \overline{v} \in V$ and $\forall v \rightarrow w \in E, \overline{w} \rightarrow \overline{v} \in E$. Length of an edge $v \rightarrow w$, in which suffix of $v$ equals the prefix of $w$, is defined as the size of $v$'s prefix that is not in the overlap [30]. An example assembly graph is shown in Figure 3.1.

---

*For a given sequence $s \in S$, its Watson-Crick complement is defined as $\overline{s} = \overline{c_1c_2...c_n} = \overline{c}_n\overline{c}_{n-1}...\overline{c}_1$, where $\overline{A} = T$, $\overline{C} = G$, $\overline{G} = C$, and $\overline{T} = A$.

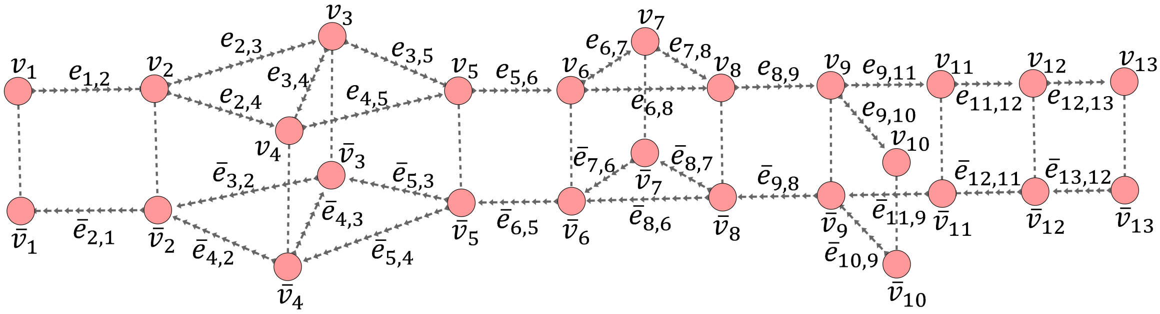


**Figure 3.1:** Example of an assembly graph constructed from an arbitrary set of sequences $S$ with cardinality $|S| = 13$. Vertices are denoted with $v_i$ and linked with dashed lines to their Watson-Crick complements $\bar{v}_i$. Edges are denoted with $e_{i,j}$, while their complenetary pairs with $\bar{e}_{j,i}$. The graph was manually drawn and annotated in Cytoscape [39]. Due to clarity, edge labels are absent in subsequent figures.

Given a set of approximate pairwise overlaps, we separate internal and contained overlaps from suffix-prefix overlaps following Algorithm 5 described in [30]. To be more resilient to erroneous data we do not carry out any overlap filtering beforehand. The maximal overhang constraint is removed but we decrease the overhang to overlap length ratio. Overlaps classified as internal are set aside for later use (see Section 3.2.1), contained sequences and their overlaps are removed, and all suffix-prefix overlaps are used as building blocks of the assembly graph.

Ideally we would reconstruct the sequenced genome by finding a Hamiltonian path in the graph [40], a path that visits each vertex only once. In computer science this problem is characterized as NP-complete and cannot be solved in feasible time [41]. Therefore, a series of simplification methods are used instead to find unambiguous paths in the graph, called *contigs*, that cover contiguous regions in the sample [42]. Usually the final assembly consists of high-confidence contigs called *unitigs*, which either span whole chromosomes or end at repetitive genomic regions [16].

### 3.1.1 Simplification methods

The assembly graph undergoes several modifications of which transitive reduction is utilized first. Given a triplet of vertices $(v_i, v_j, v_k)$, edge $v_i \rightarrow v_k$ is transitive if there exist edges $v_i \rightarrow v_j$ and $v_j \rightarrow v_k$. We can safely remove it as there exists a path over $v_j$ that connects $v_i$ and $v_k$. Although, removing transitive edges one at a time is not advisable as some of them may imply others (Figure 3.2). A linear time algorithm ($O(|E|)$) for transitive reduction has been proposed by [27], but we examine all possible candidates and determine if the path lengths are comparable (maximal difference of 12%), as depicted in Algorithm 1.

Next in order are dead ends of the assembly graph, branches of contigs that cease abruptly either due to sequencing errors or low sequencing coverage (Figure 3.3). They can be located by launching a depth first search from vertices with $deg^{-}(v) = 0$. The search stops when we reach

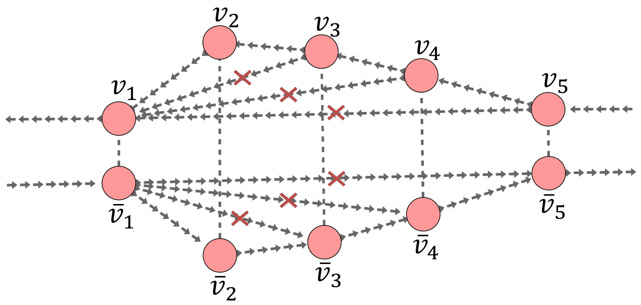


**Figure 3.2:** Part of an assembly graph containing transitive edges. For any path of three vertices $v_i \to v_j \to v_k$, a transitive edge connects the first and last vertex of that path, $v_i \to v_j$. It can be removed without any loss of information, but removing any of them separately might hinder the detection of others. In this example, edge $v_5 \to v_1$ is not transitive without edge $v_4 \to v_1$, as well as $v_4 \to v_1$ without $v_3 \to v_1$. The subgraph was manually drawn and annotated in Cytoscape [39].

**Algorithm 1** Transitive reduction of an assembly graph

```
Input: Assembly graph $G = (V, E)$.
Output: None. (Procedure updates the given graph.)
procedure AssemblyGraphTransitiveReduction($V, E$)
    for all $v \in V$ do
        for all $v \to w \in E$ do
            $w_{mark} \leftarrow 1$
        for all $v \to w \in E$ do
            for all $w \to x \in E$ do
                if $x_{mark}$ and $|v \to w| + |w \to x| \in |v \to x| \pm \varepsilon$ then
                    $(v \to x)_{mark} = 1$
        for all $v \to w \in E$ do
            $w_{mark} \leftarrow 0$
    for all $v \to w \in E$ do
        if $(v \to w)_{mark}$ then
            $E \leftarrow E \setminus \{v \to w\}$
```

a junction vertex, which has multiple outgoing edges or multiple incoming edges. The dead end is pruned only if the junction vertex has $deg^{-}(v) > 1$, meaning there should be another path leading up to it. In addition, we ignore paths longer than 5 vertices. Pseudocode is available in Algorithm 2 and is run iteratively until no changes occur in the graph.

More complex structures located in assembly graphs are called *bubbles*, an arangement of multiple paths connecting two vertices. In haploid organisms they occur due to sequencing errors, while in diploid and polyploid as a result of genomic variation. It is desireable to resolve bubbles (*pop* them) by retaining only one path. We search for vertices with two different paths using breadth first search as in Algorithm 3, similar to [43]. Afterwards, we check whether both paths span the same sequence. We deem it necessary in order to distinguish bubbles which incurred due to false overlaps. Minimizers are collected from both path sequences as in minimap [30], and the lists are sorted and merged to identify the longest overlap as in Daligner [24]. If

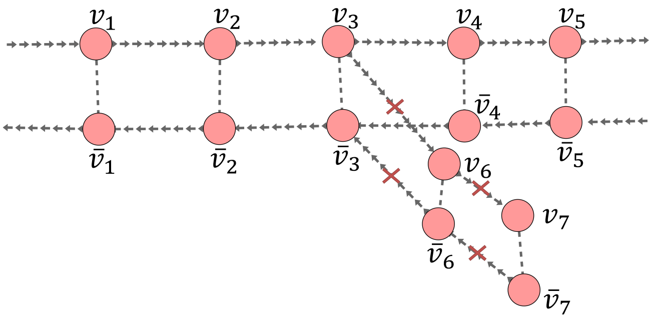


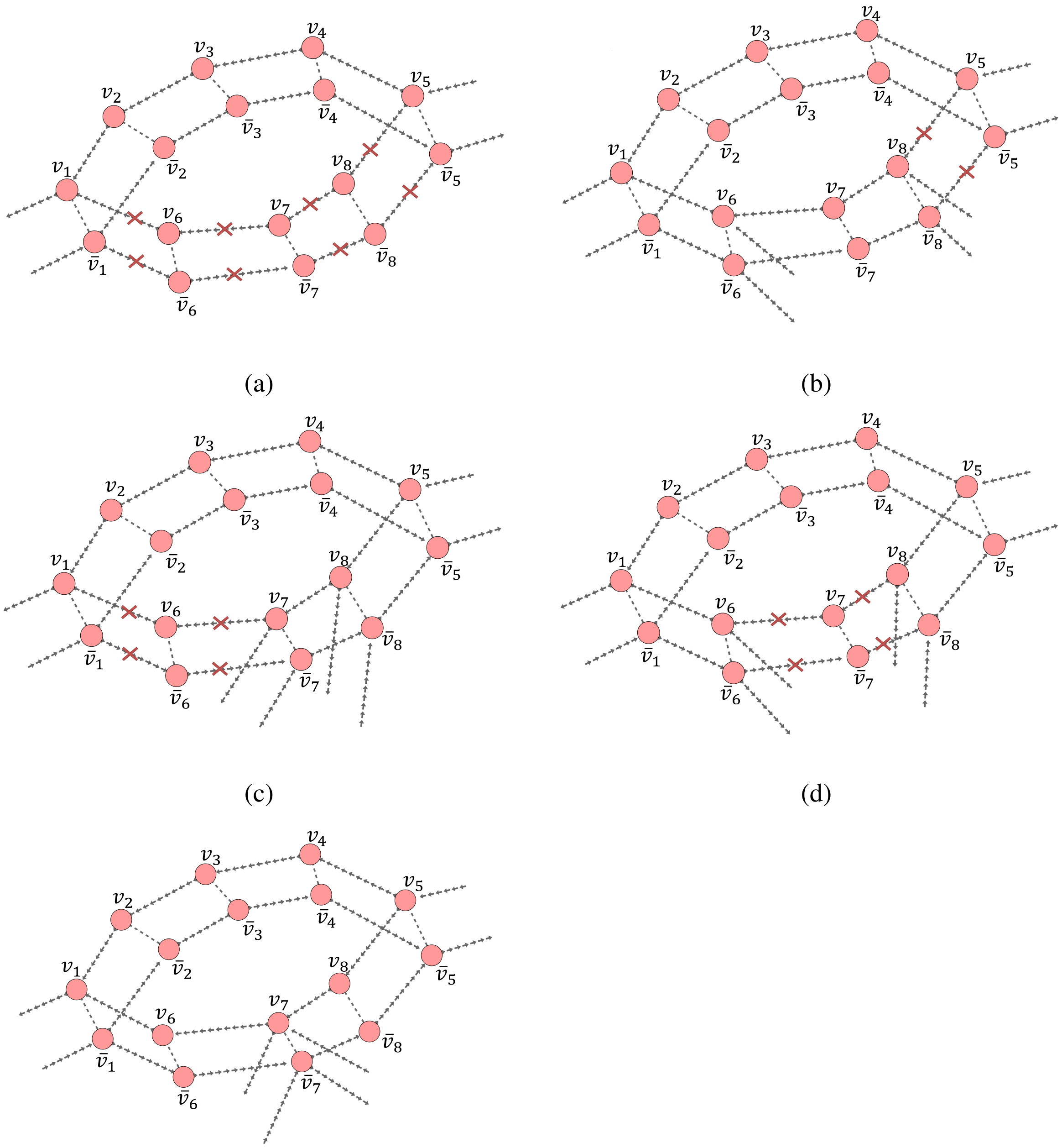
**Figure 3.3:** Part of an assembly graph containing a dead end (path $v_3 \rightarrow v_6 \rightarrow v_7$). Removing such paths should not lead to loss of information. The subgraph was manually drawn and annotated in Cytoscape [39].

**Algorithm 2** Pruning of dead ends in an assembly graph

```
Input: Assembly graph $G = (V, E)$.
Output: None. (Procedure updates the given graph.)
procedure AssemblyGraphPruning($V, E$)
    for all $v \in V, deg^-(v) = 0$ do
        $n \leftarrow 0$
        $v' \leftarrow v$
        while $v' \rightarrow w \in E$ do
            $v' \leftarrow w$
            if $deg^+(v') > 1$ or $deg^-(v') > 1$ then
                break                                  ▷ Reached a junction
            $n \leftarrow n + 1$
        if $deg^-(v') > 1$ and $n < 6$ then
            while $v \neq v'$ and $v \rightarrow w \in E$ do
                $E \leftarrow E \setminus \{v \rightarrow w\}$
                $v \leftarrow w$
```

the percentage of matching bases is at least 50% we take a step further and identify which path to remove. More precisely, we look for edges which if removed will not break any other path of the graph. Figure 3.4 depicts five such scenarios. The path of the bubble with fewer vertices is examined first, and if such edges do not exist, the other path is considered instead. Pseudocode for bubble resolution is shown in Algorithm 4. The whole algorithm is evoked until the graph remains unchanged, because removal of some bubbles can facilitate removal of others.

Eventually, we exhausted all simplification methods and want to find unitigs, unambiguous paths which contain vertices with $deg^-(v) = deg^+(v) = 1$. This is easily achievable by picking any such vertex and expanding it in both directions until we reached the end, made a circle or encountered a vertex without the defined property. Afterwards, the unitig path is collapsed into a single vertex which represents a compound sequence of all contained vertices. The sequence is a concatenation of vertex prefixes which length is encoded in the edges. Algorithm 5 depicts this process.



(a)

(b)

(c)

(d)

(e)

**Figure 3.4:** Different path formations in bubble-like structures of the assembly graph. We assume that the path $v_5 \to v_4 \to v_3 \to v_2 \to v_1$ is the one we want to keep. The path over vertices $v_8$, $v_6$ and $v_7$ is searched for edges that will not discontinue other paths of the graph if removed. Subfigure (a) depicts the simplest scenario in which the whole path can be removed. When they are several vertices with indegree $deg^-(v) \geq 2$ and outdegree $deg^+(v) = 1$, everything before the first such vertex can be removed as in subfigure (b). Similar rule applies to the case in subfigure (c) where they are several vertices with $deg^+(v) \geq 2$ and $deg^-(v) = 1$. Everything after the last such vertex is safely removable. When there is a combination of those vertex types, edges between the last vertex with $deg^+(v) \geq 2$ and the first vertex with $deg^-(v) \geq 2$ can be safely removed, as seen in (d). This only applies when vertices do not have multiple edges of the other type, that is if there is vertex with $deg^-(v) \geq 2$ and $deg^+ \geq 2$ in the path then no edges are removed as they would break other paths in the graph (subfigure (e)). All subfigures were manually drawn and annotated in Cytoscape [39].

**Algorithm 3** Detection and resolution of bubble-like structures in an assembly graph

```
Input: Assembly graph $G = (V, E)$.
Output: None. (Procedure updates the given graph.)
procedure AssemblyGraphBubblePopping($V, E$)
    for all $v \in V, deg^+(v) \geq 2$ do
        for all $v' \in V$ do                                ▷ Initialize breadth first search
            $v'_{distance} \leftarrow 0$
            $v'_{predecessor} \leftarrow$ null
        $B \leftarrow$ null
        $Q \leftarrow$ new Queue($v$)
        while $|Q| \neq 0$ do
            $v' \leftarrow$ POP_FRONT($Q$)
            for all $v' \rightarrow w \in E$ do
                if $w = v$ then
                    continue                                 ▷ Found a cycle
                if $v'_{distance} + |v' \rightarrow w| > 500000$ then
                    continue                                 ▷ Out of reach
                if $w_{predecessor} \in V$ then
                    $B \leftarrow (v', w)$                    ▷ Found sink
                    break
                $w_{distance} \leftarrow v'_{distance} + |v' \rightarrow w|$
                $w_{predecessor} = v'$
                PUSH_BACK($Q, w$)
        if $B =$ null then
            continue
        $(v', v'') \leftarrow B$                              ▷ Backtrack to find paths
        $P' \leftarrow$ new Array($v' \rightarrow v''$)
        while $v'_{predecessor} \neq$ null do
            Append($P', v'_{predecessor} \rightarrow v'$)
            $v' \leftarrow v'_{predecessor}$
        $P'' \leftarrow$ new Array
        while $v''_{predecessor} \neq$ null do
            Append($P'', v''_{predecessor} \rightarrow v''$)
            $v'' \leftarrow v''_{predecessor}$
        if $|P' \cap P''| > 2$ then                          ▷ An inner bubble is not resolved
            continue
        GraphPopBubble($P', E$) or GraphPopBubble($P'', E$)  ▷ Algorithm 4
```

**Algorithm 4** Resolution of a single bubble as described in Figure 3.4

```
Input: Candidate path P of an arbitrary bubble and the edge set $E$ of assembly graph $G$.
Output: True if edges are removed, false otherwise.
procedure AssemblyGraphPopBubble($P, E$)
    $i \leftarrow -1$
    for $k \leftarrow 0$ to $|P| - 1$ do                    ▷ Find first vertex $i$ with multiple incoming edges
        $(v \rightarrow w) \leftarrow P[k]$
        if $deg^{-}(w) > 1$ then
            $i \leftarrow k + 1$
            break
    $j \leftarrow -1$
    for $k \leftarrow 0$ to $|P| - 1$ do                    ▷ Find last vertex $j$ with multiple outgoing edges
        $(v \rightarrow w) \leftarrow P[k]$
        if $deg^{-}(w) > 1$ and $deg^{+}(w) > 1$ then
            return false
        if $deg^{+}(w) > 1$ then
            $j \leftarrow k + 1$
    if $i = j = -1$ then                                    ▷ Remove whole path
        for all $v \rightarrow w \in P$ do
            $E \leftarrow E \setminus \{v \rightarrow w\}$
    else if $i = -1$ then                                   ▷ Remove edges after $j$
        for all $v \rightarrow w \in P[j, |P|]$ do
            $E \leftarrow E \setminus \{v \rightarrow w\}$
    else if $j = -1$ then                                   ▷ Remove edges before $i$
        for all $v \rightarrow w \in P[0, i]$ do
            $E \leftarrow E \setminus \{v \rightarrow w\}$
    else if $j < i$ then                                    ▷ Remove edges between $j$ and $i$
        for all $v \rightarrow w \in P[j, i]$ do
            $E \leftarrow E \setminus \{v \rightarrow w\}$
    else
        return false
    return true
```

**Algorithm 5** Unitig creation in an assembly graph

```
Input: Assembly graph $G = (V, E)$.
Output: None. (Procedure updates the given graph.)
procedure AssemblyGraphUnitigs($V, E$)
    for all $v \in V, deg^+(v) = deg^-(v) = 1$ do
        $v' \leftarrow v$
        while $w \rightarrow v' \in E$ and $deg^+(w) \leq 2$ and $deg^-(w) \leq 2$ do
            $v' \leftarrow w$
            if $v' = v$ then
                break
        $v'' \leftarrow v$
        while $v'' \rightarrow w \in E$ and $deg^+(w) \leq 2$ and $deg^-(w) \leq 2$ do
            $v'' \leftarrow w$
            if $v'' = v$ then
                break
        $u \leftarrow$ new Vertex
        if $v' \neq v''$ then                                ▷ Reconnect edges if unitig is not circular
            if $w \rightarrow v' \in E$ then
                $E \leftarrow (E \setminus \{w \rightarrow v'\}) \cup \{w \rightarrow u\}$
            if $v'' \rightarrow w \in E$ then
                $E \leftarrow (E \setminus \{v'' \rightarrow w\}) \cup \{u \rightarrow w\}$
        while $v' \neq v''$ and $v' \rightarrow w \in E$ do                    ▷ Join sequences
            $u_{sequence} \leftarrow u_{sequence} + (v'_{sequence})[0, |v' \rightarrow w|]$
            $E \leftarrow E \setminus \{v' \rightarrow w\}$
            $v' \leftarrow w$
        if $deg^+(v'') = 0$ then
            $u_{sequence} \leftarrow u_{sequence} + v''_{sequence}$
        $V \leftarrow V \cup \{u\}$
```

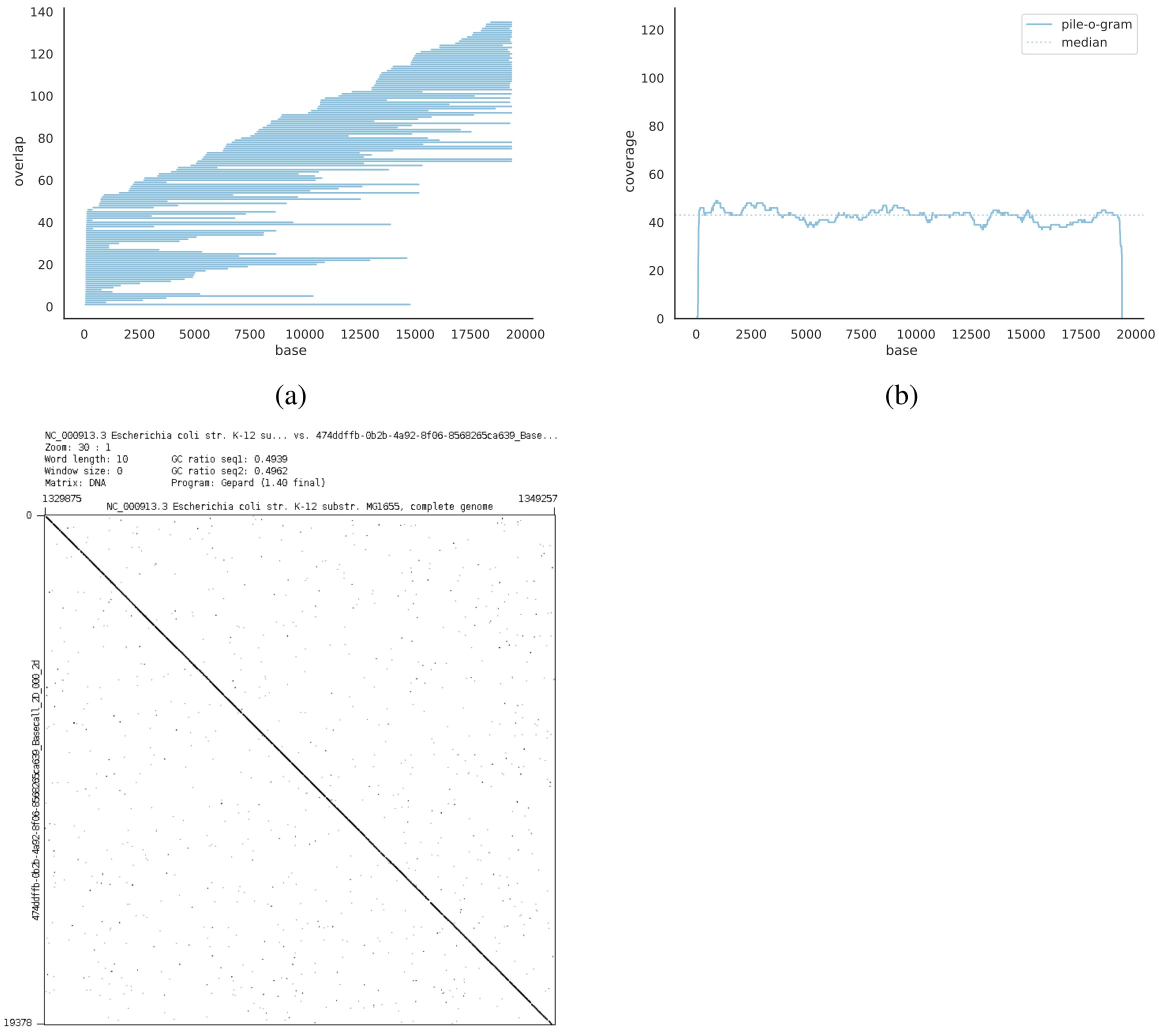
### 3.1.2 Discussion

Described methods for assembly graphs are used by almost all OLC based assemblers in some form. Sequencing errors and false approximate overlaps spawn edges in the graph that are not present in the sample, and above simplification methods have a hard time dealing with them. Therefore, additional heuristic methods are used. For example, Miniasm removes short overlaps from junction vertices if there is a much longer overlap present [30]. We employed the same technique at first but later switched to a more robust method based on graph drawings (see Section 3.3). On the other hand, HINGE utilizes the information from the set of pairwise overlaps to annotate problematic regions in sequences, and incorporates that in the best overlap graph [33]. We implemented a similar approach prior the construction of a full assembly graph.

