University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Ivan Sović

ALGORITHMS FOR *DE NOVO* GENOME ASSEMBLY FROM THIRD GENERATION SEQUENCING DATA

DOCTORAL THESIS

Zagreb, 2016

University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Ivan Sović

ALGORITHMS FOR *DE NOVO* GENOME ASSEMBLY FROM THIRD GENERATION SEQUENCING DATA

DOCTORAL THESIS

Supervisors:
Associate Professor Mile Šikić, PhD
Professor Karolj Skala, PhD

Zagreb, 2016

Sveučilište u Zagrebu

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Ivan Sović

ALGORITMI ZA *DE NOVO* SASTAVLJANJE GENOMA IZ SEKVENCIRANIH PODATAKA TREĆE GENERACIJE

DOKTORSKI RAD

Mentori:
Izv. prof. dr. sc. Mile Šikić
Prof. dr. sc. Karolj Skala

Zagreb, 2016.

The dissertation was made at the University of Zagreb Faculty of Electrical Engineering and Computing, Department of Electronic Systems and Information Processing in co-operation with Ruđer Bošković Institute, Centre for Informatics and Computing and the Genome Institute of Singapore.

Mentori:
Associate Professor Mile Šikić, PhD
Professor Karolj Skala, PhD

The dissertation has: 157 pages

The dissertation number: ___________

About the Supervisors

Mile Šikić was born in Zagreb in 1972. He received B.Sc and M.Sc degrees in electrical engineering from the University of Zagreb, Faculty of Electrical Engineering in Zagreb in 1996, and 2002 respectively. In 2008 he defended his PhD thesis "Computational method for prediction of protein-protein interaction" in computing at the same faculty.

He started his career at University of Zagreb in 1997 as research associate at Faculty of Electrical Engineering and Computing. He worked as teaching assistant in the period from 2005 to 2009, and as assistant professor from 2009-2014. From 2015 he has been working as associate professor. Using his annual leave he joined Bioinformatics Institute, A*STAR Singapore as postdoc in the period from 2011 to 2012. In 2013 he was appointed as adjunct scientist at the same institute.

He has been working as consultant and project manager on tens of industry projects in the ICT field. Also he has been leading several research projects in bioinformatics and computational biology. He was a one of leading investigators on the project „Complex network analysis of cis and trans chromatin interactions” financed by JCO Singapore. In 2014 he received Proof of Concept BICRO grant, Croatian Science Foundation grant and a donation from ADRIS Foundation. In the fields of bioinformatics and complex networks he has authored or co-authored 15 journal papers (CC, SCI, SCI expanded), 8 papers in conference proceedings and one book chapter. His work has been cited over 400 times (h-index = 8). He teaches Software Engineering, Algorithms and data structures and bioinformatics. At University of Zagreb, he created and co-created new courses Bioinformatics and Creative Laboratory. He has supervised over 50 undergraduate and master thesis, 2 master dissertations, one phd thesis, and currently is supervising seven PhD students.

Assoc. Prof. Šikić is a member of IEEE and ISCN. He participated in 2 conference international program committees and he servers as technical reviewer for various international journals. He is a member of steering committee of Science and Society Synergy Institute - ISZD.

Mile Šikić rođen je u Zagrebu 1972. Diplomirao je i magistrirao u polju elektrotehnike na Sveučilištu u Zagrebu Fakultetu elektrotehnike i računarstva (FER), 1996, odnosno 2002. Godine. Doktorsku disertacijom u polju računarstva pod naslovom „Računalna metoda za predviđanje mjesta proteinskih interakcija“ obranio je 2008. godine na istom fakultetu.

Od 1997 do 2005 je radio kao zavodski suradnik na Zavodu za elektroničke sustave i obradbu informacija, Fakulteta elektrotehnike i računarstva, Sveučilišta u Zagrebu, od 2005 do 2009 kao asistent, 2009 izabran je u zvanje docenta, a 2015 u izvanrednoga profesora. Od 1.5.2011 – 1.9.2002 kao poslijedoktorand je radio na Bioinformatics Institut, A*STAR u Singapuru, a od 2013 na istom institutu djeluje kao pridruženi znanstvenik.

Sudjelovao je i vodio nekoliko desetaka projekata s industrijom u području ICT-a te nekoliko istraživačkih projekata primarno u području bioinformatike i računalne biologije. Kao jedan od vodećih istraživača aktivno je sudjelovao na projektu „Complex network analysis of cis and trans chromatin interactions" financiranim od strane JCO Singapur. Voditelj je HRZZ uspostavnoga projekta te Proof of Concept BICRO projekta. Projekti koje je vodio su dobili donacije Zaklade Adris i Zaklade HAZU. Do sada je, većinom u području bioinformatike i računalne biologije, objavio petnaest znanstvenih radova (CC, SCI, SCI expanded), jedno poglavlje u knjizi i osam radova na konferencijama s međunarodnom recenzijom. Njegovi radovi su do sada citirani preko 400 puta (h-index = 8). Predaje na prediplomskoj i diplomskoj razini predmete vezane uz programiranje, algoritme i strukture podataka, bioinformatiku i kreativnost. U nastavu je na diplomskoj razini uveo predmete Bioinformatika te Kreativni laboratorij koji je pokrenuo zajedno s još 5 kolega s različitih sastavnica Sveučilišta u Zagrebu. Vodio je više od 70 završnih i diplomskih radova, dva magistarska rada, jedan doktorat, a trenutno je mentor sedmero doktorskih studenata.

Izv. Prof. Šikić je član stručnih udruga IEEE i ISCB. Sudjeluje u 2 međunarodna programska odbora znanstvenih konferencija te sudjeluje kao recenzent u većem broju inozemnih časopisa. Član je upravnoga vijeća Instituta sinergije znanosti i društva.

Karolj Skala was born in 1951 in Subotica. He received B.Sc., M.Sc. and Ph.D. degrees in the field of electrical engineering at the University of Zagreb Faculty of Electrical Engineering (ETF), 1974, 1979 or 1983. From November 1974 he is working at Ruđer Bošković Institute in the department of Laser research and development, since from 1991 is on head of the Centre for Informatics and Computing as a consultant in permanent title from 2008. From 1996 to 2012 he is a teacher at the Faculty of Graphic Arts in the position of full professor. Since 2013, a professor in 'permanent title at the University of Osijek. He was lead of four research projects of the Ministry of Science, Education and Sports of the Republic of Croatia. He participated in five Framework Programme of the European Union EU FP6 and seven EU FP7 projects. Now providing 3 EU H2020 projects: *EGI Engange*, Engaging the EGI Community towards an Open Science Commons Horizon 2020, 2015-2018; *INDIGO Data Cloud* INtegrating Distributed data Infrastructures for Global ExplOitation, Horizon 2020, 2015-2018; *SESAME Net*, Supercomputing Expertise for Small And Medium Enterprises Horizon 2020, 2015-2017 He is a national representative of COST IC 1305, Network for Sustainable Ultrascale Computing (NESUS). Further he was the national representative of the European Commission at Research Infrastructure Programme Committee EC FP7 (2010-2013) and he currently represent Croatia in EC at European Research Infrastructure Consortium (ERIC). He has published more than 90 scientific papers in the fields of; laser communications, scientific visualization and distributed computing systems.

Prof. Skala is programme committee member of MIPRO association and regular member of Croatian Academy of Engineering (HATZ). Chair of the Annual Conference; Distributed Computing and Visualisation Systems and participates as a reviewer in a number of international projects and scinetific journals. He has won medals from Hungarian Academy of Sciences in 2010.

Karolj Skala rođen je 1951. u Subotici. Diplomirao je, magistrirao i doktorirao u polju elektrotehnike na Sveučilištu u Zagrebu, Fakultetu elektrotehnike (ETF), 1974., 1979. odnosno 1983. godine. Od studenog 1974. godine radi na Institutu Ruđer Bošković u odjelu za Laserska i atomska istraživanja i razvoj, a od 1991 je voditelj Centra za informatiku i računartvo kao savjetnik u trajnome zvanju od 2008. Od 1996 do 2012. je nastavnik na Grafičkom fakultetu u Zagrebu u naslovnom zvanju redovitog profesora. Od 2013. je profesor u trajnome zvanju Sveučilišta u Osijeku. Bio je voditelj četiri znanstvena projekta Ministarstva znanosti, obrazovanja i sporta Republike Hrvatske. Sudjelovao je u okvirnim programima Europske unije, pet EU FP6 i sedam EU FP7 projekata. Trenutno je konzorcijski nacionalni partner koordinator tri EU H2020 projekta: *EGI Engange*, Engaging the EGI Community towards an Open Science Commons Horizon 2020, 2015-2018; *INDIGO Data Cloud* INtegrating Distributed data Infrastructures for Global ExplOitation, Horizon 2020, 2015-2018; *SESAME Net*, Supercomputing

Expertise for Small And Medium Enterprises Horizon 2020, 2015-2017.

Sudjeluje u COST IC 1305 - Network for Sustainable Ultrascale Computing (NESUS). Bio je nacionalni predstavnik u Europskoj komisiji; Research Infrastructure Programme Committee EC FP7 (2010-2013) i trenutno obnaša funkciju predstavnika u; European Research Infrastructure Consortium (ERIC). Objavio je više od 90 znastvenih radova iz područja laserskih komunikacija, znanstvene vizualizacije i distribuiranih računalnih sustava. Prof. Skala je član programskog odbora udruge MIPRO i redoviti član Akademije tehničkih znanosti Hrvatske (HATZ). Voditelj je godišnje konferencije Distributed Computing and Visualisation Systems te sudjeluje kao recenzent u većem broju inozemnih projekata i časopisa. Dobitnik je medalje Mađarske Akademije znanosti 2010. godine.

Acknowledgements

Looking back through the past years, I cannot imagine that any of this would be possible without the immense love, support, patience and encouragement, selflessly given to me by my better half - Iva. You were always there for me, and you've always been that special someone to me who made me smile and got me cheery when times were getting hard. I never took any of this for granted, and never will.

To my mom and dad, Ljiljana and Ivan, who always supported my decisions and backed me up every step of the way. You thought me to first look before I leap, to set priorities, and to look beyond limits, pushing far.

To my sister Ana who was always there when I pushed too far. I always could count on you, in any situation, which you've proven numerous times. You also thought me so much - everything from coding in Logo to signal processing.

To the rest of my family, thank you for everything!

Finally, to the people who guided me for the past six years - my supervisors: Karolj Skala from whom I learned an incredible amount about projects, priorities and how to take a stand; Niranjan Nagarajan who welcomed me into his group, enabled the major part of this research and thought me everything I know about writing journal papers; and especially Mile Šikić.

Mile, you steered me, educated me and pushed me whenever I got stuck. You thought me about critical thinking, and how to objectively filter ideas - a skill which I desperately needed. You were always open to discussions without which none of these projects would ever get realized. Thank you for introducing me to this incredible field. I consider you not only as my mentor, but a very good friend as well.

I want to thank Tomislav Lipić and Bono Lučić for endless constructive discussions throughout the years, and Krešimir Križanović, Robert Vaser and Andreas Wilm for helping me solve many of these interesting puzzles along the way.

In the end, I would like to thank all my colleagues and friends who stuck by me, even when I got so obsessed with my work.

Once again, here I thank you all from the bottom of my heart. I truly appreciate everything you have done for me. Without you, this would not have been possible.

Abstract

During the past ten years, genome sequencing has been an extremely hot and active topic, with an especial momentum happening right now. New, exciting and more affordable technologies have been released, requiring the rapid development of new algorithmic methods to cope with the data. Affordable commercial availability of the sequencing technology and algorithmic methods which can leverage the data could open doors to a vast number of very important applications, such as diagnosis and treatment of chronic diseases through personalized medicine or identification of pathogenic microorganisms from soil, water, food or tissue samples.

Sequencing the entire genome of an organism is a difficult problem, because all sequencing technologies to date have limitations on the length of the molecule that they can read (much smaller than the genomes of a vast majority of organisms). In order to obtain the sequence of an entire genome, reads need to be either stitched together (*assembled*) in a *de novo* fashion when the genome of the organism is unknown in advance, or mapped and aligned to the reference genome if one exists (reference assembly or mapping). The main problem in both approaches stems from the repeating regions in the genomes which, if longer than the reads, prevent complete assembly of the genome. The need for technologies that would produce longer reads which could solve the problem of repeating regions has resulted in the advent of new sequencing approaches – the so-called *third generation sequencing technologies* which currently include two representatives: Pacific Biosciences (PacBio) and Oxford Nanopore. Both technologies are characterized, aside from long reads, by high error rates which existing assembly algorithms of the time were not capable of handling. This caused the development of time-consuming read error correction methods which were applied as a pre-processing step prior to assembly.

Instead, the focus of the work conducted in the scope of this thesis is to develop novel methods for *de novo* DNA assembly from third generation sequencing data, which provide enough sensitivity and precision to completely omit the error-correction phase. Strong focus is put on nanopore data.

In the scope of this thesis, four new methods were developed: (I) NanoMark - an evaluation framework for comparison of assembly methods from nanopore sequencing data; (II) GraphMap - a fast and sensitive mapper for long error-prone reads; (III) Owler - a sensitive overlapper for third generation sequencing; and (IV) Racon - a rapid consensus module for correcting raw assemblies. Owler and Racon were used as modules in the development of a novel *de novo* genome assembler Aracon. The results show that Aracon reduces the overall assembly time by at least 3*x* and up to even an order of magnitude less compared to the state-of-the-art methods, while retaining comparable or better quality of assembly.

Keywords: *de novo*, assembly, PacBio, nanopore, NanoMark, GraphMap, Racon, Aracon

Produženi sažetak

Algoritmi za *de novo* sastavljanje genoma iz sekvenciranih podataka treće generacije

Tijekom proteklih desetak godina, sekvenciranje genoma postalo je iznimno aktivno i zanimljivo područje, a pravi zamah događa se upravo sada. Nedavno su se počele pojavljivati nove, uzbudljive i pristupačne tehnologije, koje povlače i potrebu za razvojem novih algoritamskih metoda koje će se moći uhvatiti u koštac s količinom i kvalitetom podataka koje one generiraju. Komercijalna isplativost i dostupnost tehnologije za sekvenciranje, kao i pratećih algoritamskih rješenja kojima bi se maksimizirao potencijal ovih tehnologija, mogao bi otvoriti vrata širokom spektru važnih primjena: od dijagnoze i tretmana kroničnih bolesti kroz personaliziranu medicinu, preko identifikacije patogenih mikroorganizama iz uzoraka tla, vode ili tkiva pa do istraživačkih projekata ciljanih otkrivanju znanja biološke pozadine živog svijeta oko nas. Za uspješnu realizaciju svih ovih primjena ključne su računalne metode za analizu i procesiranje prikupljenih podataka.

Sekvenciranje cijelog genoma nekog organizma predstavlja vrlo složen problem jer sva postojeća tehnologija za sekvenciranje sadrži jedno važno ograničenje - najveću duljinu molekule koju uređaji mogu pročitati. Duljine očitanih sekvenci (očitanja) puno su kraće od duljine genoma velike većine organizama. Kako bi se uspješno mogla dobiti cjelovita sekvenca nekog genoma, očitanja je potrebno međusobno povezati (sastaviti) na *de novo* način koji se primjerice koristi u slučaju kada genom promatranog organizma već nije poznat unaprijed. Drugi pristup temelji se na mapiranju i poravnanju očitanja s referentnim genomom promatranog organizma (prethodno visoko-kvalitetno sastavljeni genom organizma) u slučaju ako referentni genom već postoji (sastavljanje uz referencu ili mapiranje).

Glavna pretpostavka svih algoritama za sastavljanje genoma je da vrlo slični fragmenti DNA dolaze s iste pozicije unutar genoma. Ista ta sličnost predstavlja temelj za: (I) otkrivanje preklapanja između pojedinih očitanja te njihovo povezivanje u duže, slijedne sekvence (kontige, eng. *contigs*) u slučaju *de novo* sastavljanja, ili (II) za određivanje pozicije (eng. *mapping*) na referentnom genomu s kojeg je neko očitanje uzorkovano, te precizno poravnanje očitanja (eng. *alignment*) s referentnim genomom. U oba slučaja, osnovni problem javlja se u ponavljajućim regijama u genomu, koje, ako su duže od samih očitanja, onemogućuju jednoznačnu i potpunu rekonstrukciju genoma. Na primjer, u idealnom slučaju, *de novo* sastavljanje rezultiralo bi skupom kontiga od kojih svaki potpuno rekonstruira jedan od kromosoma analiziranog organizma, dok u praksi ponavljanja u genomu uzrokuju nejednoznačnosti u postupku sastavljanja. Pri tome, rezultat sastavljanja obično je skup fragmentiranih kontiga nepoznate orijentacije, odvojenih prazninama nepoznate veličine.

Potreba za tehnologijama koje bi mogle prevladati problem ponavljajućih regija rezultirala je pojavom posljednje, *treće generacije* uređaja za sekvenciranje. Treća generacija trenutno uključuje samo dva reprezentativna proizvođača uređaja: Pacific Biosciences i Oxford Nanopore Technologies. Prvi predstavnik ove generacije bili su uređaji proizvođača Pacific Biosciences (PacBio). Iako ovi uređaji generiraju znatno dulja očitanja (nekoliko desetaka tisuća baza) u odnosu na uređaje druge generacije (≈100-400 baza), razina pogreške u očitanjima znatno je veća od druge generacije sekvencera (≈10-15% u usporedbi s ≈1%). U trenutku kada se ova tehnologija pojavila, postojeće metode za sastavljanje i poravnanje nisu bile u stanju iskoristiti potencijal ovih podataka zbog vrlo visoke razine pogreške. To je uzrokovalo razvoj novih metoda za hibridno i ne-hibridno ispravljanje grešaka u podatcima i njihovo sastavljanje (HGAP, PBcR), osjetljivo poravnanje i preklapanje očitanja (BLASR), kao i prilagodbu postojećih metoda poravnanja kako bi mogle baratati ovim podatcima (BWA-MEM).

Oxford Nanopore Technologies (ONT) drugi je i trenutno posljednji komercijalni proizvođač uređaja za sekvenciranje treće generacije. ONT MinION je malen, prenosiv, USB-pogonjen i cjenovno prihvatljiv uređaj za sekvenciranje koji je u stanju proizvesti vrlo dugačka očitanja (i do nekoliko stotina tisuća baza). Ove karakteristike pružaju mu transformativni potencijal u istraživanjima, dijagnozi, te u primjenama s ograničenim resursima. No, MinION ima bitan ograničavajući nedostatak - vrlo veliku pogrešku u očitanjima (≈10-40%). Specifičnosti MinION tehnologije, kao i postupci pripreme uzoraka, omogućuju dvije vrste sekvenciranja pomoću ovog uređaja: jednosmjerno (eng. *one-directional*, 1D) pri čemu se sekvencira samo jedan lanac određenog DNA fragmenta, ili dvosmjerno (eng. *two-directional*, 2D) pri čemu su oba lanca nekog DNA fragmenta kemijski povezana i zajedno slijedno sekvencirana. Redundancija kod 2D očitanja iskorištava se za postizanje veće točnosti očitanja (≈80-88% točnosti kod R7.3 kemije) u odnosu na ≈75% točnosti kod 1D očitanja. Ovakve razine i profil pogrešaka stvorili su potrebu za razvojem još osjetljivijih algoritama za mapiranje, poravnanje i sastavljanje očitanja koji bi mogli otvoriti potpuni potencijal ove tehnologije.

U trenutku izrade rada opisanog u sklopu ove disertacije, sve metode sastavljanja genoma koje su bile primjenjive za sekvencirane podatke treće generacije ovisile su o fazi ispravljanja pogreške u podatcima kao pret-koraku sastavljanja. Ispravljanjem pogreške omogućilo se iskorištavanje veće količine ulaznih podataka, čime se kompenzirala potencijalno niska osjetljivost sljedeće faze. Proces ispravljana pogreške temelji se na mapiranju i poravnanju svih očitanja u ulaznom skupu podataka na isti taj skup, te primjenom statističkih metoda kako bi se odredio konsenzus i razrješile pogreške. Iako je ovaj pristup temeljito ispitan i omogućuje dobivanje dobrih rezultata, njegov glavni problem je vrlo velika vremenska zahtjevnost što posebice dolazi do izražaja kod sastavljanja većih genoma.

Umjesto takvog pristupa, fokus rada provedenog u sklopu izrade ove disertacije je na razvoju novih metoda i algoritama za *de novo* sastavljanje DNA iz sekvenciranih podataka treće generacije, koji će omogućiti dovoljnu osjetljivost i preciznost kako bi se potpuno preskočio korak ispravljanja pogrešaka. Cilj je pokazati kako na ovaj način možemo znatno reducirati vrijeme izvođenja u usporedbi s najboljim postojećim metodama, dok pri tome zadržavamo istu ili višu razinu točnosti.

Doprinosi ove disertacije uključuju sljedeće:

1. Radni okvir za usporedbu alata za *de novo* sastavljanje genoma definiranjem normiranih kvalitativnih testova,
2. Optimirani algoritmi i strukture podataka za *de novo* sastavljanje genoma s naglaskom na očitanjima treće generacije, i
3. Ocjena razvijenih algoritama korištenjem novog radnog okvira usporedbe.

Kako bi se postigli navedeni ciljevi, u okviru ove disertacije, razvijene su četiri nove metode:

- NanoMark - sustav za automatiziranu usporedbu i evaluaciju metoda za sastavljanje ONT podataka
- GraphMap - brza i osjetljiva metoda za mapiranje dugačkih i greškovitih očitanja
- Owler - osjetljiva metoda za preklapanje očitanja dobivenih sekvenciranjem treće generacije
- Racon - iznimno brza konsenzus metoda za ispravljanje greškovitih dugačkih sekvenci nakon što je proveden postupak sastavljanja

Owler i Racon korišteni su kao osnovni moduli za razvoj nove metode za *de novo* sastavljanje genoma nazvane *Aracon*. Rezultati testiranja pokazuju kako je Aracon smanjio sveukupno potrebno vrijeme sastavljanja genoma za faktor $1000x$ u odnosu na najtočniju metodu za ONT podatke, te između $3x - 10x$ u usporedbi s ostalim trenutno vodećim dostupnim metodama, pri čemu mu je kvaliteta sastavljenih genoma jednaka ili bolja od istih metoda.

Poglavlje 1 ("Uvod") disertacije nastoji objasniti motivaciju i potrebu za rješavanjem problema u bioinformatici, a naročito vezanih uz sastavljanje i analizu genoma organizama. U uvodu su navedena dva specifična cilja istraživanja: (I) metoda za mapiranje/poravnanje sekvenciranih podataka treće generacije s posebnim fokusom na očitanja dobivena uređajima tvrtke Oxford Nanopore Technologies, i (II) metodu za *de novo* sastavljanje genoma sekvenciranim podatcima treće generacije s posebnim fokusom na očitanja dobivena uređajima tvrtke Oxford Nanopore Technologies.

U poglavlju 2 ove disertacije ("Teorijska podloga") predstavljeni su teorijski koncepti sastavljanja genoma. Prezentirana je nužna terminologija, te je dan detaljan opis postojećih pristupa u mapiranju i poravnanju sekvenci s referentnim genomom, kao i detaljan opis pristupa za *de novo* sastavljanje genoma. Predstavljeni koncepti i pristupi nisu fokusirani samo na sekvencirane podatke treće generacije već na cjelokupno područje kako bi se omogućio uvid u ključne ideje koje su bile potrebne za realizaciju rada u sklopu ove disertacije. Na kraju svakog od potpoglavlja dan je osvrt na trenutno stanje fokusirano na podatke dobivene ONT tehnologijom kako bi se identificirali nedostatci i pobliže objasnila potreba za metodama razvijenim u ovome radu. Detaljno su opisani koraci za sastavljanje genoma koristeći paradigmu *Preklapanje-Razmještaj-Konsenzus* (eng. *Overlap-Layout-Consensus*) kao prevladavajuće paradigme za sekvencirane podatke treće generacije, koja je na temelju razmatranja u ovom poglavlju odabrana i kao paradigma koja će biti primijenjena u ovome radu. Zaključeno je kako trenutno stanje oba potpodručja (mapiranja/poravnanja i *de novo* sastavljanja) sadrži veliki potencijal za napredak koji se može ostvariti novim algoritmima i metodama.

Poglavlje 3 ("Evaluacija hibridnih i ne-hibridnih metoda za *de novo* sastavljanje genoma koristeći nanopore očitanja") daje detaljnu usporedbu postojećih metoda (eng. *state-of-the-art*) primjenjivih za sastavljanje genoma iz sekvenciranih podataka treće generacije. Provedena je sistematična evaluacija ovih metoda, a fokus je stavljen na procjenu kvalitete metoda za sastavljanje ONT podataka (LQS, PBcR, Canu, Miniasm), dok je dodatno proveden i uspješan pokušaj prilagodbe jedne metode koja je dizajnirana specifično za PacBio tehnologiju (Falcon). Osim navedenih ne-hibridnih metoda, evaluirane su i dvije hibridne metode: SPAdes i ALLPATHS-LG, koje koriste sekvencirane podatke druge generacije (Illumina) za početno sastavljanje te naknadno podatke treće generacije za povezivanje kontiga u dulje cjeline. Sva ispitivanja provedena su nad šest skupova podataka: jedan Illumina skup namijenjen samo za hibridne metode, i pet ONT skupova podataka različite veličine i kvalitete. U svrhu jednostavnije evaluacije i nadogradnje rezultata testiranja, razvijen je radni okvir (eng. *framework*) za uniformno i automatizirano pokretanje testova i formatiranje dobivenih rezultata - *NanoMark*. Rezultati su iscrpno prezentirani kako bi se omogućio izravan uvid u podobnost pojedine metode za određeni skup podataka. Sve testirane metode pokazale su se uspješnim u sastavljanju cijelog bakterijskog genoma (*E. coli* K-12) pod pravim uvjetima, no u većini slučajeva najtočnije rezultate (od ne-hibridnih metoda) producirala je LQS metoda koja je ujedno i vremenski najzahtjevnija, dok je s druge strane Miniasm iznimno brz i štedljiv na resursima, ali zbog nedostatka konsenzus faze njegovi rezultati nisu direktno sumjerljivi s drugim metodama. Od hibridnih metoda, ALLPATHS-LG je producirao najbolje rezultate, rezultirajući i s neusporedivo manje pogrešaka u odnosu na bilo koju od testiranih ne-hibridnih metoda.

Poglavlje 4 ("Preklapanje") predstavlja razvoj nove, vrlo osjetljive metode *GraphMap* za brzo mapiranje očitanja na referentni genom i njezinu prilagodbu u novu metodu nazvanu *Owler* za preklapanje očitanja u svrhu *de novo* sastavljanja genoma. GraphMap algoritam strukturiran je u slijed od pet koraka koji omogućuju postizanje visoke osjetljivosti i brzine u odnosu na postojeće metode kroz konzervativno smanjivanje broja kandidatnih lokacija za mapiranje u svakom od koraka. Ovi koraci uključuju: (I) novu adaptaciju zrnatog pretraživanja uz dozvoljene pogreške na određenim mjestima u zrnu (eng. *gapped spaced seeds*). Pronađena zrna tada se grupiraju pomoću Houghove transformacije u grube kandidatne regije na referentnom genomu. U koraku (II) svaka od ovih regija zasebno se procesira novom metodom mapiranja na graf koristeći efikasni čvorno-centrični pristup. Rezultat ovog koraka dan je u obliku niza *uporišta* (eng. *anchors*) za poravnanje. S obzirom na repetitivnost u genomima i pogreške u sekvenciranju, uporišta se zatim u koraku (III) filtriraju korištenjem *LCSk* algoritma (eng. *Longest Common Subsequence in k-length Substrings*). Dodatno rafiniranje uporišta svake regije postiže se u koraku (IV) primjenom L_1 linearne regresije ili metodom ulančavanja uporišta (eng. chaining). U posljednjem koraku (V), GraphMap odabire jednu ili više najboljih regija te provodi proces konačnog poravnanja sekvenci. Za svako poravnanje GraphMap prijavljuje i odgovarajuću aproksimaciju E-vrijednosti (eng. *E-value*) te kvalitetu mapiranja (eng. *mapping quality*). GraphMap je detaljno testiran, evaluiran i uspoređen s postojećim metodama na simuliranim i stvarnim podatcima. Na simuliranim podatcima, GraphMap je jedina metoda koja je rezultirala osjetljivošću sličnoj BLAST-u, pri čemu je i do nekoliko redova veličine brža od BLAST-a. Na stvarnim MinION podatcima, GraphMap je na svim skupovima podataka nenadmašen u osjetljivosti u odnosu na ostale metode (BWA-MEM, LAST, BLASR, DALIGNER), te mapira između $10-80\%$ sekvenciranih baza više od ostalih metoda. GraphMap omogućuje mapiranje koje je uglavnom agnostično na parametre za razliku od ostalih metoda, te uz pred-zadane vrijednosti parametara postiže konstantno visoku preciznost na svim skupovima podataka i to tipično veću od 98%. Nadalje, GraphMap je ispitan koristeći stvarne podatke u kontekstu nekoliko različitih važnih primjena: (a) otkrivanje promjena u medicinski značajnim regijama ljudskog genoma, (b) određivanje strukturnih promjena izravno iz mapiranih podataka, i (c) izravna identifikacija vrsta mapiranjem očitanja na bazu referentnih genoma. U svim ovim primjenama, GraphMap ostvaruje rezultate znatno bolje od postojećih metoda.

Problem preklapanja očitanja može se promatrati kao specijalni slučaj problema mapiranja na referencu, pri čemu se ulazni skup očitanja umjesto na referentni genom mapira sam na sebe. GraphMap metoda preuređena je i prilagođena problemu otkrivanja preklapanja te implementirana kao modul nazvan *Owler* (eng. *Overlap with long erroneous reads*). Owler način rada koristi reducirani GraphMap algoritam u kojem se preskaču neki vremenski zahtjevni koraci. Konkretno, Owler algoritam sastoji se od sljedećih koraka: (I) konstrukcija indeksa iz skupa očitanja za jedan oblik zrna s mogućnošću pogreške, (II) za svako očitanje iz skupa dohvaćaju se i sortiraju sve odgovarajuće lokacije zrna iz indeksa, (III) provođenje LCSk postupka izravno nad dohvaćenim lokacijama zrna, (IV) ulančavanje zrna kao dodatna metoda filtriranja čime se izgrađuje prvi skup preklapanja, i (V) grubo filtriranje potencijalno loših preklapanja (na temelju duljine preklapanja, početnih i završnih lokacija). Owler je uspoređen s postojećim metodama za preklapanje očitanja treće generacije (Minimap, MHAP, DALIGNER i BLASR) na stvarnim skupovima podataka (istim skupovima koji su korišteni u Poglavlju 3 za evaluaciju metoda sastavljanja). U svim testovima, Owler konzistentno zadržava visoke razine preciznosti i odziva u odnosu na ostale metode kod kojih je vidljiva velika varijabilnost. Iznimka je Minimap koji također demonstrira visoku preciznost na svim skupovima podataka, dok mu odziv znatno opada pri vrlo velikoj pokrivenosti genoma, što nije slučaj kod Owler metode.

Poglavlje 5 ("Racon - Brzi modul za konsenzus za grubo sastavljanje genoma koristeći dugačka nekorigirana očitanja") opisuje novi algoritam *Racon* (eng. *Rapid Consensus*) koji se može koristiti kao završni korak u Preklapanje-Razmještaj-Konsenzus pristupu za *de novo* sastavljanje genoma. U sklopu analize provedene u Poglavlju 2, zaključeno je kako u trenutnom stanju ovog područja nedostaje samostojeća metoda koja će omogućiti brzo i kvalitetno dobivanje konsenzusnih sekvenci iz grubo sastavljenih genoma, dobivenih primjerice Miniasm metodom. Postojeće metode za konsenzus podataka treće generacije namijenjene su uglavnom ili za ispravljanje pogrešaka u ulaznim podatcima, ili za fino poliranje već sastavljenih genoma koji su prošli konsenzus fazu (kao integriranu fazu postojećih metoda sastavljanja). Racon je temeljen na brzoj implementaciji pristupa baziranog na POA grafovima (eng. *Partial Order Alignment*) pri čemu je glavnina ubrzanja postignuta SIMD (eng. *Single Instruction Multiple Data*) akceleracijom poravnanja sekvenci na graf. Cijeli Racon algoritam sastoji se od nekoliko koraka: (I) ulazni skup očitanja poravna se sa skupom grubih kontiga, (II) svaki kontig procesira se zasebno i to tako da se podijeli na slijedne, nepreklapajuće prozore predodređene duljine. Svaki prozor procesira se zasebno, pri čemu se (III) iz sekvence kontiga izgradi POA graf na koji se zatim SIMD-akceleriranim pristupom poravna odgovarajući dio svih očitanja koja pokrivaju trenutno analizirani prozor. (IV) Za svaki prozor odredi se konsenzus sekvenca traženjem najtežeg puta kroz graf, te se (V) konsenzusi pojedinih prozora *zalijepe* u konačnu sekvencu ispravljenog kontiga. Racon je ispitan tako da je prvo uklopljen kao dodatna konsenzus faza nakon Minimap+Miniasm metode za sastavljanje genoma, te je ukupni rezultat evaluiran na stvarnim ONT i PacBio skupovima podataka različitih veličina i kvaliteta i uspoređen s drugim aktualnim metodama (Canu, Falcon i LQS). Miniasm+Racon kombinacija u gotovo svim slučajima rezultira jednakom ili boljom kvalitetom sastavljenih genoma u usporedbi s ostalim metodama, pri čemu je za čak oko red veličine brža od ostalih metoda. Također, u gotovo svim ispitanim slučajima, Miniasm+Racon rezultira ispravljenim kontizima koji su po duljini sličniji referentnom genomu od ostalih metoda. Generalni dizajn Racona omogućuje mu da osim za ispravljanje pogrešaka u kontizima bude primjenjiv i za ispravljenje pogrešaka u ulaznim očitanjima.

U poglavlju 6 ("Integracija i evaluacija - Aracon alat za sastavljanje genoma") opisana je integracija novorazvijenih metoda, predstavljenih u prethodnim poglavljima, u novi alat za *de novo* sastavljanje genoma iz sekvenciranih podataka treće generacije - *Aracon* (eng. *Assembly with Rapid Consensus*). Aracon je implementiran kao skripta koja povezuje: Owler za određivanje preklapanja iz ulaznog skupa očitanja, Miniasmov modul za određivanje razmještaja, te dvije iteracije Racona za produciranje konačnih kontiga s ispravljenom pogreškom. Miniasm modul korišten je za određivanje razmještaja jer je razvoj novog modula za ovu fazu bio izvan okvira ove disertacije, a teorija iza string-graf pristupa već dobro utemeljena i poznata. Aracon

je evaluiran koristeći istu metodologiju koja je razvijena u sklopu Poglavlja 3 (NanoMark), te je uspoređen s ostalim ne-hibridnim metodama za sastavljanje genoma iz ONT podataka: LQS, PBcR, Canu, Miniasm (bez konsenzus koraka) i Falcon. Aracon je u svim slučajima producirao kontige s točnošću gotovo jednakoj kao LQS (najtočnija metoda), pri čemu je bio čak *tri reda veličine* brži od LQS-a. U usporedbi s ostalim metodama, kontizi producirani Araconom uvijek su sličniji u odnosu na referentni genom, pri čemu je Aracon barem *tri puta* brži od sljedeće najbrže metode.

Uzevši sve navedeno u obzir, dobivenim rezultatima uspješno je potvrđena hipoteza da je moguće ostvariti visoko-kvalitetno *de novo* sastavljanje genoma bez prethodnog koraka ispravljanja pogreške u ulaznim podatcima, pri čemu se postiže znatno ubrzanje cijelog procesa. Važnost istraživanja provedenih u sklopu ove disertacije također je prepoznata i od strane znanstvene zajednice, te su dijelovi ovog istraživanja objavljeni u visoko-rangiranim znanstvenim časopisima (Nature Communications i Oxford Bioinformatics). NanoMark, GraphMap, Owler, Racon i Aracon otvoreno su dostupni pod MIT licencom na: `https://github.com/kkrizanovic/NanoMark`, `https://github.com/isovic/graphmap`, `https://github.com/isovic/racon`, `https://github.com/isovic/aracon`.

Ključne riječi: *de novo*, sastavljanje, PacBio, nanopore, NanoMark, GraphMap, Racon, Aracon

Contents

List of Figures

List of Tables

Chapter 1

Introduction

An extremely important research subject in biology is the determination of the sequence of naturally occurring *deoxyribonucleic acid* (DNA) molecules. The DNA is a long molecule consisting of a large number of simple components called *nucleotides* or *bases*, that comprise a long chain (sequence). DNA sequencing is the process of determining this sequence of nucleotides.

During the past ten years, sequencing has been an extremely hot and active topic, with an especial momentum of upswing happening right now. In the past year, new, exciting and more affordable technologies have been released, allowing large amounts of data to be acquired, but also requiring the rapid development of new algorithmic methods to cope with the data. Affordable commercial availability of the sequencing technology is the most important driving factor of an upcoming revolution in many aspects of our everyday lives. This includes the rapid and accurate detection and identification of pathogenic microorganisms from soil, water, food or tissue samples [1][2], diagnosis and treatment of chronic diseases through personalized medicine, or simply the collection of information about genomic make-up of a large number of individuals of some species in order to generate a better understanding of the variations in certain populations, and overall to generate a better and more accurate image of the known, or better yet, the unknown living world.

For all of these applications of DNA sequencing to be possible, the key components are the computational methods for processing and analysis of the acquired data. Modern DNA sequencing instruments produce read sequences (reads) which are considerably shorter than the length of the originating genomes, or of the studied genomic features [3]. In order to obtain the sequence of an entire genome, reads need to be either stitched together (*assembled*) in a *de novo* fashion when the genome of the organism is unknown in advance, or mapped and aligned to the reference genome if one exists (*reference assembly* or *mapping*). For this reason, mapping/alignment and assembly are the first and the most important steps in any genomics project.

To date, three generations of sequencing instruments exist: first (longer reads, lower error rate) was based on Sanger sequencing method; second, also called *next generation*, (short reads, low to medium error rates) based on several different technologies most of which are no longer commercially available (e.g. Roche 454, Illumina Solexa, ABi SOLiD, Ion Proton); and third (very long reads, high error rates) currently in development (e.g. Pacific Biosciences, Oxford Nanopore Technologies) [3][4][5].

Sanger sequencing was the first successful method for sequencing the DNA, developed by Frederic Sanger in 1977, and was the most widely used sequencing technique for over 25 years. The approach is based on the modification of the process of DNA synthesis. During this process, the two chains of DNA are separated, followed by the addition of nucleotides that are complements to those contained within the chains. Sanger's method contains modified nucleotides in addition to the normal ones, where each modified nucleotide is missing an oxygen on the $3'$ end, and has also been fluorescently marked [6]. Integrating these nucleotides into a DNA chain causes a halt (or termination) in the elongation process. Using capillary electrophoresis, sequences are separated by their length (mass) and the termination base is read. This process of sequencing can deliver read lengths up to ≈ 2000 bases, high raw accuracy, and allow for 384 samples to be sequenced in parallel, generating 24 bases per instrument second [6]. Great shortcomings of this method are its high price of about \$10 per 10000 bases, and long sequencing time.

In the last ten years Next Generation Sequencing (NGS) devices have dominated genome sequencing market. In contrast to previously used Sanger sequencing, NGS is much cheaper, less time consuming and not so labor intensive. Yet, when it comes to *de novo* assembly of longer genomes many researchers are being skeptical of using NGS reads. These devices produce reads a few hundred base pairs long, which is too short to unambiguously resolve repetitive regions even within relatively small microbial genomes [3]. To ameliorate the effectiveness of short reads, some NGS technologies such as Illumina allow for so-called *paired-end* sequencing where both ends of a genomic fragment ($< 2kb$) are sequenced. Most often, the two sequenced ends do not overlap and there is an (approximately) known gap distance between the two ends, the sequence of which is unknown. Larger gaps can be achieved using *mate-pair* sequencing, where a larger ($\approx 2kb - 100kb$) fragment is inserted in a BAC (Bacterial Artificial Chromosome) and sequenced from both ends. Although the use of paired-end and mate-pair technologies has improved the accuracy and completeness of assembled genomes, NGS sequencing still produces highly fragmented assemblies due to long repetitive regions. These incomplete genomes had to be finished using a more laborious approach that includes Sanger sequencing and specially tailored assembly methods. Owing to NGS, many efficient algorithms have been developed to optimize the running time and memory footprints in sequence assembly, alignment and downstream analysis steps.

The need for technologies that would produce longer reads which could solve the problem of repeating regions has resulted in the advent of new sequencing approaches – the so-called *third generation sequencing technologies*. The first among them was a single-molecule sequencing technology developed by Pacific Biosciences (PacBio). Although PacBio sequencers produce much longer reads (up to several tens of thousands of base pairs), their reads have significantly higher error rates (≈10-15%) than NGS reads (≈1%) [7]. Existing assembly and alignment algorithms were not capable of handling such high error rates. This caused the development of read error correction methods. At first, hybrid correction was performed using complementary NGS (Illumina) data [8]. Later, self-correction of PacBio-only reads was developed [9] which required higher coverage (>50x). The development of new, more sensitive aligners (BLASR [10]) and optimization of existing ones (BWA-MEM [11]) was required.

The release of Oxford Nanopore Technologies (ONT) MinION sequencers in 2014 ushered in a new era of cheap and portable long-read sequencing. Nanopore sequencers have transformative potential for research, diagnostic, and low resource applications. In 2014, ONT established an early access programme to make their technology available worldwide while still in the pre-commercial phase. In May 2015, the MinION technology was finally commercialized in the form of the *MinION MkI* device. MinIONs are small, portable, USB-powered, inexpensive sequencing devices that produce incredibly long reads (currently up to several hundred thousand base pairs), with potential applications not only in biological research, but also in personalized medicine, ecology, pharmacy and many other fields – this technology should open the door to not only new kinds of research, but also to a multi-billion dollar industry which will stem from that. The specifics of the MinION technology and library preparation procedures allow two types of sequencing to be performed: one-directional (1D) where only one strand of a DNA fragment is sequenced (the template or the complement), or two-directional (2D) where both strands of the DNA fragment are ligated together and sequenced successively. 2D sequencing produces redundant information, in that it sequences the same fragment twice (in the forward and reverse orientations), which can be, and is, leveraged to produce higher quality data. 1D reads from the MinION sequencer (with the R7.3 chemistry) have raw base accuracy less than 75%, while higher quality *pass* 2D reads (80-88% accuracy) comprise only a fraction of all 2D reads [12][13]. This again spurred the need for development of even more sensitive algorithms for mapping and realignment, such as GraphMap [14] and marginAlign [15]. Any doubt about the possibility of using MinION reads for *de novo* assembly was resolved in 2015 when Loman *et al.* demonstrated that the assembly of a bacterial genome (*E. Coli* K-12) using solely ONT reads is possible even with high error rates [16]. To achieve this, Loman *et al.* developed two error-correction methods for nanopore data: Nanocorrect as a pre-processing method to heavily reduce the input error rate for the next stage of assembly, and Nanopolish as a post-processing method to smooth out the left-over errors using signal level data.

In fact, all current assembly methods capable of handling third generation sequencing data depend on an error correction phase to increase the sensitivity of the next phase. The problem which arises in this approach is that error correction is a very time-consuming process which requires performing all-to-all mapping of the entire input dataset (or a subset of), alignment, and applying statistics in the form of consensus calling to resolve errors. Although this approach is well tested and yields good results, it is very time consuming which is especially evident when assembling larger genomes.

Instead, the focus of the work conducted in the scope of this thesis is to develop novel methods for *de novo* DNA assembly from third generation sequencing data, which provide enough sensitivity and precision to completely omit the error-correction phase, and still produce high-quality assemblies. Strong focus will be put on data obtained with Oxford Nanopore devices. It will be shown that this approach can reduce the overall running time by at least several times and up to even an order of magnitude compared to the state-of-the-art methods, while retaining comparable or better quality.

1.1 Objectives

The goal of this research is to develop efficient algorithms which can successfully handle a combination of read sequences of variable length and accuracy, with a strong focus towards longer, error-prone reads. The expected results of the project are:

1. A method for reference mapping/alignment of third generation sequencing data with strong focus on Oxford Nanopore technologies. The method will be used throughout the project for applications such as: overlapping, consensus calling and verification of results.
2. A method for *de novo* genome assembly from third generation sequencing data with strong focus on Oxford Nanopore technologies. The method will be designed with sensitivity and precision in mind, allowing the omission of the error-correction step while allowing for faster assembly of similar or better quality compared to the state-of-the-art.

The contributions of this thesis include the following:

1. A framework for systematic benchmarking of *de novo* genome assembly tools.
2. Optimized algorithms and data structures for *de novo* genome assembly with emphasis on third generation sequencing data.
3. Evaluation of developed algorithms using the novel benchmarking framework.

1.2 Organization of the Dissertation

Chapter 2 starts with a detailed background of existing state-of-the-art approaches and methods to sequence alignment and *de novo* assembly.

Chapter 3 presents the newly developed framework for benchmarking the assembly methods, as well as the results of detailed comparison of the state of the art methods.

Chapters 4 and 5 describe the two main components of the newly developed method in detail: overlap and consensus. The complete assembly method is designed in a modular manner, and as such each module was tested individually.

In Chapter 6, the modules were integrated in a single *de novo* assembly pipeline and tested as a whole. The results were compared to the state-of-the-art using the benchmarking framework developed in the scope of the Chapter 3.

Chapter 2

Background

Significant effort has been put into the development of methods for determining the exact sequence of nucleotides comprising a DNA molecule. Sequencing the entire genome of an organism is a difficult problem - as mentioned in the Introduction section, all sequencing technologies to date have limitations on the length of the molecule that they can read. These lengths are always much smaller than the genomes of a vast majority of organisms. For this reason, a *whole genome shotgun sequencing* approach is usually applied, where multiple physical copies of the analyzed genome are *randomly* broken by enzymes into small fragments of such size which can be read by modern technologies. Given a large enough number of DNA samples and the assumption of random fragmentation, the randomly created fragments from different copies of the genome will start to cover the same genomic regions and *overlap* with each other. This in turn gives us the only available information about putting those fragments back together. The average number of reads (fragments) that independently contain a certain nucleotide is called the *depth of coverage* (often only *coverage*) [17]. Combining the fragments back into a single chain of nucleotides is called *DNA assembly*. DNA assembly is a computational process performed by programs called *genome assemblers* [18]. Image 2.1 depicts this process.

The main assumption behind all assembly approaches is that highly similar fragments of the DNA originate from the same position within a genome [3]. This similarity is used to find overlaps between reads and join them into longer, contiguous sequences (*contigs*), or to align the reads to a *reference sequence* (a previously assembled high quality representation of an organism's genome). However, the similarity assumption is not entirely correct because of genomic repeats and sequencing errors. Ideally, a *de novo* assembly would consist only of contigs which fully span individual chromosomes of an organism's genome. Repeats in the genome cause ambiguities in assembly, in turn creating fragmented contigs of unknown orientation separated by gaps of unknown length. Assemblies containing gaps are called *draft assemblies*, while the process of filling the gaps is called *finishing*. The final, completed assemblies are often referred to as *closed* or *finished*.

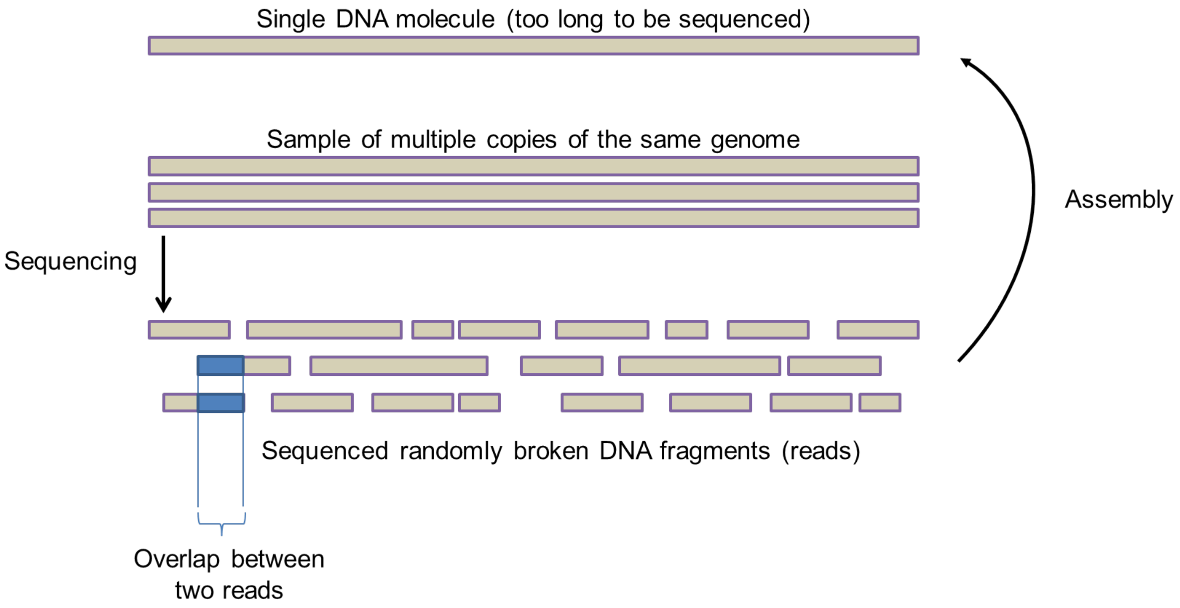

Single DNA molecule (too long to be sequenced)
Sample of multiple copies of the same genome
Assembly
Sequencing
Sequenced randomly broken DNA fragments (reads)
Overlap between two reads

Figure 2.1: Depiction of the sequencing process.

Repeats are thus one of the key problems in assembly [19]. The problem of repeats can only be resolved either by long reads which span the repeat region (first and third generation technologies), or by paired-end/mate-pair reads (NGS technologies) where each end is uniquely anchored at both sides of a repeat. Read pairs allow a span of several tens of kilobases (*kb*) between their ends, but they introduce an additional complexity because the gap between the two ends cannot be precisely sized [20]. In most bacteria and archaea, the largest repeat class is the rDNA operon, sized around $5-7kbp$ [20]. According to estimates presented in [20], having reads longer than $7kbp$ would automatically close $> 70\%$ of the complete set of bacteria and archaea present in GenBank (`http://www.ncbi.nlm.nih.gov/genbank/`).