## 3.2 Preprocessing

Leftover tangles in simplified assembly graphs are mostly the repercussions of sequencing artefacts and repetitive genomic regions. The former are called *chimeric* sequences. This are sequences consisting of multiple parts which are arranged in a way that is absent in the genome. They occur either by accidental fragment joining during sample preparation or due to misread sequence adapters. The latter cause of tangles implies overlaps between sequences that have repetitive regions on either of their ends. A repetitive region is said to be *bridged* if it is completely contained in a sequence. If not properly handled, sequences with unbridged repeats can spawn false overlaps which are not present in the sample.

An elegant way to detect chimeric and repetitive regions is with the help of *pile-o-grams*, which are extensively used in the HINGE assembler [33]. Pile-o-gram of a sequence is created by stacking all of its pairwise overlaps on top of each other (Figure 3.5a) and summing up the number of overlaps covering each base. This will yield an one dimensional signal (Figure 3.5b) which is suitable to infer usefull information from the whole dataset. Sequences that uniquely and fully map to the sequenced genome (Figure 3.5c) have almost uniform coverage across their pile-o-gram, while other have detectable fluctuations. This enables annotation of problematic regions which HINGE finds with gradients [33]. It truncates chimeric sequnces to the longest non-chimeric part, while repeat annotations are used to find sequences that overlap unbridged repeats to allow some of them multiple overlaps in an otherwise best overlap graph [33]. Our implementation uses coverage medians alongside gradients to annotate sequences. We deal with chimeric sequences the same way, but use repeat annotations to remove a portion of overlaps before graph construction. Section 3.2.1 describes that in detail.



(a)

(b)

(c)

**Figure 3.5:** Sequence pile-o-gram construction from a set of pairwise overlaps. Overlaps are stacked into a pile (subfigure (a)) and summed up over each base in the sequence (subfigure (b)). If we align the sequence to the reference genome, we can see that it is fully aligned to only one location (subfigure (c)), which results in an almost uniform pile-o-gram. Such sequences are the basis for high quality assemblies. The whole figure is tied to sequence *474ddffb-0b2b-4a92-8f06-8568265ca639* obtained from the dataset ERR1147227. Subfigures (a) and (b) were drawn in Python with Matplotlib [44], while subfigure (c) was exported from Gepard [45].

**Algorithm 6** Slope detection in a pile-o-gram

```
Input: Pile-o-gram $P$ and coverage ratio $r$.
Output: Vector of found downward and upward slopes.
procedure PILEOGRAMSLOPEDETECTION($P, r$)
    $w \leftarrow 52$
    $L \leftarrow$ new Queue
    $R \leftarrow$ new Queue
    $S \leftarrow$ new Array
    for $i \leftarrow 0$ to $k$ do
        PUSH_BACK($R, (P[i], i)$)                    ▷ Initialize right window
    for $i \leftarrow 0$ to $|P|$ do
        while $|L| \neq 0$ and (FRONT($L$))$[1] \leq i - 1 - w$ do
            POP_FRONT($L$)                           ▷ Remove bases out of reach
        while $i > 0$ and $|L| \neq 0$ and (BACK($L$))$[0] \leq P[i-1]$ do
            POP_BACK($L$)                            ▷ Remove all smaller values
        PUSH_BACK($L, (P[i-1], i-1)$)
        if $r \cdot P[i] <$ ( FRONT($L$))$[0]$ then
            APPEND($S, (i, 0)$)                      ▷ Record downward slope
        while $|R| \neq 0$ and (FRONT($R$))$[1] \leq i$ do
            POP_FRONT($R$)                           ▷ Remove bases left to current position
        while $i < |P| - w$ and $|R| \neq 0$ and (BACK($R$))$[0] \leq P[i+w]$ do
            POP_BACK($R$)                            ▷ Remove all smaller values
        PUSH_BACK($R, (P[i+w], i+w)$)
        if $r \cdot P[i] <$ ( FRONT($R$))$[0]$ then
            APPEND($S, (i, 1)$)                      ▷ Record upward slope
    return $S$
```

### 3.2.1 Sequence annotation

Pile-o-grams are represented with a vector of unsigned integers with size that equals the length of corresponding sequences. To reduce the impact on random-access memory without changing the structure of pile-o-grams, we downsample them to contain every 16-th base. Such pile-o-grams are first used to remove adapters and low coverage regions from sequences, similar to Miniasm which does that directly from overlaps [30]. Each of them is scanned through to identify the longest region covered by at least four overlaps. Everything outside that region is trimmed away. Afterwards, we search pile-o-grams again to identify coverage slopes, abrupt increases or decreases in coverage. We keep a sliding window, of fixed length $w$, left and right from the current position of the scanning procedure. The maximum value of each window is compared to the current value, and the slope is recorded if the coverage ratio is large enough. The minimal ratio depends on the type of fluctuation we are searching for, for chimeric sequences it equals to 1.84 and for repetitive regions to 1.42 (both values empirically determined). Pseudocode for slope detection is in Algorithm 6.

Collected slopes are grouped into more complex shapes such as rifts and ridges, and each group is further investigated. We declare rifts between a downward slope and an upward slope, while the opposite constitutes ridges. To decrease the number of false annotations, we utilize the information about approximate sequencing depth by calculating pile-o-gram medians[†]. This will also help to properly treat different sequencing depths and molecule copy numbers (e.g. plasmids). Sequences are grouped with suffix-prefix overlaps into connected components, and component medians are used to to determine the relevance of each rift and ridge.

A rift is declared chimeric if it contains a base with coverage bellow the component median divided by 1.84 (value empirically determined). Annotated pile-o-gram of a chimeric sequence is presented in Figure 3.6. When all chimeric rifts are annotated, sequences are shrunk to their longest non-chimeric region. Overlaps that are declared internal are examined again if any of them became a suffix-prefix overlap due to changes in chimeric sequences.

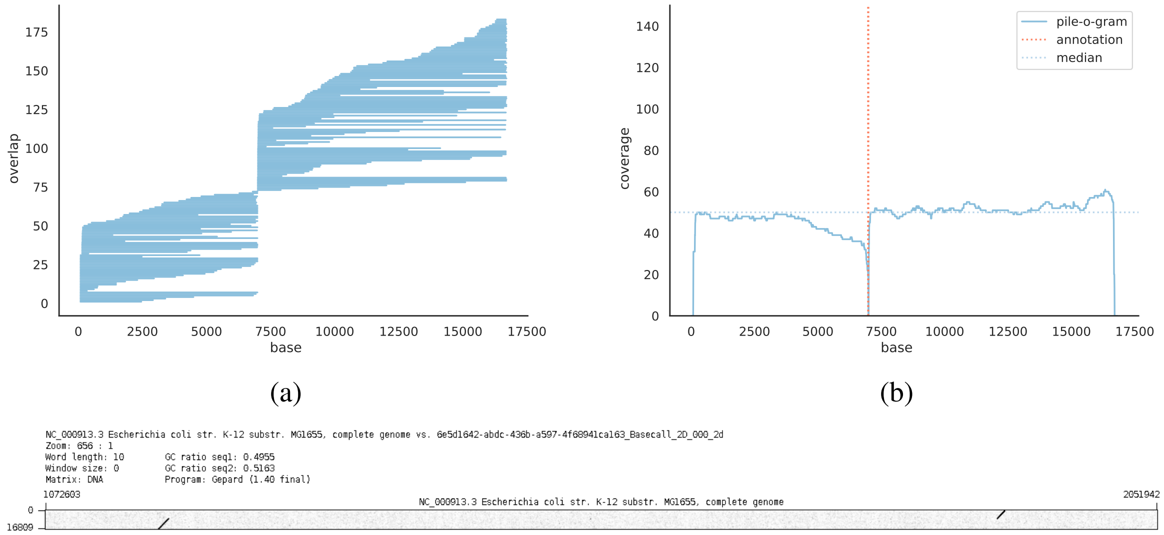
On the contrary, all bases inside a ridge are examined if their coverage is greater than the component median multiplied with 1.42 (value empirically determined). If at least 90% of bases conforms to this constraint, the ridge is declared as a repetitive genomic region, as depicted in Figure 3.7. Later we look for suffix-prefix overlaps that do not bridge annotated ridges at either sequence end (Figure 3.8a). They are removed from the overlap set if there exists at least one overlap bridging the ridge in question (Figure 3.8b). Removing overlaps without this constraint might fragment the assembly graph, if a repetitive genomic region is not bridged by any sequence.

### 3.2.2 Discussion

Even though pile-o-grams help clearing up false paths in the assembly graphs, the presented method might generate misleading annotations. For instance, sequences with low coverage regions might be wrongfully declared chimeric (Figure 3.9). The problem arises if contained sequences are removed before chimeric sequences are resolved. Leaving only the longest non-chimeric part of a sequence can lead to fragmented assemblies, if other sequences were contained in the parts that were trimmed away. This is especially troublesome for ultra-long sequences that are declared chimeric. Therefore, containment removal is aided with rift annotations. If a sequence has a chimeric rift in its pile-o-gram, it can not contain any other sequence at this point. We resolve chimeric sequences after containment removal in order to simplify retrieval of connected components.

---

[†] Methods based on coverage medians are also applied in the Unicycler assembler [46].

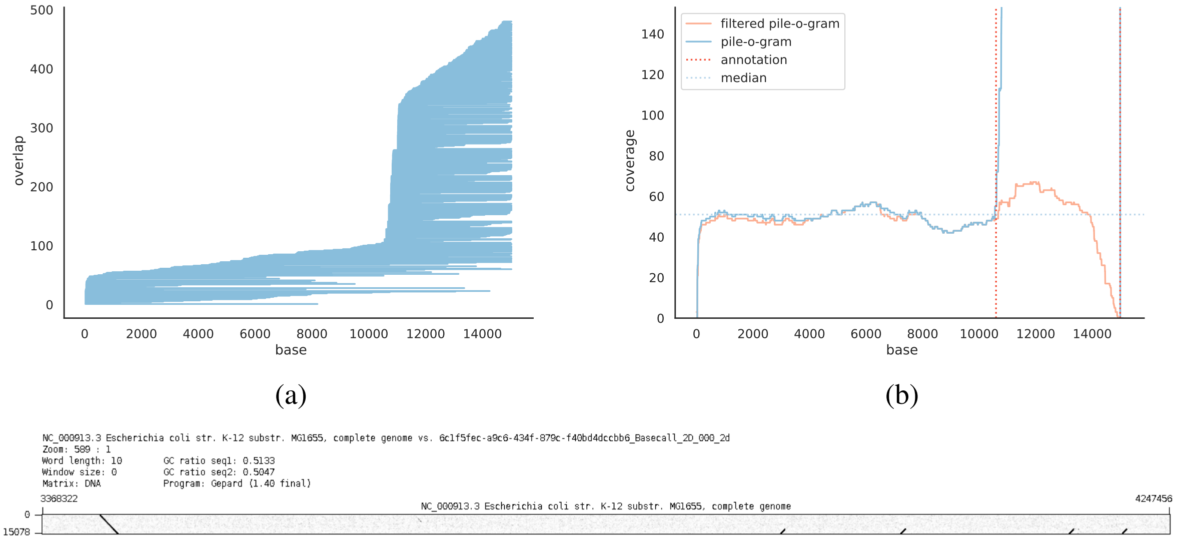


(a)

(b)

(c)

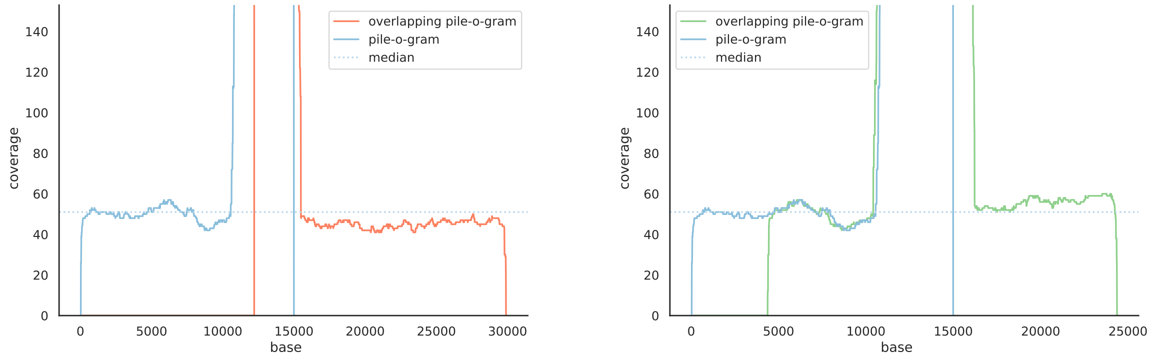
**Figure 3.6:** Pile-o-gram of a chimeric sequence. Not a single overlap covers the neighbourhood of the base at position 6900, which is an indication of a chimeric sequence (subfigure (a)). The resulting pile-o-gram has a rift around that position, which is detectable without difficulties (subfigure (b)). When the sequence is aligned to the reference genome, the alignment is split into two parts that are dislocated on the reference (subfigure (c)), verifying the assumption about the chimeric nature of the sequence. The figure is tied to sequence *6e5d1642abdc-436ba597-4f68941ca163* obtained from the dataset ERR1147227. Subfigures (a) and (b) were drawn in Python with Matplotlib [44], while subfigure (c) was exported from Gepard [45].



(a) (b)

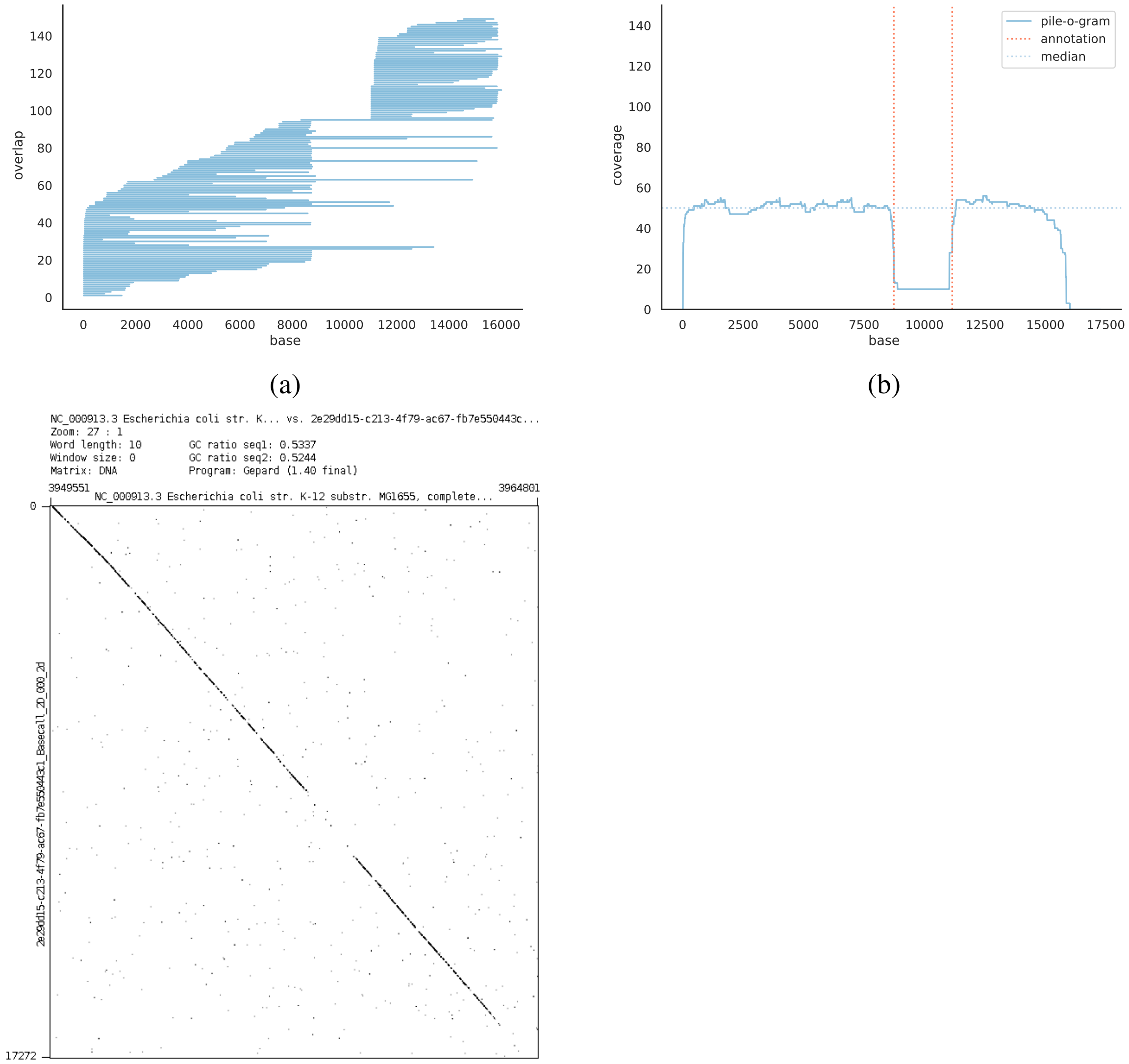
(c)

**Figure 3.7:** Pile-o-gram of a sequence containing a repetitive region. A large amount of overlaps located at the end of the sequence is a good indicator for a reptitive genomic region (subfigure (a)). When the overlaps are transformed into a pile-o-gram, the sequence end has a large ridge over the median (subfigure (b)). If k-mer filtering is employed on the whole dataset, the ridge is lost in the pile-o-gram due to lack of repetitive k-mers (pastel orange line in subfigure (b)). Aligning the sequence to the reference genome gives one primary alignment and several small ones at various positions in the genome (subfigure (c)), confirming that the end of sequence is part of a repetitive region. The whole figure is tied to sequence *6c1f5fec-a9c6-434f-879c-f40bd4dccbb6* obtained from the dataset ERR1147227. Subfigures (a) and (b) were drawn in Python with Matplotlib [44], while subfigure (c) was exported from Gepard [45].



(a) (b)

**Figure 3.8:** Overlaps between pile-o-grams that contain reptitive regions. Subfigure (a) shows a false overlap over the repetitive region, which we are trying to remove prior the construction of an assembly graph. The removal is prohibited if there are no overlaps that bridge through the repeat in question. Example of such overlap is shown in subfigure (b). Pile-o-gram drawn in blue is tied to sequence *6c1f5fec-a9c6-434f-879c-f40bd4dccbb6*, while the red and green pile-o-grams are tied to *6a7957d9-47e3-4b27-bc1c-b38f8e349884* and *8db143ed-9007-4661-8174-634f94fb6a3b*, respectively. All sequences were obtained from the dataset ERR1147227. Both subfigures were drawn in Python with Matplotlib [44].



(a)

(b)

(c)

**Figure 3.9:** Pile-o-gram of a incorrectly declared chimeric sequence. Region between 8750 and 11000 has small coverage probably due to low quality (subfigure (b)). It is declared chimeric even though there are a couple overlaps covering it (subfigure (a)). The alignment confirms that the sequence is in fact non-chimeric. It aligns to the reference genome completely and uniquely, although the low quality region is not part of the alignment (subfigure (c)). The whole figure is tied to sequence *2e29dd15-c213-4f79-ac67-fb7e550443c1* obtained from the dataset ERR1147227. Subfigures (a) and (b) were drawn in Python with Matplotlib [44], while subfigure (c) was exported from Gepard [45].

A different obstacle is tied to repetitive regions. Our implementation relies on Minimap [30] for approximate sequence matching due to its low running times. Minimap filters out the most frequent k-mers as a trade-off between sensitivity and execution speed. Ignored k-mers mostly originate from repetitive genomic regions which obstructs the detection and annotation of ridges in pile-o-grams. An example can be seen in Figure 3.7b. Decreasing the k-mer filter on the set of all sequences $S$ is unfeasible for larger genomes. The issue is bypassed by recalculating pile-o-grams after containment removal and chimeric resolution. All contained sequences are overlapped with the surviving sequences anew, a scenario in which decreasing the k-mer filter is affordable. This way even higher copy number molecules will have distinguishable ridges in their pile-o-grams.

Some tangles in the assembly graph are undetectable due to high error rates and the above approaches being heuristic. For that reason we implement a postprocessing algorithm based on graph drawings on top of everything mentioned so far.

## 3.3 Postprocessing

Even though the presented state of the assembler is able to handle a lot of graph layouts, removing edges based on overlap length could eventually yield misassemblies. The problem is in the locality of the approach. The culprit vertex which spawns a false path might be anywhere along that path, and comparing the overlap lengths from the junction to the first vertex in each path might be misleading. Therefore, we wanted to explore an approach that is more robust in identifying paths that connect distant parts of the assembly graph, and thus the genome.

The idea to draw the assembly graph in a two-dimensional Euclidean plane and use the encoded information about vertex distances came by accident. During the development of our de novo assembler, we used various tools for visualization which helped to check whether the implemented methods were doing what they are supposed to. We extensively used Cytoscape [39] to draw the assembly graph, and that was usually done at the end of the layout phase to locate problematic vertices that cause fragmentation. Once we drew the assembly graph right after construction, we noticed that some of the edges were quite elongated (a sample graph is shown in Figure 3.10). After closer examination, the elongated edges were either tied to undetected chimeric sequences or they represented unresolved overlaps of repetitive genomic regions. This was enough incentive to delve deeper into the force-directed placement algorithm [47], which is one of the underlying algorithms used in Cytoscape.

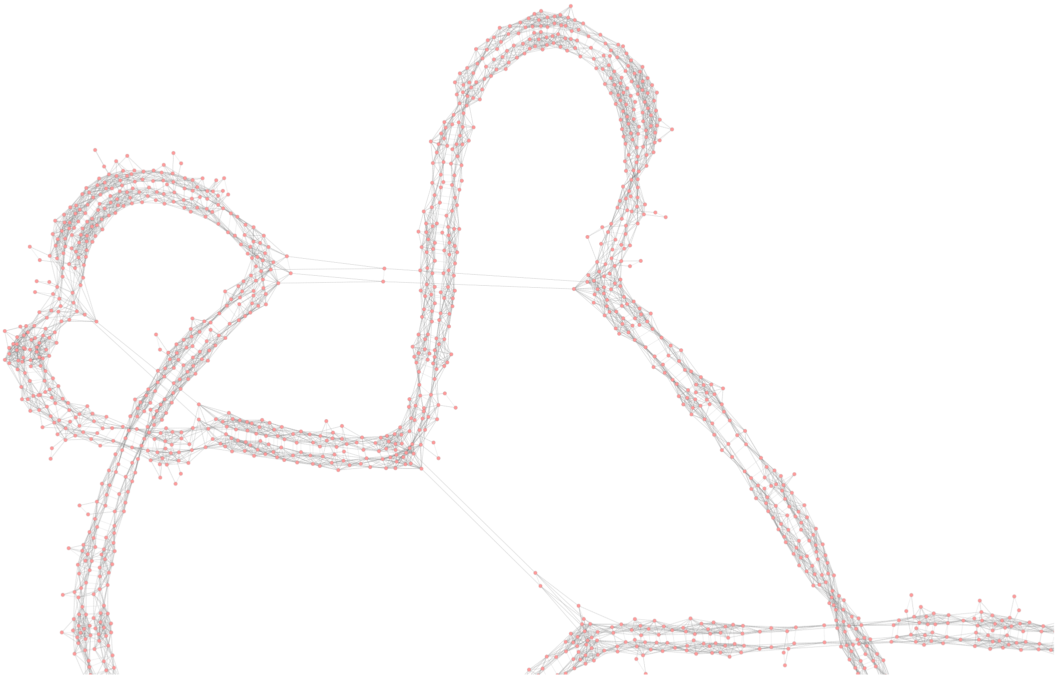


**Figure 3.10:** Magnified view of an assembly graph which shows three extended edges connecting remote parts of the graph. The assembly graph was constructed from the bacterial dataset ERR1046594 and was drawn with Cytoscape [39] (using the prefuse force-directed layout option) prior to applying any simplification methods.

### 3.3.1 Force-directed placement

Designed for drawing undirected graphs with straight edges, the force-directed placement algorithm tends to draw vertices closely together if they are connected with an edge, but not too close [47]. The authors compare the graph to a system of particles that attract and repulse each other, in which the goal is to find a static equilibrium. First, each vertex is asigned a random position on the canvas. Afterwards, in an iterative fashion, the algorithm updates vertex positions by applying forces between them. Given an optimal distance $k$, which equals to the square root of the canvas surface divided by the number of vertices $|V|$, the attractive force is defined as $f_{attractive}(d) = \frac{d^2}{k}$, while the repulsive forces is defined as $f_{repulsive}(d) = -\frac{k^2}{d}$. They cancel each other out when the distance $d$ between two vertices reaches the optimal value $k$ [47]. The total displacement of a vertex is bounded by the value $t$ (which is decreased in every iteration) and is ignored if the vertex would leave the given canvas frame. The attractive forces are only calculated between vertices that are connected with an edge ($O(|E|)$), while the repulsive forces are calculated for each pair of vertices in the graph ($O(|V|^2)$). Due to the quadratic time complexity of one iteration limiting the number of iterations is a must for larger graphs, but good drawings can be achievied with at most 100 iterations [47].

For assembly graphs we merge each vertex pair and each edge pair into a single vertex and edge, respectively, as they counterparts are redundant in the drawing. Same holds for the edge directions which are disregarded. We create the drawing at the end of the layout phase where the

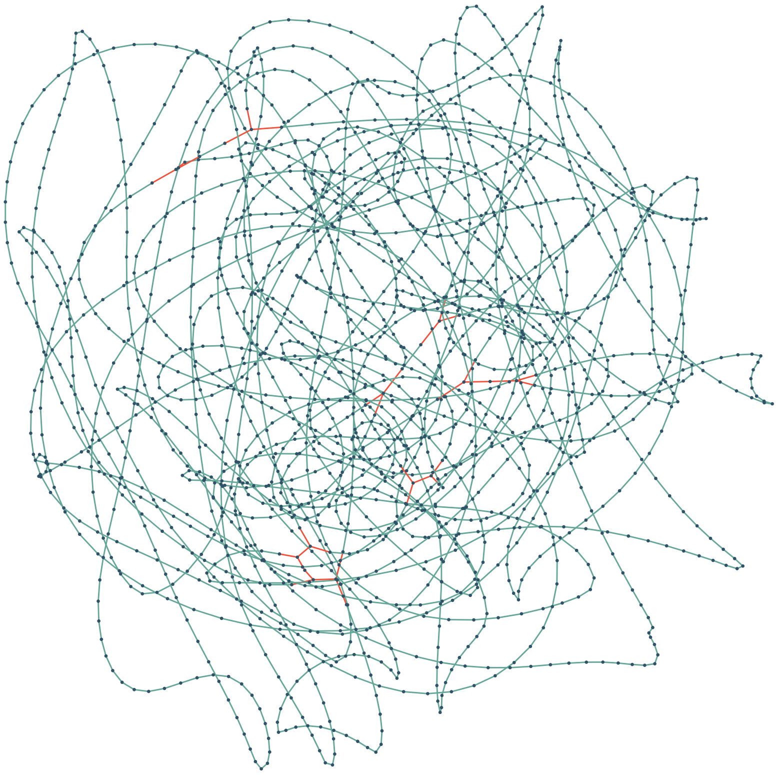


**Figure 3.11:** Condensed assembly graph displaying uniform edge lengths when using the force-directed placement algorithm [47]. Edges of interest, those tied to vertices with outdegree greater than two, are coloured in red and are really hard to distinguish. The assembly graph was constructed from the same bacterial dataset ERR1046594, but was drawn in Python with Matplotlib [44] after employing simplification methods.

average vertex degree is around two. This will result in almost equal distances between vertices which is not desirable (Figure 3.11). Consequently, we reintroduce removed transitive edges to increase the attractive forces in valid paths of the graph. The notion behind this approach is that paths existing because of chimeric sequences are quite rare and no transitive edges should support them. The same is valid for repeat induced edges, only if the majority of them have been removed as described in Section 3.2. Assembly graph from Figure 3.11 with transitive edges is shown in Figure 3.12, in which we can locate false paths clearly.

The pseudocode for assembly graph drawings can be seen in Algorithm 7. It is embarrassingly parallelizable by assigning each thread to calculate the displacement of one vertex (lines 10-18). There are a couple of modifications to the algorithm presented in [47]. We draw each

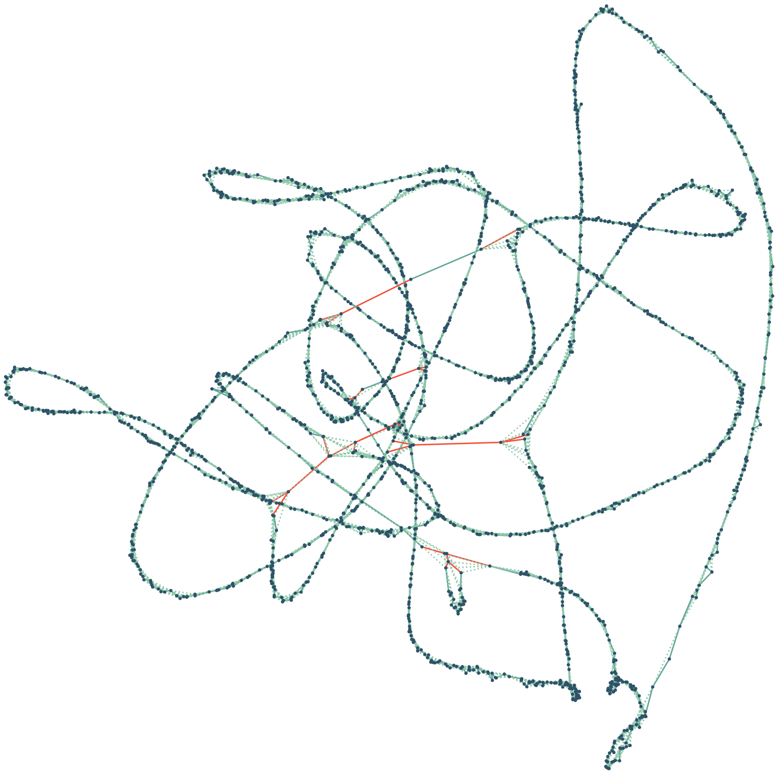


**Figure 3.12:** Condensed assembly graph with reintroduced transitive edges exhibits proper stretching of false paths drawn with the force-directed placement algorithm [47]. Transitive edges are represented with green dotted lines, while edges of interest, those tied to junction vertices, are coloured in red and are now easier to detect. The assembly graph was constructed from the bacterial dataset ERR1046594 and was drawn in Python with Matplotlib [44].

**Algorithm 7** Force-directed placement of an assembly graph

```
Input: An assembly graph $G = (V, E)$.
Output: None. (Procedure stores vertex distances to corresponding edges.)
procedure AssemblyGraphDrawing($V, E$)
    $C \leftarrow$ ConnectedComponents($V, E$)                     ▷ [48]
    for all $(V', E') \in C$ do
        $k \leftarrow \frac{1}{\sqrt{|V'|}}$
        $t \leftarrow 0.1$
        $dt \leftarrow 0.001$
        for all $v \in V'$ do
            $(v_x, v_y) \leftarrow (U(0,1), U(0,1))$               ▷ Random point on the canvas
        for $i \leftarrow 0$ to $100$ do
            for all $v \in V'$ do
                $v_{force} \leftarrow (0,0)$
                for all $w \in V' \setminus v$ do                  ▷ Repulsive forces
                    $\Delta \leftarrow (v_x - w_x,\, v_y - w_y)$
                    $v_{force} \leftarrow v_{force} + \frac{k^2}{|\Delta|} \cdot \frac{\Delta}{|\Delta|}$
                for all $v \rightarrow w \in E'$ do                ▷ Attractive forces
                    $\Delta \leftarrow (v_x - w_x,\, v_y - w_y)$
                    $v_{force} \leftarrow v_{force} - \frac{|\Delta|^2}{k} \cdot \frac{\Delta}{|\Delta|}$
                $v_{force} \leftarrow t \cdot \frac{v_{force}}{|v_{force}|}$
            for all $v \in V'$ do
                $(v_x, v_y) \leftarrow (v_x + (v_{force})_x, v_y + (v_{force})_y)$
            $t \leftarrow t - dt$
        for all $v \rightarrow w \in E'$ do
            $(v \rightarrow w)_{distance} \leftarrow |(v_x - w_x,\, v_y - w_y)|$
```

connected component separately with the aim to decrease the number of vertices in each drawing. We drop the constraints on the initial frame size and let the drawing expand as much as needed, otherwise the vertices are pushed to the frame itself and the graph loses the sought shape, as depicted in Figure 3.13. Finally, the whole algorithm is run several times to overcome bad initial layouts.

The obtained drawing is used to determine the length of each edge by calculating the Euclidean distance between connected vertex representatives in the drawing. Those distances are used to resolve junctions in the graph, that is if a vertex has multiple outgoing edges and if the length of one of them is at least twice longer than any other outgoing edge of that vertex, it is removed from the edge set $E$.

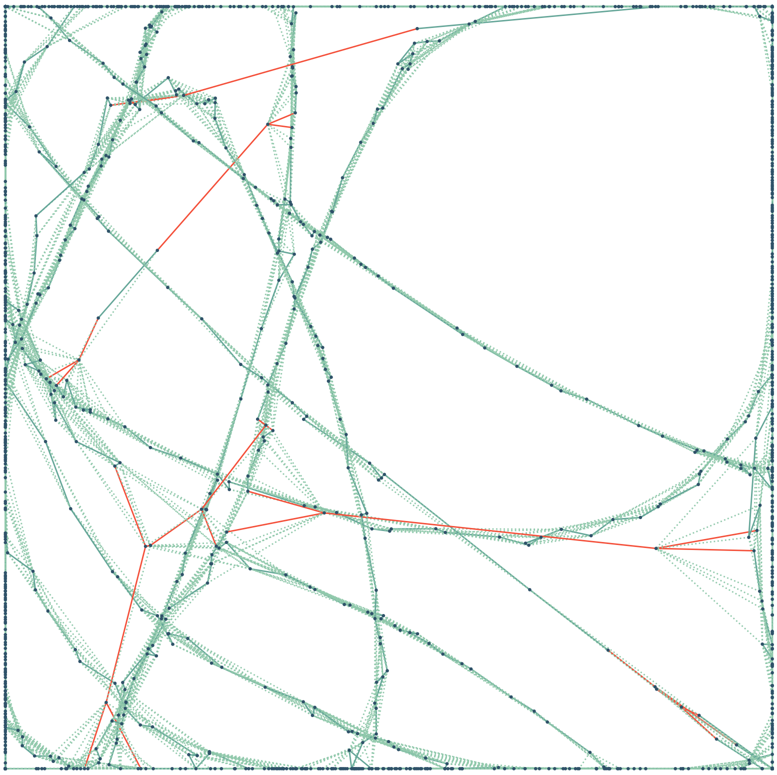


**Figure 3.13:** Condensed assembly graph drawn with the force-directed placement algorithm [47] inside a fixed canvas frame. The vertices are pushed to the frame itself and the graph loses the desired shape, although some of the false paths are still elongated. The assembly graph was constructed from the bacterial dataset ERR1046594 and was drawn in Python with Matplotlib [44].

### 3.3.2 Approximation techniques

Quadratic time complexity of the force-directed placement algorithm hinders its use on larger genomes, in which the number of vertices can range up to a few hundred thousand, thus we shifted to approximation techniques. Points of interest in the graph are junctions, vertices with outdegree greater than two. In order to decrease the execution time we shrink the unambiguous paths of the graph by replacing the vertices they consist of with a single unitig vertex. As we want to preserve the neighbourhood of all junctions to be able to determine false edges, vertices that are at most $\varepsilon$ away from any junction (shortest path between them is less than $\varepsilon$) are removed from consideration while creating unitigs. We use a modified version of Algorithm 5 which shifts the first ($v'$) and last ($v''$) vertex of a unitig inwards by $\varepsilon$ (before line 13). Unitigs consisting of less than $2\varepsilon + 1$ vertices are not collapsed. The value for $\varepsilon$ was empirically set to 42. Figure 3.14 is an example which shows that replacing groups of unessential vertices with unitigs does not change the desired drawing structure.

Despite the fact that shrinking gives a decent performance boost, some assembly graphs are just too complex. Authors of the force-directed placement algorithm proposed a grid version in which the repulsive forces are ignored outside a circle with radius $2k$ centered in each vertex [47]. If the vertices are uniformly distributed the complexity should drop to $O(|V|)$ [47]. Neglecting repulsive forces distorts the assembly graph (Figure 3.15) and the important edges are not stretched enough. For that reason, we switched to Barnes-Hut approximation which was devised for the gravitational $N$-body problem [49]. The idea is to accurately describe forces between closely arranged vertices and aproximate the forces between distant ones. Distant vertices are treated as a single point located in their center-of-mass which emits a repulsive force multiplied with their cumulative mass. The algorithm recursively splits the two-dimensional space into four equal quadrants until every vertex solely occupies a subquadrant (Figure 3.16). Data structure suitable for such division is a quadtree, a tree in which each node has up to four child nodes [50]. Quadtrees will enable a recursive method for calculation of repulsive forces starting from the whole canvas (root of the quadtree) down to each vertex (leaves of the quadtree). The method will terminate at a given node if the represented subquadrant width is less than the distance between the vertex and the center-of-mass of the subquadrant. This will reduce the running time from $O(|V|^2)$ to $O(|V|log|V|)$ and the calculated forces will differ 1% at average [49]. We modify Algorithm 7 by creating a quadtree in each iteration (before line 10). Each $v \in V$ is added to the quadtree with Algorithm 8 and afterwards the quadtree is finalized with Algorithm 9. Additionally, repulsive force calculation is replaced with Algorithm 10 (lines 12-14 of Algorithm 7).

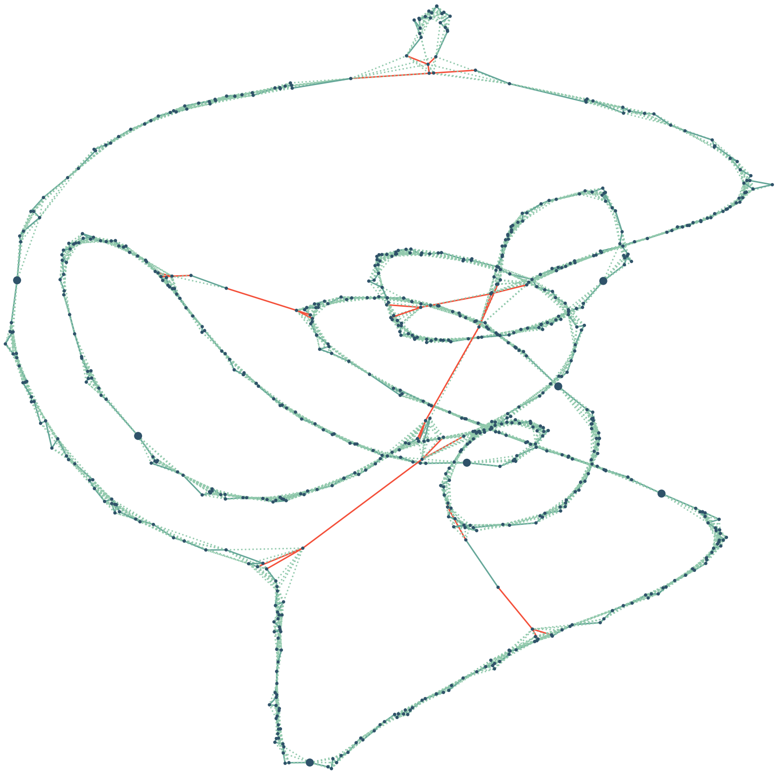


**Figure 3.14:** Condensed assembly graph drawn with the force-directed placement algorithm [47] after unitig creation. Replacing unambiguous paths of the graph with a single vertex, which is at least 42 vertices away from any junction, retains the sought drawing structure. This will boost the performance of the algorithm as the number of vertices decreased from 2430 to 1099. The assembly graph was constructed from the bacterial dataset ERR1046594 and was drawn in Python with Matplotlib [44].

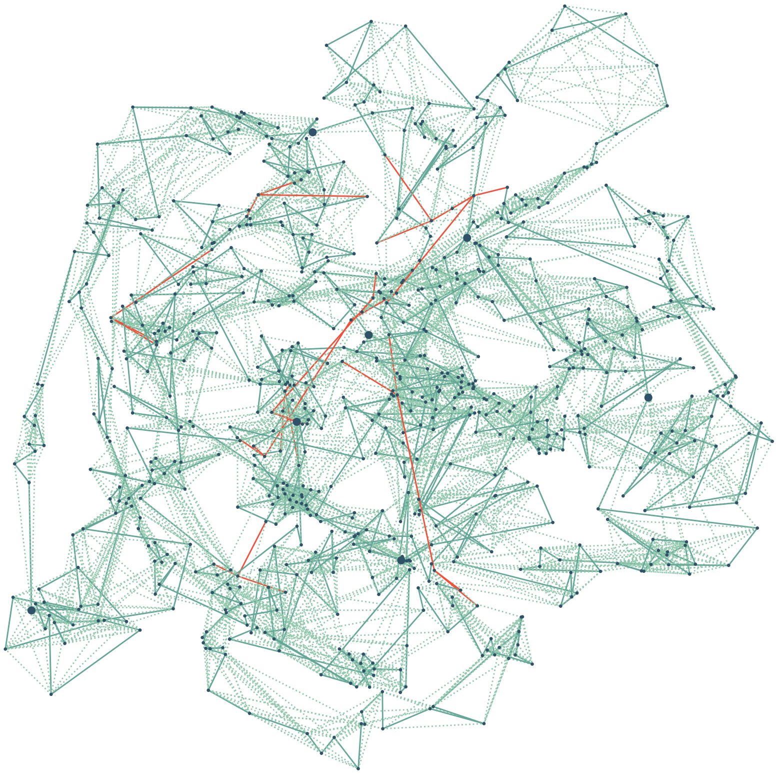


**Figure 3.15:** Condensed assembly graph drawn with the grid version of the force-directed placement algorithm [47]. Ignoring repulsive force outside a circle of radius $2k$ renders the drawing unusable for detection of false paths. The assembly graph was constructed from the bacterial dataset ERR1046594 and was drawn in Python with Matplotlib [44].