The third generation of sequencing devices have come a long way in the past several years, and now routinely enable sequencing of reads which are much longer than this limit. The latest chemistry for Pacific Biosciences (PacBio) devices allows average read lengths of $> 10kbp$ [21], while Oxford Nanopore's MinION enables practically unlimited read lengths, bound only by the laws of physics which break the long DNA molecules during the process of sample preparation. For MinION, the longest mappable reads reported so far exceed $100kbp$ in length [22], however the DNA is usually fragmented into smaller chunks in a controlled fashion to increase the throughput and quality of the data.

From this aspect, third generation sequencing holds a key to creating closed bacterial and archaeal genomes, but also the potential to do the same for mammalian or even plant genomes in the near future. Efficient and accurate assembly methods are thus of crucial importance to allow such scaling across genome lengths and dataset sizes, especially considering high error rates present in the third generation sequencing data.

Genome assembly can generally be divided into two groups: *de novo* algorithms for assembly of new genomes, and *reference assembly* (or mapping) algorithms when a reference genome already exists [23]. Whereas reference assembly relies on alignment algorithms for finding the most probable location of each read on the reference genome, *de novo* algorithms attempt to find the best possible combination of reads in the dataset to produce the most probable genomic sequence. Both approaches are fundamental in the field of computational biology, and more than that, they are tightly coupled: mapping/aligning reads would not be possible without a reference sequence to map to, whereas the problem of mapping is closely related to the problem of *overlapping* reads, which is the first step in (modern) the *de novo* assembly methods.

This chapter will present the background and the state-of-the art in both mapping/alignment and *de novo* assembly, with a primary focus on Oxford Nanopore data.

2.1 Terminology

Let $\Sigma = \{A, C, T, G\}$ be the alphabet, $s \in \Sigma$ be a nucleotide (base), and $\overline{s} \in \Sigma$ be a Watson-Crick complement of s. Watson-Crick complements are defined as: $\overline{A} = T$, $\overline{C} = G$, $\overline{T} = A$ and $\overline{G} = C$.

A DNA sequence is a string $S = s_0 s_1 \cdots s_{n-1}$, where $n = |S|$. A reverse complement of the sequence S is $\overline{S} = \overline{s_0 s_1 \cdots s_{n-1}} = \overline{s}_{n-1}\overline{s}_{n-2} \cdots \overline{s}_0$.

A substring of a string S between coordinates i and j can be denoted as $S[i...j] = s_i s_{i+1} \cdots s_{j-1}$, where $0 \leq i < j \leq |S|$. Another way often used to denote the same substring is $S[i, j]$.

A k-mer is a sequence of k nucleotides: $S_i^k = S[i...(i+k)] = s_i s_{i+1} \cdots s_{i+k-1}$. k-mers are also often referred to as *seeds* (or *shingles* in computer science).

2.2 Approaches to sequence alignment/mapping

In practice, sequences, such as biological sequences and sequencing data, rarely match perfectly to each other. In order to find the similarities and differences between two sequences, they need to be *aligned* together. Sequence alignment is, most often, the first step in any sequence analysis today:

1. From the biological aspect, the difference between two genomic sequences can carry a lot of information - the level of similarity directly reflects the evolution of species and allows the construction of phylogenetic trees, while the dissimilarities within a species can give an image of possible genetic mutations which occurred and diseases/disorders they might have caused.
2. From the technological aspect, sequencing data originating from the exact same sample, and from the exact same position in the genome might differ due to sequencing error.

On the other hand, similarities between genomic fragments originating from adjacent (overlapping) genomic regions carry the only information one can use to successfully reconstruct (assemble) a genome *de novo*.

Pairwise alignment is the process of lining up two sequences in order to achieve the maximal levels of *identity* amongst the two, where the identity is the extent to which two sequences are invariant [24]. Alignment would be a simple procedure would the only possible difference between two sequences be *substitutions* (mismatches). In this definition, the measure of difference between two sequences could be expressed as the Hamming distance between the two, which only counts the non-matching characters. Consider the following alignment of two string sequences *test* and *best*, shown in Figure 2.2.

```
Sequence 1:   test
Matching:     x|||
Sequence 2:   best
```

Figure 2.2: An example of alignment of two sequences *test* and *best*, where the only form of difference between the two sequences are substitutions. Character "|" denotes a match between two sequences, while "*x*" denotes a mismatch.

In this example, each letter of the first string sequence is matched to exactly one letter of the other. However, in general terms, substitutions are not the only type of differences which might occur during alignment - instead one needs to consider the possibilities of missing letters (or nucleotides) in any of the sequences. This type of differences are referred to as *gaps*. A *gap*, or *deletion*, in one sequence means there was an extra symbol, or an *insertion*, in the other. An example alignment including gaps is shown in Figure 2.3.

```
ATCTGAC-ATG
  |||x| |||
--CTGCCAATG
```

Figure 2.3: An example of a gapped alignment of two nucleotide sequences. A dash "$-$" in any of the sequences means there is a missing nucleotide at that position, which is present in the other sequence.

In this definition of alignment, the difference between two sequences is expressed in the terms of the *Levenshtein* distance which generalizes the Hamming distance by counting gaps as well as mismatches. Levenshtein distance is the minimum number of operations needed to transform (edit) one string into another. This distance is therefore also referred to as the *edit distance*. To calculate the edit distance, each edit operation contributes to the total count with a score of 1.

To achieve an *optimal alignment*, one first needs to define the measure of optimality. There are two perspectives used in practice:

1. Minimization of the edit distance. In this case, every edit operation is penalized by a value of -1. Minimum number of edits equals the best alignment.

2. Maximization of the *alignment score*. Match operations (same bases in both sequences at an aligned position) are awarded a positive score s, while mismatch operations are penalized with a negative penalty x, and gaps with a negative penalty e. In some types of alignment (e.g. Gotoh), every opening of a streak of gap operations is also penalized by a factor of g. Optimal alignment is susceptible to these, arbitrary and user defined parameters, but is better suited to achieve biologically relevant alignment as compared to only the edit distance alignment [25].

Optimal algorithms that solve the alignment of two sequences have been proposed by Needleman and Wunsch in 1970 for *global alignment* of sequences [26] and Smith and Waterman in 1981 for *local alignment* [27]. There are also several variations on these algorithms in the form of *semiglobal alignment*, also known as *glocal* (global-local), where gaps at the beginning and/or end of one or both sequences are not penalized. Semiglobal alignment is especially well suited for detecting sequence *overlaps*. Although all of these algorithms provide optimal solutions in some sense, they are all based on a similar dynamic programming approach and share the same algorithmic complexity of $O(mn)$, where m and n are the lengths of the input sequences. Optimal alignment, although feasible for smaller sequences, is prohibitively slow should one try to align larger sequences, such as searching a large database of sequences or aligning two entire mammalian genomes. For example, order of magnitude for the length of such genomes is $10^9 bp$, which means that the alignment could take years to complete.

In order to cope with larger input sequences (and databases of sequences), heuristic approaches were proposed by Lipman and Pearson in 1985 with their FASTP algorithm for protein alignment [28] and reworked in 1988 into the popular FASTA algorithm for alignment of nucleotide sequences [29], and later by Altschul *et al.* in 1990 with the BLAST algorithm [30]. These were the first sequence *mappers* used for local alignment, which reduce the search space for alignment using heuristic approaches by first checking short homologies between sequences, which can be performed very efficiently. There have been many efficient mappers developed and published since, such as BLAT [31], LAST [32], BWA-MEM [11], BLASR [10] and many others, most of which targeted at aligning short highly accurate sequences generated by the second generation sequencing devices.

The rest of this section will give a brief description of the alignment algorithms, mapping of sequences, and then give a description of the state-of-the-art in mapping of nanopore reads.

2.2.1 Sequence alignment

Global alignment

Global alignment is suitable for aligning two sequences of similar size which share similarities across their entire lengths. Needleman and Wunsch were the first to formulate the problem of global sequence alignment, and to propose calculating its optimal solution based on the dynamic programming [26]. The initial proposed solution had an $O(n^3)$ complexity and was impractical. The first $O(n^2)$ complexity algorithm for solving global alignment was proposed by David Sankoff [33].

The algorithm is based on constructing a matrix, calculating smaller sub-problems of the solution iteratively to find the final optimal alignment, and storing the intermediate results (scores) in the matrix. To perform the alignment between two sequences M and N, several parameters need to be defined: the score s of a match between a base on M and a base on N; the penalty x for a mismatch; and e, the penalty for "skipping" a base (gap penalty).

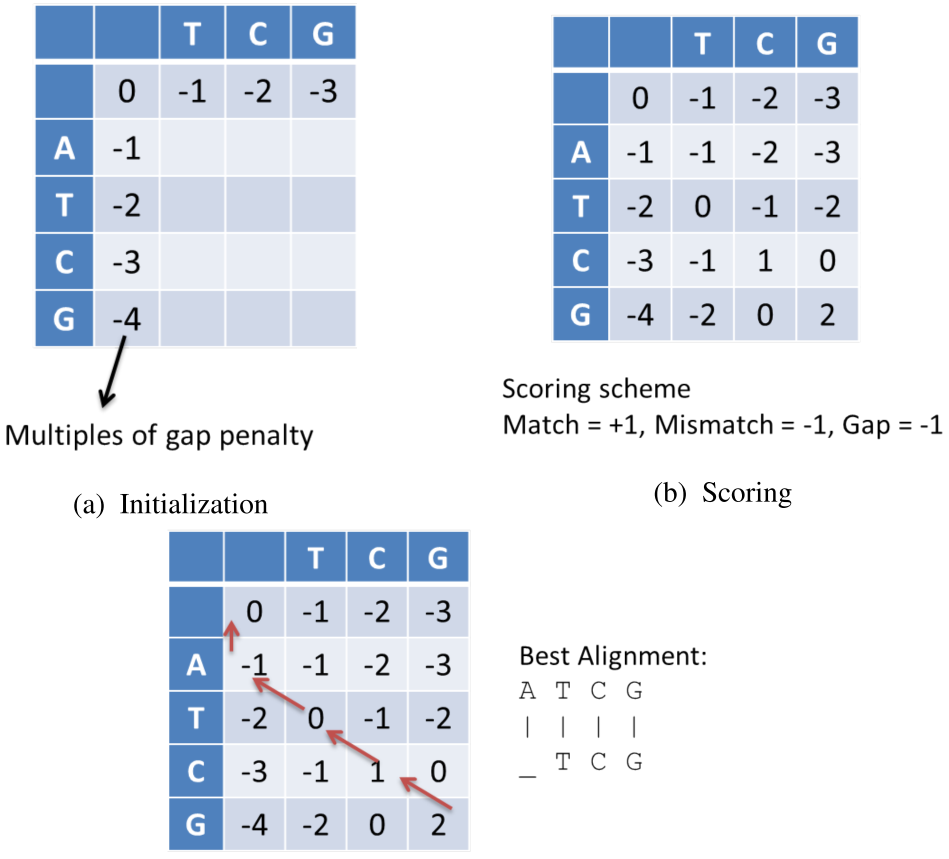
The algorithm is composed of three steps (Figure 2.4):

1. **Initialization** - a matrix $\mathcal{H}$ of $(m+1)(n+1)$ elements is initialized, where m is the length of M and n the length of N. The first row and the first column of the matrix are initialized to the multiples of the gap penalty e, as shown in Figure 2.4a.
2. **Scoring** - the elements of matrix $\mathcal{H}$ are calculated iteratively from left to right, top to bottom, using Equations 2.1 and 2.2 (Figure 2.4b). That is, the score of every element $\mathcal{H}(i,j)$ is determined based on the element above $\mathcal{H}(i-1,j)$ plus a gap penalty, the element to the left $\mathcal{H}(i,j-1)$ plus a gap penalty and the element on the upper left diagonal $\mathcal{H}(i-1,j-1)$ plus a match score/mismatch penalty. The maximum of these elements is marked as the predecessor of $\mathcal{H}(i,j)$. If the diagonal element was chosen, s is added if $M_i = N_j$ and x is added otherwise. In case the upper element or the element to the left were chosen, e is added to the score.
3. **Traceback** - the reconstruction of the actual alignment. Traceback starts at the bottom right corner of the matrix $\mathcal{H}(m,n)$, and traverses the matrix in reverse order, selecting one of its predecessors (either left, up, or diagonal) and continues the traversal until the top-left corner of the matrix is reached (Figure 2.4c).

$$\mathcal{H}(i,j) = \begin{cases} 0, & \text{if } i=0, j=0 \\ e \cdot i, & \text{if } i \neq 0, j=0 \\ e \cdot j, & \text{if } i=0, j \neq 0 \\ max \begin{cases} \mathcal{H}(i-1,j-1) + w(M_i,N_j) \\ \mathcal{H}(i-1,j) + e \\ \mathcal{H}(i,j-1) + e \end{cases} , & \text{otherwise} \end{cases} \tag{2.1}$$

$$w(M_i,N_j) = \begin{cases} s, & \text{if } M_i = N_j \\ x, & \text{otherwise} \end{cases} \tag{2.2}$$

Regarding the definition of optimality, Peter H. Sellers demonstrated the equivalence of global alignment with respect to maximizing the similarity versus minimizing the difference (distance) between two sequences when $s = 0, x = -1$ and $e = -1$ [34]. However, the equivalence only applies to the global alignment and not to the local as well.



		T	C	G
	0	-1	-2	-3
A	-1			
T	-2			
C	-3			
G	-4			

Multiples of gap penalty

(a) Initialization

		T	C	G
	0	-1	-2	-3
A	-1	-1	-2	-3
T	-2	0	-1	-2
C	-3	-1	1	0
G	-4	-2	0	2

Scoring scheme
Match = +1, Mismatch = -1, Gap = -1

(b) Scoring

		T	C	G
	0	-1	-2	-3
A	-1	-1	-2	-3
T	-2	0	-1	-2
C	-3	-1	1	0
G	-4	-2	0	2

Best Alignment:

```
A T C G
| | | |
_ T C G
```

(c) Traceback

Figure 2.4: Example of the steps in the global alignment of sequences *ATCG* and *TCG*. Arrows show the traceback of the optimal alignment between the two sequences, starting from the bottom right corner of the matrix.

Local alignment

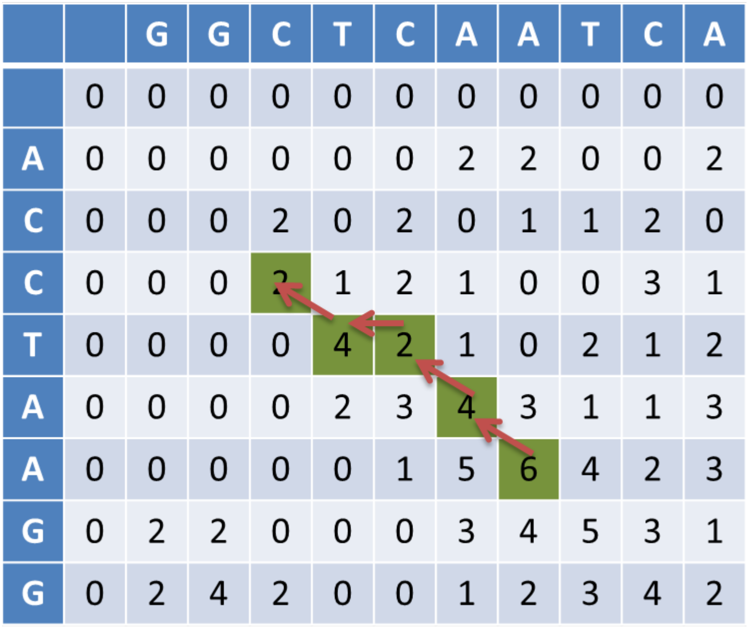
Global alignment is not well suited when sequences share similarities only in some of their smaller, more isolated (local) regions. This required a definition of a local measure of similarity as well as the development of an algorithm which could find an optimal local alignment. Such an algorithm was proposed by Smith and Waterman in 1981 [27]. Similar to the global alignment algorithm, Smith-Waterman is also based on calculating the dynamic programming (DP) matrix. However, in this case there are some subtle but important differences:

1. Gaps at the beginning of both sequences are not penalized,
2. If the score at any element of the DP matrix becomes < 0, it is set to 0,
3. Traceback begins from the maximum element of the entire matrix.

Similar to Equation 2.1, the value of an element in the Smith-Waterman alignment matrix is calculated using Equation 2.3:

$$\mathcal{H}(i,j) = \begin{cases} 0, & \text{if } i = 0 or j = 0 \\ max \begin{cases} 0 \\ \mathcal{H}(i-1,j-1) + w(M_i,N_j) \\ \mathcal{H}(i-1,j) + e \\ \mathcal{H}(i,j-1) + e \end{cases} , & \text{otherwise} \end{cases} \tag{2.3}$$

The function $w(M_i,N_j)$ is defined the same as in Equation 2.2. Also similar to global alignment, in order to obtain the actual alignment of sequences, the directions of traversing the DP matrix need to be stored during the process of filling the matrix. This makes the memory complexity of the algorithm equal to $O(mn)$ as well. An example of a local alignment matrix is given in Figure 2.5.



		G	G	C	T	C	A	A	T	C	A
	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	2	2	0	0	2
C	0	0	0	2	0	2	0	1	1	2	0
C	0	0	0	2	1	2	1	0	0	3	1
T	0	0	0	0	4	2	1	0	2	1	2
A	0	0	0	0	2	3	4	3	1	1	3
A	0	0	0	0	0	1	5	6	4	2	3
G	0	2	2	0	0	0	3	4	5	3	1
G	0	2	4	2	0	0	1	2	3	4	2

Figure 2.5: Example of a local alignment of sequences *GGCTCAATCA* and *ACCTAAGG* using a match score of $+2$, mismatch penalty of -1 and a gap penalty of -2. The first row and the first column of the matrix are initialized to 0. Arrows show the direction of the traceback, starting from the maximum element in the DP matrix.

Variations and optimizations

Many modifications and enhancements to the basic Needleman-Wunsch and Smith-Waterman algorithms have been developed over the years. Here we present some, most often used in practice.

Semiglobal or *overlap* alignment, sometimes also referred to as glocal (global-local) alignment, is a variation of the global alignment approach in which the gaps are not penalized at the beginning and/or ending of one or both sequences. Allowing arbitrary gaps at the beginning of a sequence can be achieved by setting the first row or column of the matrix to 0 (i.e. initial penalization is 0). Allowing arbitrary gaps at the ending of the sequences can be achieved by starting the traceback from the maximum element of the last row or column of the matrix, instead of the bottom right element as in normal global alignment. This alignment mode is useful for, e.g., aligning short accurate reads to a larger sequence, or determining whether two reads overlap in a suffix-prefix manner.

Alignment of biological sequences (*Gotoh*). When analyzing and comparing biological sequences, there is a general assumption that it is more likely that evolution would insert one large gap instead of several closely positioned smaller gaps. This means that the gap cost should ideally be non-linear and sub-additive, as compared to the classic Smith-Waterman which uses a linear $x \cdot e$ penalty for gaps, where x is the length of the gap and e is the gap penalty. Ideally, a good penalty would be a convex function (e.g. log) in the length of the gap. Such functions are not used in practice because the algorithm has an $O(mn^2)$ (where $m \leq n$) time complexity. In practice, the convex function is approximated using an affine function of the form $g + x \cdot e$, where g is the cost (penalty) of opening a gap and e is the cost of gap extension. Gotoh proposed in 1982 an $O(nm)$ algorithm for biologically relevant alignment which uses the affine gap penalty [25]. When aligning two sequences M and N, in addition to the alignment matrix $\mathcal{H}$, Gotoh defines two further matrices $\mathcal{E}$ and $\mathcal{F}$ which are used to track the score of best alignment which ends with a deletion in M or a deletion in N, respectively. These matrices are calculated as shown in Equations 2.4-2.6:

$$\mathcal{E}[i][j] = \max(\mathcal{E}[i][j-1] - e, \mathcal{H}[i][j-1] - g - e) \tag{2.4}$$

$$\mathcal{F}[i][j] = \max(\mathcal{F}[i-1][j] - e, \mathcal{H}[i-1][j] - g - e) \tag{2.5}$$

$$\mathcal{H}[i][j] = \max(\mathcal{E}[i][j], \mathcal{F}[i][j], \mathcal{H}[i-1][j-1] + w(N_i, M_j) \tag{2.6}$$

The traceback is performed in a similar manner to Smith-Waterman, but is expanded to account for possible switching of matrices. That is, there are three traceback matrices, one for each $\mathcal{E}$, $\mathcal{F}$ and $\mathcal{H}$ which keep track of the originating direction (be it in the same or a different matrix).

Linear memory complexity (*Hirschberg*). An important problem which occurs when aligning long genomic sequences is the quadratic memory complexity of the algorithms - for example, aligning a human and a mouse genome would require almost $9 \cdot 10^{18}$ matrix elements, each of which can consume several bytes of memory. Such large numbers are not feasible in modern computers. Instead, Hirschberg proposed an elegant and practical algorithm in 1975 which reduces the required space for global alignment from $O(nm)$ to $O(n)$ (for $n < m$), while only doubling the worst case time bound [35]. The algorithm is based on the observation that, given a global alignment $\mathcal{A}$ of sequences Q and T, the alignment $\mathcal{A}_r$ of the reverse complemented sequences $\overline{Q}$ and $\overline{T}$ is the reverse of $\mathcal{A}$. Hirschberg's algorithm is based on partitioning - the DP matrix is divided into two halves, and Needleman-Wunsch is performed twice: first the alignment $\mathcal{A}$ between $Q[0...h]$ and T is calculated, and second $\mathcal{A}_r$ between $\overline{Q[(h+1)...(|Q|-1)]}$ and $\overline{T}$. An index k of a column is chosen in such a way that the sum of the neighbouring elements of the last lines of $\mathcal{A}$ and $\mathcal{A}_r$ is maximum. This transition if part of the optimal path. The algorithm is then repeated recursively on two sets of substrings: (i) $Q[0...h]$ and $T[0...k]$, and (ii) $\overline{Q[(h+1)...(|Q|-1)]}$ and $\overline{T[(k+1)...(|T|-1)]}$, until the entire alignment path is constructed.

Banded alignment. Unlike the modifications which achieve linear memory complexity, reducing the time complexity from $O(nm)$ cannot be performed optimally. This means that heuristics need to be applied, possibly causing that a sub-optimal alignment will be reported. One approximation to reduce the time complexity is to restrict the computation of the elements to a band of diagonals around the main diagonal in each DP matrix. For a given diagonal k, only elements of the matrix that have coordinates such that $-k \leq (j-i) \leq +k$ are evaluated. If two sequences M and N are very similar, it is likely that their optimal alignment will be in a very narrow band of diagonals and that the vast majority of DP matrix elements will not be part of this alignment. Skipping the evaluation of these elements entirely can have a tremendous effect on the speed of alignment. Limiting the computation to within a band effectively places a limit on the maximum allowed gap in the alignment. By discarding the elements outside of the band, we need to calculate only $nm-(n-k)(m-k) = mk+nk-k^2$ elements. For k much smaller than $\min(m,n)$ the time complexity is then $O((n+m)k)$. The idea of a banded alignment was first proposed by Fickett in 1984 [36] and later independently developed by Ukkonen 1985 [37].

Implementation-based optimizations

Bit-vector alignment. Myers proposed a fast edit distance based bit-vector algorithm for approximate string matching in 1999 [38]. This algorithm processes the dynamic programming (DP) matrix using bit-parallelism, where multiple cells of the DP matrix are encoded in the same machine word. This enables simultaneous processing of multiple cells, effectively increasing the speed of computation. The real power of Myers' method comes from the idea that instead of storing the absolute values of scores in the rows of the DP matrix, only their differences (deltas) are kept. For edit distance alignment, this means that the only possible values of deltas are $\{-1,0,1\}$ which can be efficiently encoded, and processed by logic equations developed by Myers. The algorithm has $O(nm/w)$ time complexity, where w is the width of the machine word.

SIMD acceleration. Several methods have been proposed which utilize Single Instruction

Multiple Data (SIMD) technology, available in all modern CPUs, to accelerate the computation of the optimal alignment. The acceleration is achieved through vectorization of multiple cells of the DP matrix, similar to Myers' bit-vector algorithm. The major difference is that SIMD enables concurrent calculations on a vector of independent larger integers (8*bit*, 16*bit*,...) which fit into a SIMD word, thus enabling implementations of Smith-Waterman and other modes of alignment which require custom scoring parameters while achieving several-fold speed-up per CPU core. Rognes and Seeberg implemented the Smith-Waterman algorithm with Gotoh extensions by arranging the SIMD vectors to be parallel with the query sequence (unlike previous implementations which parallelized vectors along the diagonals of the DP matrix) [39]. Although this approach has some data dependencies it appeared to be faster because loading values along the minor diagonal is a time consuming process [40]. In 2007, Farrar proposed a "striped" approach [41] which addresses some of the computational dependencies of Rognes and Seeberg method. In general, computation is still carried out parallel to the query sequence, but instead of utilizing sequential row access like Rognes, Farrar processes several separate stripes covering different parts of the query sequence. Zhao *et al.* re-implemented and extended Farrar's algorithm into a stand-alone C/C++ library called SSW [42]. Many other SIMD implementations exist.

GPU acceleration. Modern Graphics Processing Units (GPUs) are composed of a large number of vector processors, oriented towards efficient computation. GPUs offer application performance by offloading compute-intensive portions of a program to the GPU. Since optimal dynamic programming based alignment is very compute intensive, GPU acceleration is a natural hardware complement to the algorithmic software implementation. Many GPU implementations of the alignment algorithms have appeared in the recent years, notably: SW# and SW#db [43][44], CUDASW++ [45], BarraCUDA [46], DOPA [47] and many more. The very recent SW#db method reports runtimes for protein database search comparable to that of a heuristic CPU-only BLASTP query and faster than other accelerated methods, which is a very significant result considering it utilizes exact, optimal alignment.

FPGA acceleration. Field Programmable Gate Arrays (FPGAs) provide a different, albeit more specialized avenue. FPGAs are reconfigurable devices which natively provide high levels of parallelism, but to optimally utilize FPGA architectures one needs to be familiar with the low-level design of hardware components. Not many FPGA implementations of the dynamic programming alignment exist, most of which are based on implementing the alignment process through a systolic array of identical processing elements which implement computations located on the anti-diagonals of the DP matrix. Another approach is to run a soft-core processor inside an FPGA, with custom optimized instructions implemented into the processor which will allow for accelerated computation. Example implementations of alignment algorithms include Shah *et al.* [48], SeqAlign (a student project, unpublished, http://www.chrisfenton.com/seqalign/), a M.Sc. thesis from Adam Hall [49] and Cray's commercial solution from 2005 (http://investors.cray.com/phoenix.zhtml?c=98390&p=irol-newsArticle&ID=779228). The Cray system was reported to provide a 28*x* speed-up for Smith-Waterman calculation compared to a pure CPU implementation of the same algorithm.

2.2.2 Sequence mapping

FASTA [29] and BLAST [30] were arguably the first sequence mappers, with FASTA being older of the two. Ideas presented in these works were quite innovative at the time - from the introduction of short homology searches, to very accurate statistical estimates for alignment accuracy; and they left an important legacy in the field of computational biology. Furthermore, many of these ideas are still employed today, in modern sequence mapping algorithms. As such, they present the basis of modern mapping algorithms, and will be presented in more detail.

FASTA

FASTA algorithm [29], originally published as FASTP and targeted at protein alignment [28] and later reworked into a nucleotide aligner, was the first published sequence alignment method which used the *k*-mers for identification of short homologies. The algorithm proceeds through four steps to determine the score for pairwise similarity [29]:

1. Identification of common *k*-mers between two sequences (in the original publication, *k*-mers are referred to as *ktup*s). FASTA and FASTP achieve much of their speed and selectivity in the first step. *k*-mers are looked up in a hash table to locate all identities or groups of identities between two DNA or amino acid sequences. For each match, FASTA calculates its "diagonal" in the dynamic programming matrix. The diagonal is given as $l = y - x$, where x is the position of the match on one sequence, and y its position on the other. Diagonal hits are counted, and 10 best diagonal regions are selected.
2. The 10 regions are rescored using a scoring matrix that allows conservative replacements and runs of identities shorter than *ktup* to contribute to the similarity score. For each of the best diagonal regions, a subregion with maximal score is identified - these are called "initial regions".
3. Whereas FASTP uses a single best scoring initial region to characterize the pairwise similarity, FASTA checks to see whether several initial regions may be joined together. Given the locations of the initial regions, their scores, and a "joining" penalty which is analogous to the gap penalty, FASTA calculates an optimal alignment of initial regions as a combination of compatible regions with maximal score.
4. The highest scoring database sequences are aligned using dynamic programming. The

final comparison considers all possible alignments of the query and the library sequence that fall within a band centered around the highest scoring initial region.

FASTA uses very short subsequences in the first step ($ktup = 4$ for nucleotide and $ktup = 1$ for amino acid sequences) which makes the method very sensitive, but also generates a high number of regions which are not part of an optimal alignment path, especially on larger databases. As an interesting note, one of the legacies of the FASTA mapper is the equally named FASTA file format for storing nucleotide and aminoacid sequences, still in widespread use today.

BLAST

BLAST (Basic Local Alignment Search Tool) [30], also designed for aligning both nucleotide and protein sequences, takes on a similar approach of analyzing short subsequences of length k. However, it employs a larger k ($k = 11$ for nucleotide and $k = 3$ for amino acid sequences) (in the original publication authors call this parameter w instead of k; in continuation we will use k for consistency). BLAST has had many updates and modifications (e.g. Gapped BLAST and PSI-BLAST [50]) and is still in widespread use today, especially through the NCBI web-based interface [51] for searching the entire database of known organisms with a query sequence, be it nucleotide or aminoacid. The basic BLAST algorithm is composed of three steps:

1. List - A list L of words is compiled from the query sequence. Words are short fixed-length k substrings of the query. Commonly, for searching aminoacid sequences, $k = 3$, while for nucleotides $k = 11$. For proteins, only words that score at least T when compared to some word in the query sequence are used (e.g. using the PAM-120 scoring matrix).
2. Scan - Every word from L is looked-up in the index of all words present in the database. For the search, a deterministic finite automaton is used (as opposed to e.g. a hash table).
3. Extend - All k-mer matches found in step 2 are extended into alignments. Extension is stopped when the score falls a certain distance below the highest score so far. Such alignments are referred to as High-scoring Segment Pairs (HSP).

In the basic, non-gapped version, BLAST extends words into HSPs by simply checking for matches or mismatches in the sequences, not accounting for potential insertions or deletions. BLAST then takes the highest scoring HSPs and determines their statistical significance and reports them as a list of final results together with two statistical measures: E-value and p-value (see below).

In the gapped version, BLAST takes two or more of the initial k-mer matches located on the same diagonal that are at most A bases away from each other. If such matches do not have statistically significant similarities they are discarded, otherwise Smith-Waterman algorithm is applied to obtain the final alignments. All such alignments are reported, together with their statistical measures (E-value and p-value).

E-value (Expected value), defined by the authors of BLAST, is a statistical measure used to unambiguously report statistical significance of a match [52]. This statistic was defined through analysis of how high a score is likely to arise by chance in the case of alignment of two random sequences. It was shown that, similar to how the sum of a large number of independent identically distributed (IID) random variables tends toward the normal distribution, the maximum of a large number of IID random variables tends to an *extreme value distribution* (EVD; Gumbel type distribution) [53]. Karlin and Altschul have shown that the scores of local alignments (unlike global) behave according to the EVD, which enabled the development of the statistical estimate of the probability that the alignment occurred by chance [54] (Equation 2.7):

$$P(S \geq x) = 1 - \exp(-\kappa m n e^{-\lambda x}), \tag{2.7}$$

where S is the local alignment score, m is the length of the query, n the length of the database and λ and κ are Karlin-Altschul statistical parameters [54][50] (also referred to as Gumbel parameters). The values of λ and κ depend on the scoring parameters used to generate the alignment score, and are usually pre-calculated for a given combination of the scoring parameters.

The E-value of a single distinct alignment of a score S represents the expected number of alignments which may occur with the same score when aligning a random query sequence of length m to a database of length n. The E-value can then be calculated by the Equation 2.8:

$$E = \kappa m n e^{-\lambda S} \tag{2.8}$$

Further, the values of κ and λ can also be used to normalize the score S and make scores (potentially computed using different scoring schemes) mutually comparable. The normalized score is referred to as the *bit-score* S', and can be calculated using Equation 2.9:

$$S' = (\lambda S - \ln \kappa)/\ln 2 \tag{2.9}$$

An important thing to note here is that, for ungapped alignments, Karlin-Altschul parameters can be explicitly calculated, while for gapped alignments the exact theory is not yet developed and the parameters are currently obtained only by simulation. For this reason, sets of Karlin-Altschul parameters are usually pre-computed for different gapped scoring systems (i.e. values of match score and mismatch, gap open and gap extend penalties).

Recent mapping methods

FASTA, and especially BLAST, influenced and inspired a great number of sequence mapping methods which appeared since their original publications. Some, such as BLAT (BLAST-Like Alignment Tool) [31], BLASTZ [55], PatternHunter [56] MUMmer [57][58] pre-date the second generation sequencing era. PatternHunter is especially interesting because of its introduction of spaced seeds for increased sensitivity, while MUMmer allows very fast comparison of entire genomes by constructing a suffix tree and looking-up the maximum unique matches (MUMs).

Most of the mapping tools, however, sprung up during the NGS era and were targeted at mapping and aligning short accurate reads: Novoalign (a commercial tool) [59], Bowtie and Bowtie2 [60], SOAP and SOAP2 [61], SHRiMP [62], while some were also designed to handle accurate chromosome-sized sequences and contigs: LASTZ[63], LAST [32], BWA [64] and BWA-MEM [11]. Majority of the mappers from this era were unable to handle the third generation sequencing data when it first appeared, mainly due to the error-rates present in the data. This required the development of a new long-read mapper for error-prone PacBio reads, BLASR [10]. BWA-MEM was later updated to accommodate PacBio, and subsequently Oxford Nanopore reads as well. LAST proved to be extremely sensitive and fast, even on very erroneous sequencing data should the right set of alignment parameters be used.

Since BWA-MEM, LAST and BLASR are among the most used mappers today, we will provide a short overview of these methods.

BWA-MEM follows the seed-and-extend paradigm; it initially seeds an alignment with supermaximal exact matches (SMEMs) - longest exact matches covering a certain position of a query. Since true alignments occasionally might not contain any SMEMs, BWA-MEM performs re-seeding with the longest exact matches that cover the middle base of the SMEM, and which occur at least $(k+1)$ times in the genome, where k is the number of occurrences of the original SMEM. BWA-MEM then finds colinear seeds and greedily chains them. Chains which are contained in larger chains are filtered out. The remaining seeds are extended using dynamic programming alignment.

LAST is the first method that can find and align similar regions in gigascale biological sequences, without certain severe restrictions, such as the necessity for heavy repeat masking (BLAST and similar methods) or restriction to strong similarities (usually present in DNA read mapping algorithms) [32]. Authors proposed the adaptive seeds as an alternative to fixed-length seeds commonly used in tools such BLAST, BLAT, BLASTZ and PatternHunter. The adaptive seeds can vary in length, and are extended until the number of matches in the target sequence is less than or equal to a frequency threshold f. LAST also implements the concept of spaced seeds [56] which is unified with the adaptive seed approach.

BLASR (Basic Local Alignment with Successive Refinement) is the first published method for mapping single molecule sequencing reads (PacBio third generation sequencing data) that are thousands to tens of thousands of bases long with divergence between the read and genome dominated by insertion and deletion error. BLASR also develops a combinatorial model of the sequencing error to demonstrate the effectiveness of the proposed approach. BLASR uses a suffix array or a BWT-index to find initial clusters of short exact matches between the read and the genome. It then clusters the exact matches and gives approximate coordinates in the genome for where a read should align. A rough alignment is generated using sparse dynamic programming on a set of short exact matches in the read to the region it maps to, and a final detailed alignment is generated using dynamic programming within the area guided by the sparse dynamic programming alignment.

2.2.3 Mapping nanopore reads

While some initial nanopore sequencing based applications have been reported (e.g. scaffolding and resolution of repeats in genomes [65], variant detection in clonal haploid samples [15] and *de novo* genome assembly [16]), many others remain to be explored. In particular, diploid and rare-variant calling [66], metagenome assembly and pathogen identification are all promising applications that will likely require development of new in silico techniques.

Read mapping and alignment tools are critical building blocks for many such applications. For mapping, reads from nanopore sequencing are particularly challenging due to their higher error rates and non-uniform error profiles [67]. For example, 1D reads from the MinION sequencer have raw base accuracy less than $65-75\%$; higher quality 2D reads ($80-88\%$ accuracy) comprise a fraction of all 2D reads and the total dataset, with overall median accuracy being between $70-85\%$ [65][12][68][69][13]. Reads from other short read (e.g. Illumina; $< 1\%$ error rate) and long read (e.g. PacBio; $\approx 10\%$ error rate) sequencing technologies have lower overall and mismatch ($< 1\%$) error rates. The increased read lengths in nanopore sequencing should facilitate mapping, reducing the ambiguity in location that is the major challenge for short read mappers. However, with current mappers, high error rates result in a large fraction of reads and bases ($10-30\%$) remaining unmapped or unused (e.g. 1D reads) for downstream applications [65][12][68]. This is further compounded when comparing two error-prone reads to each other or mapping to an imperfect or distant reference.

Thus, retaining sensitivity while accommodating high error or divergence rates is the key difficulty for current mapping methods. MinION error rates and profiles (i.e. ratio of insertions, deletions and substitutions) can vary across chemistries, sequencing runs, read types and even within a read. Furthermore, other nanopore and single molecule sequencing technologies may present a different distribution of error rates and profiles. Therefore, a general solution to mapping that is applicable to different error characteristics would have high utility for both current and future applications.

While alignment algorithms have been widely studied, gold-standard solutions such as dynamic programming (or even fast approximations such as BLAST) are too slow in practice for aligning high-throughput sequencing reads. To address this need, a range of read mapping tools have been developed that exploit the characteristics of second-generation sequencing reads (relatively short and accurate) by trading-off a bit of sensitivity for dramatic gains in speed [64][70]. The design decisions employed in these mappers are often tuned for specific error characteristics of a sequencing technology, potentially limiting their utility across technologies and error profiles. The less than ideal results reported in early studies using MinION data [71] could therefore be in part due to the use of mappers (e.g. BWA-MEM [11], BLASR [10] or LAST [32]) that are not suited to its error characteristics.

2.3 Approaches to *de novo* DNA assembly

Approaches to *de novo* DNA assembly are based either on greedy or on graph algorithms. Greedy algorithms were common in early genome assemblers for Sanger data [72]. They operate by joining reads into contigs iteratively, starting with the reads that have best overlaps; the process is repeated until there are no more reads or contigs that can be joined. This approach may not lead to a globally optimal solution, and is generally not applied for third generation sequencing data. Examples of assemblers that use the greedy approach include SSAKE [73], VCAKE [74] and SHARCGS [75].

Graph-based assembly algorithms are the most represented ones today. They fall into one of two categories: Overlap-Layout-Consensus (OLC) and de Bruijn graph (DBG) [18]. As a general summation of both approaches - overlaps between reads are detected (implicitly or explicitly), and then this information is used to construct a graph where vertices denote sequences (reads) and edges overlaps between them, or vice versa.

2.3.1 Overlap-Layout-Consensus approach (OLC)

Let $\mathcal{S} = \{S_1, S_2 ..., S_n\}$ be a set of non-empty strings over an alphabet Σ. An overlap graph [76] of S is a complete weighted directed graph where each string in $\mathcal{S}$ is a vertex and the length of an edge between vertices x and y, $x \rightarrow y$, is $|y| - ov(x,y)$ [77].

An example of assembly using the overlap graph is depicted in Figure 2.6. The construction of the overlap graph is the first step of the OLC approach. General process of this approach consists of three main phases [72][23][78], from which it derives its name:

1. *Overlap* - reads are compared to each other in a pairwise manner to construct an overlap graph.
2. *Layout* - the overlap graph is analysed and simplified with the application of graph algorithms to identify the appropriate paths traversing through the graph and producing an approximate layout of the reads along the genome. The ultimate goal is a single path that traverses each node in the overlap graph exactly once [18]. In most cases paths extracted from the graph do not cover the entire genome. Instead, individual paths through the

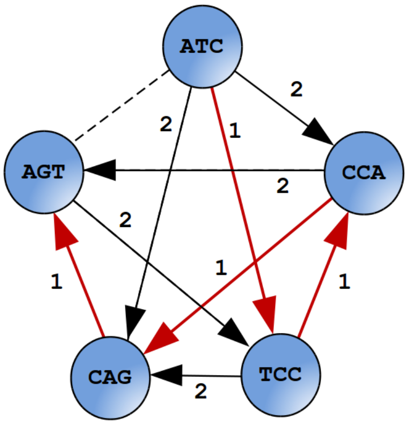


Figure 2.6: An example of assembly using the overlap graph by finding a Hamiltonian path. In this simple example, the set of input fragments consists of five reads of equal length, $\{ATC, CCA, CAG, TCC, AGT\}$, represented as nodes in the graph. Edges represent overlaps between nodes with corresponding edge weights. Edge weights for an overlap $x \rightarrow y$ are calculated as $|y| - ov(x,y)$. The result of assembly is a walk depicted in red edges starting from ATC and ending in AGT, which provides the reconstruction of the sequence $ATCCAGT$.

graph construct contiguous sequences (contigs) - partial representations of the genome. Unambiguous paths through the graph, that is, paths which do not contain extra edges along their way, are called *unitigs*.

3. *Consensus* - multiple sequence alignment of all reads covering the genome is performed, and the original sequence of the genome being assembled is inferred through the consensus of the aligned reads.

A very important observation is that the identification of a path that traverses every node in the graph only once is a computationally difficult *Hamiltonian path* problem. The problem is also known as the Travelling Salesman Problem, and is *NP-complete* [79]. For this reason the mentioned graph simplification methods are needed to create an assembly.

String graph [80] is a variant of the overlap graph approach where the nodes of the graph do not represent individual reads. Instead, there are two nodes per read: one to represent it's beginning, and one to represent it's ending. The string graph follows the same concepts of the overlap graph by removing the contained reads and transitive edges from the graph. The string graph paper [80] proposes a novel linear-time transitive reduction algorithm to remove the transient edges.

An assembly graph, introduced in [81], is a variant of the string graph with the same topology as the string graph, though the interpretation of the vertex set V is different (and similar to that of the overlap graph). In a string graph, V is the set of the two ends of sequences, while in an assembly graph V is actually composed of the forward and the reverse-complemented sequences. The assembly graph still undergoes all the simplification procedures as the overlap and the string graph.

More details on each of the OLC steps can be found in section 2.3.3.

Discussion

The three-phase definition of the OLC approach enables natural modular design of assembly tools employing this method, which allows for simpler modification and optimization of distinct assembly steps. Another advantage of OLC assemblers is that the overlaps among reads may vary in length, which equips them to handle the data from any generation of sequencing technologies.

On the other hand, the processing cost of the overlap phase is very high as it needs to be conducted for every pair of reads in the data set. Although the OLC approach is capable of handling NGS data, these datasets commonly consist of an order of magnitude more reads than were commonly generated in Sanger-based projects [18] (for which the OLC was initially designed), causing a significant increase of the overlap calculation time (quadratic complexity).

Examples of assembly methods which employ the OLC paradigm include: Celera assembler [82] (recently discontinued), Canu [83] (successor to Celera, unpublished, https://github.com/marbl/canu), Newbler [84], Edena [85], Shorty [86], Minimus [87], SGA [88], Miniasm [89] and Falcon [90].

2.3.2 The de Bruijn graph approach

Similar to the definition given for the OLC approach, let $\mathcal{S} = \{S_1, S_2 ..., S_n\}$ be a set of non-empty strings over an alphabet Σ. Given a positive integer parameter k, the de Bruijn graph $\mathcal{G} = \mathcal{B}^k(\mathcal{S})$ is a directed graph with vertices defined as a set of all k-mers generated from $\mathcal{S}$ (*k-spectrum*), and edges connecting k-mers with a perfect $(k-1)$ suffix-prefix overlap. A vertex of the de Bruijn graph is simply identified through its associated k-mer.

An example of the assembly using the de Bruijn graph is depicted in Figure 2.7. The process of constructing a de Bruijn graph consists of the following steps [91]:

1. *Construction of k-spectrum* - reads are divided into overlapping subsequences of length k.
2. *Graph node creation* - a node is created for every $(k-1)$-spectrum of each unique k-mer.
3. *Edge creation* - a directed edge is created from node a to node b if and only if there exists a k-mer such that its prefix is equal to a and its suffix to b.

This approach was first proposed by Idury and Waterman in 1995 [92][80], and was later expanded by Pevzner *et al.* in 2001 [93]. By converting the set of reads into edges of the de Bruijn graph, the assembly problem becomes equivalent to finding an Eulerian path in the graph - a path that uses every edge in the graph - for which efficient algorithms do exist [18]. In this case, assembly is a by-product of the graph construction, which proceeds quickly using a constant-time hash table lookup for the existence of each k-mer in the k-spectrum [72]. However, there can be an exponential number of distinct Eulerian paths in a graph, while only one

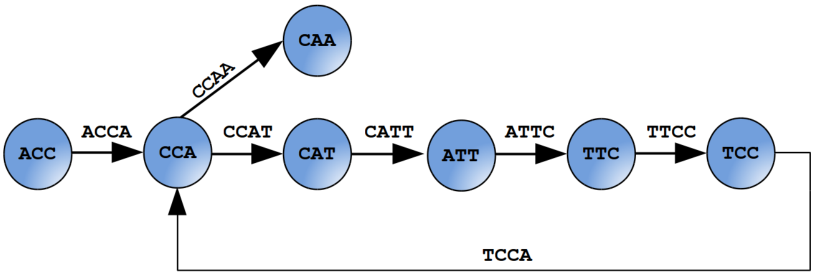


Figure 2.7: An example of assembly using the de Bruijn graph. Let $ACCATTCCA$ be a genome sequence we are trying to assemble from two reads $\{ACCATTC, ATTCCAA\}$. The k-mer spectrum for $k = 4$, is obtained from the reads: $\{ACCA, CCAT, CATT, ATTC, TTCC, TCCA, CCAA\}$, and the $(k-1)$-mer spectrum, $\{ACC, CCA, CAT, ATT, TTC, TCC, CAA\}$, required to construct the graph. The graph is constructed by placing k-mers on the edges, and $(k-1)$-mers on the vertices. The original sample sequence is fully reconstructed by traversing all edges of the graph (Eulerian path).

can be deemed to be the correct assembly of the genome, which makes this approach NP-hard as well [77]. To reduce the complexity of the problem, similar heuristics are usually applied to the constructed graph as for the overlap/string graph.

Discussion

A great advantage of the de Bruijn approach is that explicit computations of pairwise overlaps are not required, unlike is the case of the OLC approach. Finding pairwise overlaps is a computationally very expensive process, and thus the de Bruijn approach provides great performance when very large data sets are given (such as NGS data) [91].

However, de Bruijn graphs are **very sensitive to sequencing errors and repeats**, as they lead to new k-mers, adding to the graph complexity [91]. Additionally, although de Bruijn approach can handle both Sanger and NGS data, by dividing long reads into short k-mers, there is an effective loss of long range connectivity information implied by each read [18].

It was later shown that both de Bruijn and overlap graphs can be transformed into the string graph form, in which, similar to the DBG, an Eulerian path also needs to be found to obtain the assembly [80]. Major differences lie in the implementation specifics of both algorithms. Although the DBG approach is faster, OLC based algorithms perform better for longer reads (Pop, 2009). Additionally, DBG assemblers depend on finding exact-matching k-mers between reads (typically $\approx 21-127$ bases long [94]). Given the error rates in third generation sequencing data, this presents a serious limitation. The OLC approach, on the other hand, should be able to cope with higher error rates given a sensitive enough overlapper, but contrary to the DBG a time-consuming all-to-all pairwise comparison between input reads needs to be performed.

Examples of assembly methods which employ the DBG paradigm include: Euler [93], Velvet [95], AllPaths [96], AllPaths-LG [97], ABySS [98], SOAPdenovo2 [99], Ray [100] and many others.

2.3.3 Assembly of nanopore reads

Considering all the characteristics of both OLC and DBG approaches presented in sections 2.3.1 and 2.3.2, there are clear benefits of applying the OLC approach on long reads, especially error prone third generation sequencing data.

Since the focus in the past decade has been on NGS reads, most of the state-of-the-art assemblers use the DBG paradigm. Hence, there are not many OLC assemblers that could be utilized for long PacBio and ONT reads. In fact, methods developed to handle such data are mostly pipelines based on the Celera assembler, including: HGAP [101], PBcR [8] and the pipeline published by Loman *et al.* (in continuation *LQS pipeline*) [16]. Since its original publication [82], Celera has been heavily revised to support newer sequencing technologies, including modifications for second generation data [102], adoptions for third generation (single molecule) data via hybrid error correction [8], non-hybrid error correction [102][103] and hybrid approaches to assembly which combine two or more technologies [104].

All of this contributed to the popularity of Celera which led to its wide adoption in assembly pipelines for third generation sequencing data. Notably, one of the first such pipelines was the Hierarchical Genome Assembly Process (HGAP). HGAP uses BLASR to detect overlaps between raw reads during the error correction step. Unfortunately, HGAP requires input data to be in PacBio-specific formats, which prevents its application to other (e.g. nanopore) sequencing technologies. PBcR pipeline employs a similar approach to HGAP - it starts with an error correction step, and feeds Celera with corrected reads. PBcR, since recently, employs the MHAP overlapper [103] for sensitive overlapping of reads during the error-correction step. Also, recent updates allow it to handle reads from Oxford Nanopore MinION sequencers. The LQS pipeline also follows a similar approach, but implements novel error-correction (Nanocorrect) and consensus (Nanopolish) steps. Instead of BLASR and MHAP, Nanocorrect uses DALIGNER [105] for overlap detection. Nanopolish presents a new signal-level consensus method for fine-polishing of draft assemblies using raw nanopore data. The LQS pipeline also employs Celera as the middle layer, i.e. for assembly of error corrected reads.