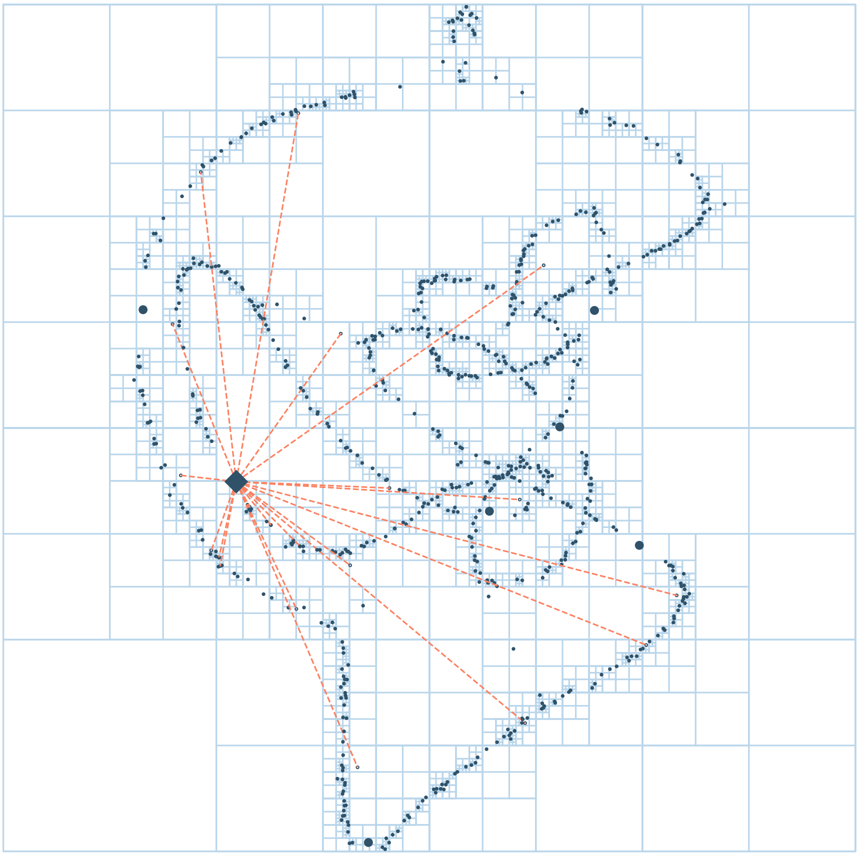


**Figure 3.16:** Condensed assembly graph drawn with Barnes-Hut approximation [49] of the the force-directed placement algorithm [47]. The canvas is recursively split into four quadrants until each vertex is a single occupant of a subquadrant. Repulsive forces are calculated depending on the distance between a given vertex and centers-of-mass of all quadrants, which enables quasilinear execution time. For example, for the vertex marked with a rhombus the number of repulsive forces calculated (denoted with red dashed lines) decreased from 1098 to only 20. The assembly graph was constructed from the bacterial dataset ERR1046594 and was drawn in Python with Matplotlib [44]. Edges of the assembly graph are not drawn for clarity.

**Algorithm 8** Adding an element to the quadtree

**Input:** Qaudtree node $q$ and vertex $v$. Quadtree nodes are tuples consisting of five values $(core, width, center, mass, subtrees)$, where $core$ is the cell center, $widht$ half the cell width, $center$ the cell center-of-mass, $mass$ the total mass of all occupants in the cell, and $subtrees$ a tuple of four children nodes.

**Output:** *True* if vertex $v$ is stored, otherwise *false*.

```
procedure QuadtreeAdd($q, p$)
    if $v_x \notin (q_{core})_x \pm q_{width}$ or $v_y \notin (q_{core})_y \pm q_{width}$ then
        return false                                        ▷ Vertex is out of quadrant bounds
    $q_{mass} \leftarrow q_{mass} + 1$
    if $q_{mass} = 1$ then
        $q_{center} \leftarrow (v_x, v_y)$
        return true
    if $q_{subtrees} = \emptyset$ then                                    ▷ Split the current cell
        $q_{subtrees} \leftarrow ($
            $(((q_{core})_x + \frac{q_{width}}{2}, (q_{core})_y + \frac{q_{width}}{2}), \frac{q_{width}}{2}, (0,0), 0, ()),$
            $(((q_{core})_x - \frac{q_{width}}{2}, (q_{core})_y + \frac{q_{width}}{2}), \frac{q_{width}}{2}, (0,0), 0, ()),$
            $(((q_{core})_x - \frac{q_{width}}{2}, (q_{core})_y - \frac{q_{width}}{2}), \frac{q_{width}}{2}, (0,0), 0, ()),$
            $(((q_{core})_x + \frac{q_{width}}{2}, (q_{core})_y - \frac{q_{width}}{2}), \frac{q_{width}}{2}, (0,0), 0, ())$
        $)$
        for all $q' \in q_{subtrees}$ do                    ▷ Move current occupant to a subquadrant
            if QuadtreeAdd($q', q_{center}$) then
                break
    for all $q' \in q_{subtrees}$ do
        if QuadtreeAdd($q', p$) then
            break
    return true
```

**Algorithm 9** Finalize quadtree by calculating centers-of-mass

**Input:** Quadtree node $q$.

**Output:** None. (Procedure calculates centers-of-mass of all nodes in the quadree.)

```
procedure QuadtreeCenter($q$)
    if $q_{subtrees} = \emptyset$ then
        return
    $q_{center} \leftarrow (0,0)$
    for all $q' \in q_{subtrees}$ do
        $q_{center} \leftarrow q_{center} + q'_{mass} \cdot q'_{center}$
    $q_{center} \leftarrow \frac{q_{center}}{q_{mass}}$
```

**Algorithm 10** Quadtree repulsive force

```
Input: Quadtree node $q$, vertex $v$ and optimal distance $k$.
Output: None. (Procedure updates the force of given vertex.)
procedure QUADTREEFORCE($q$, $v$, $k$)
    $\Delta \leftarrow (v_x - (q_{center})_x, v_y - (q_{center})_y)$
    if $|\Delta| > 2 \cdot q_{width}$ then
        $v_{force} \leftarrow v_{force} + q_{mass} \cdot \frac{k^2}{|\Delta|} \cdot \frac{\Delta}{|\Delta|}$
    else
        for all $q' \in q_{subtrees}$ do QUADTREEFORCE($q'$, $v$)
```

### 3.3.3 Discussion

Reducing the number of vertices coupled with Barnes-Hut approximation enables large graph drawings in quasilinear time. Although they look like a yarn ball to the observer (Figure 3.18), some of elongated edges can be recognized on the graph boundaries. Using graph drawings to remove false paths greatly increases the contiguity even on larger genomes, as we will see in later chapters.

We also tried applying this algorithm without any sequence preprocessing. If there are only vertices representing chimeric sequences present in the graph, the algorithm properly enlarges corresponding edges as there is abysmal chance that several of them connect identical parts of the graph. On the other hand, the algorithm struggles to handle cases in which there are a lot of repeat induced paths connecting the same graph region (Figure 3.18). Because of that, we use the graph drawings as the last simplification method of the layout phase.

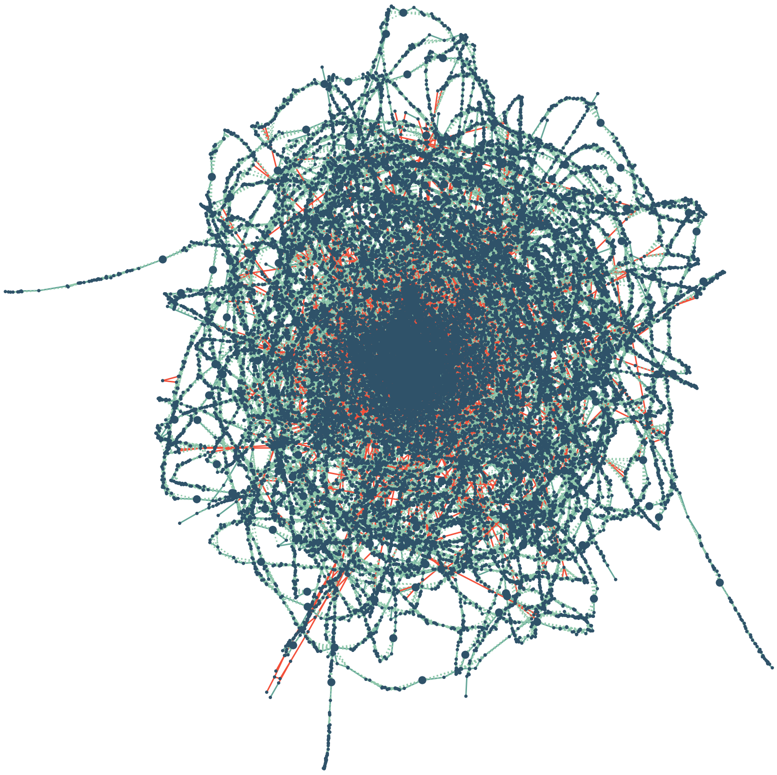


**Figure 3.17:** Condensed assembly graph of a larger genome drawn with the force-directed placement algorithm [47]. Although the false edges are hard to see, they are present and elongated enough to be removed. The assembly graph was constructed from the eukaryotic dataset SRR6702603 and was drawn in Python with Matplotlib [44].

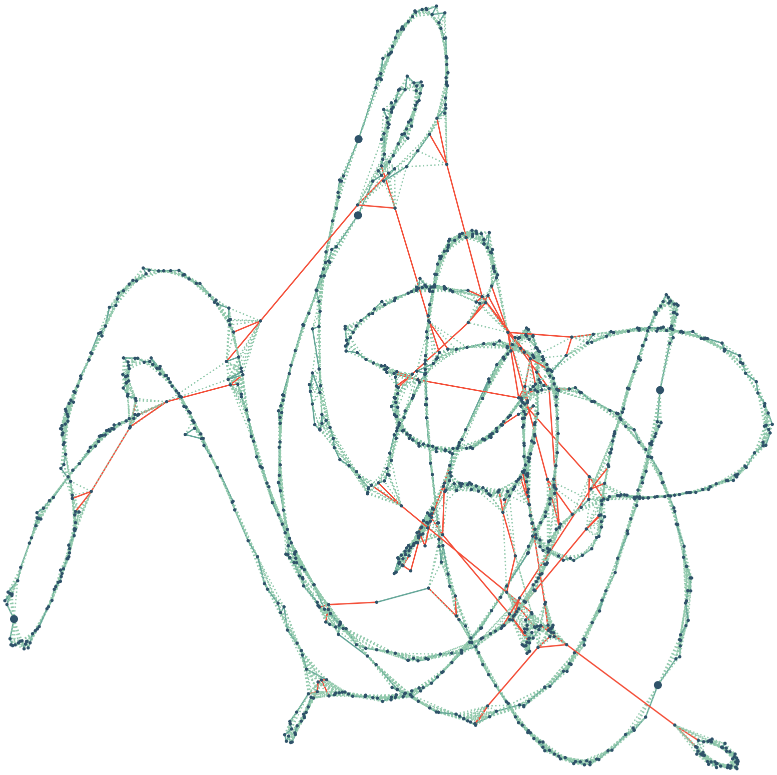


**Figure 3.18:** Condensed assembly graph with repeat induced edges drawn with the force-directed placement algorithm [47]. Increased number of such edges hinders their proper elongation in the drawing, although some of them are still resolvable. The assembly graph was constructed from the bacterial dataset ERR1046594 and was drawn in Python with Matplotlib [44].

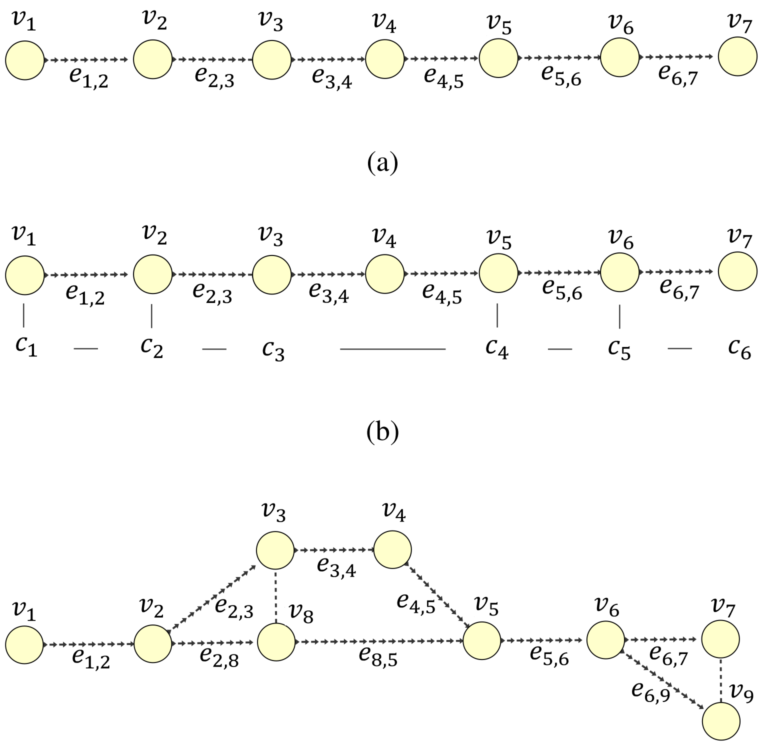
# Chapter 4

# Algorithms for consensus phase of the OLC paradigm

Unitigs extracted from a layout of third generation sequencing data are unusable for many downstream analysis due to unamended sequencing errors. To identify the right order of nucleotides amidst the noise, assemblers employ methods based on multiple sequence alignment on top of the set of unitigs. The most prosperous one is partial order alignment [20] which defines the problem through directed acyclic graphs. It extends pairwise alignment algorithms like [51] and [52] to alignment between a sequence and a graph. This prevents possible information loss present in progressive approaches that employ pairwise alignment between pairs of multiple sequence alignment representatives [20]. The consensus sequence is obtainable by finding the heaviest path in finalized partial order graphs, algorithm proposed by the same authors [21]. Because of linear time complexity of the partial order alignment with regards to the number of sequences aligned [20], we picked this algorithm as a building block to iteratively increase the accuracy of unitigs in our implementation. Reflecting on how the immense application of pairwise alignment algorithms in bioinformatics has led to many optimizations using SIMD technologies [53][54][55], it seemed natural to explore the same for partial order alignment.

## 4.1 Partial order alignment

Partial order graphs $G = (V, E)$ are directed acyclic graphs constructed from a set of sequences $S$. Vertices represent characters $c \in \Sigma$ that are connected with edges if they are consecutive in any of the sequences $s \in S$. Edge weights represent the number of sequences passing through and can be combined with base quelities obtained with sequencing [20]. Sequences are iteratively aligned and added to a growing graph [20]. Initially, an arbitrary sequence is added to an empty graph. Insertion of each succeeding sequence is guided by the partial order alignment [20]. All matching bases in the alignment are fused into existing vertices of the graph. In



(a)

(b)

(c)

**Figure 4.1:** Construction of a partial order graph from a set of two arbitrary sequences. Vertices are denoted with $v_i$, while edges with $e_{i,j}$. Dashed lines link characters that were aligned together but are mismatched. In subfigure (a) one sequence of the set is transformed into a linear graph wihtout branches. The other is aligned and afterwards included in the graph. Alignment is shown in subfigure (b) where vertical lines between $c_i$ and $v_i$ denote matches for which nothing has to be done during sequence inclusion. Subfigure (c) shows the resulting partial order graph. It has two new vertices due to mismatches $(c_3, v_3)$ and $(c_6, v_7)$, which are linked together with dashed lines. The graph has also three new edges $e_{2,8}$, $e_{6,9}$ and $e_{8,5}$. Edge $e_{8,5}$ was added due to a deletion between characters $c_3$ and $c_4$. All graphs were manually drawn and annotated in Cytoscape [39].

contrast, mismatches spawn new vertices which are linked to the vertices they are aligned to, if such do not already exist. Links between aligned vertices are not edges $e \in E$, but information stored in vertices. Deletions spawn new edges, while insertions add both vertices and edges to the graph. Figure 4.1 depicts this approach, while Algorithm 11 gives the pseudocode.

Pairwise alignment between two sequences $s$ and $t$, with lengths $n$ and $m$, is solvable with dynamic programing in $O(nm)$ time [56]. Alignments can be viewed as a set of character modifications used to obtain one sequence from the other. Modifications include matches, mismatches, insertions and deletions. Given a similarity matrix $M$ and a gap cost $g$, the algorithm computes a $(n+1) \times (m+1)$ matrix $H$ following Equation 4.1. The optimal alignment is extracted from matrix $H$ with a backtracking procedure that takes at most $O(n+m)$. Starting from the bottom right cell, the procedure determines from which of the three neighbouring cells did the current value originate, and stops when it reaches the top left cell. Modifications in regards to the stop and start cells, coupled with penalty removal in first row and column of matrix $H$ enable local [52] or semi-global version of the pairwise alignment algorithm.

$$H_{i,j} = \begin{cases} 0 & i = 0 \land j = 0 \\ j \cdot g & i = 0 \\ i \cdot g & j = 0 \\ \max \begin{cases} H_{i,j-1} + g \\ H_{i-1,j} + g \\ H_{i-1,j-1} + M_{t[i-1],s[j-1]} \end{cases} & else \end{cases} \tag{4.1}$$

Quadratic time complexity for alignment holds for linear gap modeling, which are considered inadequate for analyzing biological sequences. Ideally we would use a convex function like the logarithm, but satisfatcory results in less time can be achieved with affine gaps proposed by [57]. Matrices $F$ and $E$ model the affine function, which consists of a gap opening penalty $o$ and gap extension penalty $e$, and the time complexity increases to $O(3nm)$. Matrix recursions are updated according to Equations 4.2, 4.3 and 4.4.

$$E_{i,j} = \begin{cases} -\infty & i = 0 \lor j = 0 \\ \max \begin{cases} E_{i,j-1} + e \\ H_{i,j-1} + o \end{cases} & else \end{cases} \tag{4.2}$$

$$F_{i,j} = \begin{cases} -\infty & i = 0 \lor j = 0 \\ \max \begin{cases} F_{i-1,j} + e \\ H_{i-1,j} + o \end{cases} & else \end{cases} \tag{4.3}$$

$$H_{i,j} = \begin{cases} 0 & i = 0 \land j = 0 \\ o + (j-1) \cdot e & i = 0 \\ o + (i-1) \cdot e & j = 0 \\ \max \begin{cases} E_{i,j} \\ F_{i,j} \\ H_{i-1,j-1} + M_{t[i-1],s[j-1]} \end{cases} & else \end{cases} \tag{4.4}$$

Partial order alignment extends pairwise alignment by incorporating multiple vertex predecessors which are present in partial order graphs. As a prerequisite, the graph has to be topologically sorted using algorithm like [58]. Each vertex must be situated in the alignment matrix $H$ strictly after all of its predecessors in the graph. The running time slightly increases to $O((2\tilde{p}+1)|V|m)$, where $\tilde{p}$ is the average number of predecessors in the partial order graph [20]. Updated alignment recursions for matrices $H$ and $F$ are shown in Equations 4.5 and 4.6. Recursion for matrix $E$ remains the same as no dependencies arise in the sequence. Starting point of the backtracking procedure is changed to the highest scoring cell which has a vertex counterpart without outgoing edges. To determine the best move from a cell, the current values has to be compared against all vertex predecessors. Modifications of local and semi-global alignment algorithms are similar.

$$F_{v_{row},j} = \begin{cases} -\infty & v = null \lor j = 0 \\ \begin{cases} H_{0,j} + o & deg^{-}(v) = 0 \\ \max\limits_{w \to v \in E} \begin{cases} F_{w_{row},j} + e \\ H_{w_{row},j} + o \end{cases} & else \end{cases} & else \end{cases} \tag{4.5}$$

$$H_{v_{row},j} = \begin{cases} 0 & i = 0 \land j = 0 \\ o + (j-1) \cdot e & v = null \\ \begin{cases} o & deg^{-}(v) = 0 \\ \max\limits_{w \to v \in E} \left\{ H_{w_{row},j} + e \right. & else \end{cases} & j = 0 \\ \max \begin{cases} E_{v_{row}j} \\ F_{v_{row},j} \\ M_{v_{base},s[j-1]} + \begin{cases} H_{0,j-1} & deg^{-}(v) = 0 \\ \max\limits_{w \to v \in E} H_{w_{row},j-1} & else \end{cases} \end{cases} & else \end{cases} \tag{4.6}$$

Once all sequences are added to the partial order graph, the consensus sequence can be obtained from a topologically sorted graph. For each vertex $v \in V$ in topological order, we choose the ingoing edge $w \to v \in E$ with the largest weight and update the score of vertex $v$ accordingly [21]. Vertex $w$ is then declared as the best predecessor of $v$. Once all scores are calculated, the final consensus is generated by tracing back from highest scoring vertex $r$. In case the algorithm picks an internal vertex ($deg^{+}(r) \neq 0$), an additional method called branch completion is applied [21]. Scores of all vertices except $r$ are set to a negative value. The traversal continues from vertex $r$ with the constraint that only vertices with non negative scores can be chosen as predecessors. Pseudodoce for consensus generation is shown in Algorithm 12.

**Algorithm 11** Adding a sequence to partial order graph as defined in the alignment

**Input:** Partial order graph $G = (V, E)$, sequence $s$ and its base qualities $q$, and alignment $a$ between the graph and the sequence. Alignment is a list of pairs $(i, j)$ describing the relationship between vertices $v \in V$ and characters $c \in s$.