Until very recently, the only non-hybrid alternative to Celera-based pipelines was Falcon. Falcon is a new experimental diploid assembler developed by Pacific Biosciences, not yet officially published. It is based on a hierarchical approach similar to HGAP, consisting of several steps: (I) raw sub-read overlapping for error correction using DALIGNER, (II) pre-assembly and error correction, (III) overlapping of error-corrected reads, (IV) filtering of overlaps, (V) construction of the string graph and (VI) contig construction. Unlike HGAP, it does not use Celera as its core assembler. Since Falcon accepts input reads in the standard FASTA format and not only the PacBio-specific format like HGAP does, it can potentially be used on any base-called long-read dataset. Although originally intended for PacBio data, Falcon presents a viable option for assembly of nanopore reads, even though they have notably different error profiles.

In late 2015 the developers of Celera, PBcR and MHAP moved away from original Celera and PBcR projects and started to develop a new assembler, Canu. Canu is derived from Celera and also utilizes code from Pacific Biosciences' Falcon and Pbdagcon projects.

Also in late 2015, a new long read assembly tool called Miniasm was released and later published in the beginning of 2016 [89]. Miniasm attempts to assemble genomes from noisy long reads (both PacBio and Oxford Nanopore) without performing error-correction. **This is coincidentally a closely related topic to the one explored by the author of this thesis, and was conducted in parallel and independently. It is important to mention that the work on this thesis was well under way and near completion at the time Miniasm was released, and the author of Miniasm acknowledged our concurrent work in his paper [89]. However, the two works still differ significantly. Namely, Miniasm lacks a consensus phase which corrects the sequencing errors and instead outputs contigs containing same/similar error-rates to the original input data. On the other hand, a consensus phase, developed and described in the scope of this thesis, enables rapid correction of sequencing errors from assembled contigs, with final results comparable to or better than the state of the art, while overall being an order of magnitude faster than the state-of-the-art.**

Aside from mentioned methods, hybrid assembly approaches present another avenue to utilizing nanopore sequencing data. Liao *et al.* [106] recently evaluated several assembly tools on PacBio data, including hybrid assemblers SPAdes [107] and ALLPATHS-LG [97] for which they reported good results. Both of these are DBG-based, use Illumina libraries for the primary assembly and then attempt to scaffold the assemblies using longer, less accurate reads. Furthermore, SPAdes was recently updated and now officially supports nanopore sequencing data as the long read complement to NGS data.

Now, to take a step back and consider the OLC approach again - even though the definition of the OLC paradigm enables and promotes the modular design of algorithms for *de novo* genome assembly and their software implementations, most methods follow this approach in a very restricted fashion. For example, the Celera assembler [82] in it's core is modular, but all modules depend on information stored in Celera's *Store* databases which are not commonly used in any other assembly tools. Another similar example, inspired by Celera assembler [108], is AMOS [109] (A Modular, Open-Source whole genome assembler; the encapsulating project of the Minimus assembler; not used in third generation sequence assembly) which defines the *AMOS message format* to enable communication between its various components. The AMOS message format is again custom tailored and not generically used in other assembly projects.

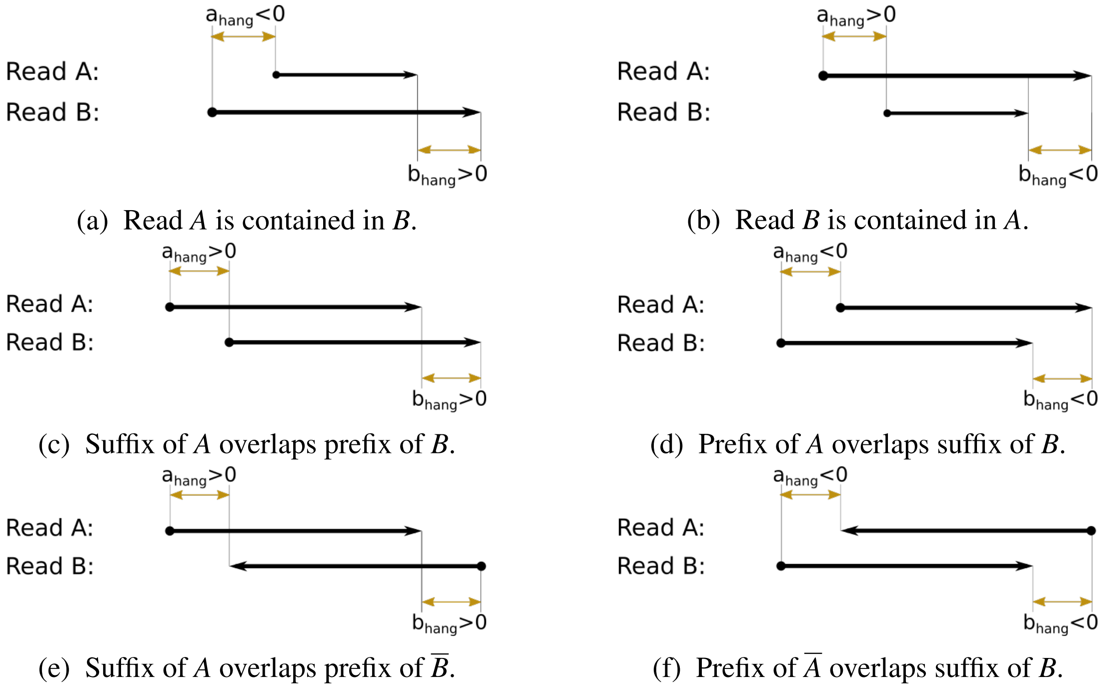
Until very recently, there was a great lack of standardization in the aspect of inter-modular information interchange. The appearance of standalone overlapping tools for third generation sequencing data (BLASR [10], DALIGNER [105], MHAP [103] and Minimap [81]) as well as standalone layout modules (Miniasm [81]) may have spun the wheel in development of such standards. Although official standards are not yet established, all of these overlappers generate very similar outputs (BLASR's $M5$ format, DALIGNER's *las*, MHAP's *mhap* and Minimap's paf) which are mutually easily convertible. Furthermore, there is an advent of a standard format for representing the layout graphs called GFA (Graphical Fragment Assembly) [110][81], currently supported by Miniasm, SGA and Canu. GFA is easily visualizible using the Bandage visualization tool [111].

Overlap

Given a set of sequences (reads), the overlap phase performs a pairwise comparison of input sequences, in an attempt to find all plausible prefix-suffix matches between them. It should report overlaps between reads which are adjacent in the real genomic sequence (true positives) and it should dismiss overlaps between reads which are not adjacent (false positive). False positive overlaps can occur due to repeats in the genome as well as high levels of sequencing errors, untrimmed adapter sequences and chimeric reads [72][81]. False negative overlaps can also occur as a consequence of high levels of sequencing errors in the data and the lack of sensitivity of an overlapper. Since the adjacency of any two read sequences is not known in advance, overlappers actually shift the detection process from checking adjacency to similarity comparison and adjacency prediction.

Early overlappers used in the Sanger sequencing era detected overlaps based on the seed-and-extend approach, commonly used in read-to-reference mapping (Section 2.2). In this approach, the algorithm for overlap detection would first look for seeds in the form of short exact matches of length k (k-mers). All reads that shared sufficiently many k-mers would be considered, and an alignment would then be constructed between such reads. Imprecise alignments in this case are allowed to account for sequencing errors, however, those alignments which do not meet certain quality requirements can easily be filtered out. Since the exact alignment is applied in this approach, the entire prefix and/or suffix of a particular read should be completely covered by the overlap.

Overlaps can be classified in three major groups: (I) *contained* overlaps, where both ends of a read are included in the alignment (Figure 2.8a - 2.8b) [76], (II) *dovetail* overlaps, where each read has exactly one of its ends in the alignment, and alignment begins at one read's end and continues until the other reads's end (Figure 2.8c - 2.8f) [76], and (III) partial overlaps (Figure 2.9) where the alignment does not extend to the ends of the reads (e.g. because of chimeric reads, presence of adapter sequences or higher sequencing errors toward read's end) [112]. In dovetail overlaps, the part of read A which is not in the alignment (overhang) is called a_{hang}, and the part of read B which is not in the alignment is called b_{hang} (Figure 2.8c). If the overhangs are positive values, then they represent the number of bases in the corresponding read which are *hanging outside* the alignment. The values of overhangs can also be negative, which represents



(a) Read A is contained in B.

(b) Read B is contained in A.

(c) Suffix of A overlaps prefix of B.

(d) Prefix of A overlaps suffix of B.

(e) Suffix of A overlaps prefix of $\overline{B}$.

(f) Prefix of $\overline{A}$ overlaps suffix of B.

Figure 2.8: Classification of overlap types: 2.8a and 2.8b depict contained overlaps where one read completely falls within the other; 2.8c - 2.8f depict all variations of dovetail overlaps between a pair of reads. $\overline{A}$ and $\overline{B}$ stand for reverse complements of A and B, respectively. The ball-point end of each read presents the start of the read, whereas the arrowhead presents its end.

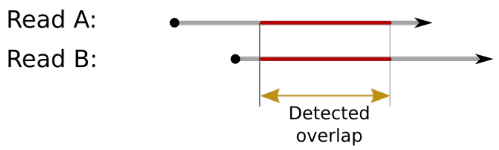


Figure 2.9: Depiction of a partial overlap between reads. Unlike dovetail overlaps, the overlapping region does not reach the ends of the reads.

the number of bases that are missing from the corresponding fragment (or, the number of bases in the other fragment that are hanging outside the alignment) (Figure 2.8d).

The seed-and-extend (mapping) approach, although effective, is prohibitively slow [113], especially on third generation sequencing data. Novel overlap algorithms had to be developed to harness the power of third generation data, especially for larger genomes. State-of-the-art in overlap detection between raw, noisy, long reads currently includes only four methods: BLASR [10], DALIGNER [105], MHAP [103] and Minimap [81].

BLASR was originally designed for mapping PacBio reads to a reference genome [10], and was later reworked to support overlapping. According to a comparison of methods presented in [103], BLASR is not ideally suited for overlapping all pairs of reads. It's sensitivity is primarily affected by the *bestn* parameter which controls how many alignments are reported for each read. BLASR was used as an overlapper in the Hierarchical Genome Assembly Process (HGAP) pipeline, one of the first assembly methods for third generation sequencing data, targeted at assembling microbial genomes from PacBio reads [101].

DALIGNER [105], published by Myers in 2014, was, contrary to BLASR, first designed as a fast overlapper for PacBio reads, and only later converted into a mapper/aligner. To be more precise, even though used for detecting overlaps, DALIGNER defines itself as being a tool for finding all significant *local alignments* between reads, which is a more general approach than only finding overlaps. DALIGNER identifies *k*-mer matches between two sets of reads by sorting their *k*-mers and merging the sorted lists [81]. It's speed is primarily derived through careful cache-friendly implementations of radix-sort and merge operations. A linear-time difference algorithm is then applied on the matching *k*-mer seeds to compute the overlap [103]. To maintain the efficiency accross datasets, DALIGNER must rely heavily on filtering the repetitive *k*-mers [103], potentially causing problems in repetitive genomic regions. DALIGNER's implementation also supports running on large-scale High Performance Computing (HPC) distributed memory environments, enabling it to be used in assembly projects of large (mammalian size) genomes. DALIGNER is used as an overlapper in the Falcon PacBio assembler [90] (`https://github.com/PacificBiosciences/FALCON`) and the LQS assembly pipeline [16] for nanopore data. Compared to BLASR, DALIGNER is more than an order of magnitude faster on a 54x whole human genome dataset (15600 CPUh vs 404000 CPUh) [105].

MHAP (MinHash Alignment Process) [103] takes on a different approach to overlapping than any previous overlapper. It uses a dimensionality reduction technique called MinHash to create a more compact representation of sequencing reads. MinHash was originally developed by Altavista to determine the similarity of web pages [114]. MHAP takes a DNA sequence and creates a MinHash *sketch*. First, all *k*-mers are converted to integer *fingerprints* using multiple, randomized hash functions. For each *k*-mer, only the minimum valued fingerprint is retained, and the collection of such min-mers for a sequence makes its sketch. Then, the Jaccard similarity measure amongst two *k*-mer sets can be estimated by computing the Hamming distance between their sketches, and the resulting estimate is strongly correlated with the number of shared *k*-mers between two sequences [103]. This is a computationally efficient technique for estimating similarity because the sketches are relatively small. The approach of generating MinHash sketches can also be viewed as a generalization of *minimizers* [115]. A limitation of this approach is that a fixed number of hash values is required to generate a sketch, regardless of the length of the sequences, which may waste space or hurt sensitivity when input sequences vary greatly in length [81].

Minimap [81] is the most recent addition to the list. It is heavily influenced by all of the above methods - similarly to MHAP, it adopts the idea of sketches but instead of MinHash uses minimizers as a reduced representation; it uses a hash table to store *k*-mers, and uses sorting extensively like DALIGNER [81]. Minimap is a versatile tool not intended only for read overlapping, but also for read-to-genome and genome-to-genome mapping. Minimap is blazingly fast - it takes only about 2 CPU minutes to overlap an entire *E. Coli* K-12 $54x$ nanopore dataset (genome size $\approx 4.6Mbp$, size of the input dataset $\approx 250Mbp$), and about 9 CPU hours to overlap an entire *C. Elegans* 81x PacBio dataset (genome size $\approx 100Mbp$, size of the input dataset $\approx 8Gbp$). Minimap starts by creating a minimizer index of the input dataset. A (w,k,ϕ)-minimizer of a string s at the position i, $i \in [0,|s|-1]$, is the smallest k-mer in a surrounding window of w consecutive k-mers from position i. Here, ϕ denotes a hash function which converts a k-mer string into a numerical form to be used for comparison. The minimizer is given as a triple (h,i,r), where h is the minimizer hash key, i is the position on the string s and r is the strand from which the minimizer originated. Such triplets are hashed by their minimizer hash key for easier lookup. The process of mapping/overlapping then consists of computing the minimizers for all input sequences, looking them up in the index, and clustering hits to identify colinear matches. The Longest Increasing Subsequence (LIS) algorithm is then applied on each cluster to generate the final mapping positions.

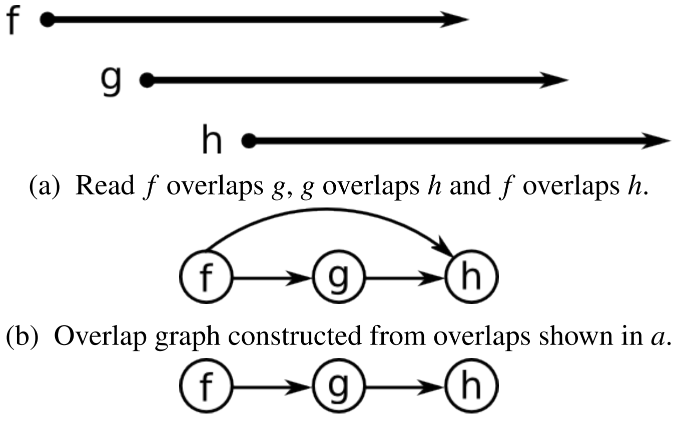
Layout graph

Myers introduced, formally, the notion of *overlap graphs* in his seminal paper from 1995 [76] as a generalization of the layout phase, and the *de novo* assembly concept overall. The overlap graphs have been used extensively since, and have had several re-definitions and updates in the form of *string graphs* [80] and *assembly graphs* [81]. While the original overlap graphs are seldom used today, both string and assembly graphs are still relying on some basic overlap graph ideas, and hold onto the general concepts of graph simplifications presented in the original 1995 paper [76].

The definitions of the overlap and string graph have been presented in section 2.3.1. Here, we will give a brief description of the assembly graph as the newest addition to the group.

Let v and w be two strings (reads) mapped to each other based on their sequence similarity. We say that *w contains v* if v can be mapped to a substring of w. Otherwise, if a suffix of v and a prefix of w can be mapped to each other, then v overlaps w. We denote this as $v \to w$. In this aspect, reads v and w can be represented as vertices in the graph, and the overlap relationship as a directed edge between them [81]. Similar to the definition of the overlap graph in [76], the length of the edge $v \to w$ equals the length of v's prefix that is not contained in the prefix-suffix match between the two sequences.

Let $G = (V,E,l)$ be a graph without multi-edges (every pair of vertices can have only one edge), where V is a set of vertices (reads), E a set of edges between them (overlaps) and ℓ : $E \to \Re_+$ is the length function. A graph G is said to be *Watson-Crick complete* if: a) for every read (vertex) v in V it's reverse-complement $\overline{v}$ is also present in V, or formally: $\forall v \in V$, $\overline{v} \in V$, and b) for every edge in E between two reads (vertices) v and w there is an edge in E between the reverse-complements of the two reads $\overline{v}$ and $\overline{w}$, or formally: $\forall (v \to w) \in E, \exists (\overline{w} \to \overline{v}) \in E$



(a) Read f overlaps g, g overlaps h and f overlaps h.

(b) Overlap graph constructed from overlaps shown in a.

(c) Overlap graph with reduced transitive edge $f \rightarrow h$.

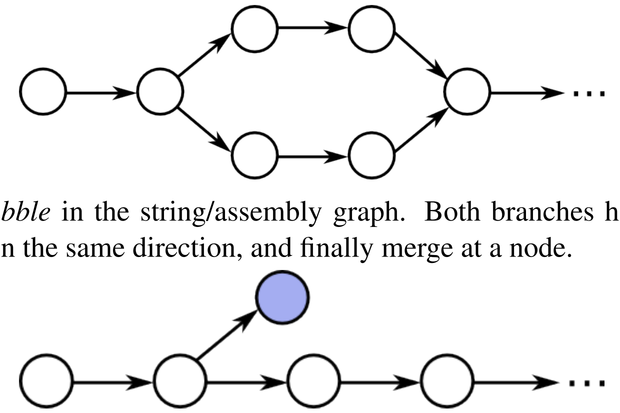
Figure 2.10: Depiction of an overlap graph containing a transitive edge.

[81]. A graph G is said to be *containment-free* if any sequence v is not contained in any other sequences in V (Figure 2.8a) [81]. If a graph G is both Watson-Crick complete and containment-free, it is an ***assembly graph*** [81]. In this definition, the de Bruijn graph can also be viewed as a special case of the assembly graph.

Aside from contained reads, the overlap graph, the string graph and the assembly graph all define the notion of reducing the *transitive edges*. A transitive edge is defined as follows: let f, g and h be read sequences (vertices). If f overlaps g ($f \rightarrow g$), g overlaps h ($g \rightarrow h$) and f overlaps h ($f \rightarrow h$), then the string graph edge $f \rightarrow h$ is unnecessary, as the edges $f \rightarrow g \rightarrow h$ can be used to spell the same sequence (Figure 2.10). A linear-time transitive reduction algorithm was proposed by Myers in 2005 to remove such edges from the graph [80]. The algorithm is based on graph coloring: for a vertex v of the graph find its longest outbound edge. Mark v and all of the vertices reachable through the out-edges of v as *inplay*. For every vertex w reachable through the out-edges of v, check if there are any vertices x reachable through the out-bound edges of w which are already marked as *inplay* and their edge lengths are shorter than the longest edge. If so, remove the longest edge from the graph.

Upon filtering contained reads and reducing transitive edges, string and assembly graphs often contain branches in the form of *bubbles* and *spurs* (also called *tips*) (Figure 2.11), which are most likely caused by sequencing errors, lack of coverage in some regions or multiple haplotypes present in the input sample. For a single haplotype genome reconstruction, these branches can cause problems in extracting long unitigs/contigs from the graph. Assembly methods usually remove short dead-end branches (e.g. Miniasm removes all such branches which contain 4 or less reads) and *pop* the bubbles using a variant of Kahn's topological sorting algorithm [81][95].

The concepts of overlap, string and assembly graphs were implemented in a number of assembly tools such as Celera, SGA, Falcon and Miniasm, however, to the best of our knowledge, only Miniasm presents a modular, standalone implementation of the layout phase. Additionally,



(a) A *bubble* in the string/assembly graph. Both branches have edges pointing in the same direction, and finally merge at a node.

(b) A *spur* or a *tip* in the string/assembly graph. A short branch which ends with a node that contains no out-bound edges (blue node in the graph).

Figure 2.11: Additional complexities in the string/assembly graphs.

while overlap and string graphs were designed for more accurate read data, and assemblers implementing them always incorporate an error-correction phase to ensure good assemblies, the assembly graph and its implementation in Miniasm were designed with noisier, third generation sequencing data in mind. The assembly graph is robust to higher levels of sequencing errors, and allows assembly from raw read overlaps, generated by any of the previously mentioned overlappers: Minimap, MHAP, DALIGNER and BLASR. In fact, Miniasm implements scripts to convert between the overlap formats used by these tools and into it's native *PAF* format [81]. The output of Miniasm is in the *GFA* format, which enables simple visualization and promotes standardization of assembler outputs.

Consensus

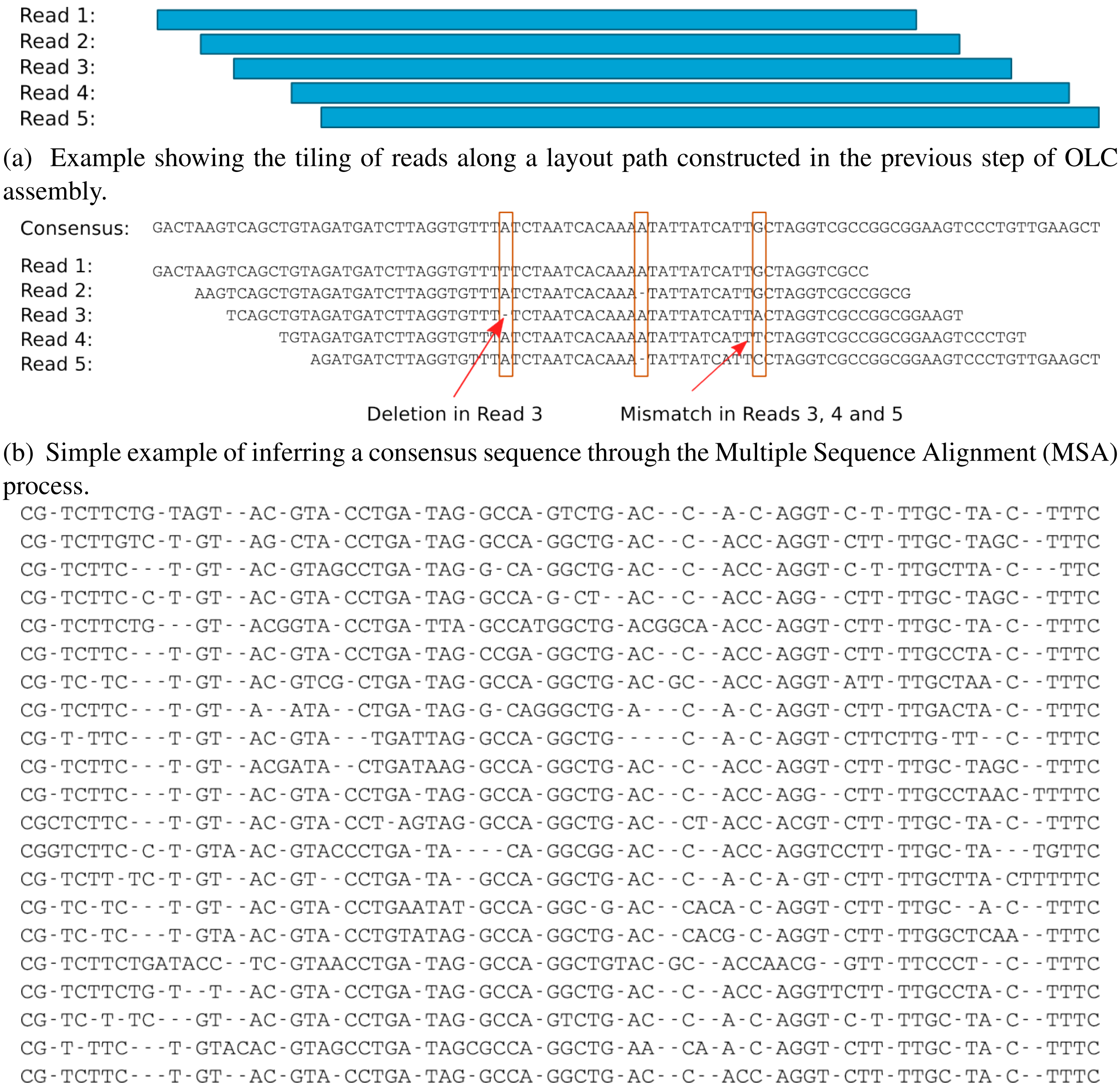
Consensus, in the context of the classic OLC assembly, almost always refers to the process of inferring the DNA sequence that is implied by the arrangement of reads along the chosen path through the graph. However, consensus developed a much broader meaning in the past several years: (I) error correction, (II) consensus phase of OLC and (III) fine polishing of post-consensus sequences. All these rely on the similar principles to some extent. In continuation of this section, we will focus on the general concepts of OLC contig consensus.

The reads along the layout path at this point may still contain sequencing errors which made it through all the previous steps into the final contig sequences, even though there might have been an error-correction step applied at a pre-processing phase. To resolve potential errors in contigs, statistical methods are applied. In general terms - oversampling the genome at sufficient coverage C will, on average, provide C reads covering each contig position. If the error model is truly random, then each read would represent an independent observation of the originating genomic region with sequencing error in the form of additive noise present in the data which can be simply overcome algorithmically [103]. However, every sequencing technology has a certain error bias, making high quality reconstructions significantly harder.

For each position in the contig, a consensus algorithm needs to answer the question "which nucleotide/gap is here?" with certain level of confidence. The confidence that an algorithm can achieve is usually estimated from control samples (e.g. results of assembly of a model organism) in terms of *quality values* (QV). QVs are calculated simply as: $Q = -10 \log p$, where p is the probability that a particular base is *wrong* (i.e. the probability of error). The quality is usually expressed in the form of "$Q10$" for the probability of $p = 10^{-1}$ of an error occurring, "$Q20$" for the probability of $p = 10^{-2}$, "$Q30$" for the probability of $p = 10^{-3}$, and so on. To express the quality of assemblies of model organisms, the probability is calculated *a posteriori* and represents the percentage of erroneous bases. Most often, an *a priori* per-base error probability estimate of sequencing reads is provided by the sequencing devices / base calling software. In this context, the quality values are often referred to as *Phred* scores for legacy reasons. The quality estimate is expressed in the same way, and usually provided together with the sequences (e.g. the FASTQ format). Unlike most sequencing devices which generate only one QV per base, PacBio's machines have the ability to estimate four different probabilities for which they provide QV's in their $bas.h5$ HDF5 format [116]: insertion, deletion, merge (probability of a merged-pulse error at the current base) and substitution. For deletions and substitutions, PacBio devices also provide the most likely alternative base at those positions. The *a priori* QV's provide extremely powerful information leveraged for best consensus calling [101], variant calling [66], and other applications.

Upon the completion of the layout phase, each contig can be viewed as a tiling of reads along the layout path, as depicted in Figure 2.12a. Commonly, an assembly method then applies a Multiple Sequence Alignment (MSA) algorithm on such tilings [23], and infers a consensus sequence using a simple majority vote or other, more complex probabilistic models which utilize QVs [101] or raw signal data [16], to call each base of the final sequence (Figure 2.12b).

Regular pairwise alignment, although much faster than MSA, is not well suited to construct a consensus sequence from a set of noisy reads with predominant insertion and deletion errors. This is due to the *reference bias*, where the aligned bases have a tendency to agree with the reference [117]. Even though the mapping of a read is accurate, its alignment will sometimes e.g. prefer to "hide" a true mismatch (Single Nucleotide Polymorphism, SNP) inside an insertion, and cause the aligned sequence to be biased toward the reference at some indel positions [118]. MSA helps avoid the reference bias by aligning a set of three or more sequences of similar length simultaneously (Figure 2.12c), allowing the inference of homology between all sequences - which is in fact just what the consensus procedure is trying to achieve. Similar to the pairwise alignment, the dynamic programming method can readily be extended to multiple sequences, but requires the computation of an L-dimensional matrix to align L sequences [119].


Read 1:
Read 2:
Read 3:
Read 4:
Read 5:
Consensus: GACTAAGTCAGCTGTAGATGATCTTAGGTGTTTATCTAATCACAAAATATTATCATTGCTAGGTCGCCGGCGGAAGTCCCTGTTGAAGCT
Read 1: GACTAAGTCAGCTGTAGATGATCTTAGGTGTTTTTCTAATCACAAAATATTATCATTGCTAGGTCGCC
Read 2: AAGTCAGCTGTAGATGATCTTAGGTGTTTATCTAATCACAAA-TATTATCATTGCTAGGTCGCCGGCG
Read 3: TCAGCTGTAGATGATCTTAGGTGTTT-TCTAATCACAAAATATTATCATTACTAGGTCGCCGGCGGAAGT
Read 4: TGTAGATGATCTTAGGTGTTTATCTAATCACAAAATATTATCATTTCTAGGTCGCCGGCGGAAGTCCCTGT
Read 5: AGATGATCTTAGGTGTTTATCTAATCACAAA-TATTATCATTCCTAGGTCGCCGGCGGAAGTCCCTGTTGAAGCT
Deletion in Read 3
Mismatch in Reads 3, 4 and 5

(a) Example showing the tiling of reads along a layout path constructed in the previous step of OLC assembly.

(b) Simple example of inferring a consensus sequence through the Multiple Sequence Alignment (MSA) process.

```
CG-TCTTCTG-TAGT--AC-GTA-CCTGA-TAG-GCCA-GTCTG-AC--C--A-C-AGGT-C-T-TTGC-TA-C--TTTC
CG-TCTTGTC-T-GT--AG-CTA-CCTGA-TAG-GCCA-GGCTG-AC--C--ACC-AGGT-CTT-TTGC-TAGC--TTTC
CG-TCTTC---T-GT--AC-GTAGCCTGA-TAG-G-CA-GGCTG-AC--C--ACC-AGGT-C-T-TTGCTTA-C---TTC
CG-TCTTC-C-T-GT--AC-GTA-CCTGA-TAG-GCCA-G-CT--AC--C--ACC-AGG--CTT-TTGC-TAGC--TTTC
CG-TCTTCTG---GT--ACGGTA-CCTGA-TTA-GCCATGGCTG-ACGGCA-ACC-AGGT-CTT-TTGC-TA-C--TTTC
CG-TCTTC---T-GT--AC-GTA-CCTGA-TAG-CCGA-GGCTG-AC--C--ACC-AGGT-CTT-TTGCCTA-C--TTTC
CG-TC-TC---T-GT--AC-GTCG-CTGA-TAG-GCCA-GGCTG-AC-GC--ACC-AGGT-ATT-TTGCTAA-C--TTTC
CG-TCTTC---T-GT--A--ATA--CTGA-TAG-G-CAGGGCTG-A---C--A-C-AGGT-CTT-TTGACTA-C--TTTC
CG-T-TTC---T-GT--AC-GTA---TGATTAG-GCCA-GGCTG-----C--A-C-AGGT-CTTCTTG-TT--C--TTTC
CG-TCTTC---T-GT--ACGATA--CTGATAAG-GCCA-GGCTG-AC--C--ACC-AGGT-CTT-TTGC-TAGC--TTTC
CG-TCTTC---T-GT--AC-GTA-CCTGA-TAG-GCCA-GGCTG-AC--C--ACC-AGG--CTT-TTGCCTAAC-TTTTC
CGCTCTTC---T-GT--AC-GTA-CCT-AGTAG-GCCA-GGCTG-AC--CT-ACC-ACGT-CTT-TTGC-TA-C--TTTC
CGGTCTTC-C-T-GTA-AC-GTACCCTGA-TA----CA-GGCGG-AC--C--ACC-AGGTCCTT-TTGC-TA---TGTTC
CG-TCTT-TC-T-GT--AC-GT--CCTGA-TA--GCCA-GGCTG-AC--C--A-C-A-GT-CTT-TTGCTTA-CTTTTTC
CG-TC-TC---T-GT--AC-GTA-CCTGAATAT-GCCA-GGC-G-AC--CACA-C-AGGT-CTT-TTGC--A-C--TTTC
CG-TC-TC---T-GTA-AC-GTA-CCTGTATAG-GCCA-GGCTG-AC--CACG-C-AGGT-CTT-TTGGCTCAA--TTTC
CG-TCTTCTGATACC--TC-GTAACCTGA-TAG-GCCA-GGCTGTAC-GC--ACCAACG--GTT-TTCCCT--C--TTTC
CG-TCTTCTG-T--T--AC-GTA-CCTGA-TAG-GCCA-GGCTG-AC--C--ACC-AGGTTCTT-TTGCCTA-C--TTTC
CG-TC-T-TC---GT--AC-GTA-CCTGA-TAG-GCCA-GTCTG-AC--C--A-C-AGGT-C-T-TTGC-TA-C--TTTC
CG-T-TTC---T-GTACAC-GTAGCCTGA-TAGCGCCA-GGCTG-AA--CA-A-C-AGGT-CTT-TTGC-TA-C--TTTC
CG-TCTTC---T-GT--AC-GTA-CCTGA-TAG-GCCA-GGCTG-AC--C--ACC-AGGT-CTT-TTGC-TA-C--TTTC
```

(c) An example of a short excerpt from a multiple sequence alignment of 21x real 2D nanopore sequencing reads. The level of sequencing errors is incomparable to the previous depiction in subfigure *b*.

Figure 2.12: Depiction of sequence MSA and consensus.

This makes the optimal algorithm scale exponentially with the complexity of $O(N^L)$, where N is the length of the input sequences, and therefore this method is NP-complete. Instead, heuristics in the form of progressive sequential alignment are applied to create approximations of the optimal solution, with a substantial reduction in computational complexity. Progressive alignment involves incorporating the input sequences one-by-one into the final model, following the inclusion order specified by a pre-computed guide tree (a tree in which nodes represent pairwise alignments between two sequences, a sequence and a profile, or two profiles) [120]. The combination of a tree-based progressive strategy and a global pairwise alignment algorithm forms the core of most available methods, such as: MAFFT, MUSCLE, ClustalW and T-Coffee [120].

However, probably the most popular MSA algorithm for generating consensus sequences from third generation sequencing data is the *Partial Order Alignment* (POA) graph approach [121][122]. It is used in several state-of-the-art methods, including Quiver [101] and Nanocorrect [16]. POA defines the MSA through a directed acyclic graph (DAG), where nodes are individual bases of input sequences, and weighted, directed edges represent whether two bases are neighbouring in any of the sequences. Weights of the edges represent the multiplicity (coverage) of each transition. The alignment is carried out directly by pairwise dynamic programming, eliminating the need to reduce the MSA to a profile [121]. One of the biggest advantages of POA is its speed (linear time complexity in terms of the number of sequences) when compared to other MSA algorithms [121]. Consensus sequences can be constructed from a built POA graph by repeatedly constructing a Viterbi (maximum likelihood) traversal, identifying sequences that match this path adequately, adjusting their contributions to the graph edge weights and iterating this process until no more sequences are left [122].

POA has successfully been applied in some of the most sensitive consensus methods for third generation sequencing data today: the Quiver consensus algorithm from PacBio [101], and the Nanocorrect error-correction algorithm for correcting nanopore reads (the LQS assembly pipeline) [16].

Quiver was originally developed for Pacific Biosciences' Circular Consensus Sequencing (CCS) analysis and later adapted to support the multimolecule consensus analyses [101]. Quiver uses a greedy algorithm to maximize the likelihood $Pr(\mathbf{R} \mid T)$, where $\mathbf{R}$ is a vector of reads and T is the unknown template sequence. For long references (contigs), the consensus is processed with tiling windows across the reference to limit the amount of memory used. The Quiver algorithm performs these steps for any window W: (I) An input set of alignments is used to identify the set of reads $\mathbf{R}$ corresponding to the window W; (II) A POA graph is constructed from $\mathbf{R}$ and a candidate template sequence inferred through the consensus ($\hat{T}_1 \leftarrow POAConsensus(R)$); and (III) Single base mutations are inserted into the candidate template iteratively ($\hat{T}_{s+1} \leftarrow \hat{T}_s + \mu$) until the likelihood converges ($Pr(\mathbf{R} \mid \hat{T}_s + \mu) > Pr(\mathbf{R} \mid \hat{T}_s)$). Quiver is designed to utilize all types of quality values provided by the PacBio sequencing machines (insertion, deletion, merge and substitution qualities), and uses a Viterbi algorithm to compute the template likelihood function and determine the consensus for the next iteration. In [101], authors report achieving a $Q50$ consensus accuracy on a *de novo E. Coli* K-12 assembly.

Nanocorrect is an error-correction method used by Loman *et al.* in [16] where the authors showed the first nanopore-only *de novo* assembly. The algorithm of Nanocorrect is relatively straightforward: (I) overlapping of input error-prone reads is conducted using DALIGNER; (II) for each read, all other overlapping reads are trimmed to the overlapping regions and reverse-complemented if needed; and (III) for each read the trimmed overlapping sequences are written to a FASTA file and input to $poaV2$ - the original POA implementation created by the authors of POA. The first consensus sequence reported by *poaV2* is then selected as the error-corrected read. Once all reads are corrected, the dataset is used as the input for the Celera assembler.

Nanopolish is a method related to Nanocorrect by being part of the same assembly pipeline published in [16]. Similar to Quiver for PacBio, Nanopolish is currently the state-of-the-art in consensus polishing methods for nanopore sequencing data, and actually the only method for this purpose. It takes as input a set of alignments (generated by BWA-MEM) and a set of raw nanopore reads, containing signal level data (*events*). Nanopolish then splits the draft assembly (contig) into $10kbp$ segments, overlapping by $200bp$, which are then processed in parallel. An anchor is selected every $50bp$ on the draft assembly's sequence. At each anchor position on the draft assembly sequence, 2D MinION reads and their template and complement current signal events are extracted and input into a Profile HMM model. The consensus algorithm generates the sequence of the draft assembly between anchors A_i and A_{i+2}. After the new consensus sequence is computed, the sequence of the assembly and the event-to-assembly mappings for A_{i+1} are updated using a Viterbi algorithm. This is done in order to progressively improve the quality of the event-to-assembly alignments recorded in the anchors.

Another graph alignment approach, inspired by POA, is DAGCon (with it's implementation *pbdagcon*) [101] developed to handle PacBio data. Similar to POA, DAGCon also constructs a directed acyclic graph. Unlike POA, DAGCon does not take as input a set of reads, but a set of alignments to a backbone sequence. First, DAGCon converts all mismatches in the alignments into insertion-deletion pairs as this simplifies the construction of the graph. Alignments are then normalized by left-aligning the indels (gaps are moved to the right-most equivalent positions). The initial linear graph is then constructed from the backbone sequence. Then, alignments are added to the graph. The dynamic programming alignment is not repeated, but the input alignments are used to find the matching bases in the backbone graph, and to insert the indels inbetween the matching bases as new branches in the graph. The multi-edges are merged to convert the possible multigraph into a regular graph, and the nodes with the same label are merged. A consensus is then inferred by traversing the weighted DAG and finding the path with the maximum score. Since, from this description, the DAGCon is not a true MSA method, it is typically biased toward the backbone sequence used in the alignment. The *pbdagcon* implementation of DAGCon was designed for correction of error-prone PacBio reads in the pre-processing step of *de novo* assembly: shorter input PacBio reads are aligned to a subset of longer "seed" reads using BLASR, and DAGCon is then applied on individual seed reads to correct the errors. *pbdagcon* module is used by the FALCON assembler, and as of recently by Canu as well.

FalconSense is another consensus algorithm developed in the scope of the FALCON project, which accelerates consensus generation by never explicitly building a multiple sequence alignment. Instead, similarly to DAGCon, it aligns reads to a template sequence which is the target of correction, and then tags individual matches and indels to determine a consensus sequence with high support. FalconSense is faster than DAGCon, but less robust [103].

PBcR is, again, a pre-processing error-correction tool which uses a multiple sequence alignment approach to correct the input data [8]. In it's original publication, PBcR was intended as a hybrid error-correction tool which corrected PacBio reads using much more accurate second generation sequencing data [8]. The short, accurate reads were aligned to the erroneous PacBio ones, and a tiling path would be extracted for each read and fed into an MSA. Originally, PBcR utilized AMOS's $make-consensus$ module, well suited for second generation data. PBcR later moved away from the hybrid approach and $make-consensus$, and used *pbdagcon* instead. PBcR is the predecessor to the Canu assembler.

However, to the best of our knowledge, currently no generic stand-alone consensus modules exist which could be applied on raw, noisy contigs generated from third generation sequencing data, such as the ones produced by Miniasm. Even though Quiver and Nanopolish *are* stand-alone modules, they: (I) require sequencer specific input, namely, Quiver depends on 4 types of quality values (insertion, deletion, merge and substitution), while Nanopolish requires raw nanopore signal-level data, and (II) both Quiver and Nanopolish are intended to be applied *after* the consensus phase of an assembler, to fine-polish the results. In our experiments with Nanopolish, polishing raw Miniasm contigs took much longer and the results were of significantly lower quality compared to using the same dataset with Canu created contigs.

A consensus method for this purpose would have to be very fast to be on par with Miniasm's speed in order to leverage the omission of the error-correction step, but also to generate results of comparable or better quality compared to the existing state-of-the-art assembly pipelines. This presents an interesting research opportunity to create impactful contributions.

Chapter 3

Evaluation of hybrid and non-hybrid methods for *de novo* assembly of nanopore reads

Assessing the state-of-the-art is a crucial step in any research. This chapter attempts to summarize and evaluate the current methods for *de novo* genome assembly from nanopore sequencing data, and give a thorough interpretation of results. The valuable information obtained through this evaluation enables better design choices during the development of a new and improved *de novo* assembly method.

We benchmarked five non-hybrid (in terms of both error correction and scaffolding) assembly pipelines as well as two hybrid assemblers which use third generation sequencing data to scaffold Illumina assemblies. Tests were performed on several publicly available MinION and Illumina datasets of *Escherichia Coli* K-12, using several sequencing coverages of nanopore data (20×, 30×, 40× and 50×). We attempted to assess the assembly quality at each of these coverages, in order to estimate the requirements for closed bacterial genome assembly. For the purpose of the benchmark, an extensible genome assembly benchmarking framework was developed. Results show that hybrid methods are highly dependent on the quality of NGS data, but much less on the quality and coverage of nanopore data and perform relatively well on lower nanopore coverages. All non-hybrid methods correctly assemble the *E. Coli* genome when coverage is above 40×, even Falcon, the non-hybrid method tailored for Pacific Biosciences reads. While it requires higher coverage compared to a method designed particularly for nanopore reads, its running time is significantly lower.

3.1 Methods

Since, to the best of our knowledge, no dedicated MinION read simulator exists, we focused our benchmark on real nanopore sequencing datasets. Although there is a number of publicly available datasets, many of them consist either of organisms/strains which do not yet have officially finished genome assemblies, or the coverage of the dataset is not high enough to provide informative nanopore-only assembly results. Aside from the *Lambda* phage (which comes as a burn-in sample for every MinION), sequencing data for the well-known clonal sample of *E. Coli* K-12 MG1655 are the most abundant. In this study, we use several most recent *E. Coli* K-12 datasets to reflect on the current state of the nanopore data as well as the quality of assembly they provide. In addition to using entire datasets, we subsampled two of the datasets to provide a larger span of coverages in order to inspect the scalability of assemblers as well as their ability to cope with the abundance or the lack of data.

3.1.1 Datasets

Benchmarking datasets were extracted from several publicly available nanopore datasets and one publicly available Illumina dataset. These are:

- *ERX*708228, *ERX*708229, *ERX*708230, *ERX*708231: 4 flowcells used in Loman *et al.* nanopore assembly paper [16].
- *E. coli* K-12 MG1655 R7.3 dataset [123].
- MARC, WTCHG dataset [13]: A dataset recently published by the MinION Analysis and Reference Consortium, consists of a compilation of data generated using several MinION sequencers in laboratories distributed world-wide.
- *E. coli* K-12 MG1655 SQK-MAP006-1 dataset: This is the most recent publicly available MinION dataset, obtained using the newest sequencing protocol. Link: `http://lab.loman.net/2015/09/24/first-sqk-map-006-experiment/`
- Illumina frag and jump libraries [106]. Link:`ftp://ftp.broadinstitute.org/pub/papers/assembly/Ribeiro2012/data/ecoli_data_alt.tar.gz`

The *benchmarking* datasets were designed with the idea to test the effect of varying coverage and data quality on the assembly process. The benchmarking datasets consist of either full datasets described above or subsampled versions of these datasets. Datasets used for benchmarking are presented in Table 3.1.

For each of the benchmarking datasets we analyzed the error rates present in the data (Appendix A, Table A.1). For this, all reads were aligned to the *E. coli K-12* reference (*NC*_000913.3) using **GraphMap** [14] (parameters "-a anchorgotoh"). Analysis shows a clear distinction between older (Dataset 1: 33% error rate) and newer (Dataset 2 and 3: 16%, Dataset 4: 11% and Dataset 5: 10% error rates) nanopore data, as well as Illumina data (3% error rate).

Table 3.1: Description of the benchmarking datasets used for evaluation.

Name	Description
Dataset 0	Illumina reads used by hybrid assemblers, consists of three libraries: (I) 1 frag library - paired-end reads (read length $101bp$, insert size $180bp$), coverage $55x$, 11861912 reads, and (II) 2 jump libraries - mate-pair reads (read length $93bp$, insert size $3000bp$), total coverage $85x$, 19779032 reads.
Dataset 1	Complete *E. coli* R7.3 dataset, contains both 1d and 2d reads (both pass and fail), total coverage $67x$ (70531 reads), of which 2d reads comprise $14x$ (11823 reads).
Dataset 2	Reads from Loman *et al.* [16] subsampled to coverage $19x$, pass 2d reads only (in total 16945 reads).
Dataset 3	Complete dataset used by Loman *et al.* [16] nanopore assembly paper, contains pass 2d reads only, coverage $29x$, 22270 reads.
Dataset 4	Reads from MARC WTCHG dataset, 2d reads extracted from pass ($33x$) and fail ($7x$) folders, total coverage $40x$, total number of 2d reads: 29635.
Dataset 5	2d reads extracted from the first run of the MAP006 dataset (MAP006-1), from pass folder only, coverage $54x$, 25483 reads in total.

3.1.2 Data preparation

For nanopore datasets, sequence data was extracted from basecalled FAST5 files using Poretools [124]. For Datasets 1, 3, 4 and 5 the entire set of reads was extracted and used for analyses. For Dataset 2, only flowcells $ERX708228$, $ERX708229$ and $ERX708230$ were used to obtain coverage close to $20x$. Datasets 1 was prepared for testing the assemblers' robustness on 1d reads. Additionally, Dataset 5 ($\approx 50x$ coverage) was subsampled to four different coverages: $32x$, $35x$, $37x$ and $40x$. This was done in order to enable a more fine-grained estimation of the amount of sequencing data required for a complete genome assembly.

Hybrid assemblers were tested using the Illumina dataset together with each nanopore test dataset. They were also run on the Illumina dataset alone, to get a reference for the assembly quality and to be able to estimate the contribution to the assembly when nanopore reads are added. All libraries in the Illumina dataset come with reads and quality values in separate files ($fasta$ and $quala$ files). These were combined into $fastq$ format using $convertFastaAndQualToFastq.jar$ tool downloaded from: `http://www.cbcb.umd.edu/software/PBcR/data/convertFastaAndQualToFastq.jar`.

3.1.3 Assembly pipelines

LQS pipeline: Pipeline developed and published by Loman *et al.* in their pivotal nanopore assembly paper (`https://github.com/jts/nanopore-paper-analysis`) [16]. The pipeline consists of Nanocorrect, WGS and Nanopolish. The version of the pipeline tested in this study uses Nanocorrect commit 47dcd7f147c and WGS version 8.2. For the base version of the pipeline, we didn't use Nanopolish.

PBcR: Implemented as a part of the WGS package (`http://wgs-assembler.sourceforge.net/wiki/index.php/PBcR`) [8]. In this study version 8.3rc2 of WGS was used. Spec file defining assembly parameters for nanopore data, was downloaded from the PBcR web page.

FALCON: To evaluate Falcon we used the FALCON-integrate project (`https://github.com/PacificBiosciences/FALCON-integrate`) (commit: 3e7dd7db190) [90]. Since no formal parameter specification for nanopore data currently exists, we derived a suitable set of parameters through trial and error (Table 3.2).

SPAdes: SPAdes v3.6.1 was downloaded from `http://bioinf.spbau.ru/en/content/spades-download-0` [107].

ALLPATHS-LG: ALLPATHS-LG release 52488 was downloaded from `https://www.broadinstitute.org/software/allpaths-lg/blog/?page_id=12` [97].

Canu: Canu was obtained from `https://github.com/marbl/canu` (commit: 70e711a382f). Canu is currently not yet published.

Miniasm: Miniasm was obtained from `https://github.com/lh3/miniasm` (commit: 17d5bd12290). For calculating read overlaps we used Minimap (`https://github.com/lh3/minimap`) (commit: 1cd6ae3bc7c) [89].

3.1.4 Evaluating the results

All assembly results were compared to the *E. coli* K-12 MG1655 NCBI reference, *NC*_000913.3. Assembly quality was evaluated using Quast 3.1 [125] and Dnadiff [126] tools. CPU and memory consumption were evaluated using a fork of the Cgmemtime tool (`https://github.com/isovic/cgmemtime.git`). For assemblies that produced one "big contig", over 4*Mbp* in length, that contig was extracted and solely compared to the reference using the Dnadiff tool.

Of all tested assembly pipelines, only the LQS pipeline has a polishing phase (Nanopolish). To make the benchmark fair and since other assemblers' results could also be polished to further improve them, all draft assemblies containing a "big contig" were polished using Nanopolish (`https://github.com/jts/nanopolish`, commit b09e93772ab4), regardless of the assembly pipeline they were generated with. At the time of testing, Nanopolish did not support 1d reads and had trouble with very small contigs. Therefore, we applied Nanopolish only to the largest contig in each assembly and skipped Dataset 1 which contains 1d reads. This does not present a problem because Dataset 1 was not successfully assembled by any of the non-hybrid assemblers.