**Output:** None. (Procedure updates the given graph.)

```
procedure PartialOrderGraphAddSequence($V, E, s, q, a$)
    if $|q| = 0$ then                                ▷ Use multiplicity if qualities are absent
        $q \leftarrow$ new Array($1, |s|$)
    if $|a| = 0$ then                                ▷ Graph is empty or sequence is unalignable
        $v \leftarrow$ null
        for $j \leftarrow 0$ to $|s|$ do
            $w \leftarrow$ new Vertex($s[j]$)
            $V \leftarrow V \cup \{w\}$
            if $v \neq$ null then
                $E \leftarrow E \cup \{v \rightarrow w\}$
                $(v \rightarrow w)_{weight} \leftarrow q[j-1] + q[j]$
            $v \leftarrow w$
        return
    $v \leftarrow$ null
    for all $(i, j) \in A$ do
        $w \leftarrow$ null
        if $i = -1$ then                             ▷ Gap in the partial order graph
            $w \leftarrow$ new Vertex($s[j]$)
            $V \leftarrow V \cup \{w\}$
        else if $j = -1$ then                        ▷ Gap in the sequence
            continue
        else                                         ▷ Vertex $v_{row} = i$ and character $c_j$ are aligned
            $w' \leftarrow w'' \in V, w''_{row} = i$
            if $w'_{base} = s[j]$ then
                $w \leftarrow w'$
            else
                for all $w'' \in w'_{aligned}$ do     ▷ Check aligned vertices for a match
                    if $w''_{base} = s[j]$ then
                        $w \leftarrow w''$
            if $w =$ null then
                $w \leftarrow$ new Vertex($s[j]$)
                $w_{aligned} \leftarrow \{w'\} \cup w'_{aligned}$
                for all $w' \in w_{aligned}$ do       ▷ Record new aligned links
                    $w'_{aligned} \leftarrow w'_{aligned} \cup \{w\}$
                $V \leftarrow V \cup \{w\}$
        if $v \neq$ null then
            if $v \rightarrow w \in E$ then           ▷ Update weight if edge is present
                $(v \rightarrow w)_{weight} \leftarrow (v \rightarrow w)_{weight} + q[j-1] + q[j]$
            else
                $E \leftarrow E \cup \{v \rightarrow w\}$
                $(v \rightarrow w)_{weight} \leftarrow q[j-1] + q[j]$
        $v \leftarrow w$
```

**Algorithm 12** Consensus calling from partial order graphs

```
    Input: Partial order graph $G = (V, E)$.
    Output: Consensus of the multiple sequence alignment encoded in the given graph.
 1: procedure PartialOrderGraphConsensus($V, E$)
 2:     $V \leftarrow$ TopologicalSort($V, E$)                                  ▷ [58]
 3:     for all $v \in V$ do
 4:         $v_{score} \leftarrow 0$
 5:         $v_{predecessor} \leftarrow$ null
 6:     $r \leftarrow$ null
 7:     for all $v \in V, deg^{-}(v) > 1$ do                ▷ Heaviest bundle algorithm [21]
 8:         for all $w \rightarrow v \in E$ do
 9:             if $v_{predecessor} =$ null or
                    $(v_{predecessor} \rightarrow v)_{weight} < (w \rightarrow v)_{weight}$ or
                    $(v_{predecessor} \rightarrow v)_{weight} = (w \rightarrow v)_{weight}$ and $(v_{predecessor})_{score} < w_{score}$ then
10:                 $v_{predecessor} \leftarrow w$
11:         $v_{score} \leftarrow (v_{predecessor})_{score} + (v_{predecessor} \rightarrow v)_{weight}$
12:         if $r =$ null or $r_{score} < v_{score}$ then
13:             $r \leftarrow v$
14:     if $deg^{+}(r) = 0$ then
15:         goto 25
16:     for all $v \in V \setminus \{r\}$ do               ▷ Branch completion algorithm [21]
17:         $v_{score} = -1$
18:     for all $v \in V, deg^{-}(v) > 1, v_{row} > r_{row}$ do
19:         $v_{predecessor} \leftarrow$ null
20:         for all $w \rightarrow v \in E$ do
21:             if $w_{score} = -1$ then
22:                 continue
23:             if $v_{predecessor} =$ null or
                    $(v_{predecessor} \rightarrow v)_{weight} < (w \rightarrow v)_{weight}$ or
                    $(v_{predecessor} \rightarrow v)_{weight} = (w \rightarrow v)_{weight}$ and $(v_{predecessor})_{score} < w_{score}$ then
24:                 $v_{predecessor} \leftarrow w$
25:     $c \leftarrow$ new String
26:     while $r \neq$ null do
27:         APPEND($c, r_{base}$)
28:         $r \leftarrow r_{predecessor}$
29:     return REVERSED($c$)
```

### 4.1.1 Vectorization

Quadratic time complexity of partial order alignment along with large sequencing yields led us to investigate possible optimization techniques. Additional dependencies in the alignment made the intra-set parallelization of Smith-Waterman alignment by [54] seem like the most transparent approach to start with. The authors place SIMD vectors parallel to the query sequence which is situated on the $y$-axis. This enables parallel computation of the matrices horizontally, while vertical dependencies in columns are handled with shift operations. They also introduced sequence profiles, $|\Sigma| \times m$ matrices containing precalculated similarity scores [54]. Our implementation of partial order alignment places the SIMD vectors parallel to the sequence as well, but the sequence is situated on the $x$-axis for clarity. Vertical and diagonal dependencies are resolved using SIMD operations, while the horizontal dependency is processed linearly. Vectorization of the algorithm decreases the time complexity from $O((2\tilde{p}+1)|V|m)$ to $O((\frac{2\tilde{p}}{k}+1)|V|m)$, where $k$ is the number of variables that fit in a SIMD vector. We later modified the horizontal computation with prefix-max algorithm, further decreasing the time complexity to $O((\frac{2\tilde{p}}{k}+\frac{\log k}{k})|V|m)$. Concerning memory, we store all three matrices $E$, $F$ and $H$ entirely into memory, yielding $O(3|V|m)$ complexity. They are needed to access predecessor rows during alignment and are used in the backtracking procedure. As elements of SIMD vectors can not be accessed directly, we load a small portion of them to unaligned arrays during backtrack. The total amount is defined with predecessor quantity of cells that are contained in the found alignment.

We vectorized global, local and semi-global partial order alignment, all with either linear, affine or convex (piecewise affine described in [59]) gap models. Supported are Intel SSE4.1 and AVX2 instruction sets. The former embeds 128-bit registers while the latter 256-bit. With sequencing data we deal either with short (16 bits) or long (32 bits) integer precision, depending on sequence length and alignment scoring parameters. The number of variables per SIMD vector $k$ is therefore either 4 or 8 for SSE4.1, and 8 or 16 for AVX2. Pseudocode for vectorized global partial order alignment with affine gaps and eight 16-bit variables per SIMD vector is shown in Algorithm 13. Backtrack procedure is unchanged and not in the focus of this optimization, and is therefore not included in the pseudocode. Other alignment modes, gap models, variable precision and instruction sets can be easily inferred from this pseudocode. Unfortunately, this does not hold for shift operations in AVX2 instruction set, where registers contain two 128-bit lanes. Shifting the whole 256-bit register does not transfer elements between lanes, but can be simulated with element shuffling. This increases the latency compared to normal shift operations and thus the AVX2 alignment version is only marginally faster than the SSE4.1 version.

**Algorithm 13** Partial order alignment using SIMD instructions

```
**Input:** Partial order graph $G = (V, E)$, sequence $s$, and alignment parameters $m$ for match, $n$ for mismatch, $o$ for gap opening and $e$ for gap extension.
**Output:** List of pairs $(i, j)$ defining the alignment between the given graph and sequence.
procedure PARTIALORDERALIGNMENT($V, E, s, m, n, o, e$)
    $V \leftarrow$ TOPOLOGICALSORT($V, E$)                                   ▷ [58]
    $b \leftarrow -\infty + 2^{10}$
    $o \leftarrow o - e$
    $Q \leftarrow [o,o,o,o,o,o,o,o]$
    $R \leftarrow [[e,e,e,e,e,e,e,e],[2e,2e,2e,2e,2e,2e,2e,2e],[4e,4e,4e,4e,4e,4e,4e,4e]]$
    $P \leftarrow$ SEQUENCEPROFILE($V_\Sigma, s, m, n$)                      ▷ [54]
    $H \leftarrow$ new Array($[b,b,b,b,b,b,b,b], (|V|+1) \cdot |s|/8$)
    $c \leftarrow$ new Array($b, |V|+1$)
    $F \leftarrow$ new Array($[b,b,b,b,b,b,b,b], (|V|+1) \cdot |s|/8$)
    $E \leftarrow$ new Array($[b,b,b,b,b,b,b,b], (|V|+1) \cdot |s|/8$)
    $M \leftarrow [[b,0,0,0,0,0,0,0],[b,b,0,0,0,0,0,0],[b,b,b,b,0,0,0,0]]$
    $l \leftarrow b$
    $i \leftarrow -1$
    $c[0] \leftarrow 0$                                        ▷ Initialize first column
    for all $v \in V$ do
        if $deg^-(v) = 0$ then
            $c[v_{row}] \leftarrow o$
        for all $w \rightarrow v \in E$ do
            $c[v_{row}] \leftarrow \max(c[v_{row}], c[w_{row}] + e)$
    $T \leftarrow [0,0,0,0,0,0,0,0]$                              ▷ Initialize first row
    for $j \leftarrow 0$ to $|s|/8$ do
        $T \leftarrow ((H[0][j]$ LSHIFT $1)$ OR $(T$ RSHIFT $7)) + Q + R[0]$
        $T \leftarrow$ MAX($T, M[0]$ OR $((T + R[0])$ LSHIFT $1)$))                ▷ Prefix-max
        $T \leftarrow$ MAX($T, M[1]$ OR $((T + R[1])$ LSHIFT $2)$))
        $T \leftarrow$ MAX($T, M[2]$ OR $((T + R[2])$ LSHIFT $4)$))
        $H[0][j] \leftarrow T$
        $T \leftarrow T - Q$
    $z \leftarrow$ new Vertex                              ▷ Temporary vertex for first row
    for all $v \in V$ do                                        ▷ Calculate alignment
        if $deg^-(v) = 0$ then
            $E \leftarrow E \cup \{z \rightarrow v\}$                            ▷ Temporary edge
        for all $w \rightarrow v \in E$ do                    ▷ Vertical and diagonal update
            $T \leftarrow [c[w_{row}],0,0,0,0,0,0,0]$
            for $j \leftarrow 0$ to $|s|/8$ do
                $T_2 \leftarrow (H[w_{row}][j]$ LSHIFT $1)$ OR $T$
                $F[v_{row}][j] \leftarrow$ MAX($F[v_{row}][j], H[w_{row}][j] + Q + R[0], F[w_{row}][j] + R[0]$)
                $H[v_{row}][j] \leftarrow$ MAX($H[v_{row}][j], F[v_{row}][j], T_2 + P[v_{base}][j]$)
                $T \leftarrow (H[w_{row}][j]$ RSHIFT $7)$
        if $z \rightarrow w \in E$ then
            $E \leftarrow E \setminus \{z \rightarrow v\}$
        $T \leftarrow [0,0,0,0,0,0,0,c[v_{row}]]$
        for $j \leftarrow 0$ to $|s|/8$ do                    ▷ Horizontal update with prefix-max
            $E[v_{row}][j] \leftarrow ((H[v_{row}][j]$ LSHIFT 1) OR ($T$ RSHIFT 7)) $+ Q + R[0]$
            $E[v_{row}][j] \leftarrow$ MAX($E[v_{row}][j], M[0]$, OR (($E[v_{row}][j] + R[0]$) LSHIFT 1))
            $E[v_{row}][j] \leftarrow$ MAX($E[v_{row}][j], M[1]$, OR (($E[v_{row}][j] + R[1]$) LSHIFT 2))
            $E[v_{row}][j] \leftarrow$ MAX($E[v_{row}][j], M[2]$, OR (($E[v_{row}][j] + R[2]$) LSHIFT 4))
            $H[v_{row}][j] \leftarrow$ MAX($H[v_{row}][j], E[v_{row}][j]$)
            $T \leftarrow$ MAX($H[v_{row}][j], E[v_{row}][j] - Q$)
        if $deg^{+}(v) = 0$ then                    ▷ Find best cell for backtrack
            $h \leftarrow$ LOAD($H[v_{row}][|s|/8 - 1]$)
            if $l < h[|s| \bmod 8]$ then
                $l \leftarrow h[|s| \bmod 8]$
                $i \leftarrow v_{row}$
    $a \leftarrow$ TRACEBACK($H, c, F, E, P, i, |s| - 1$)
    return $a$
```

### 4.1.2 Discussion

Calculating multiple sequence alignment from a set of sequences $S$ and set of unitigs $U$ is commenced by constructing a partial order graph from a single unitig $u \in U$, which serves as a backbone. All sequences $s \in S$ which overlap that unitig are iteratively added to the graph with partial order alignment. The same is done for other unitigs of the assembly. Given that sequences have average length around ten thousand nucleotides and that unitigs can have lengths up to a few million bases, reaching intractable cases is unavoidable because of quadratic time and memory complexities. To cope with this we tryied using subgraph alignments. Having approximate overlaps between a sequence and the bacbkone, we would extract a subgraph defined by the overlap begin and end positions and align the sequence to it. This approach saves time if the sequence is much smaller than the backbone, but some sequences are still too large for it to manage. Therefore, we chose to slice sequences and unitigs into smaller chunks and combine the multiple sequence alignments together, as described in Section 4.2. Another option worth exploring might be banded alginment [60], which is left for future analysis.

## 4.2 Unitig polishing

Rapid and accurate consensus generation from multiple alignment of long sequences was achieved by dividing the problem into consecutive nonoverlapping windows of predefined length. This was inspired by other polishing tools such as Nanopolish [23] and Quiver [18], which depend on raw signal data or error profiles of specific sequencing technologies. Our implementation on the other hand is sequencing platform independent, and only uses base qualities if present. Using fixed length windows allows linear scaling to larger genomes, if the coverage is constant.

Given the trade-off between accuracy and execution time, we empirically chose 500 bases long windows. This way the alignment score will not overflow short integer precision and we can fit at least eight variables into SIMD vectors (Section 4.1.1). Slicing both sequences and unitigs is guided with approximate overlaps generated with any state-of-the-art overlapper [59][25][31]. Exact sequence positions to ends of each unitig window are extracted from alignments found by Edlib, a fast edit-distance based aligned which uses bit-vectors [61]. For each window, a partial order graph is constructed from the corresponding unitig substring, giving it zero weights to fend off reference bias of pairwise alignments. Substrings of sequences that overlap the given window are iteratively added to the graph with global partial order alignment using linear gap model. Consensus of each window is collected and concatenated into a polished unitig. Both pairwise and partial order alignment computations are embarrassingly parallelizable. We assign each overlap into a separate thread task, and the same is done for each window.

Suppressing high error rates of third generation sequencing is done with few filtering methods applied before construction of partial order graphs. Approximate overlaps in which the overlap length ratio between the sequence and unitig is lower than $\varepsilon$ are discarded. In addition, only the longest overlap is retained for each sequence. Finally, if base quality values are available, sequence substrings that have average quality less than $q$ are not used in multiple sequence alignment. Pseudocode for our polishing implementation is available in Algorithm 14. Higher sequence identities are achievable with several iterations of the proposed method.

### 4.2.1 Discussion

The use of nonoverlapping windows can affect the consensus quality at window ends due to low coverage regions manifested as branches in partial order graphs. If they are located at either graph end whilst being adequately long, they will be picked by the heaviest path procedure (Algorithm 12). We tried to extend the nonoverlapping windows by 10% in total and overlap their consensus sequences, but the accuracy boost was not justified with the increase in execution time. Knowing the number of sequences covering each base in the consensus sequence, we instead apply a heuristic which trims each end of the consensus until the base coverage is large enough. The coverage threshold was declared as half the number of sequences used for polishing the corresponding window. This methods works great when the window has uniform coverage, but fails dealing with datasets in which sequences are smaller than the window size. Therefore, the implementation automatically skips trimming if sequences are obtained with second generation of sequencing. Additionally, we apply subgraph alignment as described in Section 4.1.2. Sequence division into windows will leave some substrings that are much smaller than the window, which will to lead to misplaced alignments and might influence accuracy.

**Algorithm 14** Sequence polishing with partial order alignment

**Input:** Set of target sequences $T$, set of sequences used for polishing $S$ and their qualities $Q$, and set of approximate overlaps $O$ between sets $T$ and $S$. Overlaps $o \in O$ are tuples of seven values $(t\_id, t\_begin, t\_end, s\_id, s\_begin, s\_end, strand, alignment)$, where $t\_id$ and $s\_id$ are identifiers of overlapping sequences, $(t\_begin, t\_end)$ and $(s\_begin, s\_end)$ overlap positions on forward strands of $t$ and $s$, respectively, $strand$ denotes whether either of the sequences is reversed complemented in the overlap, and optionally the $alignment$).

**Output:** Set of polished target sequences.

```
procedure CONSENSUS($T, S, Q, O$)
    $O \leftarrow$ SORT($O$)                                   ▷ Sort by $o_{s\_id}$ and $(o_{s\_end} - o_{s\_begin})$
    $f \leftarrow$ new Array(0, $|S|$)
    for all $o \in O$ do                                       ▷ Filter overlaps
        if $\frac{\min(o_{t\_end}-o_{t\_begin}, o_{s\_end}-o_{s\_begin})}{\max(o_{t\_end}-o_{t\_begin}, o_{s\_end}-o_{s\_begin})} < 0.7$ or $f[o_{s\_id}] = 1$ then
            $O \leftarrow O \setminus \{o\}$
        else
            $f[o_{s\_id}] \leftarrow 1$
            if $|o_{alignment}| = 0$ then
                $t \leftarrow (t' \in T, t'_{id} = o_{t\_id})[o_{t\_begin}, o_{t\_end}]$
                $s \leftarrow (s' \in S, s'_{id} = o_{s\_id})[o_{s\_begin}, o_{s\_end}]$
                $o_{alignment} \leftarrow$ EDLIB($t, \overline{s}$ if $o_{strand}$ else $s$)                ▷ [61]
    $P \leftarrow$ new Set
    for all $t \in T$ do
        $p \leftarrow$ new String
        for $w \leftarrow 0$ to $\lceil |t|/500 \rceil$ do
            $(V, E) \leftarrow$ new PartialOrderGraph
            PARTIALORDERGRAPHADDSEQUENCE($V, E$,                   ▷ Algorithm 11
                $t[w \cdot 500, (w+i) \cdot 500]$, new Array$(0, 500)$, [ ])
            for all $o \in O, o_{t\_id} = t_{id}$ do
                $(b, e) \leftarrow$FINDBREAKPOINTS($o_{alignment}, w$)
                if $b = -1$ and $e = -1$ then                      ▷ $s$ does not overlap $w$
                    continue
                $q \leftarrow (q' \in Q, q'_{id} = o_{s\_id})[b, e]$
                if $|q| \neq 0$ and avg$(q) < 10.0$ then           ▷ Substring quality filter
                    continue
                $s \leftarrow (s' \in S, s'_{id} = o_{s\_id})[b, e]$
                if $o_{strand}$ then
                    $s \leftarrow \overline{s}$
                    $q \leftarrow$ REVERSED($q$)
                $a \leftarrow$ PARTIALORDERALIGNMENT($V, E, s, 5, -4, -8$)     ▷ Algorithm 13
                PARTIALORDERGRAPHADDSEQUENCE($V, E, s, q, a$)           ▷ Algorithm 11
            $c \leftarrow$ PARTIALORDERGRAPHCONSENSUS($V, E$)            ▷ Algorithm 12
            APPEND($p$, TRIM(c))
        $P \leftarrow P \cup \{p\}$
    return $P$
```

# Chapter 5

# Integration and evaluation

Algorithms described in chapters 3 and 4, together with Minimap [30] used for the overlap phase of the OLC paradigm, are integrated into a de novo genome assembler called *Raven*. To summarize, Raven finds approximate pairwise overlaps to build an assembly graph that is simplified stepwise. The same overlaps are used for creation of pile-o-grams which aid in sequence preprocessing during graph construction. To decrease fragmentation, a novel approach based on graph drawings is utilized to untangle the assembly graph. Eventually, the assembly accuracy is iteratively improved by computing multiple sequence alignment on small substrings with the help of partial order graphs. Raven does not employ correction of third generation sequencing data prior assembly, but still yields accurate genome reconstructions while not being resource-intensive. It is a culmination of several independent implementations which are briefly described and evaluated in the following sections.

## 5.1 Implementation

Every tool that arose from this doctoral thesis was implemented in C++, and is publicly available under the MIT license at several GitHub web addresses. Chronologically, we first worked on algorithms for the consensus phase. Multiple sequence alignment utilizing partial order graphs [20][21] is incorporated into a library called *Spoa*, which can also be used as a standalone tool. Spoa implements Algorithms 11, 12 and 13 from Chapter 4, including three alignment types and three gap models, from which any combination can be run with or without vectorization. Spoa is used as a library in our polishing tool called *Racon* which equals Algorithm 14. Besides computing consensus sequences with Spoa, Racon performs overlap filtering, sequence shearing and consensus trimming. Racon is a versatile polishing tool mostly used to improve the accuracy of de novo assembled genomes, operating with either second or third generation of sequencing data. Spoa and Racon are accessible at `https://github.com/rvaser/spoa` and `https://github.com/lbcb-sci/racon`, respectively.

Algorithms 1-10 are the core of our layout module *Rala*. Rala modifies and extends the assembly graph of the Miniasm assembler [30], with preprocessing and postprocessing methods for graph untangling. The former is achieved with ideas from the HINGE assembler [33], while the latter relies on force-directed placement [47] of the assembly graph. Rala was designed to be run hierarchically on top of Minimap. A preconstruction step resolves chimeric and removes contained sequences. Remaining sequences are given to Minimap again to reconstruct pile-o-grams which are needed to finalize the construction of the assembly graph, which is later simplified. The source code can be found at `https://github.com/rvaser/rala`.

Initial version of a complete OLC based de novo genome assembler was named *Ra*, and is a Python script combining Minimap, Rala and Racon. Ra runs Minimap and Rala twice to find unitigs which are later polished with two iterations of Racon. At that time, the postprocessing method for graph simplification was based on overlap lengths, although a more strict version of the one used in Miniasm. Ra is available at `https://github.com/lbcb-sci/ra`.

Finalizing our assembler we wanted to omit hefty disc requirements present in large genomes. With that purpose in mind we transformed Minimap to a C++ library with some slight modifications. We called it *Ram* and it is available at `https://github.com/lbcb-sci/ram`. Source code of Rala, a library version of Racon and Ram were integrated in a single C++ project which we renamed to Raven, as Ra was a tad bit short. Raven loads the whole sequence set into memory only once at the beginning of execution. Sequences are transformed into minimizer hashes in blocks of one gigabyte. Found pairwise overlaps are immediately transformed into downsampled pile-o-grams, each with size that approximately equals to 25% of the corresponding sequence size. In addition, the 16 longest overlaps are stored per sequence which increase the memory usage slightly as each overlap consists of only 512 bytes. Longest overlaps are used for containment removal following after sequence trimming and chimeric resolution. The remaining sequences are again hashed by blocks. They are overlapped to each other and to the rest of the dataset, but with a reduced $k$-mer filter. Recreated pile-o-grams help remove repeat induced overlaps prior the construction of the assembly graph. After employing several simplification methods, we utilize force-directed layout to try and untangle the leftover junctions in the graph. All sequences are overlapped with extracted unitigs and polished with Racon. The same is done twice in a row and the final result is a set of very accurate unitigs. All mentioned overlap steps are done completely in memory without anything stored to the hard drive. They are embarrassingly parallelized by assigning one sequence to each thread. Pseudocode of Raven is presented in Algorithm 15, while the assembler itself is accessible at `https://github.com/lbcb-sci/raven`.