Table 3.2: Since Falcon was not designed for Oxford Nanopore data, we experimented with its configuration parameters to try to achieve the best assemblies. Through trial and error we derived the following set of parameters which were used in our benchmarks.

```
job_type = local
input_fofn = input.fofn
input_type = raw

length_cutoff = 1000
length_cutoff_pr = 1000

jobqueue = your_queue
sge_option_da =
sge_option_la =
sge_option_pda =
sge_option_pla =
sge_option_fc =
sge_option_cns =

pa_concurrent_jobs = 24
ovlp_concurrent_jobs = 24

pa_HPCdaligner_option =  -v -dal4 -t100 -e.70 -l100 -s100
ovlp_HPCdaligner_option = -v -dal4 -t100 -h60 -e.92 -l100 -s100

pa_DBsplit_option = -x100 -s50
ovlp_DBsplit_option = -x100 -s50
falcon_sense_option = --output_multi --min_idt 0.50 --
    local_match_count_threshold 0 --max_n_read 200 --n_core 8
overlap_filtering_setting = --max_diff 100 --max_cov 100 --min_cov 5 --
    bestn 20 --n_core 8
```

3.2 Results

We developed a benchmarking framework called "NanoMark" to easily evaluate the performance of assembly tools on nanopore (or other) data. The framework is available on GitHub at `https://github.com/kkrizanovic/NanoMark`. NanoMark is implemented as a collection of Python scripts which enable simpler downloading and installation of all required software and dependencies, enable assembly of a given dataset using one or more assemblers and provide evaluation of the results. The framework currently implements wrappers for assembly pipelines described in this paper, but can easily be expanded to others. The wrappers have a standardized interface and internally handle conversion of input/output formats and data migration, as well as measure the time and memory consumption for a pipeline. Running NanoMark on a dataset will simply loop through all available wrappers, and collect and format the results. Detailed usage information can be found on the above mentioned link.

3.2.1 Non-hybrid assembly quality

Since Nanopolish currently does not support 1d reads, and none of the other assemblers include a polishing phase, initially we focused on comparison of non-polished draft assemblies.

Table 3.3 displays assembly results on Datasets 2-5 assessed using Quast and Dnadiff. Dataset 1 analysis is omitted because of its particular characteristics. It has a greater total coverage but much lower data quality compared to other datasets (older version of the sequencing protocol, low percentage of 2d reads and the use of 1d reads). None of the non-hybrid assemblers managed to produce a good assembly using Dataset 1 (Appendix A, Table A.2). It can be concluded that low 2d coverage together with high coverage of low quality 1d reads is not sufficient to complete an assembly of a bacterial genome using currently available methods.

Table 3.3: Assembly quality assessment using Quast and Dnadiff.

Dataset	Assembler	# ctg.	N50	Genome fraction (%)	Avg. Identity 1-to-1	Total SNPs	Total Indels
2	LQS	8	1159703	**99.895**	98.08	8858	79746
	Falcon	98	11083	6.994	94.58	3263	47211
	PBcR	22	246681	0.593	93.7	14823	269053
	Canu	26	332535	90.123	95.71	5691	183774
	Miniasm	15	353994	0.002	84.21	248575	373190
3	LQS	3	**4603990**	**99.998**	**98.49**	4568	65283
	Falcon	124	13838	17.316	94.97	3206	59638
	PBcR	1	4329903	12.825	94.03	7209	262357
	Canu	10	4465231	88.655	95.77	5213	185027
	Miniasm	3	3362269	0.002	84.04	247849	367927
4	LQS	8	**4622531**	**99.938**	**99.08**	2256	40118
	Falcon	13	4538244	99.938	97.66	3710	104165
	PBcR	3	3615068	99.553	97.39	2394	117397
	Canu	2	4576679	99.915	98.42	812	71878
	Miniasm	1	4577227	0.002	88.31	185194	326066
5	LQS	5	4006324	**99.991**	99.43	1435	25106
	Falcon	1	4580230	99.655	98.84	2589	50662
	PBcR	1	4596475	99.914	98.99	1136	45542
	Canu	1	**4600945**	**99.746**	**99.29**	415	32100
	Miniasm	1	4774395	0.002	88.69	175279	328688

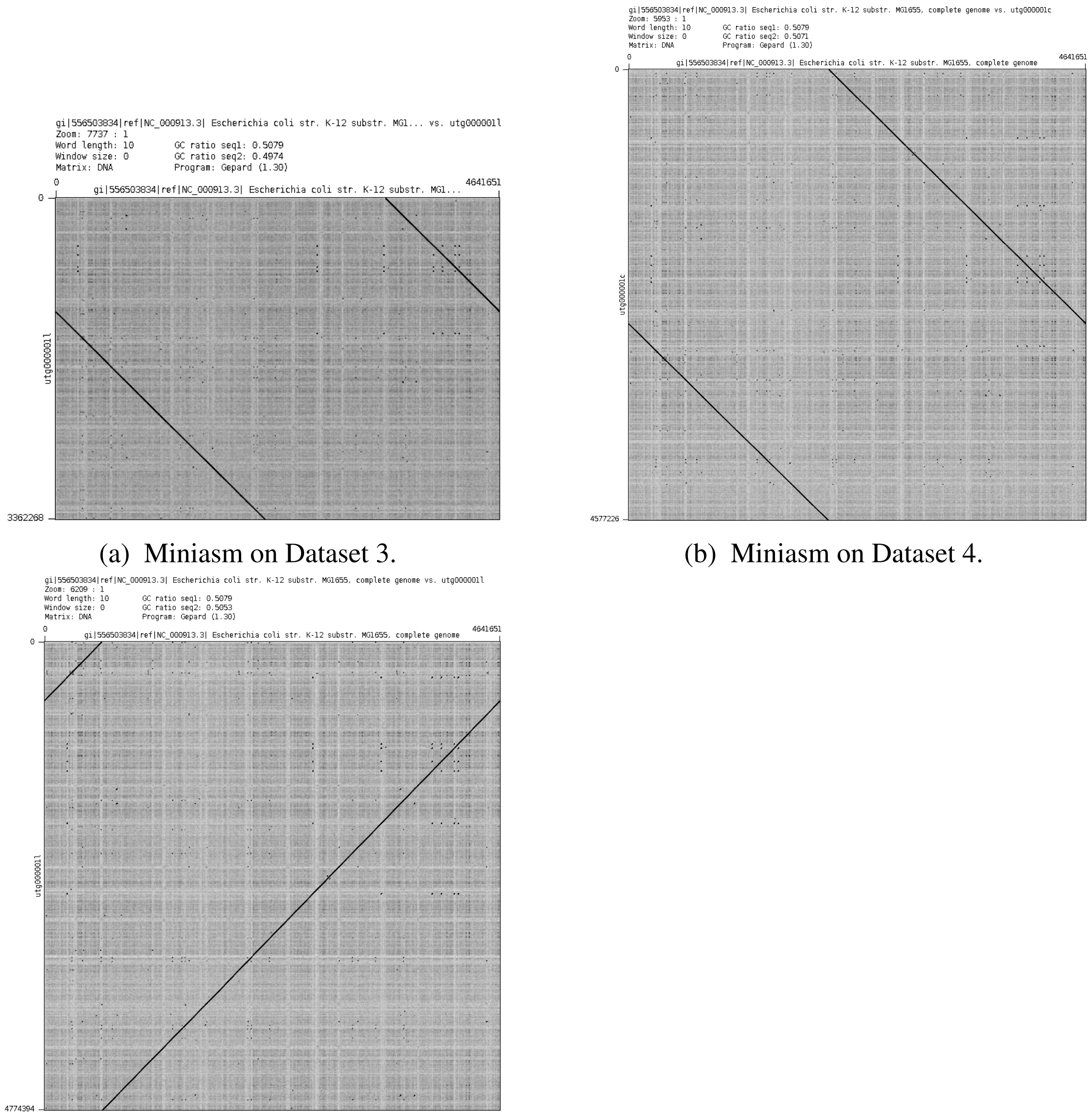
None of the non-hybrid assembly pipelines managed to complete the genome at 20*x* coverage. LQS pipeline produced the best assembly - it managed to cover almost the whole genome, albeit using 8 separate contigs. 30*x* seems to be sufficient for LQS pipeline to get very good results and for Canu and PBcR to cover most of the genome, however with the largest contig notably shorter than the reference genome (especially for PBcR). At coverages over $40x$, all tested assemblers produce good contiguous assemblies, which surprisingly includes Falcon, originally designed for PacBio data. To further estimate the coverage required by each assembler for a contiguous assembly, we used Dataset 5 subsampled to coverages $32x$, $35x$, $37x$ and $40x$. The results are shown in Appendix A, Table A.3.

Another surprising result that can be seen in Table 3.3 is a noticeable drop in assembly quality for the LQS pipeline on Dataset 5. While it managed to cover a greater part of the reference than any other pipeline on any dataset, its assembly consisted of 5 contigs, the largest of which is just over $4Mbp$. Overall, LQS assemblies demonstrate the highest average identity compared to the reference sequence, even without applying the polishing phase.

Miniasm's strengths are, on the other hand, oriented towards very fast and contiguous assembly of the genome, without increasing the per-base quality of the resulting sequences. Since it does not include an error correction step nor a consensus step, the contigs it produces contain errors similar to the input read data. This makes Miniasm hard to numerically compare to other assemblers without polishing the contigs. Table 3.3 shows that Miniasm manages to produce assemblies which contain, on average, a smaller number of contigs compared to other methods, cumulatively covering the entire genome. On coverages above $40x$, Miniasm produces a single contig assembly of the entire *E. Coli* genome. Since statistics in Table 3.3 make Miniasm hard to compare to other assemblers, we generated dotplots of the largest contigs produced for Datasets 3, 4 and 5 (Figure 3.1). This was performed in order to validate that the generated contigs were not chimeric or misassembled.

Additionally, we performed a "big contig" analysis where only the largest contig of length $\geqslant 4Mbp$ (a representative of the *E. coli* chromosome) was selected and evaluated using Dnadiff. This analysis gave a good estimate on the quality of the assembly from the aspects of chromosome completeness and breakage. Apart from Miniasm, all non-hybrid assemblies that produced a "big contig" had a comparable number of breakpoints ($10-50$) with the exception of PBcR on Dataset 3 (841) and LQS on Dataset 5 (88). It is interesting to note that in these cases the "big contig" is considerably shorter than the reference (see Appendix A, Table A.4).

Since the results for non-hybrid assembly tools show variation in assembly quality across datasets (Table 3.3), we further investigated the differences between them. As described in the Background section, there are two major differences: (I) LQS and PBcR both employ WGS (Celera) as their middle-layer assembler and Canu is a modified fork of Celera, while Falcon and Miniasm implement their own string/assembly graph layout modules; and (II) each of these pipelines, save for Miniasm, implements its own error-correction module. Taking into account that both Celera and Falcon utilize an overlap-graph based layout step, we suspected that (II) may have played a more significant role on the assembly contiguity. The error-correction pro-



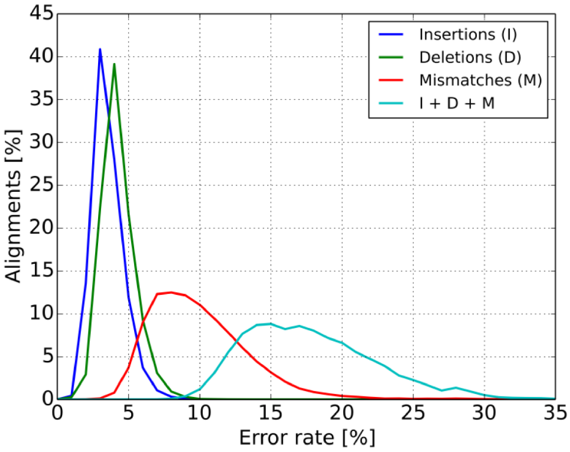
(a) Miniasm on Dataset 3.

(b) Miniasm on Dataset 4.

(c) Miniasm on Dataset 5.

Figure 3.1: Dotplots of largest contigs generated by Miniasm for: *a*) Dataset 3, *b*) Dataset 4 and *c*) Dataset 5.

cess is performed very early in each pipeline, and the quality of corrected reads can directly influence any downstream analysis. For this purpose, we analysed the error rates in raw reads from Dataset 3 (Figure 3.2) as well as the error-corrected reads generated by Nanocorrect, PBcR, Canu and Falcon error-correction modules (Figure 3.3). For analysis, all reads were aligned to the *E. coli* K-12 reference (*NC*_000913.3) using GraphMap (parameters "-a anchor-gotoh"). The results show that each error-correction module produces corrected reads with a significantly different error profile. The raw dataset (coverage 28.78*x*) contained a mixture of 3% insertions, 4% deletions and 9% mismatches (median values). While the insertion errors were mostly eliminated by all error-correctors, PBcR, Canu and Falcon exhibited higher amounts of deletion errors in their output. Nanocorrect produced the best results, reducing both



Insertion rate	Deletion rate	Mismatch rate	Match rate	Coverage
3%	4%	9%	85%	28.78x

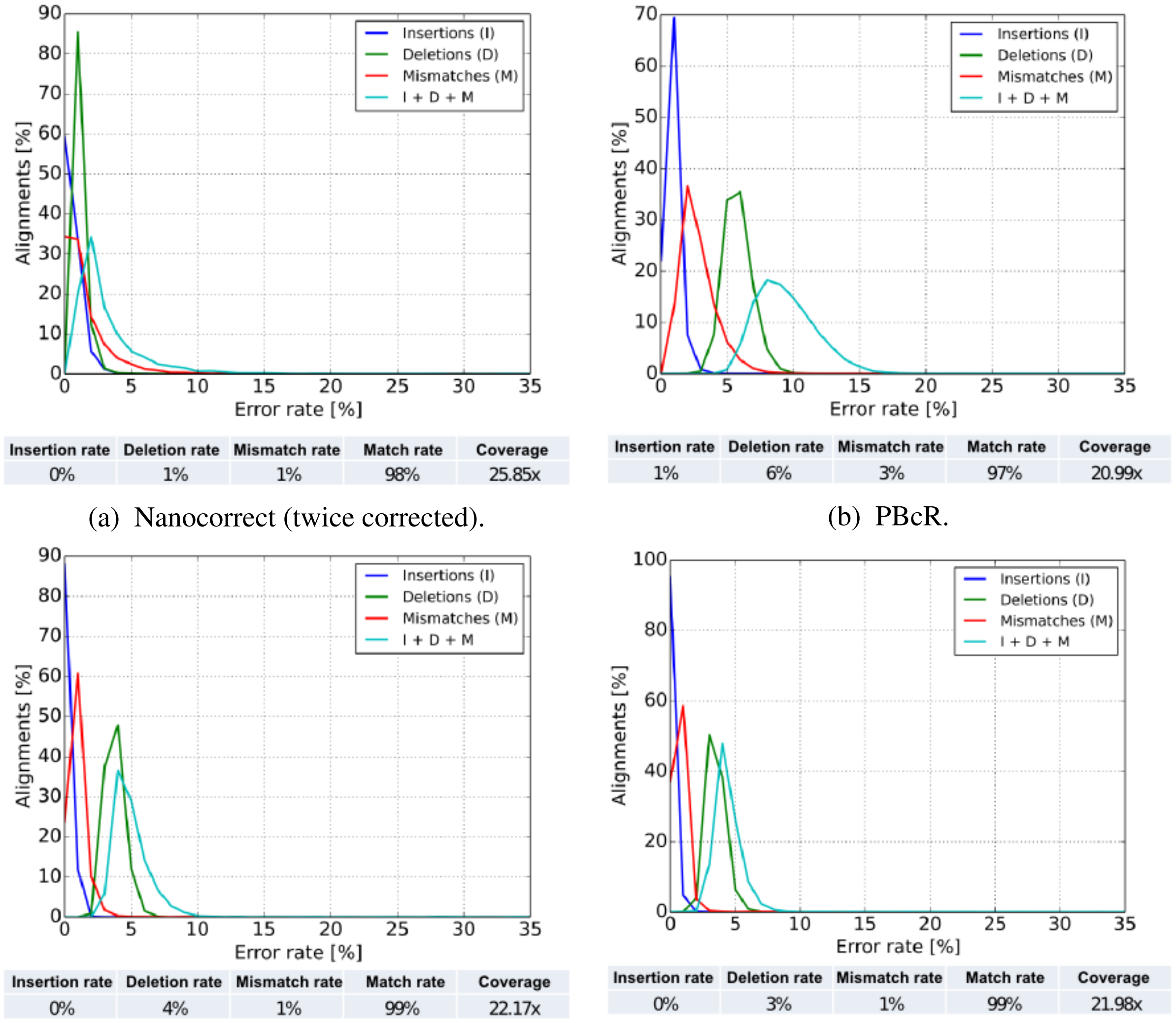
Figure 3.2: Error rate analysis of raw nanopore reads from Dataset 3. Insertion, deletion and mismatch rates in the table below are presented by the median values of the entire dataset. Coverage value refers to the average coverage of the entire raw dataset.

deletion and mismatch rates to 1%, while still maintaining a large coverage of the output error-corrected reads (25.85*x*). The error profile of Canu-corrected reads (Figure 3.3c) resembles the one obtained with Falcon error correction (Figure 3.3d). This is expected considering that Canu directly borrows components from the Falcon error-correction module.

To assess the influence of (I), we used error-corrected reads generated by Nanocorrect as the input data for Falcon and Miniasm for every dataset. We noticed that this procedure increased both the contiguity of Falcon's assembly and the average identity on all datasets (Appendix A, Table A.5). Increase in coverage of error-corrected reads provided a consistent increase of the quality of assembly in terms of largest contig length, average identity and number of variants. Although draft assemblies produced by the LQS pipeline exhibited a reduction in the size of the largest contig on Dataset 5, these assemblies also resulted in lower number of variants (SNPs and indels) compared to the Nanocorrect+Falcon combination. Miniasm benefitted from error-corrected reads as well, however, the difference in results is not as dramatic as for Falcon (Appendix A, Table A.6). Although the number of contigs for Datasets 1 and 2 increased when error-corrected reads were used, the total length of the generated contigs increased as well. Single contig full-genome assembly was achieved even on Dataset 3. The average identity of Nanocorrect+Miniasm assemblies is much higher than for Miniasm alone (and comparable to error-corrected datasets), which is expected as corrected reads are directly used to construct the contigs.

3.2.2 Hybrid pipeline comparison

Hybrid and non-hybrid assembly pipelines are not directly comparable because hybrid pipelines have an advantage in greater coverage supplied by Illumina reads. Table 3.4 gives a more de-



Insertion rate	Deletion rate	Mismatch rate	Match rate	Coverage
0%	1%	1%	98%	25.85x

(a) Nanocorrect (twice corrected).

Insertion rate	Deletion rate	Mismatch rate	Match rate	Coverage
1%	6%	3%	97%	20.99x

(b) PBcR.

Insertion rate	Deletion rate	Mismatch rate	Match rate	Coverage
0%	4%	1%	99%	22.17x

(c) Canu.

Insertion rate	Deletion rate	Mismatch rate	Match rate	Coverage
0%	3%	1%	99%	21.98x

(d) Falcon

Figure 3.3: Error rate analysis of error-corrected nanopore reads obtained with different error correction methods. Insertion, deletion and mismatch rates are presented by the median values of the entire dataset. Coverage value refers to the average coverage of the corresponding error-corrected dataset.

tailed comparison between two hybrid assemblers ALLPATHS-LG and SPAdes. Besides running both pipelines on Dataset 0 (paired-end and mate-pair reads) together with each nanopore dataset, SPAdes was also tested using only Illumina paired-end reads (without mate-pair reads). The table shows that ALLPATHS-LG produces better results than SPAdes on all datasets, from Dataset 0 without nanopore data for which SPAdes is not able to produce one sufficiently large contig, to Dataset 5 on which the difference is miniscule and apparent only in the number of SNPs and indels.

It is interesting to note that while ALLPATHS-LG requires both a paired-end and a mate-pair library to run, SPAdes seems not to be able to leverage mate-pair reads to a noticeable effect. In the presence of nanopore reads, results using paired-end Illumina library without mate-pairs seems to be equal to or even slightly better than with a mate-pair library, for all nanopore datasets. This means that in a situation where mate-pair reads are unavailable, SPAdes might be

Table 3.4: Comparing ALLPATHS-LG and SPAdes results.

Dataset	Assembler	# ctg.	N50	Genome fraction (%)	Avg. Identity 1-to-1	Total SNPs	Total Indels
0	ALLPATHS-LG	3	**4626283**	**99.219**	**99.99**	59	73
	SPAdes PE+MP	106	1105151	99.089	99.98	231	78
2	ALLPATHS-LG	2	**4639001**	**99.938**	**99.99**	**10**	**12**
	SPAdes PE only	20	4470699	99.908	99.99	430	93
	SPAdes PE+MP	18	4488904	99.912	99.98	427	110
3	ALLPATHS-LG	3	**4638937**	**99.938**	**99.99**	**6**	**38**
	SPAdes PE only	19	4474624	99.908	99.99	418	92
	SPAdes PE+MP	19	4474608	99.88	99.99	425	108
4	ALLPATHS-LG	1	**4638952**	**99.938**	**99.99**	**8**	**20**
	SPAdes PE only	20	4475777	99.908	99.99	401	66
	SPAdes PE+MP	20	4475770	99.88	99.99	399	73
5	ALLPATHS-LG	1	**4638958**	**99.938**	**99.99**	**3**	**5**
	SPAdes PE only	18	**4648869**	**99.918**	**99.99**	421	47
	SPAdes PE+MP	16	**4648863**	**99.918**	**99.99**	420	43

a good choice for a *de novo* assembler.

While none of the non-hybrid assemblers managed to produce a good assembly using Dataset 1 (Appendix A, Table A.2), both hybrid assemblers were able to use this dataset to improve their assembly. From Tables 3.3 and 3.4, it can be concluded that hybrid assembly pipelines achieve better results than non-hybrid ones. However, this is mostly because Illumina reads provide additional coverage of the genome.

3.2.3 Resource usage

To estimate efficiency of each assembly pipeline, NanoMark measures and reports *User time*, *System time*, *CPU time*, *Real time* (Wall clock time) and *Maximum memory usage* (Resident Set Size, RSS) for each assembly tool and dataset.

Table 3.5 shows CPU time and memory measurements with respect to sequencing coverage for each assembly pipeline. Miniasm proved to be by far the fastest of the tested assemblers, while LQS was, also by a large margin, the most time consuming. We note that, in general, error-correction of non-hybrid assemblers is the most time consuming step, which is especially evident in the LQS pipeline. Miniasm completely skips error-correction and consensus steps and is approximately three orders of magnitude faster than the second fastest non-hybrid tool (Canu) and two orders of magnitude faster than the fastest tested hybrid tool (SPAdes). All assembly pipelines consumed less than 20*GB* of memory, with Miniasm being the most conservative one.

Table 3.5: CPU time (hours) / Maximum memory usage (GB).

Assembler	**Coverage**			
	20	**30**	**40**	**50**
LQS	1086 / 4	2539 / 4	4438 / 4	8142 / 4
ALLPATHS-LG	13.2 / 18	26.0 / 18	44.9 / 17	144.5 / 20
PBcR	6.2 / 2	13.7 / 6	14.1 / 5	19.3 / 5
Falcon	3.1 / 6	6.4 / 10	19.7 / 10	13.8 / 13
SPAdes	0.9 / 5	1.0 / 5	1.1 / 5	1.2 / 5
Canu	5.33 / 3	11.2 / 4	28.7 / 4	9.39 / 4
Miniasm	0.009 / 2	0.015 / 2	0.026 / 3	0.044 / 4

3.2.4 Polishing the assembly

For every assembly result that contained a contig at least $4Mbp$ in length, we extracted that contig, and polished it using Nanopolish. The results were then compared to the reference using Quast and Dnadiff. The results of the analysis are shown in Table 3.6.

We can see that, without exception, Nanopolish will improve a non-hybrid assembly. Contig length (N50) will come closer to the reference length, average identity will increase while total number of SNPs and indels will decrease. On the other hand, the effect on hybrid assemblies is opposite. Contig length (N50) will usually decrease, average identity will always decrease, while total number of SNPs and indels will increase.

Although surprising at first, this result is expected if we consider that with hybrid assemblies Nanopolish is trying to improve contigs obtained from data with lower error rate (Illumina reads) using data with higher error rate (nanopore 2d reads).

3.2.5 Discussion

In this chapter we developed a benchmarking framework for *de novo* assembly tools focused on third generation sequencing data and compared several hybrid and non-hybrid *de novo* assemblers as well as assessed their ability to work with nanopore sequencing data. Each examined tool proved capable of assembling a whole bacterial genome under the right conditions. Needless to say, the choice of the best assembly tool will heavily depend upon the characteristics of the dataset. Keeping in mind that hybrid and non-hybrid assemblers are not directly comparable, we can say that ALLPATHS-LG showed overall the best results. However, it requires a rather specific set of Illumina paired-end and mate-pair short read libraries to perform the assembly, which might not always be practical to obtain. In case only paired-end reads are

Table 3.6: Quality assessment of polished assemblies.

Dataset	Assembler	Polish	N50	Genome fraction (%)	Avg. Identity	Total SNPs	Total Indels
2	Allpaths	YES	4638854	99.937	99.46	1012	24213
	Allpaths	NO	4639001	99.938	99.99	10	12
	SPAdes	YES	4489211	96.227	99.44	1011	23463
	SPAdes	NO	4488904	96.220	99.98	424	110
3	LQS	YES	4648870	99.997	99.47	1357	22622
	LQS	NO	4603990	99.998	98.49	4568	65283
	PBcR	YES	4615494	99.458	99.22	4460	31520
	PBcR	NO	4329903	12.825	94.03	7209	262357
	Canu	YES	4638184	99.783	99.28	3180	29607
	Canu	NO	4465231	88.644	95.77	5318	186843
	Allpaths	YES	4636408	99.936	99.53	790	21010
	Allpaths	NO	4638937	99.938	99.99	6	38
	SPAdes	YES	4472428	96.226	99.52	769	20291
	SPAdes	NO	4474608	96.220	99.99	424	108
4	LQS	YES	4664571	99.938	99.60	890	17712
	LQS	NO	4622531	99.938	99.08	2256	40118
	Falcon	YES	4643699	99.937	99.54	1829	19361
	Falcon	NO	4538244	99.938	97.66	3710	104165
	Canu	YES	4653892	99.934	99.58	1196	18012
	Canu	NO	4576679	99.915	98.42	812	71878
	Miniasm	YES	4667580	99.898	98.3	21331	58211
	Miniasm	NO	4577227	0.002	88.31	185194	326000
	Allpaths	YES	4647282	99.938	99.62	674	17168
	Allpaths	NO	4638952	99.938	99.99	8	20
	SPAdes	YES	4484185	96.223	99.61	655	16573
	SPAdes	NO	4475770	96.220	99.99	398	73
5	LQS	YES	4018309	86.570	99.80	453	7671
	LQS	NO	4006324	86.570	99.43	1237	21852
	Falcon	YES	4624811	99.654	99.78	834	9318
	Falcon	NO	4580230	99.655	98.84	2589	50662
	PBcR	YES	4639491	99.973	99.79	627	9131
	PBcR	NO	4596475	99.914	98.99	1136	45542
	Canu	YES	4631443	99.918	99.8	556	8547
	Canu	NO	4600945	99.786	99.29	415	32100
	Miniasm	YES	4696482	99.712	98.06	20395	70406
	Miniasm	NO	4774395	0.002	88.69	175279	328688
	Allpaths	YES	4637979	99.938	99.82	312	7859
	Allpaths	NO	4638958	99.938	99.99	3	5
	SPAdes	YES	4648653	99.929	99.82	343	7904
	SPAdes	NO	4648863	99.918	99.99	420	53

available, SPAdes might be a good choice. Of the non-hybrid assembly tools, on some datasets LQS pipeline came close to or even surpassed hybrid tools. However, extremely high CPU time used by Nanocorrect might make it prohibitively slow on larger genomes and larger datasets, in which case Canu, Falcon or PBcR could be used instead.

Miniasm must be considered apart from other assemblers. While the lack of error correction and consensus phases results in assemblies with a significant error rate, astonishing CPU efficiency makes it a perfect candidate for time-critical applications, especially if it could be enhanced with a suitably efficient consensus phase. Applying Nanopolish to Miniasm assemblies showed that they could be significantly improved, but at this point still fall behind other assemblies.

Polishing draft assemblies with Nanopolish improved the results of non-hybrid assemblies, but worsened the results of hybrid ones. Relative assembly quality remained the same, but after polishing the difference between hybrid and non-hybrid assemblies reduced substantially.

Chapter 4

Overlap

Exploiting the power of nanopore sequencing requires the development of new bioinformatics approaches to deal with its specific error characteristics. This includes the development of new methods for sequence assembly and sequence mapping/alignment.

As discussed in Chapter 2 Background, the problem of *overlapping* can also be viewed as a special case of *mapping*. In the general case of sequence mapping, mappers are presented two sets of sequences: reference sequences (e.g. chromosomes, or a genomic database) and read sequences. Overlapping of reads can be viewed as the process of mapping reads to themselves, i.e. the reference sequences *are also* the read sequences. In this case, one would only need to discard the self-hits - that is, perfect alignments of a sequence to itself. This definition of overlapping was the primary motivation to create a very sensitive and accurate mapper "GraphMap", which could then be reworked and used for raw sequence overlapping.

In this chapter, we present the development of a novel mapper for third generation sequencing data called GraphMap and it's modification for overlapping of raw third generation sequencing reads. The overlapping method is implemented as a submodule of GraphMap called "Owler". Both GraphMap and Owler are described, analyzed, thoroughly tested and compared to the state-of-the-art in the following sections.

4.1 GraphMap - Fast and sensitive mapping of nanopore sequencing reads

The GraphMap algorithm is structured to achieve high-sensitivity and speed using a five-stage 'read-funneling' approach as depicted in Figure 4.1. The underlying design principle is to have efficiently computable stages that conservatively reduce the set of candidate locations based on progressively defined forms of the read-to-reference alignment. For example, in stage I, GraphMap uses a novel adaptation of gapped spaced seeds [127] to efficiently reduce the search space (Figure 4.2) and then clusters seed hits as a form of coarse alignment (Figure 4.3). These

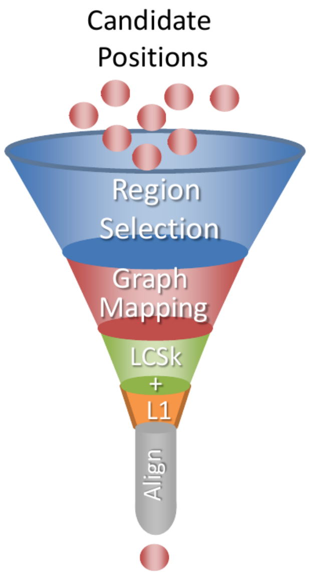


Figure 4.1: A schematic representation of stages in GraphMap. GraphMap refines candidate locations through stages and reduces the number of candidate locations to one.

are then refined in stage II using graph-based vertex-centric processing of seeds to efficiently (allowing seed-level parallelism) construct alignment anchors (Figure 4.4). GraphMap then chains anchors using a k-mer version of longest common subsequence (LCS) construction (stage III; Figure 4.5), refines alignments with a form of L_1 linear regression (stage IV; Figure 4.5) and finally evaluates the remaining candidates to select the best location to construct a final alignment (stage V). GraphMap computes a BLAST-like E-value as well as a mapping quality for its alignments.

4.1.1 Methods

Algorithm description

The GraphMap algorithm is presented in detail in Algorithm 1. It consists of five stages, each presented below.

Stage I: Region selection

GraphMap starts by roughly determining regions on the reference genome where a read could be aligned. This step is performed in order to reduce the search space for the next step of the algorithm, while still providing high sensitivity. As a first step, region selection relies on finding seeds between the query sequence and the reference, before clustering them into candidate regions. For seed finding, commonly used approaches such as maximal exact matches (MEMs) (as used in BWA-MEM [11]) or Hamming distance based spaced seeds [56] (as used in LAST [32]) were found to be either not sensitive enough or not specific enough in the presence of error rates as high as is feasible in nanopore data (e.g. see "Fixed seed k=13" for ONT 1D data in Appendix B, Table B.7). Instead, a form of gapped spaced seeds was employed, similar

Algorithm 1: GraphMap algorithm

```
    Input: Set of reference sequences $\mathcal{R}$, set of reads $\mathcal{Q}$, graph mapping node $k$-mer size $k$, number of edges
           per node $l$, error rate $e$, ambiguity factor $F$ and a parameter $P$ specifying the alignment type
           (semiglobal or anchored)
    Output: Set of alignments $\mathcal{A}$
    Function GraphMap($\mathcal{R}, \mathcal{Q}, k, l, e, F, P$) begin
1       $I_1 \leftarrow CreateGappedIndex(R, "1111110111111")$
2       $I_2 \leftarrow CreateGappedIndex(R, "11110111101111")$
3       $\mathcal{I} \leftarrow \{(I_1, "1111110111111"), (I_2, "11110111101111")\}$      ▷ A set of (index,shape) tuples
4       $\mathcal{A} \leftarrow \emptyset$
5       $lenR \leftarrow 0$
6       foreach $r \in \mathcal{R}$ do                                                 ▷ Calc. total reference len.
7           $lenR \leftarrow lenR + |r|$
8       foreach $q \in \mathcal{Q}$ do                                                 ▷ Process reads individually
9           $\mathcal{M} \leftarrow$ empty array                            ▷ Intermediate mapping locations
10          $\mathcal{G} \leftarrow RegionSelection(\mathcal{I}, \mathcal{R}, lenR, 0.75, q)$  ▷ An array of potential regions sorted by num. of hits
11          foreach $g \in \mathcal{G}$ do                                          ▷ Process each region individually
12              $\mathcal{W} \leftarrow GraphMapping(g, q, k, l)$
13              $\mathcal{W}_{lcsk} \leftarrow LCSk(W)$
14              $(c_{L1}, d_{int}) \leftarrow FitL1Line(\mathcal{W}_{lcsk}, |q|, e)$
15              if $P = semiglobal$ then
16                  $\mathcal{W}_{L1} \leftarrow L1Filtering(\mathcal{W}, c_{L1}, d_{int}, |q|, e)$
17                  $\mathcal{W}_a \leftarrow LCSk(\mathcal{W}_{L1})$                 ▷ Perform second LCSk to get anchors
18              else if $P = anchored$ then
19                  $\mathcal{W}_a \leftarrow FilterByChaining(\mathcal{W}_{lcsk}, e, 200, 2, 50)$        ▷ Get anchors by chaining
20              $(f, n_{cb}) \leftarrow CalcRegionQuality(\mathcal{W}_a, lenR, |q|, e)$  ▷ $f$ is region quality, $n_{cb}$ is num. cov. bases
21              $\mathcal{M} \leftarrow$ Append tuple $(g, \mathcal{W}_a, c_{L1}, d_{int}, f, n_{cb})$       ▷ Add an intermediate mapping
22          if $|\mathcal{M}| = 0$ then
23              continue
24          Sort $\mathcal{M} = [(g, \mathcal{W}_a, c_{L1}, d_{int}, f, n_{cb})]$ in descending order of $f$
25          $(\mathcal{W}_{a,best}, c_{L1,best}, d_{int,best}, f_{best}, n_{cb,best}) \leftarrow \mathcal{M}[0]$        ▷ Best scoring region
26          foreach $(g, \mathcal{W}_a, c_{L1}, d_{int}, f, n_{cb}) \in \mathcal{M}$ do                    ▷ Process each region individually
27              if $n_{cb} \geq (1 - F) \cdot n_{cb,best}$ then
28                  $\mathcal{A}_{region} \leftarrow Align(g, q, \mathcal{W}_a, c_{L1}, e, P)$
29                  $\mathcal{A} \leftarrow \mathcal{A} \cup \mathcal{A}_{region}$
30      return $\mathcal{A}$
```

to gapped q-gram filters for Levenshtein distance [127]. Specifically, the approach proposed in Burkhardt and Kärkkäinen [127] was extended to use both one- and two-gapped q-grams (Figure 4.2; Algorithm 1, lines $1-3$; Algorithm 2) as detailed below.

Gapped q-grams are a seeding strategy that allow for fast and very sensitive lookup of inexact matches, with variations allowed in predefined "don't care" (DC) positions of the seed. Consistent with existing terminology, the concrete layout of the inclusive and DC bases is referred to here as a shape and the number of used positions its weight. Gapped q-grams allow for DC positions within a shape to also contain insertions and deletions (indels). The approach in GraphMap for implementing Levenshtein gapped q-grams is based on constructing a hash index

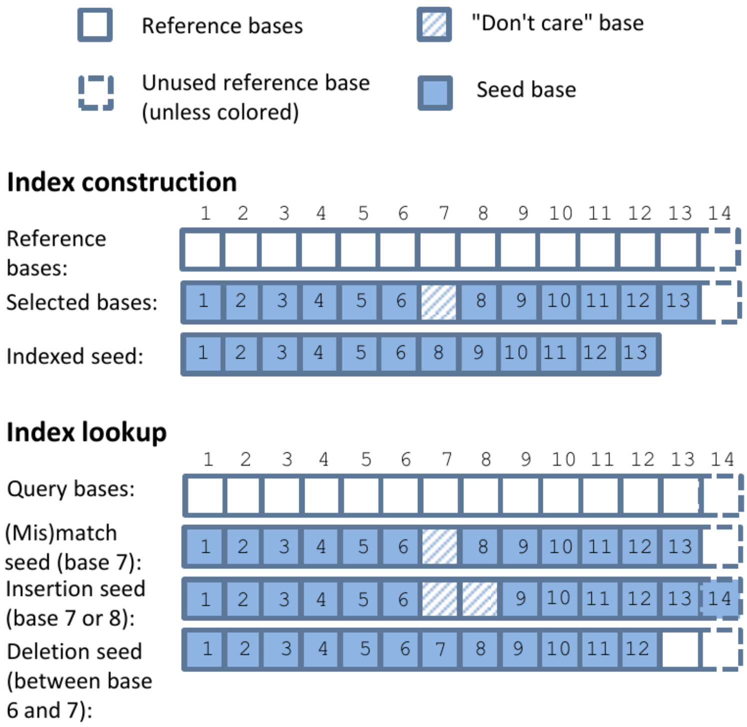


Figure 4.2: Structure of spaced seeds used for index construction and index lookup. For each position in the reference one seed is inserted into the index and for each position in the query, three seeds are looked up.

of the reference sequence, where the q-gram positions are hashed by the keys constructed from the shape's layout – only inclusive bases are taken for constructing the key, while the DC bases are simply skipped (Figure 4.2; Algorithm 2, lines $1-15$). During the lookup step, multiple keys are constructed for each shape and used for retrieval (Algorithm 2, lines $17-24$). For each DC base, three lookup keys are constructed:

1. A key constructed in the same manner as during the indexing process, which captures all seed positions and with a DC base being a match or a mismatch (e.g. "1111110111111"; see "(Mis)match seed" in Figure 4.2),
2. A key where the DC base is not skipped. This key captures up to one deletion (as indels are frequently 1bp long) at the specified position (e.g. "111111111111"; see "Deletion seed" in Figure 4.2), and
3. A key where the DC base as well as the following base is skipped. This key allows for at most one insertion and one match/mismatch (e.g. "11111100111111"; see "Insertion seed" in Figure 4.2).

In total, for each shape d^3 keys are constructed, where d is the number of DC bases. GraphMap uses two complementary shapes for the region selection process: "1111110111111" (or the 6-1-6 shape) and "11110111101111" (or the 4-1-4-1-4 shape), where 1 marks the inclusive bases and 0 the DC positions (Algorithm 1, lines $1-2$). This shape combination was selected based on empirical evaluation of a range of combinations, due to the computational

Algorithm 2: Gapped Spaced Index

```
Input: A set of sequences $\mathcal{R}$, and a shape $s$
Output: Index given as a hash table $\mathcal{I}$ where keys are seeds, and values are tuples of the seeds' originating
        sequence id and its position within that sequence $(t_{s_{id}}, t_{pos})$
Function CreateGappedIndex($\mathcal{R}, s$) begin
    $\mathcal{I} \leftarrow$ empty hash table
    for $i = 0$ to $(|R| - 1)$ do
        $r \leftarrow R[i]$
        for $j = 0$ to $(|r| - |s|)$ do                                ▷ Index the forward strand
            $h \leftarrow$ empty string
            for $k = 0$ to $(|s| - 1)$ do                              ▷ Remove "don't care" bases
                if $s[k] \neq 0$ then
                    $h \leftarrow$ Append $r[j+k]$
            $\mathcal{I}[h] \leftarrow \mathcal{I}[h] \cup (i, j)$
        for $j = 0$ to $(|\bar{r}| - |s|)$ do                          ▷ Index the reverse strand
            $h \leftarrow$ empty string
            for $k = 0$ to $(|s| - 1)$ do                              ▷ Remove "don't care" bases
                if $s[k] \neq 0$ then
                    $h \leftarrow$ Append $\bar{r}[j+k]$
            $\mathcal{I}[h] \leftarrow \mathcal{I}[h] \cup (i + |\mathcal{R}|, j)$
    return $\mathcal{I}$

Input: Index given as a hash table $\mathcal{I}$ of hashed seeds, a lookup shape $s$ and a raw seed $h_{raw}$ (still containing
       "don't care" (DC) bases)
Output: A set $\mathcal{T}$ containing hits in form of tuples $(t_{id}, t_{pos})$
Function CollectHits($\mathcal{I}, s, h_{raw}$) begin
    $\mathcal{T} \leftarrow$ empty set
    $\mathcal{S} \leftarrow$ Set of shapes for all combinations of DC bases in $s$
    for $i = 0$ to $(|S| - 1)$ do
        $h \leftarrow$ empty string
        for $j = 0$ to $(|S[i]| - 1)$ do                               ▷ Remove "don't care" bases
            if $S[i][j] \neq 0$ then
                $h \leftarrow$ Append $h_{raw}[j]$                      ▷ Lookup
        $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{I}[h]$
    return $\mathcal{T}$
```

intractability of computing the optimal shape for the Levenshtein distance [56][128] For each shape, a separate index is used in GraphMap. At every seed position, both shapes are looked up, and all hits are used in the next step for binning (Algorithm 1, line 10).

To derive a general approach for binning seed hits (Algorithm 3), we draw on the concept of a Hough Transform (HT), a method commonly used in image processing for detection of shapes such as lines, circles and ellipses. The HT defines a mapping from image points into an accumulator space, called the Hough space. In the case of line detection, if a given set of points in Cartesian space are collinear, then their relation can be expressed with a linear equation with

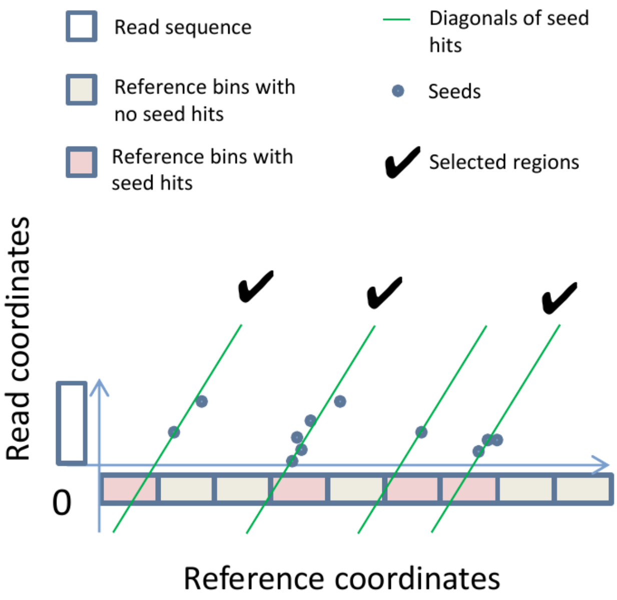


Figure 4.3: *Region selection* by clustering of candidate seeds on the reference. Diagonals with sufficient number of seed hits are used to identify regions for further processing.

common slope m and intercept c:

$$y = mx + c, \tag{4.1}$$

where (x,y) are the coordinates of a point in 2D space. HT attempts to determine parameters m and c of a line that describes the given set of points. Note that the system is generally overdetermined and thus the problem can be solved using linear regression techniques. However, the HT uses an evidence-gathering approach, which can be used to detect an arbitrary number of lines in the image instead of only one best (Figure 4.3). Equation 4.1 can be converted into its dual in parameter space:

$$c = -mx + y. \tag{4.2}$$

The intuition is as follows: given a point (x,y) in Cartesian space, its parameter space representation defines a line. If multiple Cartesian space points are given, each transforms into a different line in the parameter space. Their intersections specify potential lines in the original, Cartesian space. HT defines an accumulator space, in which m and c are rasterized so as to take only a finite range of values. HT then simply counts all the potential solutions in the accumulator space by tracing all the dual lines for each point in the Cartesian space, and increasing the vote count for each (m,c) coordinate. All HT space coordinates with count above a defined threshold can then be considered as candidate lines in the original Cartesian space.

A single seed hit can be represented with a "k-point" (q,t) in 2D space, where q is the seed's position on the read, and t is the position of the seed hit on the reference. In the case a read is completely error-free and extracted from the exact reference, its set of k-points would be perfectly collinear in such defined space. Moreover, under these ideal conditions, they would

Algorithm 3: Select most likely regions of the genome to contain the correct mapping

```
Input: A set $\mathcal{I}$ of $(gappedspacedindex, shape)$ tuples for different shapes, a set of reference sequences $\mathcal{R}$,
       total length of all reference sequences $lenR$, bin size $L$, fraction $p_{b_{max}}$ from the maximum bin count
       $b_{max}$ to output regions, and a read sequence $q$
Output: A set of regions $\mathcal{G}$
Function RegionSelection($\mathcal{I}, \mathcal{R}, lenR, L, p_{b_{max}}, q$) begin
1      $n_{bins} \leftarrow lenR/L$                                          ▷ Number of bins
2      $\mathcal{B} \leftarrow$ an array of size $n_{bins}$ with all values initialized to 0     ▷ Bin counts
3      $\mathcal{B}_u \leftarrow$ an array of size $n_{bins}$ with all values initialized to $-1$   ▷ Keeps track of last bin updates
4      $s_{max} \leftarrow$ empty string
5      foreach $(I, s) \in \mathcal{I}$ do                                   ▷ Find maximum length shape
6          if $|s| > |s_{max}|$ then
7              $s_{max} \leftarrow s$
8      for $i = 0$ to $(|q| - |s_{max}|)$ do                                 ▷ Process all k-mers for Hough Transform
9          foreach $(I, s) \in \mathcal{I}$ do
10             $h_{raw} \leftarrow q[i \cdots (i + s)]$
11             $\mathcal{T} \leftarrow CollectHits(I, s, h_{raw})$
12             foreach $(t_{id}, t_{pos}) \in \mathcal{T}$ do
13                 $c \leftarrow t_{pos} - i$                                ▷ Calculate the diagonal intercept
14                 $t_{bin} \leftarrow c/L$                                  ▷ Rasterize the intercept
15                 if $\mathcal{B}_u[t_{bin}] < i$ then                      ▷ Do not count same bin multiple times for same seed
16                     $\mathcal{B}[t_{bin}] \leftarrow \mathcal{B}[t_{bin}] + 1$     ▷ Increase the Hough Transform accumulator
17                     $\mathcal{B}_u[t_{bin}] \leftarrow i$
18     $\mathcal{G} \leftarrow \emptyset$
19     $r \leftarrow$ Concatenate all sequences in $\mathcal{R}$
20     $b_{max} \leftarrow \max(\mathcal{B})$
21     for $i = 0$ to $(|\mathcal{B}| - 1)$ do                               ▷ Generate regions
22         if $\mathcal{B}[i] > p_{b_{max}} \cdot b_{max}$ then
23             $r_{start} \leftarrow \max(i \cdot L - |q|, 0)$
24             $r_{end} \leftarrow \min((i + 1) \cdot L + |q|, |r|)$
25             $g \leftarrow r[r_{start} \cdots r_{end}]$
26             $\mathcal{G} \leftarrow \mathcal{G} \cup g$
27     Sort $\mathcal{G}$ in descending order of corresponding number of bin hits in $\mathcal{B}$
28     return $\mathcal{G}$
```

all lie on a line tilted at a 45 angle (slope $m = 1$). This collinearity also corresponds to the main diagonal in the dynamic programming alignment matrix. Since m is known, only the intercept parameter c needs to be determined to find the accurate mapping position. As c corresponds to the (already discrete) coordinates on the reference sequence, a simple integer array of the length of the reference can be used for counting votes (Figure 4.3). For each k-point, its c parameter value is determined with a simple expression (Equation 4.3; Algorithm 3, line 15):

$$c = t - q. \tag{4.3}$$

The index of the accumulator array with the highest count is the exact mapping position of the read on the reference. In this simple form, this approach mirrors the techniques used in other aligners (e.g. FASTA). However, the concept of the Hough Transform (HT) allows us to extend and generalize this notion.

We account for substitution and indel errors in this framework as follows: substitution errors cause only the reduction in the maximum vote count for the correct c value and induce noise votes in other locations on the reference. Such type of errors can be addressed using appropriate thresholding on the hit count (see below). On the other hand, indels are of special interest because they shift the alignment diagonal and cause more substantial reduction of votes for the correct location. Additionally, using an accumulator array that is of size equal to the size of the reference sequence can cause high memory consumption, especially in the case of processing large sequences in multithreaded environments.

To address both the error-rate and memory consumption issues, GraphMap rasterizes the reference sequence into partitions of length $L/3$ (where L is the read length), so that at least one partition is fully covered by the read (Algorithm 3, lines 1 and 14). For each seed hit to a bin, it increments the value of the bin corresponding to its c parameter value determined using Equation 4.3. Bins are then sorted in descending order of the number of hits (Algorithm 3, line 27). To limit the search to the most likely bins, only bins with count greater than 75% of the max count are selected for further processing (Algorithm 3, line 22). A *region* is then defined as a portion of the reference that expands the corresponding bin's start and end location by an additional read length, to compensate for potential indel errors and ensure that the entire alignment area enters the next step of mapping (Algorithm 3, lines $23-26$). In the case that the reference genome is specified as being circular by the user, GraphMap allows the region to be constructed by concatenating the beginning and the end of the reference. Regions are then processed separately until the last step of the method, when the highest scoring region is selected for alignment Algorithm 1, line 11).

Stage II: Graph-based vertex-centric construction of anchors

In this stage (Algorithm 4, line 12), candidate regions from stage I are refined by constructing alignment chains or anchors from short seed matches. To do this, GraphMap uses the notion of a "k-mer mapping graph". Given a pair of sequences (target and query), it starts by constructing a k-mer mapping graph from the target sequence (Algorithm 4, line 2). In the current implementation, the read was chosen to be the target sequence in order to reduce memory consumption. Also, the process of graph mapping is performed between a single read and multiple regions, and this organization reduces the overhead of graph construction. The vertices of the k-mer mapping graph are the k-mers of the target sequence of length T (Figure 4.4). Unlike in a de Bruijn graph, identical k-mers are not truncated into the same vertex of the graph but are kept as separate individual vertices (Figure 4.4). For every vertex $v_i, (\forall i \in (0...T-k))$, l directed outbound edges are added which connect v_i to vertices $v_{i+1}, v_{i+2}, ..., v_{i+l}$ (Figure 4.4). The rationale for such a design is as follows: in case $l = 1$ and if the query is a subset of the

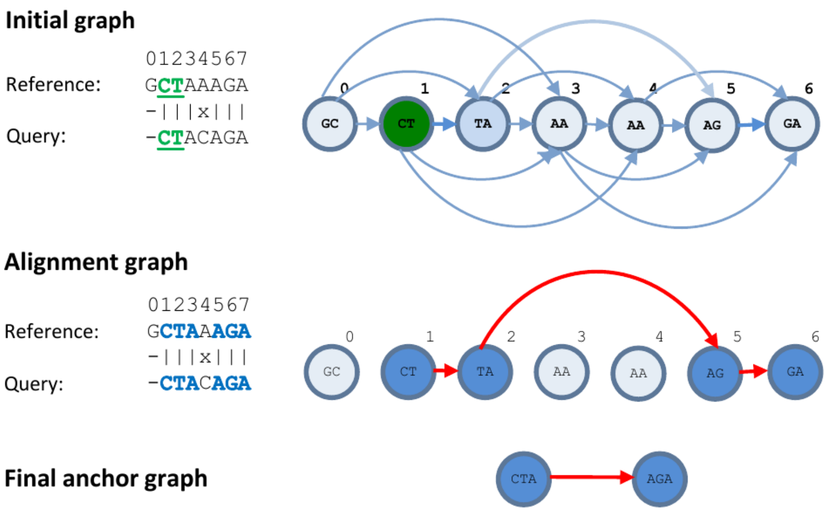


Figure 4.4: Generating alignment anchors through a fast graph based ordering of seeds (*Graph Mapping*). Seeds from the query (2-mers here; starting from the green seed) are looked up, and information in the graph propagated, to construct a maximal walk that serves as an anchor.

target with no differences or errors, the target's mapping graph would contain the same k-mers in the exact same order as in the query sequence. Thus, an exact walk exists in both sequences. However, in realistic conditions, variations and sequencing errors exist in reads. Although the majority of k-mers might still be in the same order, a simple exact linear walk through the reference's and read's mapping graphs cannot be found due to the differing k-mers present. Instead, the walk is fragmented into several smaller ones and this is particularly severe when the error rate is high, as seen in nanopore sequencing. To address this issue, the additional $(l-1)$ edges act as a bridge between vertices in the mapping graph. Thus GraphMap allows a linear walk to be found not only by following consecutive k-mers in the graph, but to jump-over those that produce poorer solutions. Figure 4.4 depicts such an example. GraphMap uses $l = 9$ by default as it was empirically found to enable anchor construction for most ONT reads.

For graph construction, GraphMap uses an index constructed from the target on the fly, using a smaller continuous seed for sensitivity (default $k = 6$, similar to the k-mer used for MinION base-calling) (Algorithm 4, lines $4-6$). In principle, any indexing method can be used and for runtime efficiency GraphMap uses perfect k-mer hashing when $k < 10$ and suffix arrays otherwise. To do graph traversal, for each consecutive k-mer in the query, a list of hits on the target sequence is obtained from the index (Algorithm 4, line 9). The vertex-centric walk then works as follows: for a chosen vertex, collect information from input edges (Algorithm 4, lines $12-16$), choose the "best" edge (Algorithm 4, line 15) and update the information it contains (Algorithm 4, lines $17-29$), and transmit this information to all outbound edges simultaneously (this is performed implicitly by modifying the vertex's information). The "best" edge is defined here to be the one belonging to the longest walk. The information that is transmitted through the edges contains the walk length, the position of the starting k-mer in both the target and the read,

Algorithm 4: Graph Mapping. Constructing a k-mer graph from sequence q and mapping g in a fast vertex-centric manner. Graph is implemented as an array of size q since it is a DAG and all in-edges for a node are its direct predecessors.

```
Input: Region sequence $g$, read sequence $q$, graph mapping $k$-mer size, the number of out edges $l$ for each
       node of the graph and the minimum number of covered bases per walk (anchor) $m$
Output: An array of walks $\mathcal{W}$ on the graph representing matching subsequences between $q$ and $g$, where
        each walk is a tuple containing start and end locations in both sequences $(q_s, q_e, g_s, g_e)$
Struct VERTEX                                          ▷ Vertex data structure with default values begin
    $ts \leftarrow -1$                                 ▷ Timestamp of the last vertex update
    $g_s \leftarrow 0$                                 ▷ Start position of a walk on g
    $g_e \leftarrow 0$                                 ▷ End position of a walk on g
    $q_s \leftarrow 0$                                 ▷ Start position of a walk on q
    $q_e \leftarrow 0$                                 ▷ End position of a walk on q
    $n_k \leftarrow 0$                                 ▷ Number of k-mers covered by a walk to this vertex
    $n_{cb,g} \leftarrow 0$                            ▷ Number of bases on g covered by k-mers to this vertex
    $n_{cb,q} \leftarrow 0$                            ▷ Number of bases q covered by k-mers to this vertex
    $reg \leftarrow -1$                                ▷ Registry ID of the walk containing this vertex
Function GRAPHMAPPING$(g, q, k, l, m)$ begin
    $\mathcal{W} \leftarrow$ empty array               ▷ Graph walks (anchors)
    $graph \leftarrow$ array of Vertex objects of length $|q|$ initialized to default
    $\mathcal{H} \leftarrow \emptyset$                 ▷ Hash table of k-mers from q
    for $i = 0$ to $(|q| - k)$ do                      ▷ Index all short k-mers in a hash
        $kmer \leftarrow q[i...(i+k)]$
        $\mathcal{H}[kmer] \leftarrow \mathcal{H}[kmer] \cup i$      ▷ Add the position to the set
    for $i = 0$ to $(|g| - k)$ do                      ▷ Process all k-mers from g
        $kmer \leftarrow g[i...(i+k)]$
        $hits \leftarrow \mathcal{H}[kmer]$
        foreach $p \in hits$ do                        ▷ p is also the vertex ID of a hit
            $v \leftarrow$ NULL     ▷ "Best" vertex (part of longest reachable walk), default "not found" value
            for $j = \max(p - l, 0)$ to $(p - 1)$ do   ▷ Check nodes on l inbound edges from p and find max
                if $graph[j].ts \leq i$ and
                $0 < (graph[p].ts - graph[j].ts) \leq l$ then      ▷ Check if walk on j was too long ago
                    if $v = NULL$ or
                    $graph[j].n_{kmers} > v.n_{kmers})$ then       ▷ Find longest previous reachable walk
                        $v \leftarrow graph[j]$
                        $d_q \leftarrow \min(p - j, k)$            ▷ Distance of current to best vertex in q coords
            if $v = NULL$ then                         ▷ No valid walks were found, start a new one on p
                With $graph[p]$:
                    $(ts, g_s, g_e, q_s, q_e, n_k, n_{cb,g}, n_{cb,q}, reg) \leftarrow (i, i, i+k, p, p+k, 1, k, k, -1)$
            else                                       ▷ Take the longest reachable walk and expand it with current vertex
                $d_g \leftarrow \min(i - v.g_s, k)$            ▷ Distance of current to best vertex in g coords
                With $graph[p]$:
                    $(ts, g_s, g_e, q_s, q_e, n_k, n_{cb,g}, n_{cb,q}, reg) \leftarrow (i, v.g_s, i+k, v.q_s, p+k, v.n_k + 1, d_g, d_q, v.ts)$
                if $graph[p].n_{cb,g} > m$ and $graph[p].n_{cb,q} > m$ then     ▷ Register the walk if long enough
                    if $graph[p].reg = -1$ then        ▷ If the walk has not yet been registered
                        $graph[p].reg \leftarrow |\mathcal{W}|$
                        $\mathcal{W} \leftarrow$ Append $graph[p]$;
                    else
                        $\mathcal{W}[graph[p].reg] \leftarrow graph[p]$;
    return $\mathcal{W}$
```

and the number of covered bases and k-mers in both sequences. Thus the runtime complexity of the vertex-update operation is $O(1)$.

After all k-mers from the query have been processed, a list of walks in the graph is collected (Algorithm 4, lines $27-30$). Walks that are too short (default < 12 bases i.e. smaller than the seeds from stage I) are excluded to avoid a large search space (Algorithm 4, line 24). Vertex-centric walks allow GraphMap to quickly construct longer alignments in the presence of higher substitution error rates, as seen in nanopore sequencing data. In the presence of low substitution error rates ($< 2\%$, as is the case for Illumina as well as PacBio reads), a single walk can cover most of, if not the entire read. For ONT reads we observed shorter walks that we refer to here as anchors (Figure 4.4).

Stage III: Extending anchors into alignments using LCSk

Each anchor reported by GraphMap in stage II represents a shared segment (or subsequence) between the target and the query sequence with known start and end positions in both sequences. Due to the presence of repeats, the set of obtained anchors is not necessarily monotonically increasing in both the target and query coordinates. For this reason, a subset of anchors that satisfy the monotonicity condition needs to be selected. The problem of identifying such a subset can be expressed as finding the *Longest Common Subsequence in k Length Substrings* [129] (LCSk). Note that this is distinct from just finding the longest common subsequence (LCS) as that ignores the information determined in the anchors and can favor alignments that have many more indels. Recently, an efficient and simple algorithm for solving a variant of the LCSk problem has been proposed [130]. In our implementation we follow this paradigm (Algorithm 1, lines 13 and 17; Algorithm 5), but instead of using substrings of fixed size k, we allow for variable length substrings. Concretely, the size of each substring is equal to the length of the corresponding anchor (Algorithm 5, line 16).

The LCSk construction presented in Algorithm 5 takes in a list of anchors $\mathcal{W}$ defined by their start and end positions in both sequences. Start and end points are observed as individual events (Algorithm 5, lines $2-8$), sorted in ascending order of their coordinates. For a given coordinate (x,y), LCSk finds the maximum of a previously solved LCSk for all events with coordinates (i,j) such that $0 \leq i < x$ and $0 \leq j < y$. Such a maximum can be found very efficiently using a Fenwick tree (Algorithm 6), as proposed in [130]. Fenwick tree data structure encodes partial solutions (such as sums or max/min values) in an array at locations predetermined by bits of the currently analyzed array index position y, and implements two operations: *get* and *update*. The non-zero bits encode predecessors in the tree, which are looked-up and compared (in case finding a max/min value) or summed (in case of finding a running sum). Fenwick has $O(\log n)$ complexity for the *get* operation (compared to a $O(n)$ naive solution for calculating a sum of numbers) (Algorithm 6, lines $3-8$), but also has an $O(\log n)$ complexity for the *update* operation (Algorithm 6, lines $9-12$). Initial Fenwick tree construction takes $O(n \log n)$ time.

Using the Fenwick data structure, LCSk procedure in Algorithm 5 then processes each event in sorted order of their coordinates (Algorithm 5, line 13), and finds a previous maximum result using Fenwick *get* (Algorithm 5, line 15). If such a result does not exist, a new value is initialized (Algorithm 5, lines $17-19$), otherwise the retrieved result is extended (Algorithm 5, lines $21-22$). For a particular anchor, Fenwick is updated and traceback recorded only when the end point of the anchor is reached (Algorithm 5, lines $24-27$). Finally, the LCSk is reconstructed from the traceback (Algorithm 5, lines $28-33$).

As a result, the reconstruction of LCSk is obtained in the form of a list of consecutive anchors in the target and the query sequence. The LCSk stage was observed to be key to GraphMap's ability to construct approximate alignments that help identify the correct mapping location. Removing this stage reduced GraphMap's precision and recall by $10-30\%$ without significantly affecting its runtime or memory usage.

Stage IV: Refinining alignments using L1 linear regression

The alignments obtained using LCSk tend to be largely accurate but since its definition lacks constraints on the distance between substrings, the alignments obtained may include outlier matches and incorrect estimation of overall alignment length (Figure 4.5). These outliers are caused by repeats or sequencing errors, but they still satisfy the monotony condition. Similar to the observation presented for region selection, the LCSk list of anchors should ideally be collinear in the 2D query-target coordinate space, with a slope of 45. All deviations from this line are caused by indel errors, and can be viewed as noise. The filtering of outlier anchors begins by fitting a 2D line with a 45 slope in the query-target space under the *least absolute deviation criteria* (LAD, L_1) (Algorithm 1, line 14; Algorithm 7). Next, a subset of anchors which are located within $d_{L1} = eT\sqrt{2}/2$ from either side of the L_1 line is selected, where e is the expected error rate (by default, conservatively set to 45%), T is the target (read) length, and the factor $\sqrt{2}/2$ is used to convert the distance from target coordinate space to a distance perpendicular to the L_1 line. A confidence interval $d_{int} = 3\sum_{i=1}^{N} d_i/N$ is calculated, where d_i is the distance from a selected anchor i to the L_1 line (the constant 3 was chosen to mimic a 3σ rule) (Algorithm 7, line 13). LCSk is then repeated once again but only on the anchors which are located within the distance $\pm d_{int}$ from the L_1 line in order to compensate for possible gaps caused by anchor filtering (Figure 4.5; Algorithm 1, lines $16-17$; Algorithm 8). The use of L_1 filtering was observed to improve the precision of alignment start and end coordinates for many reads, though the overall impact on performance was less significant in comparison to the LCSk stage.

After filtering, five empirically derived scores that describe the quality of the region are calculated. They include: the number of exact k-mers covered by the anchors n_{kmers}, the confidence interval d_{int} of anchors around the L_1 line, the length of the query sequence which matched the target (distance from the first to the last anchor) m_{len}, the number of bases covered by anchors

Algorithm 5: Performing the Longest Common Subsequence in k-length Substrings to filter a set of graph mapping walks. The original LCSk was modified to support variable length substrings instead of fixed length k

```
Input: A set $\mathcal{W}$ of tuples $(q_s, q_e, g_s, g_e)$, where $q_s$ and $q_e$ are start and end positions in sequence $q$, and $g_s$
       and $g_e$ are start and end positions in sequence $g$
Output: A set of tuples $\mathcal{W}_{lcsk}$ extracted from $\mathcal{W}$ which produce the best LCSk score
Function LCSK($\mathcal{W}$) begin
    $n \leftarrow 0$
    $\mathcal{E} \leftarrow$ empty array of tuples $(x, y, w_{id}, m)$                ▷ Events
    for $i = 0$ to $(|\mathcal{W}| - 1)$ do                                            ▷ Create events
        $(q_s, q_e, g_s, g_e) \leftarrow \mathcal{W}[i]$
        $\mathcal{E} \leftarrow$ Append $(q_s, g_s, i, BEGINNING)$              ▷ Separate start and end points
        $\mathcal{E} \leftarrow$ Append $(q_e, g_e, i, ENDING)$
        $n \leftarrow \max(n, q_e, g_e)$               ▷ Maximum coordinate value needed to init Fenwick
    Sort $\mathcal{E}$ in ascending order of tuples, $ENDING < BEGINNING$
    $\mathcal{F} \leftarrow Fenwick(n + 1)$            ▷ Data struct to keep max. previous value for every point
    $traceback \leftarrow$ array of size $|\mathcal{W}|$ with values initialized to $-1$
    $best \leftarrow 0$
    $dp_{max} \leftarrow 0$
    foreach $(x, y, w_{id}, m) \in \mathcal{E}$ do                                 ▷ p is also the vertex ID of a hit
        if $m = BEGINNING$ then
            $(prev_{dp,val}, prev_{dp,id}) \leftarrow FenwickGet(\mathcal{F}, y)$      ▷ Get the max. previous value and ID
            $k \leftarrow \mathcal{W}[w_{id}].g_e - \mathcal{W}[w_{id}].g_s$              ▷ Length of an anchor in y coordinates
            if $prev_{dp,val} = -1$ then                            ▷ There are no predecessors, i.e. first anchor
                $dp[w_{id}] \leftarrow k$
                $traceback[w_{id}] \leftarrow -1$
            else                                                      ▷ Found a predecessor, link traceback
                $dp[w_{id}] \leftarrow prev_{dp,val} + k$
                $traceback[w_{id}] \leftarrow prev_{dp,id}$
        else
            $FenwickUpdate(\mathcal{F}, y, (dp[w_{id}], id))$       ▷ Store the new value and update the maximums
            if $dp[w_{id}] > dp_max$ then
                $dp_{max} \leftarrow dp[w_{id}]$
                $best \leftarrow w_{id}$
    $\mathcal{W}_{lcsk} \leftarrow \emptyset$
    $p \leftarrow best$
    while $p! = -1$ do                                           ▷ Create the reconstruction (traceback)
        $\mathcal{W}_{lcsk} \leftarrow \mathcal{W}_{lcsk} \cup \mathcal{W}[p]$
        $p \leftarrow traceback[p]$
    $\mathcal{W}_{lcsk} \leftarrow$ Reverse order of elements in $\mathcal{W}_{lcsk}$
    return $\mathcal{W}_{lcsk}$
```

(includes only exact matching bases) n_{cb} and the read length. The last four scores are normalized to the range $[0, 1]$ with the following Equations 4.4-4.7 (Algorithm 1, line 20; Algorithm 9; Algorithm 10).

$$f_{d_{int}} = \max(0, 1 - \frac{d_{int}}{\frac{eT}{\sqrt{2}}}), \tag{4.4}$$

Algorithm 6: Fenwick tree data structure for fast retrieval of maximum previous values, used by the LCSk algorithm. Given an index i of a value a_i in an array of n elements, $[a_0, a_1, ..., a_{n-1}]$, FenwickMax provides two operations: $get(i)$ to find the index of the maximum value of $a_j | 0 \leq j < i$, and $update(i, v)$ to change the value of $a_i \leftarrow v$ and update the maximum for all $a_j | i \leq j < n$.

```
Input: Number of elements $n$ to initialize the Fenwick tree
Output: An array $\mathcal{F}$ of Fenwick-ordered tuples $(val, id)$
Function FENWICK($n$) begin
    $\mathcal{F} \leftarrow$ array of tuples $(val, id)$ of length $(n+1)$        ▷ Keeps predecessor info
    return $\mathcal{F}$

Input: An array $\mathcal{F}$ of Fenwick-ordered tuples $(val, id)$ and a coordinate $x$ in range of $[0, |\mathcal{F}|]$ for which to
       find the maximum predecessor
Output: A tuple of $(val, id)$ of the maximum predecessor
Function FENWICKGET($\mathcal{F}, x$) begin
    $x \leftarrow x + 1$
    $t \leftarrow$ empty tuple of $(0, 0)$
    while $x > 0$ do                                                  ▷ Get maximum value left of x
        $t \leftarrow \max(t, \mathcal{F}[x])$
        $x \leftarrow x - (x \& -x)$                  ▷ Remove the last non-zero bit, "&" is bitwise here
    return $t$

Input: An array $\mathcal{F}$ of Fenwick-ordered tuples $(val, id)$ and a coordinate $x$ in range of $[0, |\mathcal{F}|]$ for which to
       update the data structure, and a tuple $t$ of $(val, id)$ which will be placed at $\mathcal{F}[x]$
Function FENWICKUPDATE($\mathcal{F}, x, t$) begin
    $x \leftarrow x + 1$
    while $x < |\mathcal{F}|$ do                                      ▷ Update all values from x to the right
        $\mathcal{F}[x] \leftarrow \max(\mathcal{F}[x], t)$
        $x \leftarrow x + (x \& -x)$
```

Algorithm 7: Fitting a 45° line under the L_1 criteria.

```
Input: A set $\mathcal{W}$ of graph mapping walks specified by query start and end, and region start and end
       coordinates respectively; length of the read sequence $|q|$ and error rate $e$
Output: Intercept of the 45° $L_1$-fitted line $c_{L1}$ and confidence interval $d_{int}$ of walk distances around the $L_1$
        line
Function FITL1LINE($\mathcal{W}, |q|, e$) begin
    $\mathcal{C} \leftarrow$ empty array
    foreach $(q_s, q_e, g_s, g_e) \in \mathcal{W}$ do                  ▷ Calculate all intercepts (diagonals)
        $\mathcal{C} \leftarrow$ Append $(g_s - q_s)$
    $c_{L1} \leftarrow median(\mathcal{C})$                  ▷ Intercept (diagonal) of the $L_1$ line
    $d_{L1} \leftarrow e \cdot |q|\sqrt{2}/2$                   ▷ Maximum dist from $L_1$ line
    $\mathcal{W}_d \leftarrow$ empty array
    foreach $(q_s, q_e, g_s, g_e) \in \mathcal{W}$ do                                      ▷ Process walks
        $c \leftarrow (g_s - q_s)$
        $d \leftarrow |(c - c_{L1}) \cdot \sqrt{2}/2|$             ▷ Distance from the median line
        if $d \leq d_{L1}$ then                    ▷ Considering only hits within $d_{L1}$ to ignore big outliers
            $\mathcal{W}_d \leftarrow$ Append $d$
    $d_{int} \leftarrow 3 \cdot \sum_{i=1}^{|\mathcal{W}_d|} |\mathcal{W}_d[i]| / |\mathcal{W}_d|$      ▷ Calculate the confidence interval only on hits within $d_{L1}$
    return $(c_{L1}, d_{int})$
```

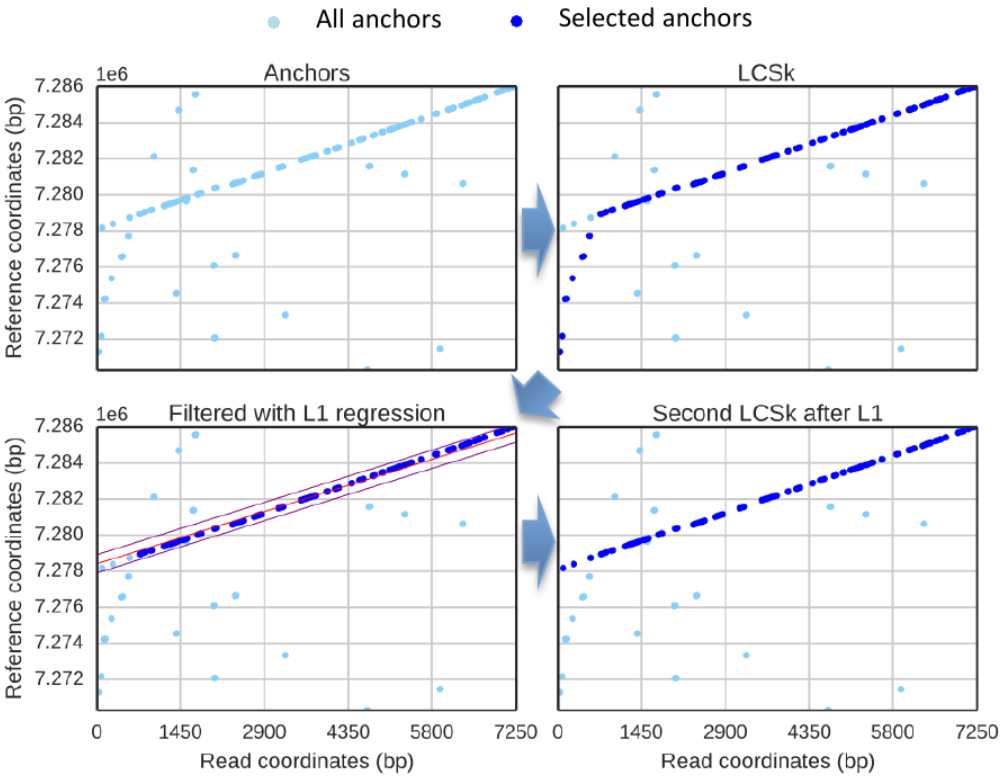


Figure 4.5: Filtering of seed matches using LCSk and L_1 regression. Anchors are chained into a monotonically increasing sequence, with outliers trimmed using L_1 regression, to get an approximate alignment.

Algorithm 8: Filtering walks using L_1 linear regression

Input: A complete set of graph mapping walks $\mathcal{W}$, intercept of the L_1 line c_{L1}, confidence interval d_{int} of walk distances around the L_1 line, length of the read sequence $|q|$ and error rate e

Output: A set of walks $\mathcal{W}_{L1}$ from $\mathcal{W}$ within L_1

```
Function L1FILTERING($\mathcal{W}, c_{L1}, d_{int}, |q|, e$) begin
    $\mathcal{W}_{L1} \leftarrow \emptyset$
    foreach $(q_s, q_e, g_s, g_e) \in \mathcal{W}$ do
        $c \leftarrow (g_s - q_s)$
        $d \leftarrow |(c - c_{L1}) \cdot \sqrt{2}/2|$        ▷ Distance from the median line
        if $d \leq d_{int}$ then
            $\mathcal{W}_{L1} \leftarrow \mathcal{W}_{L1} \cup (q_s, q_e, g_s, g_e)$
    return $\mathcal{W}_{L1}$
```

$$f_{m_{len}} = \frac{m_{len}}{T}, \tag{4.5}$$

$$f_{cb} = \min\left(\frac{n_{cb}}{m_{len}(1-e)}, 1\right), \tag{4.6}$$

$$f_T = \min\left(\frac{lnT}{lnQ}, 1\right), \tag{4.7}$$

where Q is the length of the reference sequence (query in our previous definition). The overall quality of the alignment in a region is then calculated as the product of the normalized scores (Algorithm 10, line 10):

$$f = f_{d_{int}} f_{m_{len}} f_{cb} f_T. \tag{4.8}$$

Algorithm 9: Calculate the number of bases covered by anchors

```
Input: A set of anchors $\mathcal{W}$
Output: Number of bases covered by anchors $n_{cb}$ for the given region
Function CalcCoveredBases($\mathcal{W}$) begin
    $n_{cb} \leftarrow 0$
    foreach $(q_s, q_e, g_s, g_e) \in \mathcal{W}$ do
        $n_{cb} \leftarrow n_{cb} + (q_e - q_s)$
    return $n_{cb}$
```

Algorithm 10: Calculate quality measures for a region

```
Input: A set of anchors $\mathcal{W}_a$, total length of the reference sequences $lenR$, length of the read $|q|$, confidence
       interval $d_{int}$ of anchor distances around the $L_1$ line and error rate $e$
Output: Quality factor $f$ and the number of bases covered by anchors $n_{cb}$ for the given region
Function CalcRegionQuality($\mathcal{W}_a, lenR, |q|, d_{int}, e$) begin
1       Sort $\mathcal{W}_a = [(q_s, q_e, g_s, g_e)]$ in ascending order of $q_s$
2       $(q_{s,min}, q_{e,min}, g_{s,min}, g_{e,min}) \leftarrow \mathcal{W}_a[0]$
3       $(q_{s,max}, q_{e,max}, g_{s,max}, g_{e,max}) \leftarrow \mathcal{W}_a[|\mathcal{W}_a| - 1]$
4       $m_{len} \leftarrow (q_{e,max} - q_{s,min})$                    ▷ Calc. maximum span of anchors
5       $n_{cb} \leftarrow CalcCoveredBases(\mathcal{W}_a)$
6       $f_{d_{int}} \leftarrow \max(0, 1 - d_{int}/(e \cdot |q|/\sqrt{2}))$
7       $f_{m_{len}} \leftarrow m_{len}/|q|$
8       $f_{n_{cb}} \leftarrow \min(n_{cb}/(m_{len} \cdot (1 - e)), 1)$
9       $f_{|q|} \leftarrow \min(\ln(|q|)/\ln(lenR), 1)$
10      $f \leftarrow f_{d_{int}} \cdot f_{m_{len}} \cdot f_{n_{cb}} \cdot f_{|q|}$
11      return $(f, n_{cb})$
```

Stage V: Construction of final alignment

After all selected regions have been processed they are sorted by the f value. The region with the highest value f_{max} is selected for the final alignment. The default settings for GraphMap use an implementation of Myers' bit-vector algorithm for fast alignment [38]. GraphMap also allows users a choice of aligners, including an implementation of Gotoh's semi-global alignment algorithm [25], as well as an option to construct anchored alignments (Algorithm 1, lines 19 and 28; Algorithm 11; Algorithm 12). Specifically, in the anchored approach, anchors from the LCSk step are chained and clustered, and alignments within and between cluster endpoints computed using Myers' bit-vector alignment (extensions to read ends are done in a semiglobal manner). Clustering is done by collecting neighbouring anchors where the distance between two anchors is less than a user defined threshold $p_{q,max}$ (Algorithm 11, lines 10 and 13), and the maximum distance between the two anchor's diagonals (indel gap) is less than $p_{i,max} = e \cdot d$, where e is the expected error rate in the data and d is the distance between the endpoints of the two anchors (Algorithm 11, lines $11-13$). Clusters with very few bases (Algorithm 11, lines $20-26$) were discarded for this purpose as they were found to reduce alignment accuracy.

GraphMap allows users to output all equally or similarly good secondary alignments by specifying an ambiguity factor F in the range $[0,1]$ and using that to select regions which have $n_{cb} \geq (1-F)n_{cb,best}$, where $n_{cb,best}$ is the number of bases covered by anchors in the region with the maximum f value (Algorithm 1, lines $24-29$). We denote the count of regions with n_{cb} above the ambiguity threshold as N_a.

Algorithm 11: Filtering walks by chaining

```
Input: A complete set of graph mapping walks $\mathcal{W}$, maximum expected error rate $e$, maximum allowed
       distance $p_{q,max}$ between walks to chain them, minimum number of walks $c_{min\_walks}$ and minimum
       number of covered bases $c_{min\_cb}$ in a chain to retain the chain
Output: A set of walks $\mathcal{W}_{chained}$

Function FILTERBYCHAINING($\mathcal{W}, e, p_{q,max}, c_{min\_walks}, c_{min\_cb}$) begin
1     Sort $\mathcal{W} = [(q_s, q_e, g_s, g_e)]$ in ascending order of parameters
2     $\mathcal{C} \leftarrow$ empty array                              ▷ Single chain
3     $\mathcal{C}_{all} \leftarrow$ empty array                        ▷ An array of all chains
4     for $i = 0$ to $(|\mathcal{W}| - 1)$ do
5         if $|\mathcal{C}| = 0$ then
6             $\mathcal{C} \leftarrow$ Append $\mathcal{W}[i]$
7             continue
8         $(q_{s,i}, q_{e,i}, g_{s,i}, g_{e,i}) \leftarrow \mathcal{W}[i]$
9         $(q_{s,i+1}, q_{e,i+1}, g_{s,i+1}, g_{e,i+1}) \leftarrow \mathcal{W}[i+1]$
10        $p_q \leftarrow q_{s,i+1} - q_{e,i}$            ▷ Distance between the walks in q coordinates
11        $p_i \leftarrow |(q_{e,i} - g_{e,i}) - (q_{s,i+1} - g_{s,i+1})| \cdot \sqrt{2}/2$            ▷ Distance between diagonals
12        $p_{i,max} \leftarrow e \cdot \sqrt{(q_{s,i+1} - q_{e,i})^2 - (g_{s,i+1} - g_{e,i})^2}$ ▷ Maximum allowed (indel) dist between diagonals
13        if $p_q < p_{q,max}$ and $p_i < p_{i,max}$ then
14            $\mathcal{C} \leftarrow$ Append $\mathcal{W}[i]$
15        else
16            $\mathcal{C}_{all} \leftarrow$ Append $\mathcal{C}$
17            $\mathcal{C} \leftarrow$ empty array                      ▷ Single chain
18            $\mathcal{C} \leftarrow$ Append $\mathcal{W}[i]$
19    $\mathcal{W}_{chained} \leftarrow \emptyset$
20    foreach $\mathcal{C} \in \mathcal{C}_{all}$ do
21        $n_{cb} \leftarrow 0$
22        foreach $(q_s, q_e, g_s, g_e) \in \mathcal{C}$ do
23            $n_{cb} \leftarrow n_{cb} + (q_e - q_s)$
24        if $|\mathcal{C}| \leq c_{min\_walks}$ and $n_{cb} < c_{min\_cb}$ then
25            continue
26        $\mathcal{W}_{chained} \leftarrow \mathcal{W}_{chained} \cup \mathcal{C}$
27    return $\mathcal{W}_{chained}$
```

Mapping quality

Since the region filtering process in GraphMap maintains a large collection of possible map-

Algorithm 12: Align a read to a processed region using semiglobal or anchored (global) dynamic programming (DP) alignment

```
Input: Region sequence $g$, read sequence $q$, set of anchors $\mathcal{W}_a$, intercept of the $L_1$ line $c_{L1}$, error rate $e$ and
       a parameter $P$ specifying the alignment type (semiglobal or anchored)
Output: Alignment $a$ of the read $q$ to the region $g$

Function ALIGN$(g, q, \mathcal{W}_a, c_{L1}, e, P)$ begin
    Sort $\mathcal{W}_a = [(q_s, q_e, g_s, g_e)]$ in ascending order of $q_s$
    if $\mathcal{W}_a = \emptyset$ then
        return $\emptyset$
    $\mathcal{A} \leftarrow \emptyset$
    if $P = semiglobal$ then
        $(q_{s,min}, q_{e,min}, g_{s,min}, g_{e,min}) \leftarrow \mathcal{W}_a[0]$
        $(q_{s,max}, q_{e,max}, g_{s,max}, g_{e,max}) \leftarrow \mathcal{W}_a[|\mathcal{W}_a| - 1]$
        $g_{sg,start} \leftarrow g_{s,min} - e \cdot |q|$        ▷ Start position on region for semigl. alignment
        $g_{sg,end} \leftarrow g_{s,max} + e \cdot |q|$          ▷ End position on region for semigl. alignment
        $\mathcal{A} \leftarrow$ DP semiglobal alignment between $g[g_{sg,start} \cdots g_{sg,end}]$ and $q$
    else if $P = anchored$ then
        $(q_{s,0}, q_{e,0}, g_{s,0}, g_{e,0}) \leftarrow \mathcal{W}_a[0]$
        $\mathcal{A} \leftarrow \mathcal{A} \cup$ DP semiglobal alignment between $g[0 \cdots g_{s,0}]$ and $q[0 \cdots q_{s,0}]$
        $(q_{s,|\mathcal{W}_a|-1}, q_{e,|\mathcal{W}_a|-1}, g_{s,|\mathcal{W}_a|-1}, g_{e,|\mathcal{W}_a|-1}) \leftarrow \mathcal{W}_a[|\mathcal{W}_a| - 1]$
        $\mathcal{A} \leftarrow \mathcal{A} \cup$ DP semiglobal alignment between $g[g_{e,|\mathcal{W}_a|-1} \cdots |g|]$ and $q[q_{e,|\mathcal{W}_a|-1} \cdots |q|]$
        for $i = 0$ to $(|\mathcal{W}_a| - 1)$ do
            $(q_{s,i}, q_{e,i}, g_{s,i}, g_{e,i}) \leftarrow \mathcal{W}_a[i]$
            $\mathcal{A} \leftarrow \mathcal{A} \cup$ DP global alignment between $g[g_{s,i} \cdots g_{e,i}]$ and $q[q_{s,i} \cdots q_{e,i}]$   ▷ Align an anchor
            if $i < (|\mathcal{W}_a| - 1)$ then                                ▷ Align in-between anchors
                $(q_{s,i+1}, q_{e,i+1}, g_{s,i+1}, g_{e,i+1}) \leftarrow \mathcal{W}_a[i + 1]$
                $\mathcal{A} \leftarrow \mathcal{A} \cup$ DP global alignment between $g[g_{e,i} \cdots g_{s,i+1}]$ and $q[q_{e,i} \cdots q_{s,i+1}]$
    return $\mathcal{A}$
```

ping positions on the given reference, it enables meaningful calculation of the mapping quality directly from its definition:

$$Q = -10 \log p, \tag{4.9}$$

where p is the probability of the read being mapped to the wrong position. We calculate p simply as $p = \max(10^{-4}, 1 - \frac{1}{N_a})$, i.e. "max" $Q = 40$, and report quality values according to the SAM format specification.

E-value

For each reported alignment, GraphMap calculates the E-value which is given as a custom "ZE" parameter in the output SAM file. Following the approach used in BLAST, we rescore alignments and use pre-calculated Gumbel parameters to compute E-values in the same way as in BLAST (default parameters from BLAST: $match = 5$, $mismatch = -4$, $gap_{open} = -8$ and $gap_{extend} = -6$).

Datasets

Publicly available sequencing datasets

For evaluating GraphMap and other tools, we used nine publicly available MinION sequencing datasets and 49 synthetic datasets. The nine publicly available MinION sequencing datasets include a lambda phage dataset, three *E. coli* datasets (each produced with a different version of MinION chemistry), reads for *S. enterica* Typhi, *A. bayalyi* ADP1 and *B. fragilis* BE1, and a dataset consisting of three amplicons from the human genome, as detailed below:

1. Lambda phage burn-in dataset [71]. The dataset consists of 40552 reads in total ($211Mbp$ of data), generated using an early R6 chemistry. The reference genome (NC_001416) is $49kbp$ long giving an expected coverage of > 4300.
2. *E. coli* K-12 MG1655 R7 dataset [123]. The dataset has 111128 reads ($668Mbp$) providing 144 coverage of a $4.6Mbp$ genome ($U00096.2$).
3. *E. coli* K-12 MG1655 R7.3 dataset [123]. The dataset has 70531 reads ($311Mbp$) providing 67 coverage of the genome ($U00096.2$).
4. *E. coli* K-12 MG1655 SQK-MAP006-1 dataset. The dataset consists of 116635 reads ($1.06Gbp$) providing 228 coverage of the genome ($U00096.2$). Sequencing was performed in four runs: two with natural DNA, and two with a low-input library that includes a PCR step. The dataset used in this paper consists of the first natural DNA run (MAP006-1; `http://lab.loman.net/2015/09/24/first-sqk-map-006-experiment/`).
5. *E. coli* UTI89 available on European Nucleotide Archive, accession code $ERX987748$. The dataset consists of one run of 9048 reads ($36Mbp$ of data), providing 7.4 coverage of the genome (NC_$007946.1.fa$ reference, $5.1Mbp$ genome). The dataset is composed of pass 2D reads and their template and complement sequences.
6. *S. enterica* Typhi dataset [65]. The dataset is composed of two runs of strain H125160566 (16401 reads and 6178 reads respectively) and one run of strain 08-04776 (10235 reads). When combined, this dataset consists of 32814 reads ($169Mbp$) which amounts to 35 coverage of a closely related reference sequence, *S. enterica* Typhi Ty2 (NC_004631.1; $4.8Mbp$ genome).
7. *A. baylyi* ADP1 dataset [69]. The dataset consists of 66492 reads ($205Mbp$) providing 57 coverage of a $3.6Mbp$ genome (NC_005966.1).
8. *B. fragilis* BE1 dataset [68]. The dataset consists of 21900 reads ($141Mbp$) providing 27 coverage of a $5.2Mbp$ genome ($LN877293.1$ assembly scaffold).
9. Amplicon sequencing of human HLA-A, HLA-B and CYP2D6 genes [131]. The dataset contains 36779 reads in total. As a reference, chromosomes 6 and 22 from hg19 GRCh37 *H. sapiens* reference were used [131].

Synthetic datasets

Synthetic Illumina reads were generated using the ART simulator [132] ($150bp$ single-end) and PacBio CLR reads using the PBSIM simulator [133] (with default settings). For synthetic MinION data we adopted PBSIM (as no custom ONT simulators exist currently) and used parameters learnt from LAST alignments (to avoid bias towards GraphMap) with *E. coli* K-12 R7.3 data (Appendix B, Table B.8). Reads were simulated ($n = 1000$) for six reference sequences: *N. meningitidis* serogroup A strain Z2491 (1 chromosome, $2.2Mbp$, $NC\_003116.1$), *E. coli* K-12 MG1655 (1 chromosome, $4.6Mbp$, $U00096.2$), *S. cerevisiae* S288c (16 chromosomes, $12Mbp$), *C. elegans* (6 chromosomes, $100Mbp$), *H. sapiens* Chromosome 3 ($198Mbp$, GRCh38, $CM000665.2$) and the entire *H. sapiens* genome (24 chromosomes and mitochondrial DNA, $3.1Gbp$, GRCh38).

To estimate GraphMap's scalability with respect to error rate and read length, 25 additional datasets were simulated from the *S. cerevisiae* S288C reference, for each pair (e,L) of error rate $e \in \{5\%, 10\%, 15\%, 20\%, 25\%\}$ and read lengths $L \in \{1"kbp", 2"kbp", 3"kbp", 4"kbp", 5"kbp"\}$ ($n = 10000$).

Evaluation methods

Performance on synthetic data

Mappers were evaluated for precision and recall in meeting two goals:

1. Finding the correct mapping location - a read was considered correctly mapped if its mapping position was within $\pm 50bp$ of the correct location. In case an alignment contained soft- or hard-clipped bases, the number of clipped bases was subtracted from the reported alignment position to compensate for the shift.
2. Reporting the correct alignment at a per-base-pair level - a base was considered correctly aligned if it was placed in exactly the same position as it was simulated from. Unaligned reads and soft- or hard-clipped portions of aligned reads were not taken into account for precision calculation. Recall was calculated with respect to all simulated bases in reads.

Parameter settings for mappers.

BWA-MEM was evaluated with the nanopore setting ("-x ont2d") unless otherwise stated (version:bwa-0.7.12-r1034, commit: $1e29bcc$). BLASR was evaluated with the options "-sam -bestn 1" (version: 1.3.1, commit: $f7bf1e5$) and in addition for the database search we set more stringent parameters ("-minMatch 7 -nCandidates 1"). LAST was run with a commonly used nanopore setting [123] ("-q 1 -r 1 -a 1 -b 1"; version: 475). BLAST (version: ncbi-blast-2.2.30+-x64-linux) was run with default settings for Illumina data and a more suitable nanopore setting [134] "-reward 5 -penalty -4 -gapopen 8 -gapextend 6 -dust no" for ONT and PacBio data. GraphMap (version: v0.21, commit: $0bd0503$) was run with default settings. In addition, for circular genomes we used the "-C" option, anchored alignment for calling structural variations ("-a anchor") and E-value filtering ("-z 1e0") for database search and variant calling. marginAlign [15] was run with the "–em" option on each dataset to estimate the correct parameters since data quality varied across datasets (commit: $10a7a41$). In the case of simulations, the model parameters were first calculated for every simulated data type using a sample dataset, and then marginAlign was run using corresponding models. Furthermore, since marginAlign is a realigner and uses a mapper for seeding the alignment position, we forked and expanded marginAlign to create a version that uses GraphMap instead of LAST as its seed mapper. Our modified version of marginAlign is available on GitHub: `https://github.com/isovic/marginAlign` (commit: $d69264d$). The modified version of marginAlign was also used with the "–em" option, with the additional parameter "–graphmap" to use GraphMap. We also compared against DALIGNER (commit: $d4aa487$). For synthetic data, DALIGNER was tested using three combinations of parameters: default, "-e.7 -k10" and "-e.7 -k9". As "-e.7 -k10" was found to have the best results for synthetic ONT data (Appendix B, Tables B.1, B.2, B.3 and B.4), it was used for all tests on real nanopore data.

Consensus calling using MinION data

Consensus was called using a simple majority vote of aligned bases, insertion and deletion events (insertion sequences were taken into account while counting events) and positions with < 20 coverage were not called. Our consensus caller is implemented in a script "*consensus.py*" that is freely available at `https://github.com/isovic/samscripts`. All reads were mapped to just the corresponding reference and analyzed to determine consensus sequences. The *E. coli* K-12 reference was mutated using Mutatrix (`https://github.com/ekg/mutatrix`) with parameters "–snp-rate 0.0006 –population-size 1 –microsat-min-len 0 –mnp-ratio 0 –indel-rate 0.0067 –indel-max 10" to emulate the draft nanopore-only assembly reported by Loman *et al.* [135] ($\approx$ 3750 SNPs and $\approx$ 42500 indels). Real nanopore reads were mapped to the mutated reference, and consensus variants (from "*consensus.py*") were used to construct a consensus sequence with GATK's FastaAlternateReferenceMaker tool (GATK version 3.4-46). Consensus sequences were compared to the original reference using nucmer and dnadiff [126] (MUMmer 3.0). Positions $\pm 2bp$ from the mutated position were also considered in calculating consensus errors in mutated positions to account for alignment uncertainty in homopolymer sequences.

Benchmarking mappers for pathogen identification

Bacterial genomes related to a list of water-borne pathogens were selected from NCBI's bacterial database to construct a database of 259 genomes ($550Mbp$). MinION sequencing datasets from cultured isolates were used as proxy for sequencing of pathogen-enriched clinical samples (using data for *E. coli* K-12 R7.3, *S. enterica* Typhi and *E. coli* UTI89, as specified earlier). This is a simple test case as real samples are likely to have contamination from other sources as well (e.g. human DNA). We mapped these three read datasets to the database of bacterial genomes using each of the mappers to find unique alignments and test if these could help identify the correct species and strain. For BWA-MEM, LAST, marginAlign and DALIGNER, we chose the best alignment based on alignment score (as long as alignment score and mapping quality were greater than 0) and for GraphMap and BLASR we used the unique reported alignment (mapping quality > 0). Since marginAlign and DALIGNER do not report the alignment score in their output, we rescored their alignments (parameters $match = 1$, $mismatch = -1$, $gap_{open} = -1$ and $gap_{extend} = -1$) to make them comparable.

Single nucleotide variant calling

All 2D reads from Ammar *et al.* [131] were mapped to the human genome (GRCh37.p13; chr 6 and 22) and for each read only the alignment with the highest alignment score (AS) was kept. To avoid chimeric reads as reported in the original study only reads that fully spanned the amplicon regions were used for this analysis. Variants were called using LoFreq [66] (version: 2.1.2) with the parameters "-a 0.01 -q 0 -Q 0 –no-default-filter". A custom caller for marginAlign (marginCaller) was also used to call SNVs. The detected SNVs were then compared with known variants from dbSNP and a high-confidence set for NA12878 [136] (the sequenced sample; `ftp://ftp.ncbi.nlm.nih.gov/snp/organisms/human_9606_b141_GRCh37p13/VCF/All.vcf.gz`; `ftp-trace.ncbi.nih.gov/giab/ftp/data/NA12878/variant_calls/NIST/NISTIntegratedCalls_14datasets_131103_allcall_UGHapMerge_HetHomVarPASS_VQSRv2.18_all.primitives.vcf.gz`) to identify true positives and false positives.

Structural variation detection

We modified the *E. coli* K-12 MG1655 reference by inducing 20 SV events (10 insertions and 10 deletions) of different sizes: $100bp, 300bp, 500bp, 1000bp, 1500bp, 2000bp, 2500bp, 3000bp, 3500bp, 4000bp$. All 2D reads from both *E. coli* K-12 datasets (R7 and R7.3) were combined and mapped. SVs were detected by simple consensus vote of indel events reported in spanning alignments ($\geq$ 20 bases to avoid sequencing errors). In the absence of a sophisticated SV caller for nanopore data we used a simple rule that identifies windows where $> 15\%$ of the reads at each position report an insertion (or deletion) event (at least 5 reads). To avoid fragmented events due to a local drop in allele frequency, windows which were less than window-length apart (max of the two windows) were merged. A detected event was considered a true positive if its size was within a 25% margin of the true size and its start and end locations were less than 25% of event size away from the true locations. LUMPY [137] (version: 0.2.11) was used for testing the use of split read alignments. The script "extractSplitReads_BwaMem" provided with LUMPY was used to extract split reads from BWA-MEM alignments. As the default setting ("minNonOverlap=20") did not report any results, the script was run with the setting "minNonOverlap=0" to allow split alignments to be adjacent on the read.