**Algorithm 15** Raven algorithm for de novo genome assembly from long uncorrected sequences

**Input:** Set of raw sequences $S$ obtained with third generation of sequencing, with optional base qualities $Q$.

**Output:** Set of high accuracy unitigs.

```
procedure RAVEN($S, Q$)
    $P \leftarrow \emptyset$
    for all $s \in S$ do
        $P \leftarrow P \cup \{$PILEOGRAMCREATE($s$)$\}$        ▷ Initialize downsampled pile-o-grams
    $O \leftarrow \emptyset$
    $B \leftarrow \emptyset$
    for all $s \in S$ do
        $B \leftarrow B \cup \{s\}$
        if SIZE($B$) $< 2^{30}$ then
            continue
        $H \leftarrow$ MINIMIZERHASHCREATE($B$)        ▷ [30]
        $O' \leftarrow \emptyset$
        for all $s' \in S$ do
            $O' \leftarrow O' \cup$ MINIMIZERHASHOVERLAP($s', 10^{-3}$)        ▷ [30]
        for all $p \in P$ do        ▷ Fill pile-o-grams
            $p \leftarrow$ PILEOGRAMADDOVERLAPS($p, O'$)
        $O \leftarrow$ RETAINLONGESTOVERLAPS($O \cup O', 16$)
        $B \leftarrow \emptyset$
    for all $p \in P$ do
        PILEOGRAMTRIM($p, 4$)        ▷ Described in Section 3.2.1
        PILEOGRAMSLOPEDETECTION($p, 1.84$)        ▷ Algorithm 6
    $(S', O) \leftarrow$ REMOVECONTAINEDSEQUENCES($S$, $O$, $P$)
    $C \leftarrow$ CONNECTEDCOMPONENTS($O, P$)        ▷ [48]
    for all $c \in C$ do
        $m \leftarrow \mathrm{med}(p_{median}, \forall p \in c)$
        for all $p \in c$ do
            PILEOGRAMRESOLVECHIMERICREGIONS($p, m$)        ▷ Described in Section 3.2.1
    $P \leftarrow \emptyset$
    for all $s \in S'$ do
        $B \leftarrow B \cup \{s\}$
        if SIZE($B$) $< 2^{30}$ then
            continue
        $H \leftarrow$ MINIMIZERHASHCREATE($B$)        ▷ [30]
        for all $s' \in S'$ do
            $O \leftarrow O \cup$ MINIMIZERHASHOVERLAP($s', 10^{-3}$)        ▷ [30]
        $O' \leftarrow \emptyset$
        for all $s' \in S \setminus S'$ do
            $O' \leftarrow O' \cup$ MINIMIZERHASHOVERLAP($s', 10^{-5}$)        ▷ [30]
        for all $p \in P$ do        ▷ Update pile-o-grams
            PILEOGRAMADDOVERLAPS($p, O'$)
        $B \leftarrow \emptyset$
```

```
$C \leftarrow$ ConnectedComponents($O, P$)
for all $c \in C$ do
    $m \leftarrow \mathrm{med}(p_{median}, \forall p \in c)$
    for all $p \in c$ do
        PileogramSlopeDetection($p$, 1.42)                        ▷ Algorithm 6
        PileogramRemoveFalseOverlaps($p, O$)          ▷ Described in Section 3.2.1
$(V, E) \leftarrow$ AssemblyGraphCreate($S', O$)                              ▷ [30]
AssemblyGraphTransitiveReduction($V, E$)                           ▷ Algorithm 1
while $True$ do
    $(V', E') \leftarrow (V, E)$
    AssemblyGraphPruning($V, E$)                                   ▷ Algorithm 2
    AssemblyGraphBubblePopping($V, E$)                             ▷ Algorithm 3
    if $V' = V$ and $E' = E$ then
        break
AssemblyGraphUnitigs($V, E$, 42)                          ▷ Modified Algorithm 5
for $i \leftarrow 0$ to 16 do
    AssemblyGraphDrawing($V, E$)                                   ▷ Algorithm 7
    AssemblyGraphRemoveLongEdges($V, E$)           ▷ Described in Section 3.3.1
AssemblyGraphUnitigs($V, E$)                                       ▷ Algorithm 5
$U \leftarrow$ AssemblyGraphExtractUnitigs($V, E$)
for $i \leftarrow 0$ to 2 do
    $O \leftarrow \emptyset$
    for all $u \in U$ do
        $B \leftarrow B \cup \{u\}$
        if size($B$) $< 2^{30}$ then
            continue
        $H \leftarrow$ MinimizerHashCreate($B$)                              ▷ [30]
        for all $s \in S$ do
            $O \leftarrow O \cup$ MinimizerHashOverlap($s, 10^{-3}$)         ▷ [30]
        $B \leftarrow \emptyset$
    $U \leftarrow$ Consensus($U, S, Q, O$)                                ▷ Algorithm 14
return $U$
```

## 5.2 Datasets

To evaluate the described tools we used various publicly available datasets. The complete list is provided in Table 5.1, with dataset descriptions and references. This includes the sequenced species, sequencing technology and depth, and the expected genome size. All reference genomes were obtained from the National Center for Biotechnology Information (NCBI) genome browser. Datasets prefixed with SR, ER or DR are available for download from either the Sequence Read Archive (SRA) or the European Nucleotide Archive (ENA). The rest is hosted on different web services which can be found in corresponding references. We tried to capture as many different datasets as possible, ranging from small prokaryotes to large eukaryotes while balancing the ratio between Oxford Nanopore Technologies and Pacific Biosciences.

**Table 5.1:** Third generation sequencing data used in evaluation of tools for de novo genome assembly.

| Species | Size ($\times 10^6$) | Dataset | Coverage | Technology | Reference |
|---|---|---|---|---|---|
| *Fusobacterium gonidiaformans* | 1.7 | SRR6780920 | 61 | ONT R9.4 | [62] |
| *Fusobacterium varium* | 3.3 | SRR6780912 | 27 | ONT R9.4 | [62] |
| *Bordetella pertussis* | 4.1 | ERR1475873 | 147 | PB | [63] |
| *Escherichia coli* | 4.6 | ERR1147227 | 54 | ONT R7.3 | [64] |
| | | PBEC | 161 | PB P6-C4 | [65] |
| *Salmonella enterica* | 5.1 | ERR987680 | 176 | PB | [63] |
| *Enterobacter aerogenes* | 5.3 | ERS715397 | 132 | PB | [63] |
| *Klebsiella pneumoniae* | 5.7 | SRR5665597 | 114 | ONT R9.4 | [66] |
| | | SRR5665591 | 11 | ONT R9.4 | [66] |
| | | ERR1046594 | 104 | PB | [63] |
| | | ERR1140973 | 42 | PB | [63] |
| *Saccharomyces cerevisiae* | 12.1 | ERX1910723 | 59 | ONT R9 | [67] |
| | | SRX533604 | 127 | PB P4-C2 | [65] |
| *Plasmodium falciparum* | 23.3 | SRA360189 | 320 | PB P6-C4 | [68] |
| *Caenorhabditis elegans* | 100.3 | PBCE | 81 | PB P6-C4 | [65] |
| *Arabidopsis thaliana* | 119.7 | PBAT | 90 | PB P5-C3 | [65] |
| *Drosophila melanogaster* | 143.7 | SRR6702603 | 32 | ONT R9.5 | [69] |
| | | SRR5439404 | 127 | PB P6-C4 | [70] |
| | | SRX499318 | 109 | PB P5-C3 | [65] |
| *Ipomoea nil* | 735.2 | DRA002710 | 54 | PB P5-C3 | [71] |
| *Solanum pennellii* | 938.0 | ONTSP | 119 | ONT R9.4 | [72] |
| *Homo sapiens* | 3234.8 | ONTHS | 35 | ONT R9.4 | [9] |

## 5.3 Evaluation methods

Consensus quality was evaluted with Dnadiff, a tool from the Mummer package [73], focusing on parameters like the average identity (accuracy) and the number of aligned bases of both the assembly and the reference genome. Accuracy metrics can also be found in Quast-LG [74], but Quast-LG was primarily used in misassembly and fragmentation analyses. The most frequently used meassure for comparison of different assemblies is the NG50 value, which denotes the length of the smallest conting that with all longer contigs covers half of the sequenced genome. However, we also considered the reconstructed genome fraction, number of contigs, and number of different missasemblies, namely relocations, translocations and inversions. Additionally, we assessed the completeness of each assembly based on evolutionary expectations of gene content with the help of BUSCO [75], which reports the number of complete and fragmented single-copy orthologs. Each de novo assembler was also evaluated in terms of execution time and memory consumption.

All datasets from Table 5.1 were run on Ubuntu based systems with two 6-core Intel® Xeon® CPU E5645 @ 2.40GHz processors, using 12 threads. Maximal memory consumption and CPU time were measured with the *time* command using parameter *-v*. Bellow is the complete list of all tools used in the evaluation:

- Dnadiff [73] - `https://sourceforge.net/projects/mummer`, version 3.23,
- Quast-LG [74] - `https://https://github.com/ablab/quast`, version 5.0.2
- BUSCO [75] - `https://gitlab.com/ezlab/busco`, integrated in Quast-LG
- Minimap [30] - `https://github.com/lh3/minimap`, commit: *1cd6ae3*
- Miniasm [30] - `https://github.com/lh3/miniasm`, commit: *17d5bd1*
- Canu [28] - `https://github.com/marbl/canu`, version 1.2
- FALCON [26] - `https://github.com/PacificBiosciences/FALCON-integrate`, commit: *8bb2737*.
- Nanopolish [23] - `https://github.com/jts/nanopolish`, commit: *47dcd7f*
- Flye [35] - `https://github.com/fenderglass/Flye`, version 2.6
- Wtdbg2 [36] - `https://github.com/ruanjue/wtdbg2`, version 2.5
- Shasta [38] - `https://github.com/chanzuckerberg/shasta`, version 0.3.0
- Racon [32] - `https://github.com/lbcb-sci/racon`, commit *2f41352*
- Ra [76] - `https://github.com/lbcb-sci/ra`, commit *07364a1*
- Raven - `https://github.com/lbcb-sci/raven`, version 0.0.5

Due to lack of sufficient computing power, we obtained the ONTHS assemblies of Wtdbg2 and Flye from their respective GitHub pages, including execution time and memory consumption. The same holds for plant datasets DRA002710 and ONTSP, which were only used to see the execution time ratios of each Raven component.

## 5.4 Results

Racon coupled with Minimap and Miniasm (abbreviated with MR) was compared to Canu and Falcon, state-of-the-art assemblers at the time Racon was published [32], and the results are presented in Table 5.2. The same Minimap and Miniasm combination was polished with a similar consensus module called Sparc [77], but Racon outperformed it in both accuracy and speed [32]. We also tried to use overlapping windows which resulted with a quality improvement up to 0.06%, but the increase of $10-15\%$ in running time was not justifiable. Racon can also be utilized for sequence error correction as shown in [32], but the comparison was left out because majority of assemblers nowadays skip that step in the assembly.

Ra was independently evaluated before publication in two different studies. Authors of [78] evaluated different assemblers on simulated and real bacterial datasets, from which Ra was declared the most reliable assembler. Authors of [79] tried to assemble three large plant genomes, and Ra yielded the most contiguous assemblies. Both results encouraged us to publish a brief description and evalution of Ra in [76]. Table 5.3 encapsulates that assembly evaluation on several third generation sequencing data. Here we extend it to more datasets (table entries bellow the horizontal line) and include Raven. We want to see the benefits of combining all phases into a single binary and the impact of the postprocessing assembly graph method introduced in the Raven assembler.

Eventually, we evaluated Raven in more detail on much larger eukaryotic genomes. The results are compared to newer state-of-the-art assemblers Flye, Wtdbg2 and Shasta, and are presented in Table 5.4. The evaluation includes execution time, peak memory consumption, the NG50 value, number of contigs, number of missasemblies, overall accuracy and BUSCO scores. More insights about execution time of each Raven component can be found in Table 5.5.

**Table 5.2:** Racon consensus evaluation with two state-of-the-art assemblers Canu and Falcon. Assembly metrics were obtained with Dnadiff [73] across five datasets of varying genome sizes.

| **Dataset** (Size) | **Assembler** | **Total bases** | **Aln. bases ref.** | **Aln. bases asm.** | **Avg. identity** | **CPU time** (min) |
|---|---|---|---|---|---|---|
| ERR1147227 (4641652) | MR ($1 \times R$) | 4637173 | 4640867 | 4636689 | 0.9913 | 25 |
| | MR ($2 \times R$) | **4632058** | **4641323** | 4632055 | 0.**9932** | **46** |
| | Canu | 4601503 | 4631173 | 4601365 | 0.9928 | 1328 |
| | Falcon | 4580230 | 4627613 | 4580230 | 0.9884 | 829 |
| PBEC (4641652) | MR ($1 \times R$) | 4653199 | 4641501 | 4653111 | 0.9963 | 86 |
| | MR ($2 \times R$) | **4645508** | 4641439 | 4645508 | 0.9990 | **162** |
| | Canu | 4664416 | **4641652** | 4664416 | **0.9999** | 773 |
| | Falcon | 4666788 | **4641652** | 4666788 | 0.9990 | 2908 |
| ERX1910723 (12157105) | MR ($1 \times R$) | 12172019 | 12104541 | 12108082 | 0.9788 | 28 |
| | MR ($2 \times R$) | **12167797** | 12110095 | 12115796 | 0.9804 | **44** |
| | Canu | 12224535 | **12120070** | 12196684 | **0.9861** | 13243 |
| | Falcon | 11643917 | 11885904 | 11643482 | 0.9822 | 9603 |
| SRX533604 (12157105) | MR ($1 \times R$) | 12071278 | 12023607 | 12046299 | 0.9943 | 115 |
| | MR ($2 \times R$) | **12051573** | 12025677 | 12027338 | 0.9972 | **215** |
| | Canu | 12402332 | **12127627** | 12363941 | **0.9986** | 6375 |
| | Falcon | 12003077 | 11932488 | 11910549 | 0.9970 | 14808 |
| PBCE (100272607) | MR ($1 \times R$) | 106353704 | 100017898 | 101711974 | 0.9944 | 1247 |
| | MR ($2 \times R$) | 106392402 | 99979140 | 101741297 | 0.9973 | **2004** |
| | Canu | 106687886 | **100166301** | 102928910 | **0.9989** | 37853 |
| | Falcon | **105858394** | 99295695 | 102008289 | 0.9974 | 119766 |

**Table 5.3:** Comparison between assemblers Ra and Raven. Assembly metrics were obtained with Quast-LG [74] on fourteen datasets of varying genome sizes.

| **Dataset** (Size) | **Assembler** | **Total length** | **NG50** | **Accuracy** | **CPU time** (min) | **Memory** (GB) |
|---|---|---|---|---|---|---|
| SRR5665597 (5682322) | Ra | 5453526 | 5344601 | 0.9841 | 147 | 8.32 |
| | Raven | 5476848 | 5335669 | 0.9881 | 81 | 9.36 |
| ERR1140973 (5682322) | Ra | 5548584 | 5299089 | 0.9915 | 27 | 3.45 |
| | Raven | 5545774 | 5296442 | 0.9927 | 25 | 4.38 |
| ERX1910723 (12157105) | Ra | 12166217 | 288671 | 0.9740 | 114 | 8.62 |
| | Raven | 12437509 | 533782 | 0.9913 | 87 | 9.79 |
| SRX533604 (12157105) | Ra | 12267357 | 711780 | 0.9974 | 178 | 16.87 |
| | Raven | 12288222 | 755443 | 0.9982 | 222 | 17.77 |
| SRR6702603 (143726002) | Ra | 128449879 | 1854698 | 0.9884 | 1276 | 22.94 |
| | Raven | 135925805 | 7402264 | 0.9911 | 726 | 25.91 |
| SRX499318 (143726002) | Ra | 135936009 | 2128249 | 0.9976 | 4649 | 51.92 |
| | Raven | 151262482 | 7782087 | 0.9971 | 4855 | 52.16 |
| SRR6780920 (1698329) | Ra | 1673060 | 1673060 | 0.9946 | 13 | 1.19 |
| | Raven | 1672065 | 1672065 | 0.9940 | 9 | 1.95 |
| SRR6780912 (3299801) | Ra | 3259446 | 498596 | 0.9927 | 9 | 0.90 |
| | Raven | 3332155 | 1823705 | 0.9920 | 8 | 1.48 |
| ERR1475873 (4086189) | Ra | 4163102 | 2539687 | 0.9994 | 139 | 6.66 |
| | Raven | 4190513 | 2445540 | 0.9995 | 87 | 9.74 |
| ERR987680 (5133713) | Ra | 4795265 | 4795265 | 0.9856 | 157 | 9.14 |
| | Raven | 4795725 | 4795725 | 0.9858 | 124 | 15.60 |
| ERS715397 (5280350) | Ra | 5538171 | 5295377 | 0.9245 | 106 | 7.45 |
| | Raven | 5547345 | 5294765 | 0.9240 | 103 | 10.27 |
| SRR5665591 (5682322) | Ra | 5280019 | 98427 | 0.9827 | 7 | 0.59 |
| | Raven | 5526879 | 133005 | 0.9788 | 6 | 1.47 |
| ERR1046594 (5682322) | Ra | 5876581 | 4235452 | 0.9906 | 88 | 6.98 |
| | Raven | 5856164 | 5323392 | 0.9909 | 86 | 9.27 |
| SRA360189 (23270305) | Ra | 23104151 | 825454 | 0.9894 | 1818 | 30.09 |
| | Raven | 23959339 | 1295320 | 0.9868 | 3556 | 31.42 |

**Table 5.4:** Comparison between Raven and state-of-the-art assemblers Wtdbg2, Flye and Shasta. Assembly metrics were obtained with Quast-LG [74] on five datasets of varying genome sizes.

| **Dataset** (Size) | **Assembler** | **Total length** | **Contigs** | **NG50** | **Accuracy** | **Misassemblies** | | | **BUSCO** | | **CPU time** (min) | **Memory** (GB) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | Reloc. | Transloc. | Inver. | Complete | Partial | | |
| PBCE (100286401) | Raven | 108388051 | 110 | 1914650 | 0.9979 | 362 | 208 | 95 | 0.9538 | 0.0033 | 1755 | 33.11 |
| | Flye | **102386568** | **66** | **2860001** | **0.9985** | 414 | 178 | 76 | **0.9637** | 0.0000 | 3860 | 32.16 |
| | Wtdbg2 | 106376824 | 147 | 2660891 | 0.9983 | 240 | 121 | 73 | 0.9406 | 0.0198 | 466 | **11.42** |
| | Shasta | 106442541 | 282 | 1034734 | 0.9875 | **221** | **81** | **70** | 0.7096 | 0.1551 | **319** | 35.94 |
| PBAT (119668634) | Raven | 122311874 | **80** | 8679429 | 0.9866 | 2682 | 3215 | 17 | 0.8614 | 0.0561 | 3587 | 30.35 |
| | Flye | **119566437** | 214 | 11162372 | **0.9887** | 2699 | 3146 | 20 | **0.9340** | 0.0066 | 6504 | 46.54 |
| | Wtdbg2 | 121374889 | 288 | **13977787** | 0.9869 | **2522** | **2953** | **16** | 0.9142 | 0.0165 | **1528** | **23.82** |
| | Shasta | 42476282 | 1840 | 40285 | 0.9667 | 441 | 225 | 2 | 0.0066 | 0.462 | 115 | 46.82 |
| SRR6702603 (143726002) | Raven | 135925805 | **164** | 7402264 | **0.9911** | 439 | 757 | **6** | **0.8878** | 0.0792 | **726** | 25.91 |
| | Flye | **139296030** | 624 | **16953989** | 0.9897 | 508 | 1177 | 7 | 0.8482 | 0.1155 | 3827 | 31.03 |
| | Wtdbg2 | 136380456 | 633 | 10619613 | 0.9841 | **313** | **291** | **6** | 0.5050 | 0.3564 | 943 | **17.89** |
| | Shasta | 106644146 | 1394 | 125561 | 0.9779 | 143 | 141 | 3 | 0.3366 | 0.3630 | 171 | 19.18 |
| SRR5439404 (143726002) | Raven | **139428168** | **142** | 7383867 | 0.9917 | 1883 | 2343 | **22** | 0.9934 | 0.0000 | 5275 | 56.98 |
| | Flye | 134586137 | 249 | 13842171 | **0.9927** | 1652 | **1972** | **22** | **0.9967** | 0.0000 | 9942 | 75.32 |
| | Wtdbg2 | 138182771 | 344 | **21294972** | 0.9906 | **1636** | 2066 | 24 | 0.9934 | 0.0033 | **618** | **19.34** |
| | Shasta | 132820272 | 284 | 3816008 | 0.9870 | 1554 | 1553 | 21 | 0.9307 | 0.0396 | 665 | 85.73 |
| ONTHS (3272116950) | Raven | 2765865635 | **741** | 5706195 | 0.9822 | 3159 | 1625 | 46 | 0.4323 | 0.1716 | 136706 | 285.44 |
| | Flye | **2876181491** | 2589 | **20413247** | **0.9953** | 5171 | 2890 | 65 | **0.8746** | 0.0429 | 150000 | 714 |
| | Wtdbg2 | 2701125720 | 4219 | 11044992 | 0.9858 | **2564** | **1188** | **39** | 0.4191 | 0.1914 | **53457** | **221.8** |
| | Shasta | 1617731845 | 20637 | 215694 | 0.9636 | 1237 | 341 | 12 | 0.0264 | 0.0132 | 5030 | 346.5 |

**Table 5.5:** Detailed look into execution time of Raven components. All values are measured inside the source code and represent wall clock time in seconds.

| Dataset | Overlap | Preproc. | Layout | Postproc. | Mapping | Consensus |
|---|---|---|---|---|---|---|
| SRX533604 | 178 | 66 | 4 | 14 | 42 | **876** |
| SRA360189 | 2593 | 11785 | 157 | 62 | 327 | **3287** |
| PBCE | 2735 | 1925 | 10 | 130 | 414 | **4068** |
| PBAT | 4048 | 1422 | 28 | 71 | 457 | **12599** |
| SRR6702603 | 900 | 471 | 16 | 88 | 227 | **2217** |
| SRR5439404 | 9814 | 3002 | 105 | 113 | 789 | **14227** |
| SRX499318 | 8243 | 4013 | 293 | 221 | 768 | **12486** |
| DRA002710 | **61688** | **29310** | 2544 | 3787 | 4712 | 19777 |
| ONTSP | **669990** | **155698** | 3056 | 3075 | 16373 | 119981 |
| ONTHS | **970264** | **161358** | 445 | 854 | 29503 | 95137 |

## 5.5 Discussion

By combining Racon with Miniasm and Minimap we showed that it is posible to obtain high accuracy assemblies without sequence error correction prior assembly, while being an order of magnitute faster as shown in Table 5.2. In addition, the resulting assembly sizes were more similar to corresponding reference genomes than other methods in the evaluation. Shearing unitigs into nonoverlapping windows enabled linear scalability given constant sequencing depth [32], which contributed in more conspicuous speedups on larger genomes. Racon was upgraded after the publication which resulted in decreased resource requirements and ability to use second generation sequencing data.

Evaluation between two versions of our de novo assembler, Ra and Raven, showed that writing of intermediate results to the hard drive can be omitted without any negative impact on execution time or memory consumption, but can even lead to decreased execution times. Implementing the graph drawing based simplification method increased the NG50 value of larger genomes significantly, but sometimes yields a bit longer assembly which needs to be investigated further. When compared to other state-of-the-art assemblers, Raven is positioned between Wtdbg2 and Flye in terms of execution speed, and comes second in terms of memory allocation. Raven consistently outputs the smallest number of contigs covering almost the entire genome, but has smaller NG50 values. The accuracy is comparable across all datasets, including the number of different misassembly types and BUSCO scores. Shasta was significantly faster than any other assembler, but had troubles assembling the provided datasets probably due to its design for large genomes sequenced with Oxford Nanopore Technologies at depth around 60.