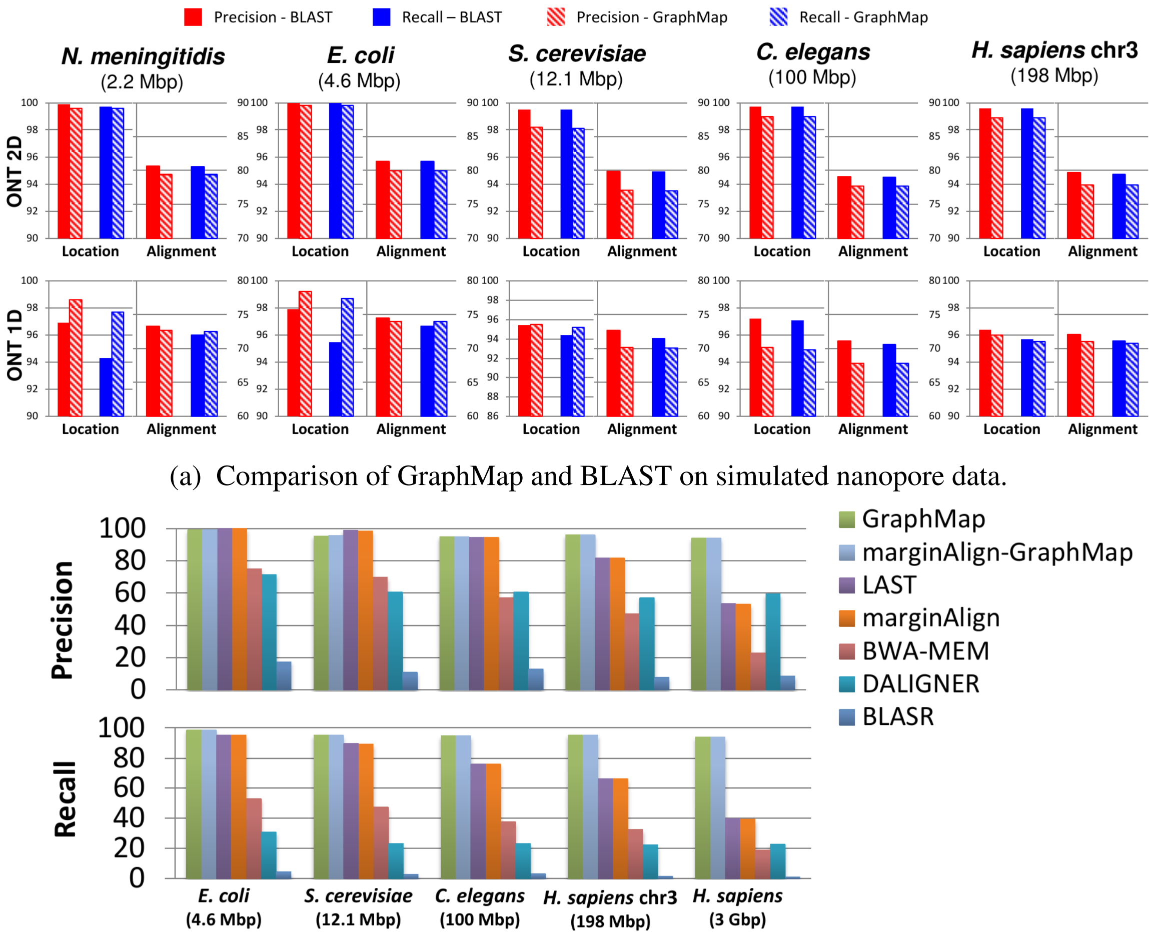
4.1.2 Results

GraphMap maps reads accurately across error profiles

GraphMap was designed to be efficient while being largely agnostic of error profiles and rates. To evaluate this feature a wide range of synthetic datasets were generated that capture the diversity of sequencing technologies (Illumina, PacBio, ONT 2D, ONT 1D) and the complexity of different genomes (Figure 4.6, Appendix B Figure B.1a). GraphMap's precision and recall were then measured in terms of identifying the correct read location and in reconstructing the correct alignment to the reference (Section 4.1.1). These were evaluated separately as, in principle, a mapper can identify the correct location but compute an incorrect alignment of the read to the reference. To provide for a gold-standard to compare against, BLAST [30] was used as a representative of a highly sensitive but slow aligner which is sequencing technology agnostic. On synthetic Illumina and PacBio data, GraphMap's results were found to be comparable to BLAST (Appendix B, Section B.1) as well as other mappers (Appendix B, Tables B.1, B.2). On synthetic ONT data, we noted slight differences ($< 3\%$) between BLAST and GraphMap, but notably, GraphMap improved over BLAST in finding the right mapping location in some cases (e.g. for *N. meningitidis* ONT 1D data; Figure 4.6a). GraphMap's precision and recall in selecting the correct mapping location were consistently $> 94\%$, even with high error rates in the simulated data. Unlike other mappers, GraphMap's results were obtained **without tuning parameters to the specifics of the sequencing technology**.

Constructing the correct alignment was more challenging for synthetic ONT datasets and correspondingly the percentage of correctly aligned bases with GraphMap (70%) is similar to the number of correct bases in the input data. The use of alternate alignment algorithms and parameters did not alter results significantly (Appendix B, Table B.9), though the use of a maximum-likelihood based realigner (marginAlign [15]) improved both alignment precision and recall (Appendix B, Table B.2). The use of marginAlign as a realigner did not improve on GraphMap's ability to identify the correct genomic location (Appendix B, Table B.1). These results highlight GraphMap's ability to identify precise genomic locations based on robust alignments without the need for customizing and tuning alignment parameters to the unknown error characteristics of the data.

For read to reference alignment, programs such as BLAST provide high sensitivity and can be feasible for small genomes, but can quickly become infeasible for larger genomes (e.g. runtime for *C. elegans* or the human genome; Appendix B, Table B.3). Read mappers such as BWA-MEM and BLASR provide a different tradeoff, scaling well to large genomes but with



(a) Comparison of GraphMap and BLAST on simulated nanopore data.

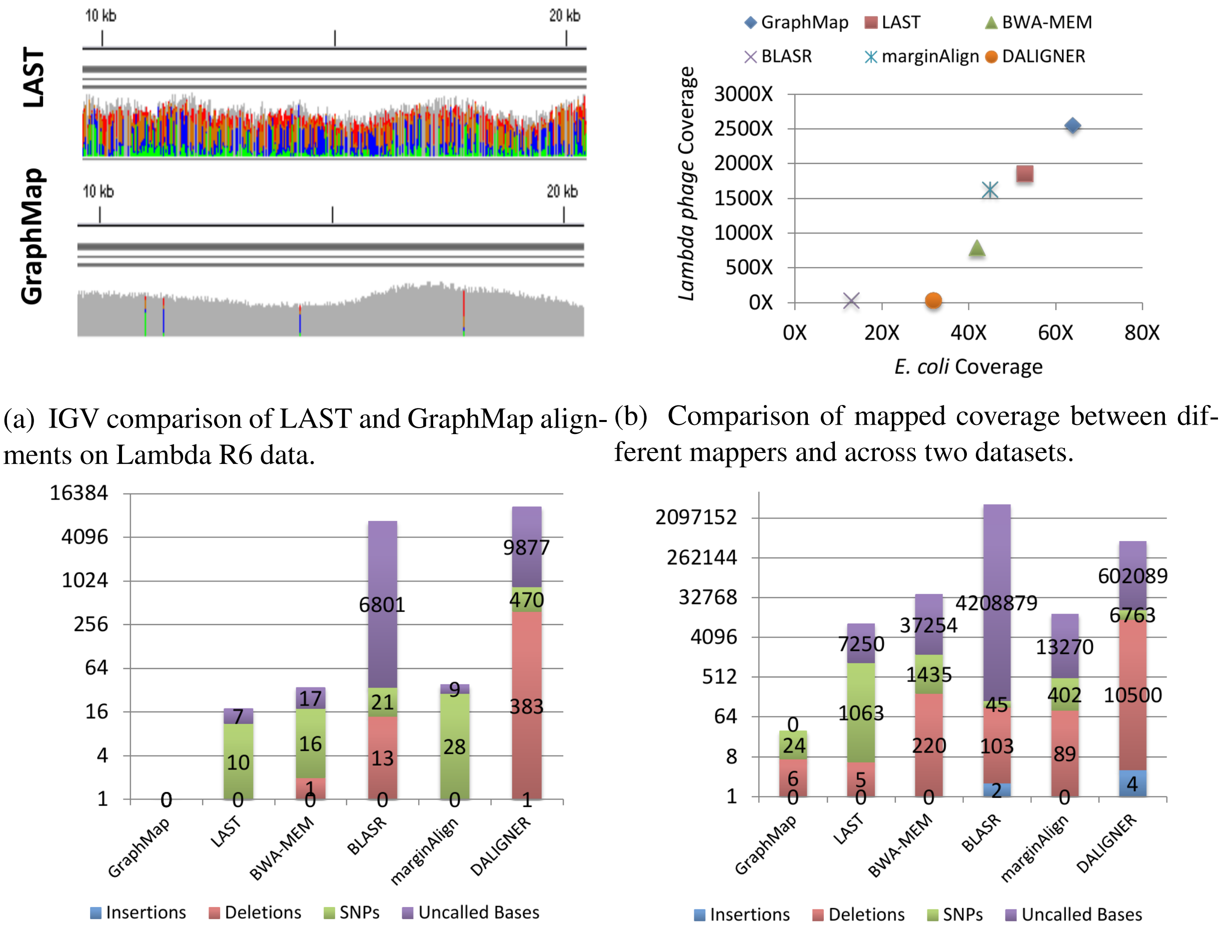
(b) Comparison of state-of-the-art methods on simulated nanopore data.

Figure 4.6: Evaluating GraphMap's precision and recall on synthetic ONT data. *a*) GraphMap (shaded bars) performance in comparison to BLAST (solid bars) on ONT 2D and 1D reads. Genomes are ordered horizontally by genome size from smallest to largest. For each dataset, the graph on the left shows performance for determining the correct mapping location (within 50*bp*; *y*-axis on the left) and the one on the right shows performance for the correct alignment of bases (*y*-axis on the right; see 4.1.1 Methods section). *b*) Precision and recall for determining the correct mapping location (within 50*bp*) for various mappers on synthetic ONT 1D reads.

low sensitivity and precision for high error rates (Figure 4.6b, Appendix B, Tables B.1, B.2 and B.3). This could partly be due to specific parameter settings as is the case for BLASR, which was designed for PacBio data. Mappers such as BWA-MEM on the other hand, have different settings optimized for different sequencing technologies (Appendix B, Figure B.1b). Despite this, BWA-MEM's performance degrades rapidly even in the ONT setting (Figure 4.6b), providing precision and recall $< 25\%$ for mapping to the human genome (Appendix B, Table B.1). DALIGNER [105], a highly sensitive overlapper which additionally supports read mapping, also provided precision and recall that degraded quickly with read error rate and genome size (Figure 4.6b; Appendix B, Table B.1). LAST, originally designed for aligning genomes, fared better in these settings, but still exhibits lower recall for large genomes (30% reduction compared to GraphMap; Figure 4.6b) and precision $< 54\%$ for mapping to the human genome (Appendix B, Table B.1). The use of a realigner (marginAlign) generally improved alignment precision and recall but results for finding the correct genomic location were similar to that of the original mapper (marginAlign uses LAST by default). GraphMap was the only program that uniformly provided high sensitivity and recall (Figure 4.6b), even for mapping to the human genome, while scaling linearly with genome size (Appendix B, Figure B.1c; Tables B.1, B.2, B.3 and B.4). Experiments with a range of read lengths and error rates also demonstrate that GraphMap scales well across these dimensions (runtime and memory usage; Appendix B, Table B.10), though mapping to large genomes currently requires the use of large memory systems ($\approx 100GB$ for human genome). Extrapolating this, mapping data from a MinION run of 100000 reads to the human genome should take < 5 hours and $<$ \$7 on an Amazon EC2 instance (r3.4xlarge) using GraphMap.

Sensitivity and mapping accuracy on nanopore sequencing data

GraphMap was further benchmarked on several published ONT datasets against mappers and aligners that have previously been used for this task (LAST, BWA-MEM and BLASR; see 4.1.1 Methods section), as well as a highly sensitive overlapper for which we tuned settings (DALIGNER; see 4.1.1 Methods section). In the absence of ground truth for these datasets, mappers were compared on the total number of reads mapped (sensitivity), and their ability to provide accurate (to measure precision of mapping and alignment) as well as complete consensus sequences (as a measure of recall). Overall, as seen in the simulated datasets, LAST was the closest in terms of mapping sensitivity compared to GraphMap, though GraphMap showed notable improvements. The differences between GraphMap and LAST were apparent even when comparing their results visually, with LAST alignments having low consensus quality even in a high coverage setting (Figure 4.7a, plot generated using IGV [138]). Across datasets, GraphMap mapped the most reads and aligned the most bases, improving sensitivity by $10-80\%$ over LAST and even more compared to other tools (Figure 4.7b; Appendix B, Figure B.2; Section B.2). This led to fewer uncalled bases compared to LAST, BWA-MEM, BLASR, DALIGNER and marginAlign even in an otherwise high-coverage dataset (Figures 4.7c and 4.7d). In addition, GraphMap analysis resulted in $>$ 10-fold reduction in errors on the *lambda* phage and *E. coli* genome (Figure 4.7c) and reported less than 40 errors on the *E. coli* genome compared to more than a 1000 errors for LAST and BWA-MEM (Figure 4.7d). With ≈ 80 coverage of the *E. coli* genome, GraphMap mapped $\approx 90\%$ of the reads and called consensus bases for the whole genome with < 1 error in 100000 bases ($Q50$ quality). The next best aligner i.e. LAST did not have sufficient coverage (20) on > 7000 bases and reported consensus with a quality of $\approx Q36$. BWA-MEM aligned less than 60% of the reads and resulted in the calling of > 200 deletion errors in the consensus genome. Similar results were replicated in other genomes and datasets as well (Appendix B, Figure B.2).



(a) IGV comparison of LAST and GraphMap alignments on Lambda R6 data.

(b) Comparison of mapped coverage between different mappers and across two datasets.

(c) Consensus calling errors on the *Lambda* phage dataset.

(d) Consensus calling errors on the *E. Coli* K-12 dataset.

Figure 4.7: Sensitivity and mapping accuracy on nanopore sequencing data. *a*) Visualization of GraphMap and LAST alignments for a lambda phage MinION sequencing dataset (using IGV). Grey columns represent confident consensus calls while colored columns indicate lower quality calls. *b*) Mapped coverage of the *lambda* phage and the *E. coli* K-12 genome (R7.3 data) using MinION sequencing data and different mappers. *c*) Consensus calling errors and uncalled bases using a MinION lambda phage dataset and different mappers. *d*) Consensus calling errors and uncalled bases using a MinION *E. coli* K-12 dataset (R7.3) and different mappers.

As another assessment of mapping and alignment accuracy, error profiles of 1D and 2D ONT reads were computed for GraphMap and compared to those for LAST and marginAlign. As observed before [15], substantial variability in the shape and modes of error rate distributions were seen across different mappers, though GraphMap's alignments resulted in lower mismatch rate estimates compared to LAST (Appendix B, Figure B.3). GraphMap's distributions were also more similar to those of marginAlign (used as a reference standard), indicating that GraphMap mapping and alignments are at least as accurate as those from LAST. Overall, deletion and mismatch rates for ONT data were observed to be higher than insertion rates, a pattern distinct from the low mismatch rates seen in PacBio data [133] and explaining why mappers tailored for PacBio data may not work well for ONT data (Appendix B, Figure B.3).

Note that the consensus calling results reported here are not comparable to those for programs such as Nanopolish [135] and PoreSeq [139], that solve the harder problem of correcting the consensus in the presence of assembly and sequencing errors. To account for a "reference bias", where an error-free reference may preferentially enable some programs to report alignments that give an accurate consensus, consensus calling was repeated on a mutated reference (see see 4.1.1 Methods section). Overall, GraphMap was observed to have similar behavior as other mappers in terms of reference bias, with comparable number of errors (SNPs, insertions and deletions) in mutated and non-mutated positions (Appendix B, Table B.11). These results further confirm that GraphMap's high sensitivity does not come at the expense of mapping or alignment accuracy. In terms of runtime requirements, GraphMap was typically more efficient than BWA-MEM and slower than LAST on these datasets (Appendix B, Table B.12). Memory requirements were typically $< 5GB$, with GraphMap and BWA-MEM being intermediate between LAST/BLASR (least usage) and marginAlign/DALIGNER (most usage; Appendix B, Tables B.5 and B.6). Analysis of reads that were only mapped by GraphMap when compared to those that were mapped by both GraphMap and LAST revealed characteristics of reads that are more amenable to GraphMap analysis. In particular, these reads were found to be slightly shorter on average ($3.4kbp$ vs $5.7kbp$), more likely to have windows with higher than average error rate (27% vs 14%), and have a greater proportion of 1D reads (90% vs 76%; *E. coli* R7.3 dataset). Overall, GraphMap provided improved sensitivity of mapping on all ONT datasets (Appendix B, Section B.2), without sacrificing alignment accuracy.

SNV calling in the human genome with high precision

Diploid variant calling using ONT data has multiple potential hurdles including the lack of a dedicated read mapper or diploid variant caller for it [139]. Not surprisingly, a recent report for calling single nucleotide variants (SNVs) from high-coverage targeted sequencing of the diploid human genome reported that existing variant callers were unable to call any variants and a naive approach requiring $1/3$ of the reads to support an allele could lead to many false positive variants [131]. To evaluate if improved read mappings from GraphMap could increase sensitivity and precision, data reported in Ammar *et al.* [131] was reanalyzed using a rare variant caller (LoFreq [66]) that is robust to high error rates, and compared against a set of gold standard calls [140] for this sample (NA12878). Targeted nanopore sequencing reads were mapped by GraphMap to the correct location on the human genome with high specificity, despite the presence of very similar decoy locations (94% identity between *CYP2D6* and *CYP2D7* [131]; Appendix B, Figure B.4). GraphMap provided the most on-target reads, aligning $15-20\%$ more reads than the next best mapper (BWA-MEM) for the three amplified genes (*CYP2D6*, *HLA-A*, *HLA-B*; Appendix B, Figure B.4). These were then used to call heterozygous variants in these challenging regions of the human genome with high precision (96% with GraphMap;

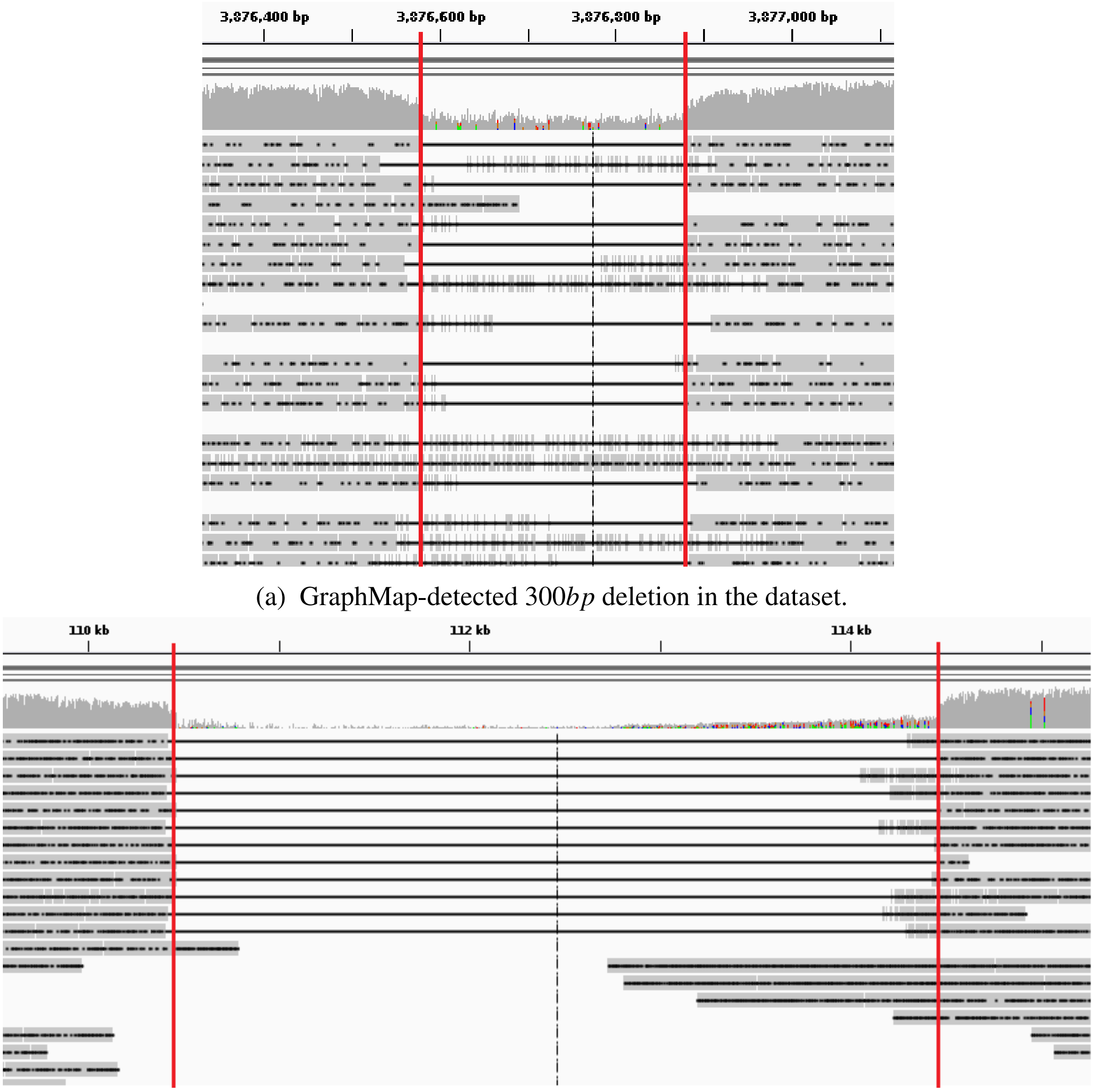
Table 4.1). GraphMap alignments identified many more true positive SNVs than other mappers, with comparable or higher precision (76% improvement compared to BWA-MEM and LAST) and a 15% increase in sensitivity over DALIGNER, which has slightly lower precision (93%; Table 4.1). While the use of a custom variant caller for marginAlign (marginCaller) improved its results in terms of sensitivity, it came at the expense of low precision (36%; Table 4.1). Subsampling GraphMap mappings to the same coverage as BWA-MEM provided comparable results (42 vs 47 true positives and 2 vs 2 false positives) indicating that GraphMap's improved mapping sensitivity (2 compared to other mappers) played a role in these results. The ability to sensitively and precisely call SNVs with GraphMap, provides the foundation for reconstructing haplotypes with long reads, and opens up the investigation of complex and clinically important regions of the human genome using nanopore sequencing.

Table 4.1: Comparison of various mappers for single nucleotide variant calling. Results are based on amplicon sequencing data for a human cell line (NA12878) for the genes *CYP2D6*, *HLA-A* and *HLA-B*. Precision values are likely to be an underestimate of what can be expected genome-wide due to the repetitive nature of the regions studied and the incompleteness of the gold-standard set. Results for marginAlign using marginCaller are shown in parentheses.

	LAST	marginAlign	BWA-MEM	BLASR	DALIGNER	GraphMap
Precision (%)	94	100 (36)	96	100	93	96
True Positives	49	1 (107)	47	43	75	86

GraphMap enables sensitive and accurate structural variant calling

Long reads from the MinION sequencer are, in principle, ideal for the identification of large structural variants (SVs) in the genome [141], but existing mappers have not been systematically evaluated for this application [65]. Read alignments produced by mappers are a critical input for SV callers. To compare the utility of various mappers, their ability to produce spanning alignments or split alignments indicative of a structural variation (insertions or deletions) was evaluated using real *E. coli* data mapped to a mutated reference. As shown in Table 4.2, mappers showed variable performance in their ability to detect SVs through spanning alignments. In comparison, GraphMap's spanning alignments readily detected insertions and deletions over a range of event sizes ($100bp - 4kbp$), providing perfect precision and a 35% improvement in recall over the next best mapper (BLASR; Table 4.2). LAST alignments were unable to detect any events under a range of parameter settings but post-processing with marginAlign improved recall slightly (5%; Table 4.2). BWA-MEM alignments natively provided 10% recall at 67% precision. Post-processing BWA-MEM alignments with LUMPY improved recall to 45%, using information from split reads to predict events. GraphMap produced spanning alignments natively that accurately demarcated the alignment event and did this without reporting any false positives (Figures 4.8a and 4.8b, Table 4.2).



(a) GraphMap-detected 300bp deletion in the dataset.

(b) GraphMap-detected 4kbp deletion in the dataset.

Figure 4.8: Structural variant calling using nanopore sequencing data and GraphMap. An IGV view of GraphMap alignments that enabled the direct detection of a 300bp deletion (delineated by red lines). *b*) GraphMap alignments spanning a $\approx 4kbp$ deletion (delineated by red lines).

Sensitive and specific pathogen identification with ONT data

Due to its form factor and real time nature, an application of MinION sequencing that has garnered interest in the community is in the identification of pathogens in clinical samples. Sequencing errors (particularly in 1D data) and the choice of read mapper could significantly influence results in such an application and lead to misdiagnosis. GraphMap's high specificity in read mapping as seen in the results for Ammar *et al.* (Appendix B, Figure B.4) suggested that it could be useful in this setting. Clonal sequencing data on the MinION and a database of microbial genomes was used to create several synthetic benchmarks to evaluate the performance

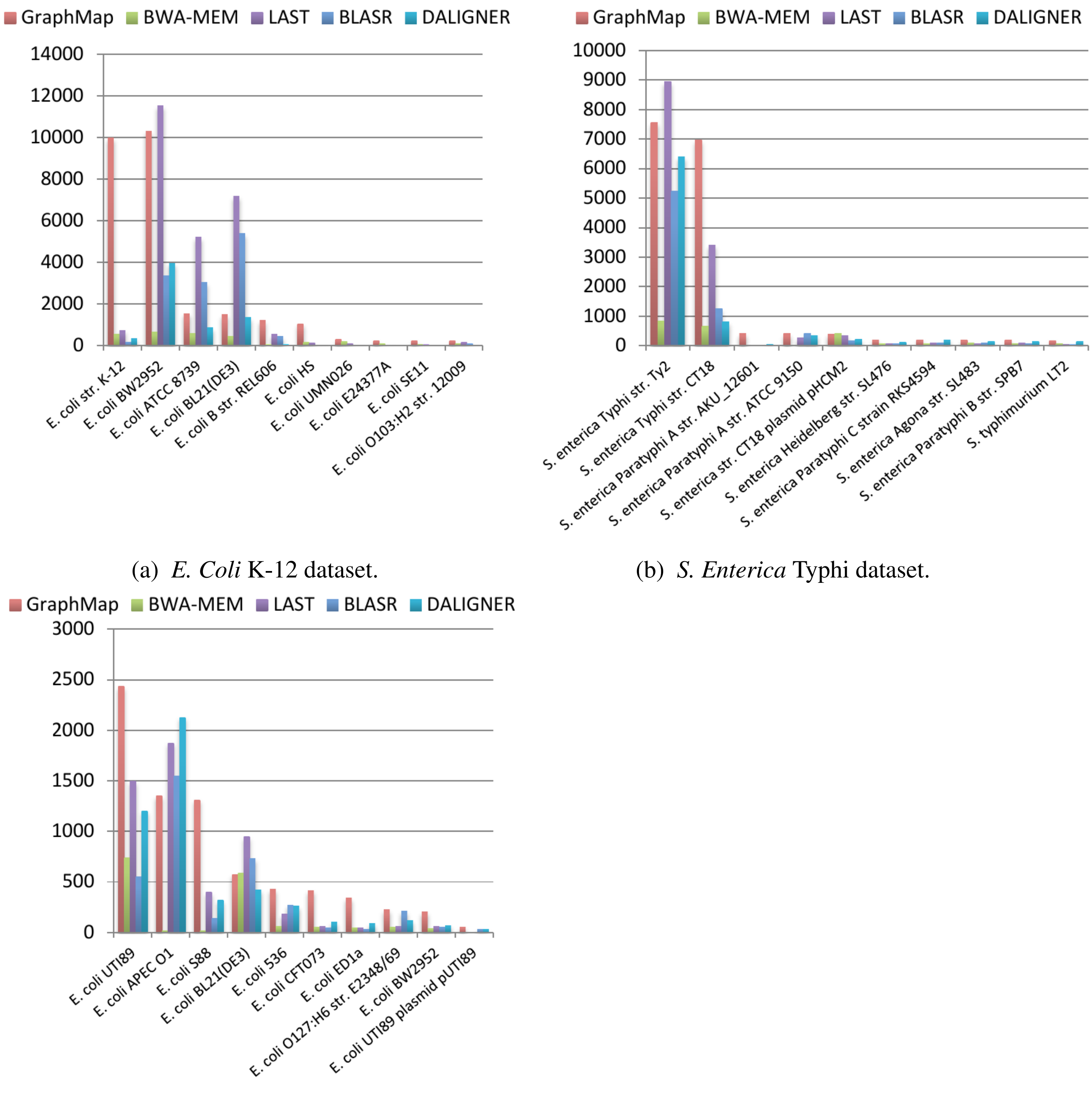
Table 4.2: Comparison of various mappers for structural variant calling. Results are based on mapping a MinION dataset for *E. coli* K-12 (R7.3) on a mutated reference containing insertions and deletions in a range of sizes ($[100bp, 300bp, 500bp, 1kbp, 1.5kbp, 2kbp, 2.5kbp, 3kbp, 3.5kbp, 4kbp]$; 20 events in total). Bold values indicate the best results for each metric. The $F1$ score is given by a weighted average of precision and recall. Values in parentheses for BWA-MEM show the results using LUMPY.

	LAST	**marginAlign**	**BWA-MEM**	**BLASR**	**DALIGNER**	**GraphMap**
Precision (%)	0	50	67 (90)	94	0	**100**
Recall (%)	0	5	10 (45)	75	0	**100**
$F1$ Score (%)	0	9	17 (60)	83	0	**100**

of various mappers for this application. For species level identification, all mappers reported high precision (typically $> 95\%$) but recall varied over a wide range from 20% to 90% (Table 4.3). GraphMap had the highest recall and $F1$ score in all datasets, providing an improvement of $2 - 18\%$ over other mappers. The improvement was more marked when a perfect reference was not part of the database (e.g. *S. enterica* Typhi, Table 4.3) For this application, BWA-MEM was the next best mapper while LAST and BLASR exhibited $> 25\%$ reduced recall compared to GraphMap (Table 4.3). Not surprisingly, strain level identification using MinION data appears to be much more difficult and in some cases a closely related strain can attract more reads than the correct strain (Figure 4.9a). However, in the datasets tested, GraphMap assigned most reads to a handful of strains that were very similar to the correct strain (Figures 4.9a and 4.9c; 99.99% identity for *E. coli* K-12 and BW2952). Moreover, the use of strain specific sequences was able to unambiguously identify the correct strain from this subset (e.g. there were no reads mapping to $NC\_012759.1 : 4.13Mbp - 4.17Mbp$, a region unique to BW2952), indicating that this approach could be used to systematically identify pathogens at the strain level.

Table 4.3: Precision and Recall for species identification using MinION reads. Bold values indicate the best results for each dataset and metric. Results for marginAlign were nearly identical to that of LAST (within 1%) and have therefore been omitted.

	E. coli K-12 (R7.3)			*S. enterica* Typhi			*E. coli* UTI89		
	Prec.	**Rec.**	**F1**	**Prec.**	**Rec.**	**F1**	**Prec.**	**Rec.**	**F1**
BLASR	93	22	36	**99**	28	44	98	55	70
LAST	94	37	53	97	34	51	95	65	78
DALIGNER	80	10	17	**99**	28	43	98	55	71
BWA-MEM	94	47	63	98	45	61	98	85	91
GraphMap	**95**	**51**	**67**	97	**56**	**72**	**99**	**88**	**93**



(a) *E. Coli* K-12 dataset.

(b) *S. Enterica* Typhi dataset.

(c) *E. Coli* UTI89 dataset.

Figure 4.9: Species identification using nanopore sequencing data and GraphMap. Number of reads mapping to various genomes in a database (sorted by GraphMap counts and showing top 10 genomes) using different mappers (GraphMap, BWA-MEM, LAST, DALIGNER and BLASR) and three MinION sequencing datasets for *a*) *E. coli* K-12 (R7.3) *b*) *S. enterica* Typhi and *c*) *E. coli* UTI89. Note that GraphMap typically maps the most reads to the right reference genome (at the strain level) and the *S. enterica* Typhi dataset is a mixture of sequencing data for two different strains for which we do not have reference genomes in the database. Results for marginAlign were nearly identical to that of LAST (within 1%) and have therefore been omitted.

4.1.3 Discussion

The design choices in GraphMap, including the use of new algorithmic ideas such as gapped spaced seeds, graph mapping and LCSk, provide a new tradeoff between mapping speed and sensitivity that is well-suited to long nanopore reads. For mapping error-prone synthetic long reads to the human genome, GraphMap was the only mapper that exhibited BLAST-like sensitivity, while being orders of magnitude faster than BLAST. On nanopore sequencing data from the MinION system, GraphMap was unmatched in terms of sensitivity, mapping more than 90% of reads and 95% of bases on average. Compared to other mappers, this lead to a $10-80\%$ increase in mapped bases (e.g. 18% increase on a recent MinION MkI dataset; Appendix B, Section B.2). This is a significant improvement – typically mapping programs are highly optimized and increase in sensitivity of even a few percentage points can be hard to achieve. Additionally, sensitivity is a key requirement for mapping tools and mapping-based analysis, as reads that cannot be mapped are unavailable for use in downstream applications. A drawback of the current implementation of GraphMap is the requirement of large-memory machines for mapping to large genomes ($\approx 100GB$ for the human genome). The use of more memory-efficient index structures (e.g. FM-index) can significantly reduce this requirement (for a modest increase in runtime) and this option is currently under implementation.

GraphMap's speed and sensitivity do not come at the expense of location and alignment precision, as demonstrated by extensive experiments with synthetic and real datasets. For determining the correct genomic location, GraphMap's precision is typically greater than 98% and it is able to distinguish between candidate locations that are more than 94% identical on the human genome. For alignment precision, GraphMap's performance scales according to sequencing error rate, is comparable to BLAST and other mappers (BWA-MEM, LAST, BLASR and DALIGNER), and was observed to be robust to the choice of alignment algorithms and parameters. GraphMap mappings provided a better starting point for the realigner marginAlign [15] and should do so for consensus calling algorithms such as Nanopolish [135] and PoreSeq [139] as well.

In general, GraphMap's improved sensitivity should benefit a range of applications for nanopore data and a few of these were explored in this study. In particular, variant calling and species identification with error-prone data can be affected by errors in mapping and alignment. Despite the lack of custom variant callers, read mappings from GraphMap were shown to provide sensitive and precise single-nucleotide variant calls on complex regions of the human genome. In addition, GraphMap alignments readily spanned insertions and deletions over a wide range of sizes (100bp-4kbp) allowing for the direct detection of such events, without assembly or split read analysis. With the development of new nanopore-specific variant calling tools, GraphMap's improved sensitivity should continue to provide a useful starting point for these applications. Furthermore, GraphMap alignments were used to identify the species-level origin of reads with high precision and recall. The sensitivity of mapping with GraphMap can be a key advantage in applications where MinION sequencing reads are used in real-time to identify pathogens [142], particularly in combination with rapid protocols for generating 1D reads on the MinION. With further downstream processing, these read mappings could be used for strain-level typing and characterization of antibiotic resistance profiles [142], meeting a critical clinical need.

In principle, the approach used in GraphMap could be adapted for the problem of **computing overlaps and alignments between reads**. As was recently shown, nanopore sequencing reads can be used to construct high-quality assemblies *de novo* [135] and sensitive hashing techniques have been used for the assembly of large genomes [103]. GraphMap's sensitivity and specificity as a mapper could thus serve as the **basis for fast computation of overlap alignments** and *de novo* assemblies in the future.

4.2 Owler - Overlap With Long Erroneous Reads

Encouraged by the results obtained with GraphMap, we took the initial design and modified it to make it more suitable for sequence overlapping. Refactoring the GraphMap algorithm to support sequence overlapping had two main goals set:

1. Sensitivity - Capturing as many true positive overlaps as possible even if this means that precision is somewhat lower. False positive overlaps can arbitrarily be filtered in the downstream analysis steps, while low sensitivity results are a permanent loss of information which cannot be subsequently compensated [103].
2. Speed - To allow for fast *de novo* assemblies without error-correction, a raw read overlapper should take significantly less time than the typical (error-correction)+overlapping approach.

Error-correction, although scales linearly with the genome size (Table 4.4), is a very time-consuming process, and prohibitively slow for large-scale mammalian-sized genome analyses. A fast and sensitive overlapper should have a dramatic impact on the entire *de novo* assembly process by opening the possibility of avoiding the error-correction entirely.

In this section we develop two approaches to overlapping:

1. Based on the full GraphMap algorithm with slight modifications to allow overlapping and skipping self-overlaps. This overlap procedure is slower, but allows full control over the quality of the resulting overlaps through all parameters defined in GraphMap, including the E-value and the mapping quality.
2. A trimmed GraphMap algorithm to allow for higher processing speeds, but offers less control over the accuracy of the output.

Table 4.4: Scalability of PBcR error-correction across three genomes of different sizes. The datasets are composed of PacBio sequences of *Lambda* phage (`http://www.cbcb.umd.edu/software/PBcR/data/sampleData.tar.gz`), *E. Coli* K-12 (`https://github.com/PacificBiosciences/DevNet/wiki/E.-coli-Bacterial-Assembly`), and *S. Cerevisiae* W303 (`https://github.com/PacificBiosciences/DevNet/wiki/Saccharomyces-cerevisiae-W303-Assembly-Contigs`), subsampled to $40x$ coverage. Fraction of bases corrected reports the size of the dataset which was output from the error-correction process compared to the initial, raw dataset.

Species	Genome size	CPU time [min]	Fraction of bases corrected
Lambda	48.5 kbp	11.29	71%
E. Coli K-12	4.6 Mbp	1337.51	82%
S. Cerevisiae W303	11.5 Mbp	3482.28	63%

4.2.1 Methods

Algorithm description

Overlapping based on the full GraphMap algorithm - "GraphMap overlap"

To allow for overlapping but still to employ the entire sensitivity of GraphMap, only *two* slight modifications to the algorithm were made:

- In the region selection step, hits with $q_{id} >= t_{id}$ would not be counted. Here, q_{id} is the ID of the read currently being analyzed (query), and t_{id} ID of the target read, or, the read where a seed from q has a hit. This slight change would disallow self-overlaps (when $q_{id} = t_{id}$) and also reduce the redundant information (e.g. overlap between reads $(2,3)$ is the same as between reads $(3,2)$; by skipping the second overlapping we increase the execution speed by $2x$).
- Implementing the MHAP output format for reporting overlaps

Other modifications were only in the form of tweaking parameters already present in GraphMap:

- Enabling secondary alignments - a single read can be mapped multiple times (to all other reads covering the region). By default, this option is turned off in GraphMap.
- Disable filtering by low mapping quality - large coverage datasets would always result in $mapq = 0$, which is the default filter in GraphMap.
- Lowering the ambiguity threshold - by default allowed ambiguity is set low (2% from the top result). Increasing this threshold will capture more divergent overlaps.

Overlap based on the trimmed pipeline - "Owler"

Owler mode (Overlap With Long Erroneous Reads) skips the graph-mapping and alignment steps of GraphMap completely, and uses only one gapped spaced index. This mode of overlapping is aimed at more accurate read sequences, such as Oxford Nanopore 2d and PacBio data.

The full pipeline, shown in Algorithm 13, consists of the following steps:

1. Construct a gapped spaced index of the reads for only one shape (6-mers, "1111110111111"), as described in Section 4.1.1 (Algorithm 13, line 1).
2. For a read, collect all gapped spaced seed hits (Algorithm 13, lines 5-11). Sort them according to the ID of the target sequence (Algorithm 13, line 12).
3. For a single target sequence, perform the Longest Common Subsequence in k-length substrings (LCSk) on the set of seeds (as described in Section 4.1.1) (Algorithm 13, lines 13-24).
4. Perform chaining of seeds which were left after LCSk to filter potential outliers (as described in Section 4.1.1) (Algorithm 13, line 25).
5. If, for an overlap, certain lenient conditions are not met, filter the overlap (Algorithm 13, lines 30-31; Algorithm 14). These conditions include: (I) checking whether the number of bases covered by seeds is above a minimum threshold (Algorithm 14, lines 1-2), (II) checking whether the overlap length is above some minimum threshold (Algorithm 14, lines 7-8), (III) checking if the ratio of overlap lengths in both sequences is suspiciously high (higher than expected error rate e) (Algorithm 14, lines 9-12), (IV) checking whether the overlap has malformed start and end overhangs (e.g. there is a repeat in the middle of two reads belonging to different genomic locations) (Algorithm 14, lines 13-14), (V) checking whether there are large insertion/deletion events in-between seeds (structural variants) (Algorithm 14, lines 15-21).
6. Form an overlap defined by: query ID, target ID, number of covered bases in the overlap, number of seeds in overlap, query orientation, query start, query end, query sequence length, target orientation, target start, target end, and target sequence length (Algorithm 13, line 32).
7. Output overlaps in MHAP or PAF formats.

4.2.2 Implementation and reproducibility

Owler was implemented in C++. Evaluation tools and scripts were developed in C++ and Python. All tests were ran using Ubuntu based systems with two 8-core Intel(R) Xeon(R) E5-2640 v2 CPUs @ 2.00GHz with Hyperthreading, using 16 threads where possible. Version of methods used in comparison:

- Minimap - `https://github.com/lh3/minimap.git`, commit: $1cd6ae3bc7c7$
- MHAP - `https://github.com/marbl/MHAP`, version 2.1, commit: $7d8b0c31a407$
- DALIGNER - `https://github.com/thegenemyers/DALIGNER`, commit: $84133cbc0de4$, DAZZ_DB commit: $70cb962a7f57$
- BLASR - `https://github.com/PacificBiosciences/blasr`, commit: $f7bf1e56871d$

Algorithm 13: Owler algorithm

```
Input: Set of read sequences for indexing $\mathcal{R}$, set of read sequences for querying $\mathcal{Q}$, expected error rate $e$
Output: Set of overlaps $\mathcal{O}$
Function OWLER($\mathcal{R}, \mathcal{Q}, e$) begin
    $s \leftarrow$ "1111110111111"
    $\mathcal{I} \leftarrow CreateGappedIndex(\mathcal{R}, s)$
    $\mathcal{O} \leftarrow \emptyset$                                  ▷ Final set of overlaps
    for $q_{id} = 0$ to $(|\mathcal{Q}| - 1)$ do                        ▷ Process reads individually
        $q \leftarrow \mathcal{Q}[q_{id}]$
        $\mathcal{W} \leftarrow \emptyset$                              ▷ A set of all hits
        for $i = 0$ to $(|q| - |s|)$ do                               ▷ Process all k-mers of a read
            $h_{raw} \leftarrow q[i...(i+s)]$
            $\mathcal{T} \leftarrow CollectHits(\mathcal{I}, s, h_{raw})$
            foreach $(t_{id}, t_{pos}) \in \mathcal{T}$ do
                $\mathcal{W} \leftarrow \mathcal{W} \cup (i, i+|s|, t_{pos}, t_{pos}+|s|, t_{id})$
        Sort $\mathcal{W} = [(q_s, q_e, t_s, t_e, t_{id})]$ in ascending order of $t_{id}$
        $r_{id,prev} \leftarrow 0$
        for $i = 0$ to $(|\mathcal{W}| - 1)$ do                         ▷ Process all hits on the same target
            $(q_s, q_e, t_s, t_e, t_{id}) \leftarrow \mathcal{W}[i]$
            if $i = 0$ then
                $i_{start} \leftarrow i$
                $t_{id,prev} \leftarrow t_{id}$
                continue
            if $t_{id} \neq t_{id,prev}$ then
                $\mathcal{W}_t \leftarrow \emptyset$                    ▷ Set of hits for a particular target
                foreach $(q_s, q_e, t_s, t_e, t_{id}) \in \mathcal{W}[i_{start}...(i-1)]$ do
                    $\mathcal{W}_t \leftarrow \mathcal{W}_t \cup (q_s, q_e, t_s, t_e)$      ▷ Create a new set without $t_{id}$
                $\mathcal{W}_{lcsk} \leftarrow LCSk(\mathcal{W}_t)$
                $\mathcal{W}_a \leftarrow FilterByChaining(\mathcal{W}_{lcsk}, e, 200, 2, 50)$     ▷ Get anchors by chaining
                $rev_q \leftarrow false$
                $rev_r \leftarrow t_{id,prev} \geq |\mathcal{R}|$   ▷ $\mathcal{I}$ indexed both fwd and rev.comp. of $\mathcal{R}$. For rev.comp. $t_{id} \geq |\mathcal{R}|$
                $r_{id} \leftarrow t_{id} \mod |\mathcal{R}|$          ▷ Actual index of the reference where the hit is
                $n_k \leftarrow |\mathcal{W}_a|$                        ▷ Number of seeds
                $n_{cb} \leftarrow CalcCoveredBases(\mathcal{W}_a)$     ▷ Number of bases covered by seeds
                if $CheckOverlap(\mathcal{W}_a, |q|, |\mathcal{R}[r_{id}]|, e, n_{cb}, 0.10, 0.33, 0.10) = OK$ then
                    $\mathcal{O} \leftarrow \mathcal{O} \cup (q_{id}, r_{id}, n_{cb}, n_k, rev_q, \mathcal{W}_a[0].q_s, \mathcal{W}_a[|\mathcal{W}_a| - 1].q_e, |q|, rev_r, \mathcal{W}_a[0].r_s, \mathcal{W}_a[|\mathcal{W}_a| - 1].r_e, |\mathcal{R}[r_{id}]|)$
    return $\mathcal{O}$
```

- GraphMap overlap - Implemented in GraphMap, `https://github.com/isovic/graphmap`, commit: $1d16f07888b6$
- GraphMap owler - Implemented in GraphMap, `https://github.com/isovic/graphmap`, commit: $1d16f07888b6$

All tools were ran using their default parameters, except for Minimap and BLASR. Both tools have default settings for mapping and not overlapping, and therefore required adjustment. Minimap was run using "-Sw5 -L100 -m0 -t16" which are the suggested parameters for overlapping (`https://github.com/lh3/minimap.git`). For BLASR, we used "-m 4 -bestn 30

Algorithm 14: Helper function to determine the validity of an overlap

```
Input: A set of seeds $\mathcal{W}_a$, query read length $|q|$, target read length $|r|$, expected error rate $e$, number of
       bases covered by seeds $n_{cb}$, minimum allowed overlap percent from any sequence $m_{perc\_ovl}$,
       maximum allowed percent overhang from any end of any sequence $m_{perc\_ovhng}$ and minimum
       percent of bases covered by seeds in any of the sequences $m_{perc\_cb}$
Output: Value OK is the overlap passed all tests, BAD otherwise.

Function CheckOverlap($\mathcal{W}_a, |q|, |r|, e, n_{cb}, m_{perc\_ovl}, m_{perc\_ovhng}, m_{perc\_cb}$) begin
    if $n_{cb} < m_{cb} \cdot \min(|q|,|r|)$ then                                        ▷ Min num covered bases
        return BAD
    $A_{start} \leftarrow \mathcal{W}_a[0].q_s$
    $A_{end} \leftarrow \mathcal{W}_a[|\mathcal{W}_a|-1].q_e$
    $B_{start} \leftarrow \mathcal{W}_a[0].r_s$
    $B_{end} \leftarrow \mathcal{W}_a[|\mathcal{W}_a|-1].r_e$
    if $|A_{end} - A_{start}| < m_{perc\_ovl} \cdot |q|$ or $|B_{end} - B_{start}| < m_{perc\_ovl} \cdot |r|$ then      ▷ Min overlap len
        return BAD
    $o_{min} \leftarrow \min(|A_{end} - A_{start}|, |B_{end} - B_{start}|)$              ▷ Minimum overlap length
    $o_{max} \leftarrow \max(|A_{end} - A_{start}|, |B_{end} - B_{start}|)$              ▷ Maximum overlap length
    if $(1 - o_{min}/o_{max}) > e$ then                                                  ▷ Max overlap indel error rate
        return BAD
    if $(A_{start} > m_{perc\_ovhng} \cdot |q|$ and $B_{start} > m_{perc\_ovhng} \cdot |r|)$ or
    $((|q| - A_{end}) > m_{perc\_ovhng} \cdot |q|$ and $|r - B_{end}| > m_{perc\_ovhng} \cdot |r|)$ then   ▷ Handle suspicious overhangs
        return BAD
    for $i = 1$ to $(|\mathcal{W}| - 1)$ do
        $(q_{s,i-1}, q_{e,i-1}, r_{s,i-1}, r_{e,i-1}) \leftarrow \mathcal{W}_a[i-1]$
        $(q_{s,i}, q_{e,i}, r_{s,i}, r_{e,i}) \leftarrow \mathcal{W}_a[i]$
        $d_{min} \leftarrow \min(|q_{s,i} - q_{e,i-1}|, |r_{s,i} - r_{e,i-1}|)$        ▷ Minimum dist between seeds
        $d_{max} \leftarrow \max(|q_{s,i} - q_{e,i-1}|, |r_{s,i} - r_{e,i-1}|)$        ▷ Maximum dist between seeds
        if $(1 - d_{min}/d_{max}) > e$ then                                              ▷ Maximum gap ratio
            return BAD
    return OK
```

-nCandidates 30 -nproc 16". HGAP assembler sets the "-bestn" parameter equal to the sequencing coverage [101]. To attempt to be more lenient, we allowed 1.5x the sequencing coverage and specified "-bestn" and "-nCandidates" for each dataset accordingly.

Datasets

We tested Owler and other overlappers on five publicly available Oxford Nanopore datasets. The datasets used here are the same as Datasets 1 - 5 used in Section 3.1.1 used to evaluate the entire assembly pipelines. For consistency, we present them in Table 4.5 as well.

Evaluation methods

Should the source locations of two reads in a genomic sample be known, and the two reads adjacent on a region of the genome with their start and end positions intersecting, we can say that these reads *overlap*.

Table 4.5: Description of the benchmarking datasets used for evaluation.

Name	Description
Dataset 1	Complete *E. coli* R7.3 dataset, contains both 1d and 2d reads (both pass and fail), total coverage 67*x* (70531 reads), of which 2d reads comprise 14*x* (11823 reads) [123].
Dataset 2	Reads from Loman *et al.* [16] subsampled to coverage 19*x*, pass 2d reads only (in total 16945 reads).
Dataset 3	Complete dataset used by Loman *et al.* [16] nanopore assembly paper, contains pass 2d reads only, coverage 29*x*, 22270 reads.
Dataset 4	Reads from MARC WTCHG dataset, 2d reads extracted from pass (33*x*) and fail (7*x*) folders, total coverage 40*x*, total number of 2d reads: 29635 [13].
Dataset 5	2d reads extracted from the first run of the MAP006 dataset (MAP006-1), from pass folder only, coverage 54*x*, 25483 reads in total. `http://lab.loman.net/2015/09/24/first-sqk-map-006-experiment/`

Since the knowledge of the originating locations of fragments is, in most cases, not available in advance, one can only estimate the source positions of the reads by mapping them to the reference genome. A list of estimated "truth" overlaps (see below) can then be determined by analyzing the mapping positions. Similar approach was applied in [103] to estimate the sensitivity (*recall*) and PPV (positive predictive value, also referred to as *precision*) of MHAP, DALIGNER and BLASR, however, instead of explicitly testing each reported overlap, the authors randomly subsampled the test set and generated approximate results.

We borrow the approach of reference-guided overlap testing as in [103], but we evaluate the full test datasets for their precision ($prec = TP/(TP+FP)$), recall ($rec = TP/T_{truth}$), and a joint F1 measure ($F1 = \frac{2 \cdot prec \cdot rec}{prec+rec}$). Here, TP presents the true positives in the test dataset, FP the false positives and T_{truth} the total number of "truth" overlaps expected to be reported.