Inspecting the execution time of all virtual components of Raven in Table 5.5, we can observe that in small prokaryotic and fungal datasets the consensus phase is the most dominant component. Increasing the size of the genome suits the linear complexity of it, and the overlap phase becomes the most dominant component. The increase in execution time is also palpable in the preprocesing step, where pile-o-grams are constructed anew by finding a subset of pairwise overlaps with a relaxed *k*-mer filter. We think that both the overlap and the preprocessing phase have room for improvements. Graph construction and simplification methods of the layout phase take minor part in the cumulative execution time. While the postprocessing time overtakes the time needed for the layout, it is still greatly dominated by other components.

# Chapter 6

# Conclusion

We have presented the algorithms behind our de novo assembler Raven. It is based on the overlap-layout-consensus paradigm and designed for long erroneous sequences produced by Pacific Biosciences or Oxford Nanopore Technologies. Raven does not employ error correction prior assembly, has memory requirements bounded by the input sequence set and rapidly generates consensus sequences with better or comparable accuracy than the state-of-the art. We have successfully managed to prove set out research hypotheses and developed several open-source tools. The predefined contribution of the research conducted in this thesis is fulfilled and presented in the next section.

## 6.1 Contribution of the thesis

**Memory efficient algorithm for layout phase of the overlap-layout-consensus paradigm achieving a low number of fragments in large genome assemblies.**

A detailed description of algorithms used for the layout phase with long erroneous reads produced by third generation of sequencing is given in Chapter 3. The focus was to enable large genome assemblies with considerate memory requirements. We extended the assembly graph introduced by Miniasm [30] with ideas from the HINGE [33] for preprocessing, and a novel algorithm for graph postprocessing. Pile-o-grams are utilized for identification of chimeric sequences and repeat-induced overlaps. The goal was to decrease the number of tangles in the assembly graph prior construction. Pile-o-grams are downsampled and constructed iteratively from blocks of overlaps, either loading them from a file or computing them in memory. The whole set of overlaps is never stored entirely in the memory. Only a small portion of overlaps are stored for containment removal, and afterwards the suffix-prefix overlaps are computed anew. The graph is simplified the regular way, with pruning of dead ends and bubble popping. The leftover junctions are resolved based on vertex distances in a force directed placement in a two dimensional plane [47], an algorithm which was optimized to run in quasilinear time. Mentioned algorithms were integrated in a standalone module Rala, and the complete OLC based assembler Ra, which was later upgraded to Raven. Preliminary results without the postprocessing method were presented in:

- Vaser, R., Šikić, M., "Yet another de novo genome assembler", in 2019 11th International Symposium on Image and Signal Processing and Analysis (ISPA), 2019, pp. 147-151, available at: `https://doi.org/10.1109/ISPA.2019.8868909`

**Fast algorithm for consensus phase of the overlap-layout-consensus paradigm ensuring high accuracy of large genome assemblies.**

Algorithms for the consensus phase with long error-prone reads obtained with third generation of sequencing data are presented in Chapter 4. We based our standalone consensus module on partial order alignment [20], which was implemented in a separate tool called Spoa. Partial order alignment was optimized with SIMD instructions, supporting three alignment modes and three gap models, following the intraset parallelization proposed for pairwise alignment by [54] and applying the prefix-max algorithm horizontally. The consensus module Racon divides sequences into short nonoverlapping windows, filters out erroneous portions of them and computes the consensus from multiple sequence alignment with the help of Spoa. Coupled with the Minimap-Miniasm pipeline, which does not employ error correction prior assembly, we showed that accurate genome assemblies are possible with an order of magnitude speedup when compared to state-of-the-art. We later integrated Racon in our assemblers Ra and Raven. The results were published in:

- Vaser, R., Sović, I., Nagarajan, N., Šikić, M., "Fast and accurate de novo genome assembly from long uncorrected reads", Genome Research, Vol. 27, 2017, pp. 737-746, available at: `https://doi.org/10.1101/gr.214270.116`

**System for de novo assembly of large genomes from data produced by third generation of sequencing.**

Chapter 5 provides a look into the development of our complete overlap-layut-consensus based assembler Raven, accompanied with a full pseudocode. We integrated the drawing based simplification method for assembly graphs in Raven and therefore compared the results with Ra. Results show that the contiguity greatly increases with little impact on execution time. We thoroughly evaluated Raven and newer state-of-the-art tools as well. Raven competes with execution time, memory consumption and accuracy with both Flye [35] and Wtdbg2 [36], consistently producing the fewest contigs but having a bit smaller NG50 values. Although, on majority of the datasets the Wtdbg2 assembler is considerably faster due to its sparse representation of sequences. A glimpse in execution time of different components in Raven gives hope for possible future improvements.

## 6.2 Future research

First and foremost the overlap phase and then the reconstruction of pile-o-grams should be improved as they constitute the majority of time on larger genomes. Our belief is that a fast containment removal algorithm could save a lot of wasted time in the overlap phase as the majority of reads are contained in others. This could be achieved by finding approximate overlaps using *k*-mers located at ends of short sequences and the full *k*-mer set of the longest sequences. The decreased amount of reads would later facilitate the retrieval of pairwise suffix-prefix overlaps needed for assembly graph construction. On the other hand, increasing the coverage of repetitive regions in pile-o-grams by aligning all contained sequences to the set of uncontained sequences might be computationally demanding, as seen on the *Plasmodium falciparum* dataset. A better approach would shrink the number of reads for which we recreate pile-o-grams, which might be possible by identifying reads that have a high number of frequent *k*-mers that are filtered out in the overlap phase.

The graph untangling procedure based on the graph drawing should be explored more in detail. Removing edges based on a simple length threshold between two vertices might be replaced with a more robust method which takes into account the neighbourhood of vertices in question. This could resolve even more corner cases which are hard to detected with preprocessing methods.

Although the speed of the consensus module Racon is remarkable, additional improvements might be possible with banded alignment applied to partial order graphs. Researches of ClaraGenomics have already massively parallelized partial order alignments on GPUs, which we have integrated in Racon resulting in almost an order of magnitude speedup. Additionally, ClaraGenomics implemented pairwise alignment and started designing retrieval of approximate overlaps with minimizers, improvements that might enable the whole Raven assembler to be run on the GPU needing only a tiny portion of the current execution times.

# Bibliography

[1] Maxam, A. M., Gilbert, W., "A new method for sequencing dna", Proceedings of the National Academy of Sciences, Vol. 74, No. 2, 1977, pp. 560-564, available at: https://doi.org/10.1073/pnas.74.2.560

[2] Sanger, F., Nicklen, S., Coulson, A. R., "Dna sequencing with chain-terminating inhibitors", Proceedings of the National Academy of Sciences, Vol. 74, No. 12, 1977, pp. 5463-5467, available at: https://doi.org/10.1073/pnas.74.12.5463

[3] Sedlazeck, F., Lee, H., Darby, C., Schatz, M., "Piercing the dark matter: bioinformatics of long-range sequencing and mapping", Nature Reviews Genetics, Vol. 19, 2018, p. 329–346, available at: https://doi.org/10.1038/s41576-018-0003-4

[4] Loman, N., Misra, R., Dallman, T., Constantinidou, C., Gharbia, S., Wain, J., Pallen, M., "Performance comparison of benchtop high-throughout sequencing platforms", Nature biotechnology, Vol. 30, 2012, pp. 434-439, available at: https://doi.org/10.1038/nbt.2198

[5] Nagarajan, N., Pop, M., "Sequence assembly demystified", Nature reviews. Genetics, Vol. 14, p. 157–167, available at: https://doi.org/10.1038/nrg3367

[6] Rhoads, A., Au, K., "Pacbio sequencing and its applications", Genomics, proteomics & bioinformatics, Vol. 13, 2015, pp. 278-289, available at: https://doi.org/10.1016/j.gpb.2015.08.002

[7] Lu, H., Giordano, F., Ning, Z., "Oxford nanopore minion sequencing and genome assembly", Genomics, Proteomics & Bioinformatics, Vol. 14, 2016, pp. 265-279, available at: https://doi.org/10.1016/j.gpb.2016.05.004

[8] Wenger, A. M., Peluso, P., Rowell, W. J., Chang, P.-C., Hall, R. J., Concepcion, G. T., Ebler, J., Fungtammasan, A., Kolesnikov, A., Olson, N. D., Töpfer, A., Alonge, M., Mahmoud, M., Qian, Y., Chin, C.-S., Phillippy, A. M., Schatz, M. C., Myers, G., DePristo, M. A., Ruan, J., Marschall, T., Sedlazeck, F. J., Zook, J. M., Li, H., Koren, S., Carroll, A., Rank, D. R., Hunkapiller, M. W., "Accurate circular consensus long-read sequencing improves variant detection and assembly of a human

genome", Nature Biotechnology, Vol. 37, No. 10, 2019, pp. 1152-1162, available at: https://doi.org/10.1038/s41587-019-0217-9

[9] Jain, M., Koren, S., Miga, K., Quick, J., Rand, A., Sasani, T., Tyson, J., Beggs, A., Dilthey, A., Fiddes, I., Malla, S., Marriott, H., Nieto, T., O'Grady, J., Olsen, H., Pedersen, B., Rhie, A., Richardson, H., Quinlan, A., Loose, M., "Nanopore sequencing and assembly of a human genome with ultra-long reads", Nature Biotechnology, Vol. 36, 2018, p. 338–345, available at: https://doi.org/10.1038/nbt.4060

[10] Sanger, F., Coulson, A. R., Hong, G. F., Hill, D. F., Petersen, G. B., "Nucleotide sequence of bacteriophage $\lambda$ dna", Journal of Molecular Biology, Vol. 162, No. 4, 1982, pp. 729-773, available at: https://doi.org/10.1016/0022-2836(82)90546-0

[11] Myers, E. W., "Toward simplifying and accurately formulating fragment assembly", Journal of Computational Biology, Vol. 2, No. 2, 1995, pp. 275-290, available at: https://doi.org/10.1089/cmb.1995.2.27

[12] Pevzner, P., Tang, H., Waterman, M., "An eulerian path approach to dna fragment assembly", Proceedings of the National Academy of Sciences of the United States of America, Vol. 98, 2001, pp. 9748-9753, available at: https://doi.org/10.1073/pnas.171285098

[13] Hierholzer, C., Wiener, C., "Über die möglichkeit, einen linienzug ohne wiederholung und ohne unterbrechung zu umfahren", Mathematische Annalen, Vol. 6, 1873, pp. 30-32, available at: https://doi.org/10.1007/BF01442866

[14] Koren, S., Schatz, M., Walenz, B., Martin, J., Howard, J., Ganapathy, G., Wang, Z., Rasko, D., Mccombie, W., Jarvis, E., Phillippy, A., "Hybrid error correction de novo assembly of single-molecule sequencing reads", Nature biotechnology, Vol. 30, 2012, pp. 693-700, available at: https://doi.org/10.1038/nbt.2280

[15] Goodwin, S., Gurtowski, J., Ethe-Sayers, S., Deshpande, P., Schatz, M., Mccombie, W., "Oxford nanopore sequencing, hybrid error correction, and de novo assembly of a eukaryotic genome", Genome research, Vol. 25, 2015, p. 1750–1756, available at: https://doi.org/10.1101/gr.191395.115

[16] Myers, E. W., Sutton, G. G., Delcher, A. L., Dew, I. M., Fasulo, D. P., Flanigan, M. J., Kravitz, S. A., Mobarry, C. M., Reinert, K. H. J., Remington, K. A., Anson, E. L., Bolanos, R. A., Chou, H.-H., Jordan, C. M., Halpern, A. L., Lonardi, S., Beasley, E. M., Brandon, R. C., Chen, L., Dunn, P. J., Lai, Z., Liang, Y., Nusskern, D. R., Zhan, M., Zhang, Q., Zheng, X., Rubin, G. M., Adams, M. D., Venter, J. C., "A whole-genome

assembly of drosophila", Science, Vol. 287, No. 5461, 2000, pp. 2196-2204, available at: https://doi.org/10.1126/science.287.5461.2196

[17] Pop, M., Phillippy, A., Delcher, A. L., Salzberg, S. L., "Comparative genome assembly", Briefings in Bioinformatics, Vol. 5, No. 3, 2004, pp. 237-248, available at: https://doi.org/10.1093/bib/5.3.237

[18] Chin, C.-S., Alexander, D., Marks, P., Klammer, A., Drake, J., Heiner, C., Clum, A., Copeland, A., Huddleston, J., Eichler, E., Turner, S., Korlach, J., "Nonhybrid, finished microbial genome assemblies from long-read smrt sequencing data", Nature methods, Vol. 10, 2013, p. 563–569, available at: https://doi.org/10.1038/nmeth.2474

[19] Chaisson, M., Tesler, G., "Mapping single molecule sequencing reads using basic local alignment with successive refinement (blasr): Theory and application.", BMC bioinformatics, Vol. 13, 2012, p. 238, available at: https://doi.org/10.1186/1471-2105-13-238

[20] Lee, C., Grasso, C., Sharlow, M. F., "Multiple sequence alignment using partial order graphs ", Bioinformatics, Vol. 18, No. 3, 2002, pp. 452-464, available at: https://doi.org/10.1093/bioinformatics/18.3.452

[21] Lee, C., "Generating consensus sequences from partial order multiple sequence alignment graphs", Bioinformatics, Vol. 19, No. 8, 2003, pp. 999-1008, available at: https://doi.org/10.1093/bioinformatics/btg109

[22] Altschul, S., Gish, W., Miller, W., Myers, E., Lipman, D., "Basic local aligment search tool", Journal of molecular biology, Vol. 215, 1990, pp. 403-410, available at: https://doi.org/10.1016/S0022-2836(05)80360-2

[23] Loman, N., Quick, J., Simpson, J., "A complete bacterial genome assembled de novo using only nanopore sequencing data", Nature methods, Vol. 12, 2015, p. 733–735, available at: https://doi.org/10.1038/nmeth.3444

[24] Myers, E. W., "Efficient local alignment discovery amongst noisy long reads", in Algorithms in Bioinformatics. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 52-67, available at: https://doi.org/10.1007/978-3-662-44753-6_5

[25] Berlin, K., Koren, S., Chin, C.-S., Drake, J., Landolin, J., Phillippy, A., "Assembling large genomes with single-molecule sequencing and locality sensitive hashing", Nature biotechnology, Vol. 33, 2015, pp. 623-630, available at: https://doi.org/10.1038/nbt.3238

[26] Chin, C.-S., Peluso, P., Sedlazeck, F., Nattestad, M., Concepcion, G., Clum, A., Dunn, C., O'Malley, R., Figueroa-Balderas, R., Morales-Cruz, A., Cramer, G., Delledonne, M., Luo, C., Ecker, J., Cantu, D., Rank, D., Schatz, M., "Phased diploid genome assembly with single-molecule real-time sequencing", Nature Methods, Vol. 13, 2016, pp. 1050-1054, available at: https://doi.org/10.1038/nmeth.4035

[27] Myers, E. W., "The fragment assembly string graph", Bioinformatics, Vol. 21, No. suppl_2, 2005, pp. ii79-ii85, available at: https://doi.org/10.1093/bioinformatics/bti1114

[28] Koren, S., Walenz, B., Berlin, K., Miller, J., Bergman, N., Phillippy, A., "Canu: Scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation", Genome Research, Vol. 27, 03 2017, pp. 722-736, available at: https://doi.org/10.1101/gr.215087.116

[29] Miller, J. R., Delcher, A. L., Koren, S., Venter, E., Walenz, B. P., Brownley, A., Johnson, J., Li, K., Mobarry, C., Sutton, G., "Aggressive assembly of pyrosequencing reads with mates", Bioinformatics, Vol. 24, No. 24, 2008, pp. 2818-2824, available at: https://doi.org/10.1093/bioinformatics/btn548

[30] Li, H., "Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences", Bioinformatics, Vol. 32, No. 14, 2016, pp. 2103-2110, available at: https://doi.org/10.1093/bioinformatics/btw152

[31] Sović, I., Šikić, M., Wilm, A., Fenlon, S. N., Chen, S. L., Nagarajan, N., "Fast and sensitive mapping of nanopore sequencing reads with graphmap", Nature communications, Vol. 7, 2016, p. 11307, available at: https://doi.org/10.1038/ncomms11307

[32] Vaser, R., Sović, I., Nagarajan, N., Sikic, M., "Fast and accurate de novo genome assembly from long uncorrected reads", Genome Research, Vol. 27, 2017, pp. 737-746, available at: https://doi.org/10.1101/gr.214270.116

[33] Kamath, G. M., Shomorony, I., Xia, F., Courtade, T. A., Tse, D. N. C., "Hinge: long-read assembly achieves optimal repeat resolution.", Genome research, Vol. 27, No. 5, 2017, pp. 747-756, available at: https://doi.org/10.1101/gr.216465.116

[34] Lin, Y., Yuan, J., Kolmogorov, M., Chaisson, M., Pevzner, P., "Assembly of long error-prone reads using de bruijn graphs", Proceedings of the National Academy of Sciences, Vol. 113, 2016, pp. E8396-E8405, available at: https://doi.org/10.1073/pnas.1604560113

[35] Kolmogorov, M., Yuan, J., Lin, Y., Pevzner, P., “Assembly of long, error-prone reads using repeat graphs”, Nature Biotechnology, Vol. 37, 2019, pp. 540-546, available at: https://doi.org/10.1038/s41587-019-0072-8

[36] Ruan, J., Li, H., “Fast and accurate long-read assembly with wtdbg2”, bioRxiv, 2019, available at: https://doi.org/10.1101/53097

[37] Chin, C.-S., Khalak, A., “Human genome assembly in 100 minutes”, bioRxiv, 2019, available at: https://doi.org/10.1101/705616

[38] Shafin, K., Pesout, T., Lorig-Roach, R., Haukness, M., Olsen, H. E., Bosworth, C., Armstrong, J., Tigyi, K., Maurer, N., Koren, S., Sedlazeck, F. J., Marschall, T., Mayes, S., Costa, V., Zook, J. M., Liu, K. J., Kilburn, D., Sorensen, M., Munson, K. M., Vollger, M. R., Eichler, E. E., Salama, S., Haussler, D., Green, R. E., Akeson, M., Phillippy, A., Miga, K. H., Carnevali, P., Jain, M., Paten, B., “Efficient de novo assembly of eleven human genomes using promethion sequencing and a novel nanopore toolkit”, bioRxiv, 2019.

[39] Shannon, P., Markiel, A., Ozier, O., Baliga, N. S., Wang, J. T., Ramage, D., Amin, N., Schwikowski, B., Ideker, T., “Cytoscape: A software environment for integrated models of biomolecular interaction networks”, Genome Research, Vol. 13, No. 11, 2003, pp. 2498-2504, available at: https://doi.org/10.1101/gr.1239303

[40] Pop, M., “Genome assembly reborn: recent computational challenges”, Briefings in Bioinformatics, Vol. 10, No. 4, 2009, pp. 354-366, available at: https://doi.org/10.1093/bib/bbp026

[41] Garey, M. R., Johnson, D. S., Computers and Intractability; A Guide to the Theory of NP-Completeness. New York, NY, USA: W. H. Freeman & Co., 1990.

[42] Myers, E. W., “Whole-genome dna sequencing”, Computing in Science and Engg., Vol. 1, No. 3, 1999, pp. 33-43, available at: https://doi.org/10.1109/5992.764214

[43] Zerbino, D. R., Birney, E., “Velvet: algorithms for de novo short read assembly using de bruijn graphs.”, Genome research, Vol. 18, No. 5, 2008, pp. 821-9, available at: https://doi.org/10.1101/gr.074492.107

[44] Hunter, J. D., “Matplotlib: A 2d graphics environment”, Computing in Science & Engineering, Vol. 9, No. 3, 2007, pp. 90-95, available at: https://doi.org/10.1109/MCSE.2007.55

[45] Krumsiek, J., Arnold, R., Rattei, T., “Gepard: a rapid and sensitive tool for creating dotplots on genome scale”, Bioinformatics, Vol. 23, No. 8, 2007, pp. 1026-1028, available at: https://doi.org/10.1093/bioinformatics/btm039

[46] Wick, R. R., Judd, L. M., Gorrie, C. L., Holt, K. E., “Unicycler: Resolving bacterial genome assemblies from short and long sequencing reads”, PLOS Computational Biology, Vol. 13, No. 6, 2017, pp. 1-22, available at: https://doi.org/10.1371/journal.pcbi.1005595

[47] Fruchterman, T. M. J., Reingold, E. M., “Graph drawing by force-directed placement”, Softw. Pract. Exper., Vol. 21, No. 11, 1991, pp. 1129-1164, available at: https://doi.org/10.1002/spe.4380211102

[48] Hopcroft, J. E., Tarjan, R. E., “Efficient algorithms for graph manipulation (algorithm 447)”, Commun. ACM, Vol. 16, 1973, pp. 372-378, available at: https://doi.org/10.1145/362248.362272

[49] Barnes, J. K., Hut, P., “A hierarchical o(n log n) force-calculation algorithm”, Nature, Vol. 324, 1986, pp. 446-449, available at: https://doi.org/10.1038/324446a0

[50] Finkel, R. A., Bentley, J. L., “Quad trees a data structure for retrieval on composite keys”, Acta Informatica, Vol. 4, 1974, pp. 1-9, available at: https://doi.org/10.1007/BF00288933

[51] Needleman, S. B., Wunsch, C. D., “A general method applicable to the search for similarities in the amino acid sequence of two proteins”, Journal of molecular biology, Vol. 48, No. 3, 1970, pp. 443-53, available at: https://doi.org/10.1016/0022-2836(70) 90057-4

[52] Smith, T. F., Waterman, M. S., “Identification of common molecular subsequences”, Journal of molecular biology, Vol. 147, No. 1, 1981, pp. 195-7, available at: https://doi.org/10.1016/0022-2836(81)90087-5

[53] Wozniak, A., “Using video-oriented instructions to speed up sequence comparison”, Bioinformatics, Vol. 13, No. 2, 1997, pp. 145-150, available at: https://doi.org/10.1093/bioinformatics/13.2.145

[54] Rognes, T., Seeberg, E., “Six-fold speed-up of Smith–Waterman sequence database searches using parallel processing on common microprocessors”, Bioinformatics, Vol. 16, No. 8, 2000, pp. 699-706, available at: https://doi.org/10.1093/bioinformatics/16.8.699

[55] Farrar, M., “Striped Smith–Waterman speeds database searches six times over other SIMD implementations”, Bioinformatics, Vol. 23, No. 2, 2006, pp. 156-161, available at: https://doi.org/10.1093/bioinformatics/btl582

[56] Sankoff, D., “Matching sequences under deletion-insertion constraints”, Vol. 69, No. 1, 1972, pp. 4-6, available at: https://doi.org/10.1073/pnas.69.1.4

[57] Gotoh, O., “An improved algorithm for matching biological sequences”, Journal of Molecular Biology, Vol. 162, No. 3, 1982, pp. 705-708, available at: https://doi.org/10.1016/0022-2836(82)90398-9

[58] Kahn, A. B., “Topological sorting of large networks”, Communications of the ACM, Vol. 5, No. 11, 1962, pp. 558-562, available at: https://doi.org/10.1145/368996.369025

[59] Li, H., “Minimap2: pairwise alignment for nucleotide sequences”, Bioinformatics, Vol. 34, No. 18, 2018, pp. 3094-3100, available at: https://doi.org/10.1093/bioinformatics/bty191

[60] Ukkonen, E., “Algorithms for approximate string matching”, Information and Control, Vol. 64, No. 1, 1985, pp. 100-118, international Conference on Foundations of Computation Theory, available at: https://doi.org/10.1016/S0019-9958(85)80046-2

[61] ŠoŠić, M., Šikić, M., “Edlib: a C/C++ library for fast, exact sequence alignment using edit distance”, Bioinformatics, Vol. 33, No. 9, 2017, pp. 1394-1395, available at: https://doi.org/10.1093/bioinformatics/btw753

[62] Todd, S. M., Settlage, R. E., Lahmers, K. K., Slade, D. J., “Fusobacterium genomics using minion and illumina sequencing enables genome completion and correction”, mSphere, Vol. 3, No. 4, 2018, available at: https://doi.org/10.1128/mSphere.00269-18

[63] “Public Health England reference collections - NCTC3000”, available at: https://www.sanger.ac.uk/resources/downloads/bacteria/nctc accessed on: 1 October 2019.