The list of "truth" overlaps for a dataset is compiled as follows. First, reads are mapped to the reference genome using three mappers: GraphMap, BWA-MEM and LAST. Unlike [103] which uses only one mapper, we use three to account for their intrinsic biases, such as possible low mapping sensitivity in some regions, or mapping ambiguities. For each mapper: (I) start and end positions of all mappings are determined; (II) If any pair of reads has an intersection of start and end intervals, a "truth" overlap is marked. The process is repeated for all three mappers. The three generated lists are merged by creating a union, meaning, if an overlap was detected using any of the mappers, we consider it a "truth" overlap. Likewise, we compile a list of all unmapped reads. When classifying overlaps from a test set, if an overlap is coincident with an unmapped read, the overlap itself would not be present in the "truth" list and might be, potentially wrongly, marked as a false positive. However, since none of the mappers managed to successfully map the read, it might still produce true overlaps which we cannot evaluate.

A test set is then evaluated by looking up every reported overlap in the "truth" list. If the test set contains multiple overlaps between two reads (e.g. (A,B) and (B,A), where A and B are reads), the overlap is counted only once. Self-overlaps (e.g. (A,A)) are also not counted. If an overlap is not present in the "truth" list, but it is coincident with a read which was marked as unmapped, the overlap is flagged as "Unknown" and not used for precision and recall calculations.

In addition to these statistical measures, we also evaluated the required CPU time and memory to conduct overlapping on each dataset.

4.2.3 Results

A surprisingly large number of true overlaps (details on the method used to determine "true" overlaps are given in Section *Evaluation methods*) between reads ($> 10^7$) obtained from Dataset 4 (which has only 40x coverage), led us to believe that there were short regions of extremely high coverage in the dataset (e.g. because of amplicon sequencing, unfiltered control sequence DNA (csDNA) or other contamination). Indeed, after mapping the reads to the reference genome using GraphMap and inspecting the per-base coverage, we found that there was a region "$gi|545778205|gb|U00096.3| : 577259 - 580681$" with average coverage above $4000x$. Figure 4.10 (generated using Tablet [143]), shows the coverage of this region visually. Considering that overlapping is a pairwise operation, this coverage is consistent with the expected number of true overlaps on that dataset.

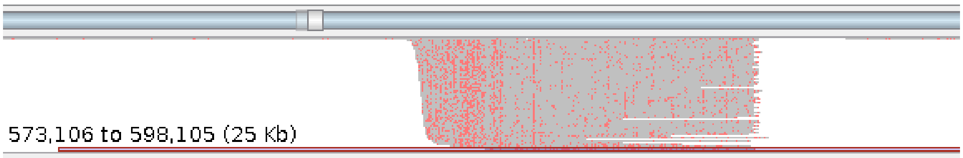


Figure 4.10: A Tablet plot of a region of extremely high coverage in Dataset 4.

MHAP, Minimap, DALIGNER, BLASR, GraphMap overlap and GraphMap owler were successfully run on all five datasets. Table 4.6 summarizes the Precision, Recall, F1, time and memory measures of all overlappers for each dataset. Values marked in bold show Owler's results compared to the best other scoring overlapper (not taking "GraphMap overlap" into account). It can be seen from the table that Owler has a consistently high precision and recall rates across all datasets, whereas other overlappers, except Minimap, show high variance. Minimap's results deteriorated only on Dataset 4 which has a short region of large coverage ($\approx 4000x$) and millions of overlaps stemming from that region. Our other overlapping method, "GraphMap overlap" also has a reduced recall on this dataset. This is due to the "–max-regions 100" parameter used to limit the number of regions GraphMap needs to inspect to 100 (similar to BLASR's

"-nbest" parameter).

Results are by far the worst on Dataset 1 for any overlapper. However, Owler and Minimap showed highest and similar precision and recall. The overall results on this dataset are not surprising, as this is a very old dataset composed of a combination of 1d and 2d reads, of average error rate of $\approx 33\%$ (Appendix A, Table A.1).

Datasets 2 and 3 show better results for all overlappers compared to Dataset 1. Also, the results are similar between Dataset 2 and Dataset 3 for all overlappers (error rate of the datasets is $\approx 16\%$), which is reasonable considering that Dataset 2 is actually a subsampled version of Dataset 3. Owler shows highest recall, while precision is high and similar to the one of Minimap.

Dataset 4 (error rate $\approx 11\%$) shows significant improvement in precision for all overlappers (especially DALIGNER) and in recall (especially MHAP). Owler still has the highest precision, while the recall is only slightly lower than MHAP's.

Dataset 5, although having the lowest error rate of $\approx 10\%$, results in an interesting twist in precision for all overlappers except Minimap which scored highest, followed by Owler. This is perhaps due to the different profile of errors in Dataset 5 (3% insertions, 2% deletions and 5% mismatches) compared to Datset 4 (2% insertions, 3% deletions and 6% mismatches). Other overlappers score better in terms of recall on this Dataset compared to any previous one, with Minimap scoring the highest, and Owler following with a small difference.

Owler's sensitivity and precision unfortunately come at the expense of time - although usually being $2-3x$ faster than BLASR, other overlappers (Minimap, MHAP and DALIGNER) were faster on these datasets.

4.2.4 Discussion

In this section we developed two new overlapping methods, both based on the GraphMap mapper for third generation sequencing data. The initial overlapping mode in GraphMap ("GraphMap overlap" in Table 4.6) showed promising results in terms of precision and recall, considering that, at the time of development only DALIGNER and BLASR were published. Although GraphMap allows tuning of the precision vs. recall via the E-value parameter and the parameter which controls the number of regions to inspect, the overall overlapping time is relatively high. This is due to the laborious steps used in the entire GraphMap pipeline to ensure high sensitivity in the mapping context. However, not all steps are necessary in the context of overlapping.

For this reason, we developed an overlapper (Owler) which runs on the reduced pipeline, aimed at achieving higher speeds, while at the same time providing optimizations for the overlapping context, which enable higher precision, recall and overall $F1$ scores when compared to GraphMap overlap and other overlapping methods on most datasets.

Table 4.6: Results of comparison between the newly developed *Owler* and *GraphMap overlap* methods and the current state-of-the-art for overlapping raw third generation sequencing reads.

		Precision [%]	Recall [%]	F1 [%]	CPU time [min]	Memory [MB]
Dataset 1	**Owler**	**98.45**	**3.88**	**7.47**	113.79	16328.52
E. Coli	**GraphMap overlap**	96.07	8.90	16.29	1253.46	12476.82
R7.3	**Minimap**	**99.24**	**3.89**	**7.48**	2.20	4600.59
	MHAP	93.68	1.49	2.93	45.74	65375.40
	DALIGNER	9.54	0.13	0.25	12.32	14817.90
	BLASR	53.04	2.20	4.23	278.66	2985.11
Dataset 2	**Owler**	**97.74**	**71.41**	**82.52**	21.64	4916.00
20x subsampled	**GraphMap overlap**	91.87	72.93	81.31	365.86	3936.54
Loman et al.	**Minimap**	**98.08**	**62.37**	**76.25**	0.61	1143.58
	MHAP	90.76	47.77	62.60	13.30	34964.13
	DALIGNER	87.27	56.99	68.95	5.08	4291.71
	BLASR	81.24	38.63	52.36	62.78	864.09
Dataset 3	**Owler**	**98.00**	**71.59**	**82.74**	43.98	7142.74
30x	**GraphMap overlap**	93.52	72.49	81.68	501.80	5551.98
Loman et al.	**Minimap**	**98.58**	**65.26**	**78.53**	1.01	2161.65
	MHAP	91.19	47.12	62.14	19.25	45696.46
	DALIGNER	87.57	59.94	71.17	10.53	6360.15
	BLASR	81.88	36.47	50.46	119.78	1281.07
Dataset 4	**Owler**	**99.58**	**85.74**	**92.15**	168.81	9961.81
40x	**GraphMap overlap**	95.40	8.29	15.26	709.64	7600.49
MARC	**Minimap**	96.92	4.86	9.26	1.73	2488.88
	MHAP	97.72	**87.64**	**92.41**	61.32	74427.35
	DALIGNER	**98.25**	50.30	66.54	59.05	16717.54
	BLASR	89.11	4.46	8.50	199.65	1791.64
Dataset 5	**Owler**	**96.71**	**86.28**	**91.20**	181.12	13021.03
50x	**GraphMap overlap**	94.60	82.45	88.11	937.54	9894.71
MAP006	**Minimap**	**98.47**	**90.64**	**94.39**	3.09	3410.43
	MHAP	77.73	83.42	80.47	35.77	69543.33
	DALIGNER	79.18	84.67	81.83	29.30	30050.71
	BLASR	85.17	44.91	58.81	323.19	2380.59

Chapter 5

Racon - Fast consensus module for raw *de novo* genome assembly of long uncorrected reads

Fast *de novo* assembly, recently published by Li [89], set in motion an avalanche of possible applications which yet need to be explored in depth, such as: rapid assembly of mammalian and plant genomes; fast structural variation detection; higher quality quick metagenomic identification (by first assembling the dataset and then using the generated contigs for database search); and "read until" sequencing of Oxford Nanopore's MinION systems where a fast assembly could help decide if enough data have been collected. Also, as noted by Li [89], the doors to fast assembly are only open if a fast consensus tool, matching the speed of Minimap and Miniasm, is developed as well.

In this chapter we discuss the development of *Racon*, our new standalone consensus module, intended for correcting raw third generation data assemblies. We show that Racon, when paired with Miniasm, provides *an order of magnitude* (see Results) faster assemblies while being of comparable or better quality to the assemblers which employ both the error correction and the consensus phases (Canu [83] (successor to Celera, unpublished), FALCON [90] and Loman *et al.* pipeline [135]). In addition to contig consensus, the generality of Racon allows it to also be used as a fast and accurate read error-correction tool.

5.1 Methods

Racon is based on the *Partial Order Alignment* (POA) graph approach [121][122], similar to some of the most sensitive consensus methods for third generation sequencing data today: Quiver [101] and Nanocorrect [135]. In this section, a detailed overview of the algorithm, the aspects of implementation and the evaluation methods are given.

5.1.1 Algorithm description

An overview of Racon's steps is given in Figure 5.1. The entire process is also shown in detail in Algorithm 15, and is as follows. To perform consensus calling (or error-correction), Racon depends on an input set of query-to-target overlaps (query is the set of reads, while a target is either a set of contigs in the consensus context, or a set of reads in the error-correction context). Overlaps can be generated with any overlapper which supports either MHAP or PAF output formats, such as Minimap [89], MHAP [103] or GraphMap [14]. In all our tests we used Minimap as it was the fastest. Racon then loads the overlaps and performs simple filtering (Algorithm 15, lines $1-3$; Algorithm 16): (I) at most one overlap per read is kept in consensus context (in error-correction context this particular filtering is disabled), and (II) overlaps which have $|1-\min(d_q,d_t)/\max(d_q,d_t)| \geq e$ are removed, where d_q is the length of the overlap in the query, d_t length in the target, and e a user-specified error-rate threshold. For each overlap which survived the filtering process, a fast edit-distance based alignment is performed [38] (Algorithm 15, lines $4-11$). Alignments are grouped by contigs, and each contig is processed separately in the same manner.

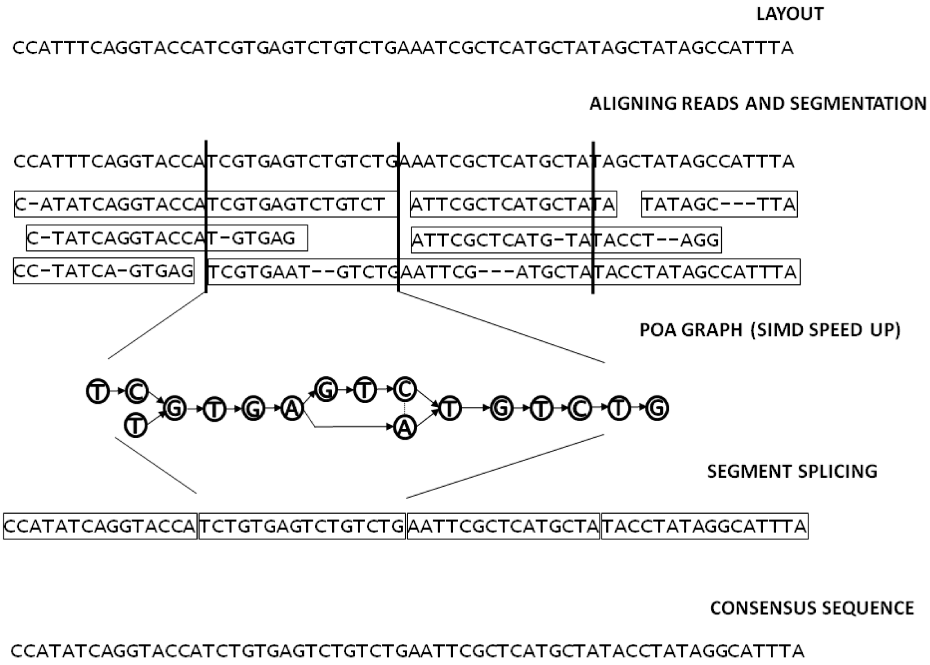

LAYOUT
CCATTTCAGGTACCATCGTGAGTCTGTCTGAAATCGCTCATGCTATAGCTATAGCCATTTA
ALIGNING READS AND SEGMENTATION
CCATTTCAGGTACCATCGTGAGTCTGTCTGAAATCGCTCATGCTATAGCTATAGCCATTTA
C-ATATCAGGTACCATCGTGAGTCTGTCT
ATTCGCTCATGCTATA
TATAGC---TTA
C-TATCAGGTACCAT-GTGAG
ATTCGCTCATG-TATACCT--AGG
CC-TATCA-GTGAG
TCGTGAAT--GTCTGAATTCG---ATGCTATACCTATAGCCATTTA
POA GRAPH (SIMD SPEED UP)
T C T G T G A G T C A T G T C T G
SEGMENT SPLICING
CCATATCAGGTACCA
TCTGTGAGTCTGTCTG
AATTCGCTCATGCTA
TACCTATAGGCATTTA
CONSENSUS SEQUENCE
CCATATCAGGTACCATCTGTGAGTCTGTCTGAATTCGCTCATGCTATACCTATAGGCATTTA

Figure 5.1: Overview of the Racon consensusprocess.

Inspired by Quiver and Nanopolish, Racon divides a raw input sequence (contigs, reads, etc.) into consecutive, non-overlapping windows of length w (Algorithm 15, line 16). In the context of consensus calling, each window is processed separately in it's own thread, while in the context of read error-correction, an entire read (regardless of the number of windows) is entirely assigned to a thread at a time for efficiency purposes.

Algorithm 15: Racon algorithm for fast and accurate consensus of noisy contigs

```
Input: Set of noisy target sequences (e.g. contigs) $\mathcal{T}$ to perform consensus on, set of read sequences $\mathcal{Q}$
       where each $q \in \mathcal{Q}$ is a tuple consisting of $(seq, qual)$ (sequence data and quality values), set of
       overlaps $\mathcal{O}$ between $\mathcal{Q}$ and $\mathcal{T}$, window size $w$, minimum allowed quality value $qv_{min}$, mode $m$ to
       run the Racon in (can be either "consensus" or "errorcorrection"), and maximum expected error
       rate $e$. An overlap $o \in \mathcal{O}$ is a tuple of 12 values: $(q_{id}, t_{id}, p_{cb}, n_k, q_{rev}, q_s, q_e, q_{len}, t_{rev}, t_s, t_e, t_{len})$,
       where $q_{id}$ and $t_{id}$ are the query and target sequence IDs, $p_{cb}$ percentage of query bases covered by
       the overlap, $n_k$ number of seeds in the overlap, $q_{rev}$ and $t_{rev}$ are 0 if query and target are forward
       oriented and 1 otherwise, $q_s, q_e, t_s, t_e$ are the start and end coordinates of the overlap in the query
       and target, and $q_{len}, t_{len}$ total query and target sequence lengths.
Output: Set of corrected target sequences $\mathcal{T}_c$

Function RACON($\mathcal{T}, \mathcal{Q}, \mathcal{O}, w, qv_{min}, m, e$) begin
1       $\mathcal{O}_f \leftarrow FilterErroneousOverlaps(\mathcal{O}, e)$
2       if $m \neq$ "errorcorrection" then
3           $\mathcal{O}_f \leftarrow FilterUniqueOverlaps(\mathcal{O}_f)$

4       $\mathcal{A} \leftarrow$ empty hash table of sets                 ▷ Aligned overlaps grouped by $t_{id}$
5       foreach $o \in \mathcal{O}_f$ do                                 ▷ Align filtered overlaps
6           $(q_{id}, t_{id}, p_{cb}, n_k, q_{rev}, q_s, q_e, q_{len}, t_{rev}, t_s, t_e, t_{len}) \leftarrow o$
7           $q_{seq} \leftarrow \mathcal{Q}[q_{id}].seq[q_s ... q_e]$
8           $q_{qual} \leftarrow \mathcal{Q}[q_{id}].qual[q_s ... q_e]$
9           $t_{seq} \leftarrow \mathcal{T}[t_{id}][t_s ... t_e]$
10          $aln \leftarrow$ bit-vector global alignment between $q_{seq}$ and $t_{seq}$
11          $\mathcal{A}[t_{id}] \leftarrow \mathcal{A}[t_{id}] \cup (o, aln, q_{seq}, q_{qual}, t_{seq})$

12      $\mathcal{T}_c \leftarrow \emptyset$                              ▷ A set for consensus sequences
13      foreach $t_{id} \in \mathcal{A}$ do                              ▷ Process each target sequence individually
14          $\mathcal{A}_t \leftarrow \mathcal{A}[t_{id}]$
15          $t_c \leftarrow$ empty string
16          for $w_{id} = 0$ to $\lceil \mathcal{T}[t_{id}]/w \rceil$ do   ▷ Process a target in non-overlapping windows
17              $w_s \leftarrow w_{id} \cdot w$                          ▷ Window start position on target seq
18              $w_e \leftarrow (w_{id} + 1) \cdot w - 1$                ▷ Window end position on target seq

19              $s_{tseq} \leftarrow \mathcal{T}[t_{id}][w_s ... w_e]$    ▷ Noisy target subsequence for SPOA
20              $s_{tqual} \leftarrow$ an array of $|s_{tseq}|$ elements initialized to 0   ▷ Dummy QVs

21              $\mathcal{S} \leftarrow \emptyset$
22              foreach $(o, aln, q_{seq}, q_{qual}, t_{seq}) \in \mathcal{A}_t$ do   ▷ Get subsequences of all reads
23                  $(s_{qseq}, s_{qqual}) \leftarrow$ extract a substring of $s_{qseq}$ which aligns between $w_s$ and $w_e$ on target
                      with corresponding quality values $q_{qqual}$
24                  if $average(s_{qqual}) \geq qv_{min}$ then
25                      $\mathcal{S} \leftarrow \mathcal{S}(s_{qseq}, s_{qqual})$

26              $\mathcal{G} \leftarrow BuildPoaGraph(s_{tseq}, s_{tqual})$
27              foreach $(s_{qseq}, s_{qqual}) \in \mathcal{S}$ do               ▷ Align all sequences to graph
28                  $\mathcal{G} \leftarrow SPOAAlignSequence(\mathcal{G}, s_{qseq}, s_{qqual})$

29              $cons \leftarrow GenerateConsensus(\mathcal{G})$
30              $t_c \leftarrow$ concatenate $cons$ to $t_c$

31          $\mathcal{T}_c \leftarrow \mathcal{T}_c \cup t_c$

32      return $\mathcal{T}_c$
```

Given start (w_s) and end (w_e) coordinates of a window on target, all alignments overlapping this region are extracted, including the substring of the target sequence (Algorithm 15, lines $17-25$). An initial simple POA graph is first constructed from the substring of the target

Algorithm 16: Filter overlaps for alignment in Racon

```
Input: A set of overlaps $\mathcal{O}$ and expected maximum error rate $e$
Output: Set of filtered overlaps $\mathcal{O}_f$
Function FilterErroneousOverlaps($\mathcal{O}, e$) begin
    $\mathcal{O}_f \leftarrow \emptyset$
    foreach $o \in \mathcal{O}$ do
        $(q_{id}, t_{id}, p_{cb}, n_k, q_{rev}, q_s, q_e, q_{len}, t_{rev}, t_s, t_e, t_{len}) \leftarrow o$
        $o_{min} \leftarrow \min(|q_e - q_s|, |t_e - t_s|)$        ▷ Minimum overlap length
        $o_{max} \leftarrow \max(|q_e - q_s|, |t_e - t_s|)$        ▷ Maximum overlap length
        if $(1 - o_{min}/o_{max}) > e$ then        ▷ Max overlap indel error rate
            continue
        $\mathcal{O}_f \leftarrow \mathcal{O}_f \cup o$
    return $\mathcal{O}_f$

Input: A set of overlaps $\mathcal{O}$
Output: Set of filtered overlaps $\mathcal{O}_f$
Function FilterUniqueOverlaps($\mathcal{O}$) begin
    $\mathcal{H} \leftarrow$ empty hash table
    foreach $o \in \mathcal{O}$ do
        $(q_{id}, t_{id}, p_{cb}, n_k, q_{rev}, q_s, q_e, q_{len}, t_{rev}, t_s, t_e, t_{len}) \leftarrow o$
        if $t_{id} \in \mathcal{H}$ then        ▷ Keep only one overlap per target
            $(q_{h,id}, t_{h,id}, p_{h,cb}, n_{h,k}, q_{h,rev}, q_{h,s}, q_{h,e}, q_{h,len}, t_{h,rev}, t_{h,s}, t_{h,e}, t_{h,len}) \leftarrow \mathcal{H}[t_{id}]$
            if $n_k > n_{h,k}$ then
                $\mathcal{H}[t_{id}] \leftarrow o$
        else
            $\mathcal{H}[t_{id}] \leftarrow o$
    $\mathcal{O}_f \leftarrow \emptyset$
    foreach $t_{id} \in \mathcal{H}$ do
        $\mathcal{O}_f \leftarrow \mathcal{O}_f \cup \mathcal{H}[t_{id}]$
    return $\mathcal{O}_f$
```

sequence (Algorithm 15, lines 26). Based on the alignment of a query read overlapping the region, coordinates w_s and w_e can be mapped to actual bases q_s and q_e on the query. Query is then trimmed to those coordinates and a tuple (s_{qseq}, s_{qqual}) compiled, where s_{qseq} is the sequence of the query in range $[q_s, q_e]$, and s_{qqual} are the qualities of the corresponding sequence. A set of tuples is collected for all queries overlapping the region, and used as the input for building a local POA graph using global alignment (Algorithm 15, lines $27-28$; Algorithm 18). The consensus sequence is then called by finding a heaviest traversal through the POA graph, and output as the corrected sequence corresponding to the window (Algorithm 15, lines $29-30$; Algorithm 20).

POA performs Multiple Sequence Alignment (MSA) through a directed acyclic graph (DAG), where nodes are individual bases of input sequences, and weighted, directed edges represent whether two bases are neighbouring in any of the sequences. Weights of the edges represent the multiplicity (coverage) of each transition. Alternatively, weights can be set according to the base qualities of sequenced data. The alignment is carried out directly through dynamic programming (DP) between a new sequence and a pre-built graph. In comparison, whereas the regular DP pairwise alignment has time complexity of $O(3nm)$, where n and m are the lengths of the sequences being aligned, the sequence to graph alignment has a complexity of $O((2n_p + 1)n|\mathcal{V}|)$, where n_p is an average number of predecessors in the graph and $|\mathcal{V}|$ is the number of nodes in the graph. Consensus sequences are obtained from a built POA graph by performing a topological sort and processing the nodes from left to right. For each node v, the highest-weighted in-edge e of weight e_w is chosen, and a score is assigned to v such that $scores[v] = e_w + scores[w]$, where w is the source node of the edge e [122]. The node w is marked as a predecessor of v, and a final consensus is generated by performing a traceback from the highest scoring node r. In case r is an internal node (r has out edges), Lee [122] proposed an approach of *branch completion*, where all scores for all nodes except $scores[r]$ would be set to a negative value, and the traversal would continue from r the same as before, with the only exception that nodes with negative score could not be added as predecessors to any other node. Consensus generation procedure is shown in detail in Algorithm 20. One of the biggest advantages of POA compared to other MSA algorithms is its speed - linear time complexity in the number of sequences [121]. However, even though faster than other MSA, the implementations of POA in current error-correction modules, such as Nanocorrect, are prohibitively slow for larger datasets.

In order to increase the speed of POA, we made two significant modifications to the algorithm: (I) a *Single Instruction Multiple Data (SIMD) modification* of the base POA algorithm and (II) *subgraph alignment*. To the best of our knowledge, this is the first SIMD POA modification in literature - we named our implementation *SPOA* accordingly. SPOA was developed in close collaboration with Robert Vaser, who proposed the initial idea and implemented the SIMD alignment algorithm.

Our SIMD modification (Figure 5.2; Algorithm 18) is based on the Rognes and Seeberg Smith-Waterman intra-set parallelization approach [39]. As in the original Rognes-Seeberg paper, SPOA places the SIMD vectors parallel to the query sequence (the read), while instead of the reference sequence a graph is placed on the other dimension of the DP matrix (Figure 5.2). The SWAT-otimizations described by Rognes and Seeberg [39] are left out in order to avoid branching in the inner loop and code duplication as three different alignment modes are implemented in SPOA. In our implementation, matrices **H** and **F**, which are used for tracking the maximum local-alignment scores ending in gaps [39], are stored entirely in memory which increases the memory complexity from linear to quadratic (Algorithm 18, line 5). These matrices are needed to access scores of predecessors of particular nodes during alignment. Unlike regular Gotoh alignment, for each row in the POA DP matrix all its predecessors (via in-edges of the corresponding node in graph) need to be processed as well (Algorithm 18, line 18). All

Algorithm 17: Constructs a POA graph from a given sequence (as described in [121])

```
Struct PoaNode begin
    $id \leftarrow -1$                              ▷ ID of the node
    $base \leftarrow$ "–"                           ▷ Nucleotide base represented by the node
    $\mathcal{E}_i \leftarrow$ empty array          ▷ Array of in edges
    $\mathcal{E}_o \leftarrow$ empty array          ▷ Array of out edges
    $\mathcal{N}_{id} \leftarrow$ empty array       ▷ Array of IDs of nodes aligned to this one
Struct PoaEdge begin
    $s \leftarrow -1$                               ▷ ID of the begin node
    $e \leftarrow -1$                               ▷ ID of the end node
    $w \leftarrow 0$                                ▷ Total weight of the edge

Input: A sequence $s$ used to build the graph and corresponding quality values $q$ for each base of $s$ to be
        used as edge weights
Output: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of vertices and $\mathcal{E}$ a set of edges
Function BuildPOAGraph($s, q$) begin
    $\mathcal{V} \leftarrow \emptyset$
    $\mathcal{E} \leftarrow \emptyset$
    for $i = 0$ to $(|s| - 1)$ do
        $\mathcal{V} \leftarrow$ Add a new PoaNode to the set, where $id \leftarrow i$ and $base \leftarrow s[i]$
        if $i > 0$ then
            $\mathcal{E} \leftarrow$ Add a new PoaEdge to the set, beginning at node $(i-1)$ with weight
              $w \leftarrow (q[i-1] + q[i])$
    $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
    return $\mathcal{G}$
```

columns are then processed using SIMD operations in a query-parallel manner described in [39], and the values of Gotoh's vertical matrix (Algorithm 18, line 22) and a partial update to Gotoh's main scoring matrix (Algorithm 18, lines 24 and 26) are calculated. SIMD operations in Algorithm 18 process 8 cells of the DP matrix at a time (16-bit registers). A temporary variable is used to keep the last cell of the previous vector for every predecessor (Algorithm 18, lines $23-25$), which is needed to compare the upper-left diagonal of the current cell to the cell one row up. Processing the matrix horizontally is not performed using SIMD operations due to data dependencies (each cell depends on the result of the cell left to it), and are instead processed linearly (Algorithm 18, lines $27-38$). We use shifting and masking to calculate every particular value of a SIMD vector individually (Algorithm 18, lines 32 and 35). After the alignment is completed, the traceback is performed (Algorithm 18, line 39) and integrated into the existing POA graph (Algorithm 18, line 40, Algorithm 19).

The resulting time complexity for all alignment modes is $O((\frac{2n_p}{k}+1)NM)$, where k is the number of variables that fit in a SIMD vector. Overall maximal speed-up can be computed as $(2n_p+1)/(2n_p/k+1)$. SPOA supports Intel SSE version 4.1 and higher, which embed 128 bit registers. Both short (16 bits) and long (32 bits) integer precisions are supported by SPOA and therefore k equals 8 and 4 variables respectively (Algorithm 18, line 3)(8 bits precision is insufficient for the intended application of SPOA and is therefore not used).

Algorithm 18: Align a sequence to a pre-constructed graph using SIMD instructions. max_s, or_s, add_s, sub_s, $lshift_s$ and $rshift_s$ are SIMD operations.

```
Input: A POA graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a sequence $s_{seq}$ to align to the graph and the quality values $s_{qual}$ of the
       sequence (edge weights)
Output: A POA graph $\mathcal{G}_a = (\mathcal{V}_a, \mathcal{E}_a)$ with incorporated $s_{seq}$ and $s_{qual}$
Function SPOAALIGNSEQUENCE($\mathcal{G}, s_{seq}, s_{qual}, match, mismatch, gap_{open}, gap_{extend}, mode$) begin
    $b \leftarrow -2^{15} + 1000$        ▷ Big negative value which fits in a SIMD register but will not overflow
    $x \leftarrow |\mathcal{V}|$
    $y \leftarrow |s_{seq}|/8$        ▷ 8 variables per SIMD vector
    $\Sigma \leftarrow \{A, C, T, G\}$        ▷ Alphabet
    $\mathcal{H}, \mathcal{F} \leftarrow$ matrices of $x \times y$ SIMD vectors
    $opn, ext \leftarrow$ two single SIMD vectors, elements initialized to $gap_{open}$ and $gap_{extend}$, respectively
    $fcv \leftarrow$ is an integer array of $x$ elements initialized to 0        ▷ First column values
    $X, score \leftarrow$ two single SIMD vectors initialized to 0
    $\mathcal{M} \leftarrow$ 8 SIMD vectors, init'ed to $[[b,0,0,0,0,0,0,0],[b,b,0,0,0,0,0,0],...,[b,b,b,b,b,b,b,b]]$▷ Masks
    $\mathcal{P} \leftarrow$ Calculate query profile of $|\Sigma| \times y$ SIMD vector elements for $s_{seq}$
    $\mathcal{G}_s = (\mathcal{V}_s, \mathcal{E}_s) \leftarrow$ topologically sorted $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
    if $mode = NW$ then
        for $i = 0$ to $(y-1)$ do
            $\mathcal{H}[0][i] \leftarrow gap_{open} + (i+1) \cdot gap_{extend}$
            $fcv[i] \leftarrow gap_{open} + (i+1) \cdot gap_{extend}$
    for $i = 0$ to $(|\mathcal{V}_s| - 1)$ do
        $v \leftarrow \mathcal{V}_s[i]$
        for $p_{id} = 0$ to $(|v.\mathcal{E}_i| - 1)$ do        ▷ Process all predecessors of current node
            $p \leftarrow v.\mathcal{E}_i[p_{id}].s$
            $X \leftarrow rshift_s(simd\_vector(fcv[p]), 7)$
            for $j = 0$ to $(y-1)$ do        ▷ Solve the element of vertical Gotoh matrix
                $\mathcal{F}[i][j] \leftarrow$        ▷ Analogous to Equation 2.5
                $max_s(\mathcal{F}[i][j], add_s(max_s(add_s(\mathcal{H}[p][j], \mathcal{G}_{open}), \mathcal{F}[p][j]), \mathcal{G}_{ext}))$
                $T_1 \leftarrow rshift_s(\mathcal{H}[p][j], 7)$
                $\mathcal{H}[i][j] \leftarrow or_s(lshift_s(\mathcal{H}[p][j], 1), X)$
                $X \leftarrow T_1$
                $\mathcal{H}[i][j] \leftarrow max_s(\mathcal{H}[i][j], max_s(add_s(\mathcal{H}[p][j], \mathcal{P}[v.base][j]), \mathcal{F}[i][j]))$
        $E \leftarrow simd\_vector(fcv[i])$
        for $j = 0$ to $(y-1)$ do        ▷ Solve the element of horizontal Gotoh matrix
            $E \leftarrow add_s(add_s(or_s(lshift_s(\mathcal{H}[i][j], 1), rshift_s(E, 7)), opn), ext)$
            $T_2 \leftarrow E$
            $ext_{temp} \leftarrow ext$
            for $k = 0$ to 7 do        ▷ SIMD parallelization cannot be used due to dependencies
                $ext_{temp} \leftarrow lshift_s(ext_{temp}, 1)$
                $T_2 \leftarrow add_s(lshift_s(T_2, 1), ext_{temp})$
                $E \leftarrow max_s(E, or_s(\mathcal{M}[k], T_2))$
            $\mathcal{H}[i][j] \leftarrow max_s(\mathcal{M}[j], E)$
            $E \leftarrow max_s(\mathcal{M}[i][j], sub_s(E, opn))$
            $score \leftarrow max_s(score, \mathcal{H}[i][j])$
    $\mathcal{A} \leftarrow$ traceback using standard Gotoh approach, taking in account that a cell can have multiple
      predecessors. SIMD vectors are converted to normal ints when needed. Return is an array of tuples
      $(s_{seq,id}, v_{id})$ where $s_{seq,id}$ is the id of a base on $s_{seq}$ or $-1$ if a gap, and $v_{id}$ is the node id in graph
      where $s_{seq}[s_{seq,id}]$ was aligned to (or $-1$ if gap in graph)
    $AddAlignmentToGraph(G_s, alignment, s_{seq}, s_{qual})$
    return $\mathcal{G}_s$
```

Algorithm 19: Adds an aligned sequence to an existing POA graph

```
Input: A POA graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, alignment $aln$ in the form of an array of tuples $(s_{seq,id}, v_{id})$ where $s_{seq,id}$ is
       the id of a base on $s_{seq}$ or $-1$ if a gap, and $v_{id}$ is the node id in graph where $s_{seq}[s_{seq,id}]$ was aligned
       to (or $-1$ if gap in graph), a sequence $s_{seq}$ to align to the graph and the quality values $s_{qual}$ of the
       sequence (edge weights)

Function AddAlignmentToGraph($\mathcal{G} = (\mathcal{V}, \mathcal{E}), aln, s_{seq}, s_{qual}$) begin
    $v_{id,s} \leftarrow -1$                                      ▷ Start node ID
    $v_{id,h} \leftarrow -1$                                      ▷ Head node ID
    $w_{prev} \leftarrow 0$                                       ▷ Previous weight
    for $i = 0$ to $|aln|$ do                                     ▷
        $(s_{seq,id}, v_{id}) \leftarrow aln[i]$
        if $s_{seq,id} = -1$ then
            continue
        if $v_{id} = -1$ then                  ▷ There is a gap in the graph, a new node needs to be added
            $v \leftarrow$ new $POANode$, initialize $v.id \leftarrow |\mathcal{V}|$
            $\mathcal{V} \leftarrow \mathcal{V} \cup v$
        else
            $v \leftarrow NULL$
            foreach $a \in \mathcal{V}[v_{id}].\mathcal{N}_{id}s$ do
                if $\mathcal{V}[a].base = s_{seq}[s_{seq,id}]$ then
                    $v \leftarrow \mathcal{V}[a]$
            if $v = NULL$ then                                    ▷ Base is aligned to a mismatch node
                $v \leftarrow$ new $POANode$, initialize $v.id \leftarrow |\mathcal{V}|$ and set $v.\mathcal{N}_{id}$ to include all $\mathcal{V}[v_{id}].\mathcal{N}_{id}$
                foreach $a \in \mathcal{V}[v_{id}].\mathcal{N}_{id}$ do
                    $\mathcal{V}[a].\mathcal{N}_{id} \leftarrow$ Append $v.id$
        if $v_{id,s} = -1$ then
            $v_{id,s} \leftarrow v.id$
        if $v_{id,h} \neq -1$ then
            $e \leftarrow$ new $PoaEdge$ with $e.s \leftarrow v_{id,h}, e.e \leftarrow v.id, e.w \leftarrow (w_{prev} + s_{qual}[s_{seq,id}])$
            $\mathcal{E} \leftarrow \mathcal{E} \cup e$                                   ▷ Add edge
        $v_{id,h} \leftarrow v.id$
        $w_{prev} \leftarrow s_{qual}[s_{seq,id}]$
```

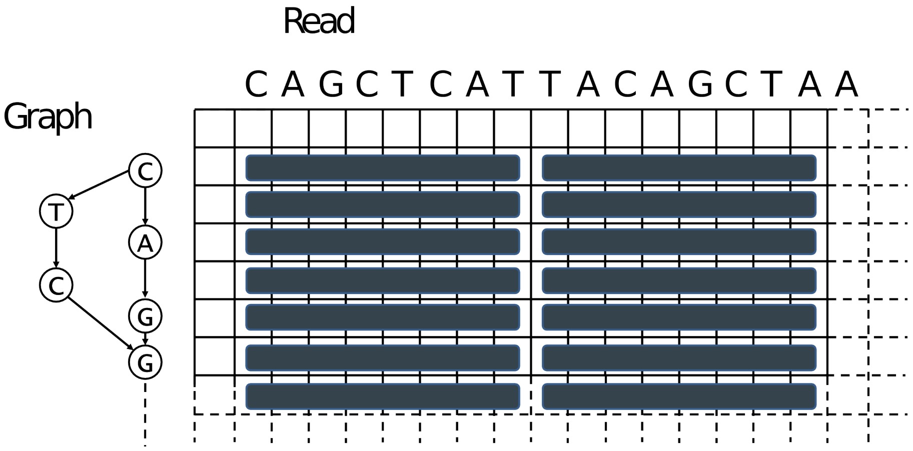


Figure 5.2: Depiction of the SIMD vectorization used in our POA implementation.

Algorithm 20: Generates a consensus sequence from a given POA graph

```
Input: A POA graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of vertices (nodes) and $\mathcal{E}$ a set of edges
Output: A consensus sequence $cons$

Function GenerateConsensus($\mathcal{G}$) begin
    $\mathcal{G}_s = (\mathcal{V}_s, \mathcal{E}_s) \leftarrow$ topologically sorted $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
    $scores \leftarrow$ an array of length $|\mathcal{V}_s|$ with all values initialized to 0
    $predecessor \leftarrow$ an array of length $|\mathcal{V}_s|$ with all values initialized to $-1$
    $r \leftarrow -1$                          ▷ Index of the end node of the consensus (if $\neq -1$)
    for $i = 0$ to $(|\mathcal{V}_s| - 1)$ do                ▷ TRAVERSE_HB algorithm from [122]
        $e_{max} \leftarrow NULL$
        foreach $e \in \mathcal{V}_s[i].\mathcal{E}_i$ do       ▷ Find an in-edge with maximum weight (local optimization)
            if $e_{max} = NULL$ or $e.w > e_{max}.w$ or $(e.w = e_{max}.w$ and $scores[e.s] >= scores[e_{max}.s])$ then
                $e_{max} \leftarrow e$
        if $e_{max} \neq NULL$ then
            $predecessor[i] \leftarrow e_{max}.s$     ▷ Traceback, $e_{max}.s$ is the beginning node of the in-edge.
            $scores[i] \leftarrow scores[e_{max}.s] + e_{max}.w$
            if $r = -1$ or $scores[i] > scores[r]$ then
                $r \leftarrow i$
    while $r \neq NULL$ and $|\mathcal{V}_s[r].\mathcal{E}_o| > 0$ do   ▷ If r is not an end node, do BRANCH_COMPLETION [122]
        for $i = 0$ to $(|\mathcal{V}_s| - 1)$ do            ▷ Set all scores except max to a neg. value
            $scores[i] \leftarrow -1$ if $i \neq r$
        for $i = (r+1)$ to $(|\mathcal{V}_s| - 1)$ do         ▷ Process the rest of the top. sorted graph
            $e_{max} \leftarrow NULL$
            foreach $e \in \mathcal{V}_s[i].\mathcal{E}_i$ do   ▷ Find an in-edge with maximum weight (local optimization)
                if $scores[e.s] = -1$ then                ▷ The main difference from HeaviestBundle
                    continue
                if $e_{max} = NULL$ or $e.w > e_{max}.w$ or
                $(e.w = e_{max}.w$ and $scores[e.s] >= scores[e_{max}.s])$ then
                    $e_{max} \leftarrow e$
            if $e_{max} \neq NULL$ then
                $predecessor[i] \leftarrow e_{max}.s$   ▷ Traceback, $e_{max}.s$ is the beginning node of the in-edge.
                $scores[i] \leftarrow scores[e_{max}.s] + e_{max}.w$
                if $r = -1$ or $scores[i] > scores[r]$ then
                    $r \leftarrow i$
    $cons \leftarrow$empty string
    while $r \neq NULL$ and $predecessor[r] \neq -1$ do        ▷ Traverse predecessors to generate consensus
        $cons \leftarrow$ Add $\mathcal{V}_s[r].base$ to front of $cons$
        $r \leftarrow predecessor[r]$
    $cons \leftarrow$ Add $\mathcal{V}_s[r].base$ to front of $cons$ if $r \neq NULL$
    return $cons$
```

The second modification implemented in SPOA, *subgraph alignment*, allows a much faster read-to-graph alignment in case a rough mapping position of the read is known in advance. The target start and end positions for alignment are then located on the graph (with the surrounding neighbourhood), and the read is aligned only with such defined subgraph. Both vectorization and subgraph extraction are provided in any of the three alignment modes implemented in SPOA: global, local and semiglobal alignment.

5.1.2 Implementation and reproducibility

Racon and SPOA are both implemented in C++. All tests were ran using Ubuntu based systems with two 6-core Intel(R) Xeon(R) E5645 CPUs @ 2.40GHz with Hyperthreading, using 12 threads where possible. Version of methods used in comparison:

- Minimap - `https://github.com/lh3/minimap.git`, commit: $1cd6ae3bc7c7$
- Miniasm - `https://github.com/lh3/miniasm.git`, commit: $17d5bd12290e$
- Canu - `https://github.com/marbl/canu.git`, version 1.2, commit: $ab50ba3c0cf0$.
- FALCON-integrate project - `https://github.com/PacificBiosciences/FALCON-integrate.git`, commit: $8bb2737fd1d7$.
- Nanocorrect - `https://github.com/jts/nanocorrect.git`, commit: $b09e93772ab4$.
- Nanopolish - `https://github.com/jts/nanopolish.git`, commit: $47dcd7f147c$.
- MUMmer - DNAdiff version 1.3, NUCmer version 3.1.

5.1.3 Datasets

Four real, publicly available PacBio and Oxford Nanopore datasets were used for evaluation. These are:

1. *Lambda* phage, Oxford Nanopore, ENA submission ERA476754, coverage $113x$ of the NC_001416 reference genome ($48502bp$). Link: `ftp://ftp.sra.ebi.ac.uk/vol1/ERA476/ERA476754/oxfordnanopore_native/Lambda_run_d.tar.gz`. This dataset was subsampled to coverages 30x and 81x for testing.
2. *E. coli* K-12 MG1655 SQK-MAP006-1 dataset, Oxford Nanopore, R7.3 chemistry, $54x$ pass 2D coverage of the genome ($U00096.3$, $4.6Mbp$). Link: `http://lab.loman.net/2015/09/24/first-sqk-map-006-experiment/`
3. *E. Coli* K-12 PacBio P6C4 dataset, $160x$ coverage of the genome ($U00096.3$). The dataset was generated using one SMRT Cell of data gathered with a PacBio RS II System and P6-C4 chemistry on a size selected $20kbp$ library of *E. coli* K-12. Link: `https://s3.amazonaws.com/files.pacb.com/datasets/secondary-analysis/e-coli-k12-P6C4/p6c4_ecoli_RSII_DDR2_with_15kb_cut_E01_1.tar.gz`
4. C. Elegans, a Bristol mutant strain, $81x$ coverage of the genome ($gi|449020133$) The dataset was generated using 11 SMRT cells P6-C4 chemistry on a size selected $20kbp$ library. Link: `https://github.com/PacificBiosciences/DevNet/wiki/C.-elegans-data-set`

.

5.1.4 Evaluation methods

The quality of called consensus was evaluated using Dnadiff [126], a well established method for assessment of assembly quality. The parameters we took into consideration for comparison include: total number of bases in the query, aligned bases on the reference, aligned bases on the query and average identity. In addition, we measured the time required to perform the entire assembly process by each pipeline.

The quality of error-corrected reads was evaluated by aligning them to the reference genome using GraphMap ([14]) with settings "-a anchorgotoh", and counting the match, mismatch, insertion and deletion operations in the resulting alignments.

5.2 Results

We developed a fast consensus module called *Racon* (Rapid Consensus) which, when paired with Miniasm, provides *an order of magnitude* (Table 5.1) faster assemblies while being of comparable or better quality to the assemblers which employ both the error correction and the consensus phases (Canu, FALCON and Loman *et al.* pipeline). Furthermore, applying Nanopolish on Miniasm+Racon outputs achieves the same Avg. Identity as the full Loman *et al.* pipeline (Table 5.2). Moreover, we show here that Miniasm+Racon assemblies in almost all cases result in contigs of length more similar to the reference genome than any of the other state-of-the-art methods (Table 5.1), with the only exception being Falcon on the *C. Elegans* dataset with a marginally better result. However, on this dataset, Racon was 45x faster than Falcon.

Further, Racon is not limited only to contig consensus applications - it's general design and extreme efficiency allow it to be used as an error-correction module as well. This allows it's applications as a substitute step for PacBio's *pbdagcon* and *FalconSense* error-correction (Table 5.3)[101][103] used in FALCON and Canu, or Nanocorrect module used in Loman *et al.* [135] (which was recently deprecated). Comparison of error rate improvements achieved by various error correction tools was also recently evaluated here [144]. Nanocorrect had by far the best results on the older nanopore sequencing data. We show that Racon manages to achieve the quality of corrected reads similar to that of Nanocorrect and other methods with only one iteration of correction, while being two orders of magnitude faster than Nanocorrect and pertaining highest reference coverage of all methods (Table 5.3).

Racon is intended as a standalone consensus module and is not explicitly tied to Miniasm. It reads multiple input formats (GFA, FASTA, FASTQ, SAM, MHAP and PAF), allowing simple interoperability and modular design of new pipelines.

We evaluated Racon in terms of generated consensus quality (Tables 5.1 and 5.2), speed of a full assembly pipeline consisting of Minimap, Miniasm and Racon (Table 5.4), and

Table 5.1: Assembly and consensus results across 4 datasets of varying length and type.

		Miniasm+Racon 1 iteration	Miniasm+Racon 2 iterations	Canu	Falcon
Lambda	Genome size [bp]	48502	48502	48502	48502
ONT	Total bases [bp]	47903	47891	25077	7212
	Aligned bases ref. [bp]	48438 (99.87%)	48434 (99.86%)	25833 (53.26%)	7483 (15.43%)
	Aligned bases query [bp]	47903 (100.00%)	47891 (100.00%)	25077 (100.00%)	7212 (100.00%)
	Avg. Identity	97.56	97.93	96.87	95.77
	CPU time [sec]	11.77	23.97	171.720	137.590
E. Coli	Genome size [bp]	4641652	4641652	4641652	4641652
ONT	Total bases [bp]	4637170	4631920	4601503	4580230
	Aligned bases ref. [bp]	4640867 (99.98%)	4641323 (99.99%)	4631173 (99.77%)	4627613 (99.70%)
	Aligned bases query [bp]	4636686 (99.99%)	4631917 (100.00%)	4601365 (100.00%)	4580230 (100.00%)
	Avg. Identity	99.13	99.33	99.28	98.84
	CPU time [sec]	2187.74	4136.06	79690.490	49732.542
E. Coli	Genome size [bp]	4641652	4641652	4641652	4641652
PacBio	Total bases [bp]	4653302	4645369	4664416	4666788
	Aligned bases ref. [bp]	4641501 (100.00%)	4641439 (100.00%)	4641652 (100.00%)	4641652 (100.00%)
	Aligned bases query [bp]	4653086 (100.00%)	4645369 (100.00%)	4664416 (100.00%)	4666788 (100.00%)
	Avg. Identity	99.63	99.91	99.99	99.90
	CPU time [sec]	6942.82	13976.22	46392.100	174468.470
C. Elegans	Genome size [bp]	100272607	100272607	100272607	100272607
PacBio	Total bases [bp]	106351448	106388732	106687886	105858394
	Aligned bases ref. [bp]	100017642 (99.75%)	99989791 (99.72%)	100166301 (99.89%)	99295695 (99.03%)
	Aligned bases query [bp]	101712974 (95.64%)	101749015 (95.64%)	102928910 (96.48%)	102008289 (96.36%)
	Avg. Identity	99.43	99.73	99.89	99.74
	CPU time [sec]	94951.25	159680.87	2271168.83	7185955.55

Table 5.2: Polished assemblies for all methods (using Nanopolish).

		Raw Miniasm	Miniasm+Racon 2 iterations	Canu	Falcon	Loman et. al pipeline
E. Coli K-12	Genome size [bp]	4641652	4641652	4641652	4641652	4641652
ONT MAP006	Total bases [bp]	4696482	4641756	4631443	4624811	4695512
54x	Aligned bases ref. [bp]	4635941 (99.88%)	4641312 (99.99%)	4633324 (99.82%)	4627571 (99.70%)	4641325 (99.99%)
	Aligned bases query [bp]	4687686 (99.81%)	4641756 (100.00%)	4631361 (100.00%)	4624811 (100.00%)	4695463 (100.00%)
	Avg. Identity	98.06	99.80	99.80	99.78	99.80

Table 5.3: Comparison of error-correction modules on *E. Coli* K-12 MAP006 R7.3 54x dataset. Values presented in the table are median values of the error and match rate estimates. Time measurements for Falcon and Canu were not available, as their error correction modules are run as a wrapped part of the pipeline, whereas Nanocorrect is a stand-alone module.

	CPU time [h]	Coverage	Insertion rate (%)	Deletion rate (%)	Mismatch rate (%)	Match rate (%)	Error rate (I+D+M) (%)
Raw	-	53.55x	5.23	2.83	4.89	89.81	13.16
Racon	13	50.20x	0.58	0.60	0.15	99.26	1.31
Nanocorrect	8100	44.74x	0.14	0.43	0.03	99.83	0.62
Falcon	-	46.95x	0.04	1.11	0.06	99.90	1.23
Canu	-	35.53x	0.06	1.25	0.08	99.85	1.40

Table 5.4: Timings of various parts of the Miniasm+Racon assembly pipeline.