[64] “Loman Labs”, available at: http://lab.loman.net/2015/09/24/first-sqk-map-006-experiment accessed on: 1 October 2019.

[65] “Pacific Biosciences DevNet”, available at: https://github.com/PacificBiosciences/DevNet/wiki/Datasets accessed on: 1 October 2019.

[66] Wick, R. R., Judd, L. M., Gorrie, C. L., Holt, K. E., “Completing bacterial genome assemblies with multiplex minion sequencing”, Microbial Genomics, Vol. 3, No. 10, 2017, available at: https://doi.org/10.1099/mgen.0.000132

[67] Istace, B., Friedrich, A., d’Agata, L., Faye, S., Payen, E., Beluche, O., Caradec, C., Davidas, S., Cruaud, C., Liti, G., Lemainque, A., Engelen, S., Wincker, P., Schacherer, J., Aury, J.-M., “de novo assembly and population genomic survey of natural yeast isolates

with the Oxford Nanopore MinION sequencer", GigaScience, Vol. 6, No. 2, 2017, pp. 1-13, available at: https://doi.org/10.1093/gigascience/giw018

[68] Vembar, S. S., Seetin, M., Lambert, C., Nattestad, M., Schatz, M. C., Baybayan, P., Scherf, A., Smith, M. L., "Complete telomere-to-telomere de novo assembly of the Plasmodium falciparum genome through long-read (>11kb), single molecule, real-time sequencing", DNA Research, Vol. 23, No. 4, 2016, pp. 339-351, available at: https://doi.org/10.1093/dnares/dsw022

[69] Solares, E., Chakraborty, M., Miller, D., Kalsow, S., Hall, K., Perera, A., Emerson, J., Hawley, R. S., "Rapid low-cost assembly of the drosophila melanogaster reference genome using low-coverage, long-read sequencing", G3-Genes Genomes Genetics, Vol. 8, 2018, pp. 3143-3154, available at: https://doi.org/10.1534/g3.118.200162

[70] Chakraborty, M., Vankuren, N., Zhao, R., Zhang, X., Kalsow, S., Emerson, J., "Hidden genetic variation shapes the structure of functional elements in drosophila", Nature Genetics, Vol. 50, 2018, pp. 20-25, available at: https://doi.org/10.1038/s41588-017-0010-y

[71] Hoshino, A., Jayakumar, V., Nitasaka, E., Toyoda, A., Noguchi, H., Itoh, T., Shin-I, T., Minakuchi, Y., Koda, Y., Nagano, A., Yasugi, M., Honjo, M., Kudoh, H., Seki, M., Kamiya, A., Shiraki, T., Carninci, P., Asamizu, E., Nishide, H., Sakakibara, Y., "Genome sequence and analysis of the japanese morning glory ipomoea nil", Nature Communications, Vol. 7, 2016, p. 13295, available at: https://doi.org/10.1038/ncomms13295

[72] Schmidt, M. H.-W., Vogel, A., Denton, A. K., Istace, B., Wormit, A., van de Geest, H., Bolger, M. E., Alseekh, S., Maß, J., Pfaff, C., Schurr, U., Chetelat, R., Maumus, F., Aury, J.-M., Koren, S., Fernie, A. R., Zamir, D., Bolger, A. M., Usadel, B., "De novo assembly of a new solanum pennellii accession using nanopore sequencing", The Plant Cell, Vol. 29, No. 10, 2017, pp. 2336-2348, available at: https://doi.org/10.1105/tpc.17.00521

[73] Delcher, A., Salzberg, S., Phillippy, A., "Using mummer to identify similar regions in large sequence sets", Current protocols in bioinformatics / editoral board, Andreas D. Baxevanis ... [et al.], Vol. Chapter 10, 2003, p. Unit 10.3, available at: https://doi.org/10.1002/0471250953.bi1003s00

[74] Mikheenko, A., Prjibelski, A., Saveliev, V., Antipov, D., Gurevich, A., "Versatile genome assembly evaluation with QUAST-LG", Bioinformatics, Vol. 34, No. 13, 2018, pp. i142-i150, available at: https://doi.org/10.1093/bioinformatics/bty266

[75] Simão, F. A., Waterhouse, R. M., Ioannidis, P., Kriventseva, E. V., Zdobnov, E. M., "BUSCO: assessing genome assembly and annotation completeness with single-copy orthologs", Bioinformatics, Vol. 31, No. 19, 2015, pp. 3210-3212, available at: https://doi.org/10.1093/bioinformatics/btv351

[76] Vaser, R., Šikić, M., "Yet another de novo genome assembler", in 2019 11th International Symposium on Image and Signal Processing and Analysis (ISPA), 2019, pp. 147-151, available at: https://doi.org/10.1109/ISPA.2019.8868909

[77] Ye, C., Ma, Z., "Sparc: A sparsity-based consensus algorithm for long erroneous sequencing reads", PeerJ, Vol. 4, 2016, available at: https://doi.org/10.7717/peerj.2016

[78] Wick, R., "rrwick/Long-read-assembler-comparison: Initial release", available at: https://doi.org/10.5281/zenodo.2702443 2019.

[79] Belser, C., Istace, B., Denis, E., Dubarry, M., Baurens, F.-C., Falentin, C., Genete, M., Berrabah, W., Chevre, A.-M., Delourme, R., Deniot, G., Denoeud, F., Duffé, P., Engelen, S., Lemainque, A., Manzanares-Dauleux, M., Martin, G., Morice, J., Noel, B., Aury, J.-M., "Chromosome-scale assemblies of plant genomes using nanopore long reads and optical maps", Nature Plants, Vol. 4, 2018, p. pages879–887, available at: https://doi.org/10.1038/s41477-018-0289-4

# List of Figures

3.1. Example of an assembly graph constructed from an arbitrary set of sequences $S$ with cardinality $|S| = 13$. Vertices are denoted with $v_i$ and linked with dashed lines to their Watson-Crick complements $\overline{v}_i$. Edges are denoted with $e_{i,j}$, while their complenetary pairs with $\overline{e}_{j,i}$. The graph was manually drawn and annotated in Cytoscape [39]. Due to clarity, edge labels are absent in subsequent figures. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

3.2. Part of an assembly graph containing transitive edges. For any path of three vertices $v_i \rightarrow v_j \rightarrow v_k$, a transitive edge connects the first and last vertex of that path, $v_i \rightarrow v_j$. It can be removed without any loss of information, but removing any of them separately might hinder the detection of others. In this example, edge $v_5 \rightarrow v_1$ is not transitive without edge $v_4 \rightarrow v_1$, as well as $v_4 \rightarrow v_1$ without $v_3 \rightarrow v_1$. The subgraph was manually drawn and annotated in Cytoscape [39]. . 13

3.3. Part of an assembly graph containing a dead end (path $v_3 \rightarrow v_6 \rightarrow v_7$). Removing such paths should not lead to loss of information. The subgraph was manually drawn and annotated in Cytoscape [39]. . . . . . . . . . . . . . . . . . 14

3.4. Different path formations in bubble-like structures of the assembly graph. We assume that the path $v_5 \rightarrow v_4 \rightarrow v_3 \rightarrow v_2 \rightarrow v_1$ is the one we want to keep. The path over vertices $v_8$, $v_6$ and $v_7$ is searched for edges that will not discontinue other paths of the graph if removed. Subfigure (a) depicts the simplest scenario in which the whole path can be removed. When they are several vertices with indegree $deg^{-}(v) \geq 2$ and outdegree $deg^{+}(v) = 1$, everything before the first such vertex can be removed as in subfigure (b). Similar rule applies to the case in subfigure (c) where they are several vertices with $deg^{+}(v) \geq 2$ and $deg^{-}(v) = 1$. Everything after the last such vertex is safely removable. When there is a combination of those vertex types, edges between the last vertex with $deg^{+}(v) \geq 2$ and the first vertex with $deg^{-}(v) \geq 2$ can be safely removed, as seen in (d). This only applies when vertices do not have multiple edges of the other type, that is if there is vertex with $deg^{-}(v) \geq 2$ and $deg^{+} \geq 2$ in the path then no edges are removed as they would break other paths in the graph (subfigure (e)). All subfigures were manually drawn and annotated in Cytoscape [39]. . . . . . 15

3.5. Sequence pile-o-gram construction from a set of pairwise overlaps. Overlaps are stacked into a pile (subfigure (a)) and summed up over each base in the sequence (subfigure (b)). If we align the sequence to the reference genome, we can see that it is fully aligned to only one location (subfigure (c)), which results in an almost uniform pile-o-gram. Such sequences are the basis for high quality assemblies. The whole figure is tied to sequence *474ddffb-0b2b-4a92-8f06-8568265ca639* obtained from the dataset ERR1147227. Subfigures (a) and (b) were drawn in Python with Matplotlib [44], while subfigure (c) was exported from Gepard [45]. . . . . . . . . . . . . . . . . . . . . . . . . . . . 20

3.6. Pile-o-gram of a chimeric sequence. Not a single overlap covers the neighbourhood of the base at position 6900, which is an indication of a chimeric sequence (subfigure (a)). The resulting pile-o-gram has a rift around that position, which is detectable without difficulties (subfigure (b)). When the sequence is aligned to the reference genome, the alignment is split into two parts that are dislocated on the reference (subfigure (c)), verifying the assumption about the chimeric nature of the sequence. The figure is tied to sequence *6e5d1642abdc-436ba597-4f68941ca163* obtained from the dataset ERR1147227. Subfigures (a) and (b) were drawn in Python with Matplotlib [44], while subfigure (c) was exported from Gepard [45]. . . . . . . . . . . . . . . . . . . . . . . . . . 23

3.7. Pile-o-gram of a sequence containing a repetitive region. A large amount of overlaps located at the end of the sequence is a good indicator for a reptitive genomic region (subfigure (a)). When the overlaps are transformed into a pile-o-gram, the sequence end has a large ridge over the median (subfigure (b)). If k-mer filtering is employed on the whole dataset, the ridge is lost in the pile-o-gram due to lack of repetitive k-mers (pastel orange line in subfigure (b)). Aligning the sequence to the reference genome gives one primary alignment and several small ones at various positions in the genome (subfigure (c)), confirming that the end of sequence is part of a repetitive region. The whole figure is tied to sequence *6c1f5fec-a9c6-434f-879c-f40bd4dccbb6* obtained from the dataset ERR1147227. Subfigures (a) and (b) were drawn in Python with Matplotlib [44], while subfigure (c) was exported from Gepard [45]. . . . . . . . . . . . . 24

3.8. Overlaps between pile-o-grams that contain reptitive regions. Subfigure (a) shows a false overlap over the repetitive region, which we are trying to remove prior the construction of an assembly graph. The removal is prohibited if there are no overlaps that bridge through the repeat in question. Example of such overlap is shown in subfigure (b). Pile-o-gram drawn in blue is tied to sequence *6c1f5fec-a9c6-434f-879c-f40bd4dccbb6*, while the red and green pile-o-grams are tied to *6a7957d9-47e3-4b27-bc1c-b38f8e349884* and *8db143ed-9007-4661-8174-634f94fb6a3b*, respectively. All sequences were obtained from the dataset ERR1147227. Both subfigures were drawn in Python with Matplotlib [44]. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24

3.9. Pile-o-gram of a incorrectly declared chimeric sequence. Region between 8750 and 11000 has small coverage probably due to low quality (subfigure (b)). It is declared chimeric even though there are a couple overlaps covering it (subfigure (a)). The alignment confirms that the sequence is in fact non-chimeric. It aligns to the reference genome completely and uniquely, although the low quality region is not part of the alignment (subfigure (c)). The whole figure is tied to sequence *2e29dd15-c213-4f79-ac67-fb7e550443c1* obtained from the dataset ERR1147227. Subfigures (a) and (b) were drawn in Python with Matplotlib [44], while subfigure (c) was exported from Gepard [45]. . . . . . . . . . . . . 25

3.10. Magnified view of an assembly graph which shows three extended edges connecting remote parts of the graph. The assembly graph was constructed from the bacterial dataset ERR1046594 and was drawn with Cytoscape [39] (using the prefuse force-directed layout option) prior to applying any simplification methods. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27

3.11. Condensed assembly graph displaying uniform edge lengths when using the force-directed placement algorithm [47]. Edges of interest, those tied to vertices with outdegree greater than two, are coloured in red and are really hard to distinguish. The assembly graph was constructed from the same bacterial dataset ERR1046594, but was drawn in Python with Matplotlib [44] after employing simplification methods. . . . . . . . . . . . . . . . . . . . . . . . . . 28

3.12. Condensed assembly graph with reintroduced transitive edges exhibits proper stretching of false paths drawn with the force-directed placement algorithm [47]. Transitive edges are represented with green dotted lines, while edges of interest, those tied to junction vertices, are coloured in red and are now easier to detect. The assembly graph was constructed from the bacterial dataset ERR1046594 and was drawn in Python with Matplotlib [44]. . . . . . . . . . . 29

3.13. Condensed assembly graph drawn with the force-directed placement algorithm [47] inside a fixed canvas frame. The vertices are pushed to the frame itself and the graph loses the desired shape, although some of the false paths are still elongated. The assembly graph was constructed from the bacterial dataset ERR1046594 and was drawn in Python with Matplotlib [44]. . . . . . . . . . . 31

3.14. Condensed assembly graph drawn with the force-directed placement algorithm [47] after unitig creation. Replacing unambiguous paths of the graph with a single vertex, which is at least 42 vertices away from any junction, retains the sought drawing structure. This will boost the performance of the algorithm as the number of vertices decreased from 2430 to 1099. The assembly graph was constructed from the bacterial dataset ERR1046594 and was drawn in Python with Matplotlib [44]. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33

3.15. Condensed assembly graph drawn with the grid version of the force-directed placement algorithm [47]. Ignoring repulsive force outside a circle of radius $2k$ renders the drawing unusable for detection of false paths. The assembly graph was constructed from the bacterial dataset ERR1046594 and was drawn in Python with Matplotlib [44]. . . . . . . . . . . . . . . . . . . . . . . . . 34

3.16. Condensed assembly graph drawn with Barnes-Hut approximation [49] of the the force-directed placement algorithm [47]. The canvas is recursively split into four quadrants until each vertex is a single occupant of a subquadrant. Repulsive forces are calculated depending on the distance between a given vertex and centers-of-mass of all quadrants, which enables quasilinear execution time. For example, for the vertex marked with a rhombus the number of repulsive forces calculated (denoted with red dashed lines) decreased from 1098 to only 20. The assembly graph was constructed from the bacterial dataset ERR1046594 and was drawn in Python with Matplotlib [44]. Edges of the assembly graph are not drawn for clarity. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35

3.17. Condensed assembly graph of a larger genome drawn with the force-directed placement algorithm [47]. Although the false edges are hard to see, they are present and elongated enough to be removed. The assembly graph was constructed from the eukaryotic dataset SRR6702603 and was drawn in Python with Matplotlib [44]. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38

3.18. Condensed assembly graph with repeat induced edges drawn with the force-directed placement algorithm [47]. Increased number of such edges hinders their proper elongation in the drawing, although some of them are still resolvable. The assembly graph was constructed from the bacterial dataset ERR1046594 and was drawn in Python with Matplotlib [44]. . . . . . . . . . . . . . . . . 39

4.1. Construction of a partial order graph from a set of two arbitrary sequences. Vertices are denoted with $v_i$, while edges with $e_{i,j}$. Dashed lines link characters that were aligned together but are mismatched. In subfigure (a) one sequence of the set is transformed into a linear graph wihtout branches. The other is aligned and afterwards included in the graph. Alignment is shown in subfigure (b) where vertical lines between $c_i$ and $v_i$ denote matches for which nothing has to be done during sequence inclusion. Subfigure (c) shows the resulting partial order graph. It has two new vertices due to mismatches $(c_3, v_3)$ and $(c_6, v_7)$, which are linked together with dashed lines. The graph has also three new edges $e_{2,8}$, $e_{6,9}$ and $e_{8,5}$. Edge $e_{8,5}$ was added due to a deletion between characters $c_3$ and $c_4$. All graphs were manually drawn and annotated in Cytoscape [39]. . . . 41

# List of Tables

5.1. Third generation sequencing data used in evaluation of tools for de novo genome assembly. . . . . 55

5.2. Racon consensus evaluation with two state-of-the-art assemblers Canu and Falcon. Assembly metrics were obtained with Dnadiff [73] across five datasets of varying genome sizes. . . . . 58

5.3. Comparison between assemblers Ra and Raven. Assembly metrics were obtained with Quast-LG [74] on fourteen datasets of varying genome sizes. . . . . 59

5.4. Comparison between Raven and state-of-the-art assemblers Wtdbg2, Flye and Shasta. Assembly metrics were obtained with Quast-LG [74] on five datasets of varying genome sizes. . . . . 60

5.5. Detailed look into execution time of Raven components. All values are measured inside the source code and represent wall clock time in seconds. . . . . 61

# Biography

Robert Vaser was born on the 2nd of May 1991 in Čakovec. In 2010 he enrolled at the University of Zagreb Faculty of Electrical Engineering and Computing. He finished the Computer Science module in 2013, granting him the Bachelor of Science in Computing degree. Title of his thesis was "Evaluation of protein database search tools". The same year he enrolled in the master degree programme, at the same faculty and the same module. In 2015, Robert graduated and obtained the title Master of Science in Computing, magna cum laude. The topic of his thesis was "De novo transcriptome assembly". Since the end of 2015, Robert is enrolled in the Ph.D. programme at University of Zagreb Faculty of Electrical Engineering and Computing. At the same institute, he was employed and worked on the project Algorithms for Genome Sequence Analysis (UIP-11-2013-7353). Since the middle of 2019, he continued his work on the project Advanced Methods and Technologies in Data Science and Cooperative Systems (KK.01.1.1.01.009). In 2016, he was awarded with "Faculty of Electrical Engineering and Computing Science Award for outstanding achievement in research work or innovations in the last two years, especially for outstanding scientific contribution to research in the field of bioinformatics and computer biology". Robert published two research and five conference papers.

## Publications

### Journal papers

1. Vaser, R., Sović, I., Nagarajan, N., Šikić, M., "Fast and accurate de novo genome assembly from long uncorrected reads", Genome Research, Vol. 27, 2017, pp. 737-746, available at: `https://doi.org/10.1101/gr.214270.116`
2. Vaser, R., Adusumalli, S., Leng, S., Šikić, M., Ng, P., "Sift missense predictions for genomes", Nature protocols, Vol. 11, 2015, pp. 1-9, available at: `https://doi.org/10.1038/nprot.2015.123`

## Conference papers

1. Vaser, R., Šikić, M., “Yet another de novo genome assembler”, in 2019 11th International Symposium on Image and Signal Processing and Analysis (ISPA), 2019, pp. 147-151, available at: `https://doi.org/10.1109/ISPA.2019.8868909`
2. Ristov, S., Vaser, R., Šikić, M., “Trade-offs in query and target indexing for the selection of candidates in protein homology searches”, in 2017 Prague Stringology Conference, 2017, pp. 118-125.
3. Vaser, R., Pavlović, D., Šikić, M., “SWORD—a highly efficient protein database search”, Bioinformatics, Vol. 32, 2016, pp. i680-i684, available at: `https://doi.org/10.1093/bioinformatics/btw445`
4. Križanović, K., Marinović, M., Bulović, A., Vaser, R., Šikić, M., “Tgtp-db — a database for extracting genome, transcriptome and proteome data using taxonomy”, in 2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2016, pp. 452-456.
5. Pavlović, D., Vaser, R., Korpar, M., Šikić, M., “Protein database search optimization based on cuda and mpi”, in 2013 36th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2013, pp. 1278-1280.

# Životopis

Robert Vaser rođen je 2. svibnja 1991. godine u Čakovcu. Upisao je Fakultet elektrotehnike i računarstva Sveučilišta u Zagrebu 2010. godine. Završio je modul računarske znanosti 2013. godine i stekao titulu sveučilišnog prvostupnika inženjera računarstva. Naslov obranjenog završnog rada bio je "Evaluacija aplikacija za pretraživanje baze proteinskih sljedova". Iste godine upisao je isti modul na diplomskom studiju dotičnog fakulteta. 2015. godine Robert je diplomirao te stekao titulu magistra inženjera računarstva, s velikom pohvalom. Tematika njegovog diplomskog rada bila je "De novo sastavljanje transkriptoma". Krajem 2015. godine upisao je doktorski studij Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu. Zaposlen je na istom fakultetu radeći na projektu Algoritmi za analizu slijeda genoma (UIP-11-2013-7353). Od sredine 2019. godine nastavio je raditi na projektu Napredne metode i tehnologije u znanosti o podatcima i kooperativnim sustavima (KK.01.1.1.01.009). 2016. godine nagrađen je s "Nagradom za znanost Fakulteta elektrotehnike i računarstva za iznimno postignuće u istraživačkom radu ili inovacijama u prethodne dvije godine, a posebno za izuzetan znanstveni doprinos istraživanjima u području bioinformatike i računalne biologije". Robert je objavio dva znanstvena te pet konferencijskih radova.