	Lambda ONT	*E. Coli* ONT	*E. Coli* PacBio	*C. Elegans* PacBio
Miniasm CPU time	2.11	175.83	700.96	32910.79
Minimap 1st iter. CPU time	0.11	16.66	42.64	913.35
Racon 1st iter. CPU time	9.55	1995.25	6199.22	61127.11
Minimap 2nd iter. CPU time	0.13	19.14	47.68	999.66
Racon 2nd iter. CPU time	12.07	1929.18	6985.72	63729.96
Total CPU time	23.97	4136.06	13976.22	159680.87

scalability of Racon with respect to the genome size (Table 5.5), on several real PacBio and nanopore datasets (see Methods). We compared against FALCON and Canu as being the current state-of-the-art in *de novo* assembly. Detailed comparisons of both to the Loman *et al.* pipeline are given elsewhere [144] and omitted here, because Falcon and Canu are both two orders of magnitude faster than the Loman *et al.* pipeline.

Table 5.5: Scalability of Racon accross genome sizes for same coverage is linear. Table shows results for one iteration of Racon.

	Lambda ONT 81x	*E. Coli* ONT 81x	*C. Elegans* PacBio 81x
Genome size	48 kbp	4.64 Mbp	100.2 Mbp
Racon CPU time [s]	27.19	1572.22	39537.75

5.3 Discussion

The goal of Racon is to generate genomic consensus which is of similar or better quality compared to the output generated by assembly methods which employ both error correction and consensus steps, while providing a speed-up of several times compared to those methods. Racon implements SIMD accelerated Partial Order Alignment graph consensus (SPOA) and can be run iteratively to achieve better performance.

We evaluated our consensus method on real PacBio and nanopore datasets, and show that the results are comparable to other state-of-the-art methods, while being an order of magnitude faster. We also demonstrate that the execution of our consensus approach scales linearly with genome size.

Here we stress out that the entire pipeline, consisting of Minimap + Miniasm + **Racon** could have an immense impact on the current state of *de novo* assembly, and allow higher quality analysis for higher throughput third generation sequencing data coming with the releases of PacBio Sequel and Oxford Nanopore's PromethION systems.

Racon and SPOA are available open source under the MIT license at: `https://github.com/isovic/racon.git` and `https://github.com/rvaser/spoa.git`.

Chapter 6

Integration and evaluation - Aracon assembler

Building on the results from previous chapters, here we proceed in development of a novel *de novo* genome assembly method. The main components of the method are GraphMap Owler, for raw read overlapping, and Racon, for the correction of raw contigs. Since the development of a layout phase was out-of-scope of this thesis, we borrow the layout component from Miniasm [89] to test the proof-of-concept that high-quality *de novo* genome assemblies are possible, even without error-correction in the pre-processing step of an assembly pipeline. We call the new method *Aracon* (Assembly with Rapid Consensus).

Aracon is thoroughly evaluated in the following sections using the methods presented in Chapter 3 in terms of assembly quality, memory usage and time consumption to obtain the assemblies.

6.1 Methods

Aracon implements the *Overlap-Layout-Consensus* paradigm by integrating the two new developed components, GraphMap Owler and Racon, together with Miniasm's layout module, into a fully functional assembly pipeline. The workflow of our design is depicted in Figure 6.1 and presented in Algorithm 21. All components developed in the scope of this thesis are designed in a modular fashion, offering the option of being run as individual stand-alone programs. Both GraphMap (Owler) and Racon can accept input and produce output in standard assembly formats (FASTA, FASTQ, SAM, MHAP, PAF, GFA), allowing easy inter-modular connection of components. Miniasm's layout module is designed with similar goals in mind [89].

Aracon is implemented as a simple script, conducting the pipeline between the modules. Aracon takes as input a FASTQ file of raw nanopore or PacBio reads. It then performs the overlapping process directly from the raw reads using the GraphMap Owler module (Algorithm

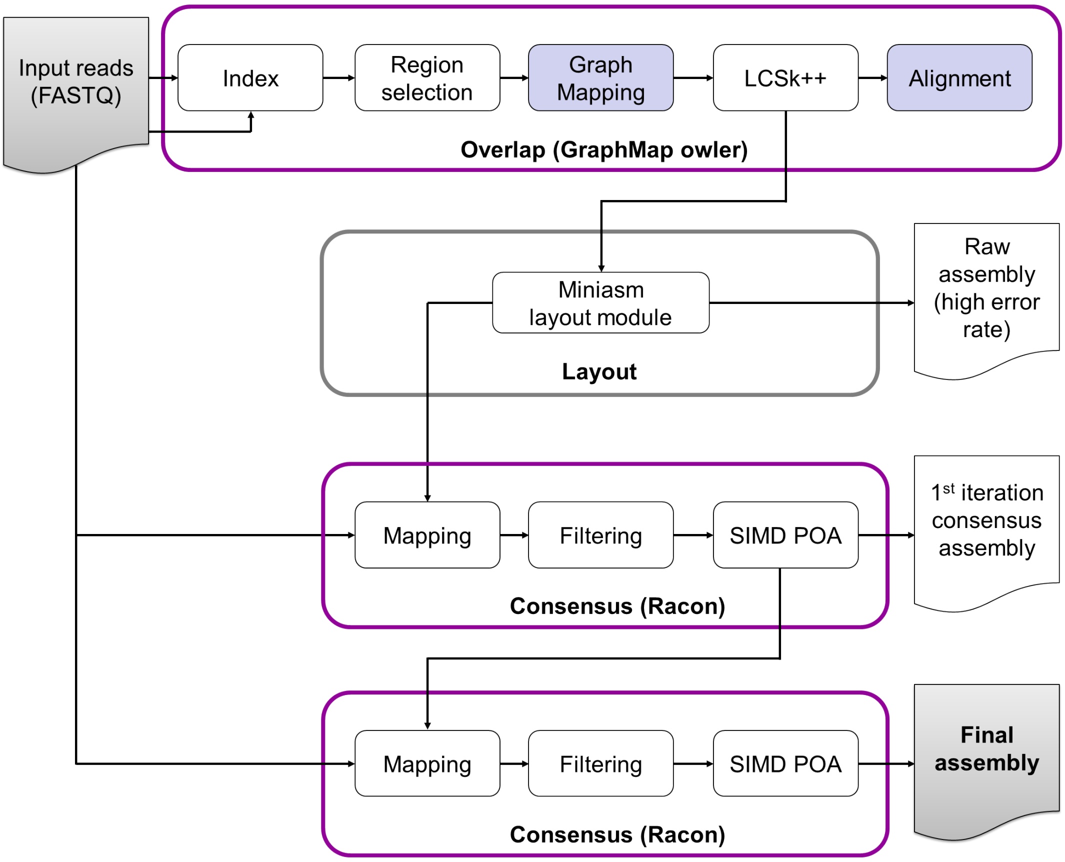


Figure 6.1: Depiction of the Aracon assembly workflow.

21, line 1). Overlaps are stored in a PAF overlap file, which can then be read by Miniasm layout (Algorithm 21, line 2). Miniasm produces and stores the raw assembly in a GFA formatted file on disk. The raw assembly is composed of tiled reads, with the error rate the same as in the input data. Racon then takes the GFA file and the original reads FASTQ file, and performs an iteration of consensus, to produce a corrected, much more accurate assembly and outputs it in FASTA format (Algorithm 21, lines 3 – 4). Racon is then run again, this time using the corrected assembly FASTA and the original reads in the FASTQ file, and performs the second

Algorithm 21: Aracon assembly pipeline

```
Input: A set of reads $\mathcal{R}$ in FASTQ format
Output: Assembly contigs $\mathcal{A}$ in FASTA/FASTQ/GFA format
Function ARACON($\mathcal{R}$) begin
1   $\mathcal{O} \leftarrow Owler(\mathcal{R}, \mathcal{R}, 0.45)$                                   ▷ Calculate overlaps
2   $\mathcal{A}_r \leftarrow Miniasm(\mathcal{R}, \mathcal{O})$                                  ▷ Calculate layout (raw assembly)
3   $\mathcal{M}_{i1} \leftarrow Minimap(\mathcal{A}_r, \mathcal{R})$                             ▷ Fast approximate mapping of reads to ref
4   $\mathcal{A}_{i1} \leftarrow Racon(\mathcal{A}_r, \mathcal{R}, \mathcal{M}_{i1}, 500, 10, "consensus", 0.45)$    ▷ First iteration of consensus
5   $\mathcal{M}_{i2} \leftarrow Minimap(\mathcal{A}_{i1}, \mathcal{R})$                          ▷ Fast approximate mapping of reads to ref
6   $\mathcal{A}_{i2} \leftarrow Racon(\mathcal{A}_{i1}, \mathcal{R}, \mathcal{M}_{i2}, 500, 10, "consensus", 0.45)$   ▷ Second iteration of consensus
7   $\mathcal{A} \leftarrow \mathcal{A}_{i2}$
8   return $\mathcal{A}$
```

iteration of consensus (Algorithm 21, lines 5 – 6). The resulting consensus from this phase is then deemed to be the *final assembly* (Algorithm 21, lines 7 – 8).

6.1.1 Implementation and reproducibility

Aracon assembly pipeline is implemented in Python. It integrates several components (GraphMap, Miniasm and Racon) which were implemented in C/C++ and added as submodules to Aracon. A wrapper for Aracon was implemented into our benchmarking framework NanoMark, used to conduct assembly tests and evaluation. All tests were ran using Ubuntu based systems with two 6-core Intel(R) Xeon(R) E5645 CPUs @ 2.40GHz with Hyperthreading, using 12 threads where possible. Version of methods used in comparison:

- Aracon - `https://github.com/isovic/aracon`, commit: $0dfda4eeaa17$
- NanoMark - `https://github.com/kkrizanovic/NanoMark`, commit: $77265aca8cbc$

6.1.2 Datasets

For evaluation purposes, we used Datasets 1 - 5 defined in Section 3.1.1. For consistency, we present them in Table 6.1 as well.

Table 6.1: Description of the datasets used for evaluation of Aracon.

Name	Description
Dataset 1	Complete *E. coli* R7.3 dataset, contains both 1d and 2d reads (both pass and fail), total coverage 67*x* (70531 reads), of which 2d reads comprise 14*x* (11823 reads) [123].
Dataset 2	Reads from Loman *et al.* [16] subsampled to coverage 19*x*, pass 2d reads only (in total 16945 reads).
Dataset 3	Complete dataset used by Loman *et al.* [16] nanopore assembly paper, contains pass 2d reads only, coverage 29*x*, 22270 reads.
Dataset 4	Reads from MARC WTCHG dataset, 2d reads extracted from pass (33*x*) and fail (7*x*) folders, total coverage 40*x*, total number of 2d reads: 29635 [13].
Dataset 5	2d reads extracted from the first run of the MAP006 dataset (MAP006-1), from pass folder only, coverage 54*x*, 25483 reads in total. `http://lab.loman.net/2015/09/24/first-sqk-map-006-experiment/`

6.1.3 Assembly pipelines

The list of assembly pipelines we compared against in Chapter 3 was reduced to five state-of-the-art methods: Loman *et al.* pipeline (in continuation, LQS), PBcR, FALCON, Canu and

Miniasm. Since the newly developed Aracon assembler is non-hybrid, and is therefore not comparable to the hybrid ones, SPAdes and ALLPATHS-LG were left out of comparison.

LQS pipeline: Pipeline developed and published by Loman *et al.* in their pivotal nanopore assembly paper (`https://github.com/jts/nanopore-paper-analysis`) [16]. The pipeline consists of Nanocorrect, WGS and Nanopolish. The version of the pipeline tested in this study uses Nanocorrect commit 47dcd7f147c and WGS version 8.2. For the base version of the pipeline, we didn't use Nanopolish.

PBcR: Implemented as a part of the WGS package (`http://wgs-assembler.sourceforge.net/wiki/index.php/PBcR`) [8]. In this study version 8.3rc2 of WGS was used. Spec file defining assembly parameters for nanopore data, was downloaded from the PBcR web page.

FALCON: To evaluate Falcon we used the FALCON-integrate project (`https://github.com/PacificBiosciences/FALCON-integrate`) (commit: 3e7dd7db190) [90]. Since no formal parameter specification for nanopore data currently exists, we derived a suitable set of parameters through trial and error (Table 3.2).

Canu: Canu was obtained from `https://github.com/marbl/canu` (commit: 70e711a382f). Canu is currently not yet published.

Miniasm: Miniasm was obtained from `https://github.com/lh3/miniasm` (commit: 17d5bd12290). For calculating read overlaps we used Minimap (`https://github.com/lh3/minimap`) (commit: 1cd6ae3bc7c) [89].

6.1.4 Evaluation methods

All assembly results were compared to the *E. coli* K-12 MG1655 NCBI reference, *NC*_000913.3. Assembly quality was evaluated using Quast 3.1 [125] and Dnadiff [126] tools. CPU and memory consumption was evaluated using a fork of the Cgmemtime tool (`https://github.com/isovic/cgmemtime.git`). The LQS pipeline was run without applying the polishing phase to make it comparable to other methods.

6.2 Results

Tables 6.2 and 6.3 give detailed information about *de novo* genome assemblies of Datasets 1 -5 obtained using Aracon, and comparison of Aracon to the state-of-the-art methods.

Based on the results for Datasets 2 - 5, and reflecting on the Genome fraction, Avg. Identity, Total SNPs and Total Indels, one can conclude that Aracon in all cases produces assemblies which very closely resemble those of LQS (the most accurate method), while always producing larger Avg. Identity than PBcR, Falcon, Canu or Miniasm (Table 6.2). Avg. Identity between

Table 6.2: Detailed comparison of the newly developed Aracon assembler and the state-of-the-art methods. Genome fraction and Avg. Identity 1-to-1 are marked in bold for Aracon as well as for the best other assembler.

Dataset	Assembler	# ctg.	Largest contig	Total length	N50	Genome fraction (%)	Avg. Identity 1-to-1	Total SNPs	Total Indels
1	**Aracon**	100	81199	2695009	27612	**97.377**	**93.11**	47069	143677
	LQS	236	201101	5085719	41967	19.278	**93.98**	59887	186721
	Falcon	1	1703	1703	1703	-	92.81	13	117
	PBcR	78	41968	1224958	18161	0.004	86.8	30719	139617
	Canu	61	24645	585225	10922	**99.91**	90.81	5736	46744
	Miniasm	87	64239	1895814	22361	0.004	82.77	106720	155383
2	**Aracon**	14	880544	4466788	352504	**97.377**	**97.5**	15417	97968
	LQS	8	1673524	4660747	1159703	**99.895**	**98.08**	8858	79746
	Falcon	98	46831	1038959	11083	6.994	94.58	3263	47211
	PBcR	22	983314	4311837	246681	0.593	93.7	14823	269053
	Canu	26	966076	4407396	332535	90.123	95.71	5691	183774
	Miniasm	15	1100378	4505577	353994	0.002	84.21	248575	373190
3	**Aracon**	1	4589826	4589826	4589826	**99.889**	**97.94**	10932	85133
	LQS	3	4603990	4622649	4603990	**99.998**	**98.49**	4568	65283
	Falcon	124	53984	1495565	13838	17.316	94.97	3206	59638
	PBcR	1	4329903	4329903	4329903	12.825	94.03	7209	262357
	Canu	10	4465231	4465231	4465231	88.655	95.77	5213	185027
	Miniasm	3	3362269	4615469	3362269	0.002	84.04	247849	367927
4	**Aracon**	2	4480397	4621464	4480397	**99.922**	**98.91**	3455	46186
	LQS	8	4622531	4659590	4622531	**99.938**	**99.08**	2256	40118
	Falcon	13	4538244	4549221	4538244	**99.938**	97.66	3710	104165
	PBcR	3	3615068	4549034	3615068	99.553	97.39	2394	117397
	Canu	2	4576679	4580258	4576679	99.915	98.42	812	71878
	Miniasm	1	4577227	4577227	4577227	0.002	88.31	185194	326066
5	**Aracon**	1	4574857	4574857	4574857	**98.754**	**99.32**	1625	29613
	LQS	5	4006324	4681348	4006324	**99.991**	**99.43**	1435	25106
	Falcon	1	4580230	4580230	4580230	99.655	98.84	2589	50662
	PBcR	1	4596475	4596475	4596475	99.914	98.99	1136	45542
	Canu	1	4600945	4600945	4600945	99.746	99.29	415	32100
	Miniasm	1	4774395	4774395	4774395	0.002	88.69	175279	328688

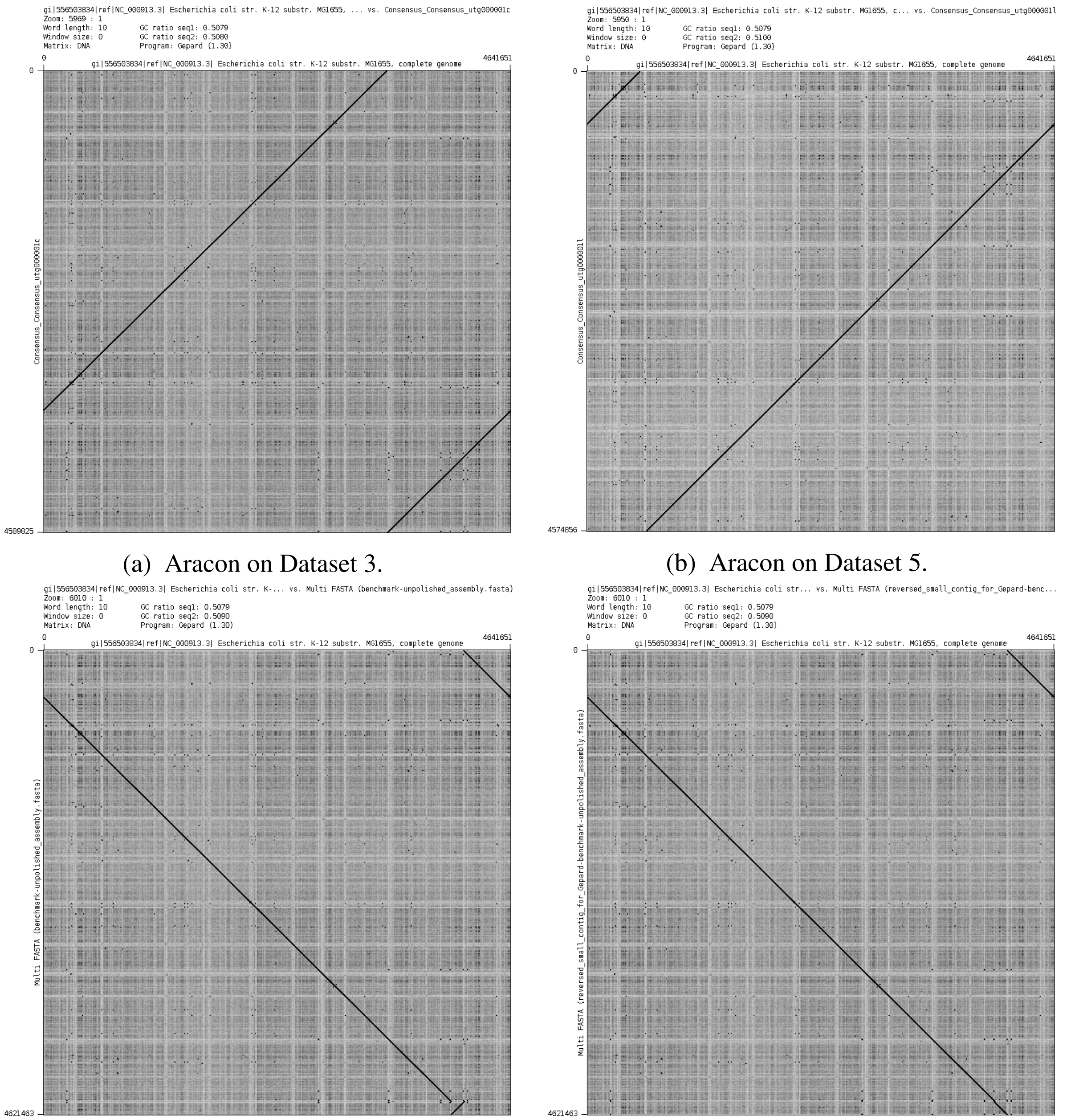
Table 6.3: CPU time (hours) / Maximum memory usage (GB). Bold values present the results for Aracon and the best other method, except for Miniasm. Although most efficient, Miniasm was excluded because it is not directly comparable to other methods since it does not employ neither an error-correction nor a consensus step. Dataset 1 was also not marked in bold, as none of the assemblers managed to assemble the dataset with quality, and the results may not be comparable.

	Dataset 1	Dataset 2	Dataset 3	Dataset 4	Dataset 5
Coverage	67x	20x	30	40	50
Assembler					
Aracon	1.94 / 16	**0.66 / 5**	**1.27 / 7**	**3.49 / 10**	**3.97 / 13**
LQS	450.13 / 4	1086 / 4	2539 / **4**	4438 / **4**	8142 / **4**
Falcon	1.34 / 4	**3.1** / 6	**6.4** / 10	19.7 / 10	13.8 / 13
PBcR	1.88 / 1	6.2 / **2**	13.7 / 6	1**4.1** / 5	19.3 / 5
Canu	1.93 / 3	5.33 / 3	11.2 / **4**	28.7 / **4**	**9.39** / **4**
Miniasm	0.029 / 3	0.009 / 2	0.015 / 2	0.026 / 3	0.044 / **4**

Aracon and LQS differs in as little as 0.11% on Dataset 5, to at most 0.58% on Dataset 2. Taking this into account, for such minor difference in the obtained results, Aracon achieves more than ***three orders of magnitude*** quicker assemblies (Table 6.3). Compared to other methods (not including Miniasm as it does not perform an error-correction or a consensus step), Aracon is at least *three times as fast* as the next best method. Memory consumption is always $< 16GB$, enabling high quality bacterial genome assemblies on regular workstations or laptops.

Similar to other assembly pipelines, the poor quality of the data in Dataset 1 and the low coverage of Dataset 2 were not overcome by Aracon either. Again, however, Aracon's performance on these datasets was comparable to that of LQS and better than other methods.

Aracon managed to achieve a **single-contig bacterial genome assembly** on Datasets 3 and 5, while on Dataset 4 the assembly consisted of only two contigs. We further inspected assemblies of Datasets 3, 4 and 5 visually using Gepard [145], to verify whether the assemblies included potential structural errors (Figure 6.2). Figures 6.2a (Dataset 3) and 6.2b (Dataset 5) show a perfect circular chromosome assembly of the *E. Coli* genome, while Figure 6.2c shows a break in between the two contigs. One can notice the inversion of the smaller contig - this is due to the arbitrary choice of a strand used to traverse the layout of the assembly. We reverse-complemented the smaller contig to inspect whether it will fit nicely to the larger one. Figure 6.2d shows the dot plot of such orientation of contigs - although split in two, the assembly aligns perfectly circular to the *E. Coli* reference genome.



(a) Aracon on Dataset 3.

(b) Aracon on Dataset 5.

(c) Aracon on Dataset 4.

(d) Aracon on Dataset 4 with reversed small contig.

Figure 6.2: Inspecting the quality of Aracon assemblies visually using the dotplot representation. Datasets 3 and 5 were assembled into single contigs which fully match the *E. Coli* reference genome, as shown in *a* and *b*. Dataset 4 was assembled into two contigs: a large one covering almost the entire genome, and a much shorter one. Together they provide a full assembly of the *E. Coli*. Figure *c* shows the assembly as-is produced by Aracon. Since strands for each contig are arbitrary, the second (smaller) contig was reversed and the dotplot re-generated (*d*). All figures show that there are no misassemblies of the genome or structural variations.

6.3 Discussion

This chapter presented the integration of all components developed within the scope of this thesis, into a novel *de novo* genome assembly method called *Aracon*. Aracon is able to produce high-quality contiguous genome assemblies from raw third generation sequencing data without an error-correction preprocessing step. It is also at least *three times faster* than the fastest alternative method which utilizes error-correction and consensus steps, while providing better results.

With these results, we managed to construct a proof-of-concept that high-quality *de novo* genome assemblies are possible without having an error-correction pre-processing step.

Aracon is available open-source under the MIT license at: `https://github.com/isovic/aracon`.

Chapter 7

Conclusion

Since the time I enrolled on a Ph.D. programme back in 2011, the times have changed drastically for sequencing technologies and related algorithmic methods. The survey of the field I conducted back then is now very much out-of-date, and many methods I have researched became obsolete. At the time, Pacific Biosciences had just released their first commercial third generation sequencing device with the promise of reads much longer than the NGS technologies could have provided. They truly did deliver - current PacBio machines can produce reads of up to $\approx 60kbp$ in length with error rate of $\approx 11\%$, which is a significant improvement compared to some of their first public data ($\approx 1000bp$ in length and $\approx 18\%$ error rate). The long reads were especially tempting for genome assembly as well as many other interesting applications, however, new and more sensitive algorithms had to be developed to cope with the characteristics of the data.

A more recent challenge, which once again stirred things up in the realm of sequencing, happened in 2014, when Oxford Nanopore Technologies released their nanopore-based MinION third generation sequencing device - at first through an early-access programme, and later in 2015 commercially. Unlike other technologies, MinION is very small and portable, and the device itself costs two orders of magnitude less than any previous technology. MinIONs should provide virtually unlimited read lengths, bound only by the shear laws of physics which cause the breaking of the long DNA molecules during sample handling. Indeed, the longest basecalled reads publicly released were $> 300kbp$ long. Reads of this length could help to reconstruct truly difficult and repetitive regions of complex eukaryotic organisms. However, the challenge lays again in the error rates, which were at first significantly higher than PacBio's ($\approx 35\%$), which required the development of even more sensitive methods for sequencing data analysis. The error rates in ONT data have since come down to similar levels as those of PacBio ($\approx 11\%$).

As mentioned above, one of the greatest applications of long reads, be it error-prone or not, is in genome assembly. The approaches used to assemble such data so far were all based on the idea of error-correcting the reads in a pre-processing step, and then using the rest of the

Overlap-Layout-Consensus (or the de Bruijn) paradigm to do the actual assembly.

It was the goal of this thesis to test the hypothesis whether it is possible to achieve high quality *de novo* genome assembly from long error-prone reads *without* error-correcting them first - instead, leaving the correction part only for the final, consensus phase. Error correction of reads is generally a slow process, because either: (I) the full set of input reads needs to be corrected by aligning reads mutually in a pairwise manner and then applying the multiple sequence alignment to resolve the ambiguities, which would scale poorly with the increase of the dataset size; or (II) only a subset of input reads will be error-corrected, e.g. only the ones above certain length (seed reads in Falcon), which is very dependant on the distribution of read lengths in the input dataset. Instead, developing a more sensitive overlapping method could produce enough information to construct a layout of the genome, and then apply a fast consensus method to reduce the errors in the assembly. This describes the general approach of the research conducted in this thesis.

We have successfully managed to prove this hypothesis by developing novel algorithms and implementing them into open-source tools. The importance of our research was also recognized by high-impact journals, where we, so far, published the framework for automated evaluation of *de novo* assembly methods and GraphMap, a method for fast and sensitive mapping of nanopore sequencing reads.

Below are pre-defined contributions of the research conducted in this thesis, and the alignment of performed work in relation to those contributions. All contributions and requirements have been met fully, while the conducted work also generated even more of the added value and impact on the state-of-the-art. As an example, even before it was officially published (while in the pre-print stage), GraphMap influenced the development of Minimap and Miniasm, which we compare to here as the state-of-the-art. Minimap and Miniasm coincidentally and independently developed *de novo* assembly without error-correction. This shows that research in modern world, and especially genomics, is no longer linear, but instead, is becoming more of a directed *cyclic* graph. Perhaps in the near future, the formats of dissertations and research papers will be revised to accommodate such research practices.

7.1 Contributions of the dissertation

A framework for systematic benchmarking of *de novo* genome assembly tools

A systematic evaluation of the state-of-the-art in *de novo* genome assembly from third generation sequencing data was conducted in the scope of Chapter 3. The focus of the research was put on assessing the existing methods for assembly of nanopore data, while also attempting to utilize a non-nanopore assembly method for the same purpose (namely Falcon, designed for

PacBio data). This step provided key information about the existing methods, and potential avenues to take when developing novel algorithms. In the scope of research, a framework called *NanoMark* was developed, which simplifies the benchmarking of different tools by automatically setting up the data, installing the assembly tools and wrapping the execution of various assemblers. NanoMark measures memory and time statistics, and performs quality assessments on the results of the assembly. Wrapper scripts can easily be added to the framework, as was later done to asses the performance of the Aracon assembler developed in the scope of this thesis (Chapter 6). The results were published in:

- Sović, Ivan; Križanović, Krešimir; Skala, Karolj; Šikić, Mile. Evaluation of hybrid and non-hybrid methods for de novo assembly of nanopore reads. Bioinformatics. 11 (2016)

Optimized algorithms and data structures for *de novo* genome assembly with emphasis on third generation sequencing data

Chapters 4, 5 and 6 describe the development of the novel assembly method called *Aracon*.

Chapter 4 starts with the development of a novel, highly sensitive mapper for long, error-prone reads called *GraphMap*, which is then subsequently modified and used for sequence overlapping. GraphMap introduces several novel concepts in the context of sequence mapping: use of gapped spaced seeds, Hough Transform, graph mapping and L_1 linear regression all offer a new combination of algorithmic improvements which enabled very sensitive mapping and overlapping of error-prone reads. The final developed overlapper is called *Owler* (Overlap With Long Erroneous Reads), and is used in the Aracon assembler developed in the scope of this thesis. GraphMap was published in:

- Sović, Ivan; Šikić, Mile; Wilm, Andreas; Fenlon, Shannon Nicole; Chen, Swaine; Nagarajan, Niranjan. Fast and sensitive mapping of nanopore sequencing reads with GraphMap. Nature Communications. 7 (2016)

Chapter 5 describes the development of a very fast standalone consensus module called *Racon* (Rapid Consensus), intended for correcting errors in raw assemblies, such as the ones produced by the Miniasm's layout step. We developed an optimized method which can utilize Single Instruction Multiple Data (SIMD) instructions of a CPU to accelerate the building of a Partial Order Alignment (POA) graph. POA is used to construct the final consensus sequence. Racon is dependant on the base quality values provided by sequencing devices to construct a more confident consensus sequence. Thoroughly testing Racon showed that it consistently achieves results which are comparable or better than state-of-the-art methods, while being up to an order of magnitude faster.

Chapter 6 describes the development of the new *Aracon* assembler (Assembly with Rapid Consensus). Aracon is implemented as a script which combines the previously described components into a single assembler: Owler is used for read overlapping, Miniasm's layout step to construct the raw contigs, and Racon is used twice to provide error-corrected output. Miniasm's layout module is used for the layout step, as the development of a layout module was outside of the scope of this thesis. String graph and assembly graph theory is described elsewhere, and efficient implementations of the algorithm already exist.

Evaluation of developed algorithms using the novel benchmarking framework

Chapter 6 provides a thorough evaluation of the novel Aracon assembly method, and its comparison to the state-of-the-art. For evaluation purposes, NanoMark (developed in the scope of Chapter 3) was used. Aracon proved to be as good as currently the best assembly method (Loman *et al.* pipeline), while being *three orders of magnitude faster*. Compared to other methods, Aracon consistently demonstrated similar high-quality results, where others' results would vary depending on the dataset. Compared to those methods, Aracon was about 3*x* faster.

7.2 Future research avenues

The high contiguity of nanopore-only assemblies provides a number of other important opportunities. For example, even small bioinformatics laboratories can now study genomic structural variations and rearrangements, or identify large antibiotic resistance islands in genomes, for which exact base variations are not of such high importance; all in-house.

What might be more interesting from an algorithmic perspective is the development of very fast and accurate algorithms for assembly of very large genomes, such as plants. Plant genomes can be even 10*x* larger than the human genome, resulting in enormous amounts of data which need to be stored and processed. Setting aside storage issues for now, aligning or assembling of such large amounts of data would require the development of extremely efficient indexing structures. The problem of computer memory organization arises in this aspect, as lookups will almost always wind up outside the cached regions of the memory, causing $\approx 100x$ reductions in speed only because of the shear size of the input data. Low-level optimizations will have to be explored to enable such analyses.

Another important and very related avenue is the development of methods which could handle *ploidy* in the input sample (number of sets of chromosomes). For example, the reference human genome ($\approx 3Gbp$) is only a haploid representation of the full diploid human genome. Plant genomes can have an even larger ploidy number, such as *Coffea arabica* which is *tetraploid*, or *Triticum aestivum* (bread wheat) which is hexaploid. Currently only one diploid-aware assembler exists (Falcon), while the polyploidy still needs to be explored from the assembly perspective.

We can expect that with further development of nanopore technology (and other long read sequencing technologies) read quality will increase and the technology will become more accessible and more affordable. This will make *de novo* assembly using nanopore reads faster, more precise and applicable to larger genomes.

Appendix A

Additional benchmarking results

Table A.1: Error rate analysis of raw read datasets. Numbers in the table represent median values.

Dataset	Insertion rate	Deletion rate	Mismatch rate	Error rate	Match rate
Dataset 0	0%	0%	3%	3%	97%
Dataset 1	4%	8%	21%	33%	74%
Dataset 2	3%	4%	9%	16%	85%
Dataset 3	3%	4%	9%	16%	85%
Dataset 4	2%	3%	6%	11%	91%
Dataset 5	3%	2%	5%	10%	90%

Table A.2: Detailed Quast results for all assembly pipelines and all datasets.

Dataset	Assembler	# ctg.	Largest contig	Total length	N50	# mismatches per 100kbp	# indels per 100kbp	Genome fraction (%)
Dataset 1	LQS	236	201101	5085719	41967	895.83	2533.71	19.278
Dataset 1	Falcon	1	1703	1703	1703	-	-	-
Dataset 1	PBcR	78	41968	1224958	18161	613.5	3067.48	0.004
Dataset 1	Allpaths	4	4639965	4648288	4639965	2.14	4.81	99.887
Dataset 1	SPAdes	19	4465980	4640184	4465980	11.08	2.61	99.91
Dataset 1	Canu	61	24645	585225	10922	526.32	526.32	0.004
Dataset 1	Miniasm	87	64239	1895814	22361	-	-	-
Dataset 2	LQS	8	1673524	4660747	1159703	134.58	1110.98	99.895
Dataset 2	Falcon	98	46831	1038959	11083	271.38	2647.58	6.994
Dataset 2	PBcR	22	983314	4311837	246681	170.67	2777.88	0.593
Dataset 2	Allpaths	2	4639001	4641330	4639001	0.24	0.19	99.938
Dataset 2	SPAdes	18	4488904	4662975	4488904	10.78	1.77	99.912
Dataset 2	Canu	25	966076	4407396	332535	112.12	2472.68	90.123
Dataset 2	Miniasm	15	1100378	4505577	353994	-	1265.82	0.002
Dataset 3	LQS	3	4603990	4622649	4603990	74.31	878.72	99.998
Dataset 3	Falcon	124	53984	1495565	13838	212.01	2615.14	17.316
Dataset 3	PBcR	1	4329903	4329903	4329903	85	2861.25	12.825
Dataset 3	Allpaths	3	4638937	4643557	4638937	0.17	0.8	99.938
Dataset 3	SPAdes	19	4474608	4647328	4474608	10.74	1.55	99.88
Dataset 3	Canu	10	4465231	4465231	4465231	68.7	2171.55	88.655
Dataset 3	Miniasm	3	3362269	4615469	3362269	1149.43	1149.43	0.002
Dataset 4	LQS	8	4622531	4659590	4622531	47.64	642.47	99.938
Dataset 4	Falcon	13	4538244	4549221	4538244	44.19	1188.82	99.938
Dataset 4	PBcR	3	3615068	4549034	3615068	48.15	1576.45	99.553
Datase t4	Allpaths	1	4638952	4638952	4638952	0.19	0.26	99.938
Dataset 4	SPAdes	20	4475770	4648587	4475770	10.09	1.21	99.88
Dataset 4	Canu	2	4576679	4580258	4576679	12.76	966.67	99.915
Dataset 4	Miniasm	1	4577227	4577227	4577227	-	-	0.002
Dataset 5	LQS	5	4006324	4681348	4006324	29.07	390.31	99.991
Dataset 5	Falcon	1	4580230	4580230	4580230	48.3	703.58	99.655
Dataset 5	PBcR	1	4596475	4596475	4596475	24.11	628.85	99.914
Dataset 5	Allpaths	1	4638958	4638958	4638958	0.19	0.22	99.938
Dataset 5	SPAdes	16	4648863	4651405	4648863	10.2	0.93	99.918
Dataset 5	Canu	1	4600945	4600945	4600945	8.38	487.18	99.746
Dataset 5	Miniasm	1	4774395	4774395	4774395	1219.51	1219.51	0.002

Table A.3: Determining the lowest coverage for which each assembly pipeline produces a sufficiently good assembly. At 29*x* coverage only LQS produces a good assembly, while at 40*x* coverage all non-hybrid assemblers produce a good assembly. To investigate this further, we prepared several datasets with coverages between 30*x* and 40*x*. These datasets were subsampled from Dataset 5 in a way that a larger dataset is a superset of a smaller one. Falcon, PBcR, Canu and Miniasm were then run on these Datasets, while LQS was left out because it already produced a good assembly at 29*x* coverage.

Dataset	**Coverage**	**Assembler**	**# ctg.**	**Largest contig**	**Total length**	**N50**	**Genome fraction (%)**	**Avg. Identity**
Dataset 3	29x	LQS	3	4603990	4622649	4603990	99.998	98.49
Dataset 3	29x	Falcon	124	53984	1495565	13838	17.316	94.97
Dataset 3	29x	PBcR	1	4329903	4329903	4329903	12.825	94.03
Dataset 3	29x	Canu	10	4465231	4576345	4465231	88.655	95.77
Dataset 3	29x	Miniasm	3	3362269	4615469	3362269	0.002	84.04
Dataset 5	32x	Falcon	2	3622965	4565105	3622965	99.432	98.67
Dataset 5	32x	PBcR	3	3065306	4603459	3065306	99.738	98.87
Dataset 5	32x	Canu	1	4606239	4606239	4606239	99.927	99.22
Dataset 5	32x	Miniasm	1	4764930	4764930	4764930	0.015	88.52
Dataset 5	35x	Falcon	2	4581657	4601328	4581657	99.777	98.7
Dataset 5	35x	PBcR	1	4589654	4589654	4589654	99.861	98.85
Dataset 5	35x	Canu	1	4606461	4606461	4606461	99.895	99.23
Dataset 5	35x	Miniasm	1	4766818	4766818	4766818	0.018	88.55
Dataset 5	37x	Falcon	2	3173910	4599063	3173910	99.736	98.73
Dataset 5	37x	PBcR	1	4589878	4589878	4589878	99.795	98.85
Dataset 5	37x	Canu	1	4606506	4606506	4606506	99.887	99.23
Dataset 5	37x	Miniasm	1	4766897	4766897	4766897	0.022	88.55
Dataset 5	40x	Falcon	1	4581577	4581577	4581577	99.736	98.76
Dataset 5	40x	PBcR	1	4588205	4588205	4588205	99.807	98.89
Dataset 5	40x	Canu	1	4593671	4593671	4593671	99.624	99.24
Dataset 5	40x	Miniasm	1	4773983	4773983	4773983	0.027	88.53

Table A.4: Detailed Dnadiff results for all “Big Contigs”. For each assembly which contained a contig of at least 4Mbp in length, that contig was extracted and compared to the reference using Dnadiff. The table does not contain entries for assemblers which did not produce contigs of 4Mbp for a certain dataset.

Dataset	Assembler	Total bases, query	Aligned bases, ref.	Aligned bases, query	Unaligned bases, ref.	Unaligned bases, query	Avg. Ident. 1-to-1	Bkp. ref.	Bkp. query	Total SNPs query	Total Indels query
Dataset 0	Allpaths	4626283	99.94%	99.51%	0.06%	0.49%	99.99	140	139	59	73
Dataset 1	Allpaths	4639965	100.00%	99.92%	0.00%	0.08%	99.98	42	42	65	261
Dataset 1	SPAdes	4465980	96.36%	100.00%	3.64%	0.00%	99.98	48	48	451	173
Dataset 2	Allpaths	4639001	100.00%	100.00%	0.00%	0.00%	99.99	20	20	10	12
Dataset 2	SPAdes	4488904	96.36%	99.99%	3.64%	0.01%	99.98	48	48	424	110
Dataset 3	LQS	4603990	100.00%	100.00%	0.00%	0.00%	98.49	11	10	4568	65283
Dataset 3	PBcR	4329903	98.20%	99.40%	1.80%	0.60%	94.03	841	841	7209	262357
Dataset 3	Allpaths	4638937	100.00%	100.00%	0.00%	0.00%	99.99	20	20	6	38
Dataset 3	SPAdes	4474608	96.36%	100.00%	3.64%	0.00%	99.99	46	46	424	108
Dataset 3	Canu	4465231	99.54%	99.92%	0.46%	0.08%	95.77	471	471	5318	186843
Dataset 4	LQS	4622531	100.00%	100.00%	0.00%	0.00%	99.08	23	22	2256	40118
Dataset 4	Falcon	4538244	100.00%	100.00%	0.00%	0.00%	97.66	27	27	3710	104165
Dataset 4	Allpaths	4638952	100.00%	100.00%	0.00%	0.00%	99.99	22	22	8	20
Dataset 4	SPAdes	4475770	96.36%	100.00%	3.64%	0.00%	99.99	46	46	398	73
Dataset 4	Canu	4576679	99.98%	100.00%	0.02%	0.00%	98.42	33	32	812	71878
Dataset 4	Miniasm	4577227	91.51%	91.47%	8.49%	8.53%	88.31	1364	1362	185194	326066
Dataset 5	LQS	4006324	86.93%	100.00%	13.07%	0.00%	99.43	88	88	1237	21852
Dataset 5	Falcon	4580230	99.70%	100.00%	0.30%	0.00%	98.84	22	22	2589	50662
Dataset 5	PBcR	4596475	99.93%	99.97%	0.07%	0.03%	98.99	47	47	1136	45542
Dataset 5	Allpaths	4638958	100.00%	100.00%	0.00%	0.00%	99.99	20	20	3	5
Dataset 5	SPAdes	4651405	100.00%	100.00%	0.00%	0.00%	99.99	36	36	420	53
Dataset 5	Canu	4600945	99.79%	100.00%	0.21%	0.00%	99.29	49	48	415	32100
Dataset 5	Miniasm	4774395	91.74%	90.82%	8.26%	9.18%	88.69	1316	1317	175279	328688

Table A.5: Falcon assembly using Nanocorrect processed reads as the input datasets.

Dataset	#ctg	Largest contig	Total length (>= 0 bp)	N50	Genome fraction (%)	Avg. identity	Total SNPs	Total Indels
Dataset 1	159	83912	2847962	27859	28.645	94.83	31162	118900
Dataset 2	32	477908	4270077	182215	92.741	98.1	8428	72872
Dataset 3	1	4585235	4585235	4585235	99.798	98.43	6201	66064
Dataset 4	3	4593235	4604056	4593235	99.631	99.04	3310	41189
Dataset 5	2	4610069	4619739	4610069	99.56	99.37	3222	26019

Table A.6: Miniasm assembly using Nanocorrect processed reads as the input datasets.

Dataset	#ctg	Largest contig	Total length (>= 0 bp)	N50	Genome fraction (%)	Avg. identity	Total SNPs	Total Indels
Dataset 1	97	129643	3532190	41369	14.445	93.94	50938	159504
Dataset 2	23	1102663	4585084	252947	96.285	97.94	10808	81751
Dataset 3	1	4613692	4613692	4613692	96.637	98.27	7539	70819
Dataset 4	1	4604261	4604261	4604261	98.98	99.04	2986	40965
Dataset 5	1	4650865	4650865	4650865	98.424	99.14	4229	35342

Appendix B

Supplementary information for GraphMap

B.1 Evaluating GraphMap on synthetic datasets

On synthetic datasets emulating error profiles from Illumina and PacBio sequencing, we noted that GraphMap and BLAST have high precision and recall ($\approx 98\%$) for both location and alignment measures and are almost indistinguishable in these metrics (Figure B.1a). The slight variations in performance that were observed were not defined by the size of the genomes that were studied. In addition, despite the marked differences in error profiles for Illumina and PacBio, the observed performance metrics were comparable, highlighting the robustness of GraphMap and its similarity to the gold-standard BLAST. Other mappers (BWA-MEM, LAST, DALIGNER and BLASR) exhibit similarly consistent results on Illumina data and PacBio data, with the exception of BLASR being slightly worse on PacBio data (by up to 10% for the human genome). BLASR's results could be a result of it being tuned to specific features of PacBio data that are not adequately captured in our simulation.

B.2 GraphMap's sensitivity on ONT datasets

GraphMap and other mappers (BWA-MEM, LAST, DALIGNER and BLASR) were evaluated on a range of publicly available ONT datasets for their performance (runtime, memory usage) and sensitivity for read mapping. Across all datasets, GraphMap was able to map the most reads and bases, typically mapping more than 95% of the bases and 85% of the reads in a dataset (Figures 4.7b, B.2, Tables B.5 and B.6). This was despite the exclusion of secondary alignments in GraphMap results and their presence in results for LAST, BWA-MEM and DALIGNER (also used for genome coverage calculations). Overall, LAST was the next best mapper, typically mapping more than 60% of bases (accounting for all secondary alignments; Tables B.5 and

Table B.1: Precision/Recall in mapping of synthetic reads to the correct genomic location ($\pm 50bp$).

Datatype	Genome	BLASR	BLAST	BWA-MEM	GraphMap	LAST	DALIGNER-default	DALIGNER-k10	DALIGNER-k9	marginAlign	marginAlign w/ GraphMap
Illumina	N. meningitidis	98.0 / 98.0	97.6 / 97.6	98.2 / 98.2	97.7 / 97.6	97.4 / 97.4	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	94.8 / 94.8	97.7 / 97.6
Illumina	E. coli	98.7 / 98.7	99.0 / 99.0	98.8 / 98.8	98.5 / 98.2	98.9 / 98.9	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	98.6 / 98.6	98.5 / 98.2
Illumina	S. cerevisiae	96.5 / 96.5	96.5 / 96.3	96.0 / 96.0	96.0 / 95.8	96.3 / 96.1	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	96.0 / 95.8	96.0 / 95.8
Illumina	C. elegans	98.0 / 98.0	98.1 / 97.6	98.0 / 98.0	97.2 / 96.4	98.3 / 98.2	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	92.5 / 92.4	97.2 / 96.4
Illumina	H. sapiens (chr3)	99.0 / 99.0	99.0 / 99.0	99.0 / 99.0	98.4 / 98.1	98.9 / 98.9	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	84.9 / 84.9	98.4 / 98.1
Illumina	H. sapiens	98.1 / 98.1	-	98.3 / 98.3	97.7 / 97.4	98.7 / 98.5	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	92.3 / 92.1	97.7 / 97.4
PacBio	N. meningitidis	93.3 / 93.2	99.6 / 99.6	99.0 / 97.7	99.1 / 99.1	99.4 / 99.4	69.7 / 26.5	99.8 / 45.0	99.8 / 44.7	99.4 / 99.4	99.1 / 99.1
PacBio	E. coli	94.8 / 94.7	100.0 / 100.0	100.0 / 98.8	99.7 / 99.7	100.0 / 100.0	66.1 / 20.9	100.0 / 42.6	100.0 / 42.6	99.9 / 99.9	99.7 / 99.7
PacBio	S. cerevisiae	93.9 / 93.9	98.9 / 98.9	98.9 / 97.6	98.2 / 98.2	98.8 / 98.7	69.3 / 24.6	99.6 / 44.5	99.3 / 44.4	98.1 / 98.0	98.2 / 98.2
PacBio	C. elegans	91.0 / 90.9	99.8 / 99.8	98.9 / 97.5	99.2 / 99.2	99.3 / 99.2	74.6 / 23.2	99.8 / 44.8	99.8 / 45.4	98.4 / 98.3	99.2 / 99.2
PacBio	H. sapiens (chr3)	86.8 / 86.8	99.7 / 99.7	98.5 / 96.4	98.7 / 98.7	98.8 / 98.4	71.3 / 24.8	100.0 / 44.6	100.0 / 44.5	94.5 / 94.1	98.7 / 98.7
PacBio	H. sapiens	87.4 / 87.2	-	98.2 / 95.5	98.4 / 98.4	96.2 / 95.2	70.4 / 27.4	100.0 / 43.3	100.0 / 44.4	94.6 / 93.7	98.3 / 98.3
ONT 2D	N. meningitidis	69.3 / 35.4	99.9 / 99.7	99.8 / 96.3	99.6 / 99.6	99.9 / 99.9	60.7 / 3.7	99.2 / 49.0	99.6 / 49.0	99.9 / 99.9	99.6 / 99.6
ONT 2D	E. coli	82.8 / 36.2	100.0 / 100.0	100.0 / 96.2	99.8 / 99.8	100.0 / 99.8	54.1 / 3.3	99.0 / 48.0	99.0 / 48.3	100.0 / 99.8	99.8 / 99.8
ONT 2D	S. cerevisiae	74.7 / 33.7	99.5 / 99.5	99.5 / 95.2	98.2 / 98.1	99.7 / 99.5	52.8 / 3.8	96.4 / 45.2	96.6 / 45.4	99.2 / 99.0	98.2 / 98.1
ONT 2D	C. elegans	66.5 / 33.9	99.7 / 99.7	97.7 / 93.0	99.0 / 99.0	99.3 / 98.3	43.2 / 1.9	96.1 / 46.7	96.4 / 46.0	99.3 / 98.3	99.0 / 99.0
ONT 2D	H. sapiens (chr3)	60.1 / 28.0	99.6 / 99.6	96.2 / 90.1	98.9 / 98.9	96.0 / 94.4	46.3 / 3.1	96.5 / 47.4	96.2 / 40.6	95.7 / 94.1	98.9 / 98.9
ONT 2D	H. sapiens	58.5 / 29.6	-	82.9 / 80.2	97.8 / 97.8	91.4 / 89.1	20.0 / 0.1	96.3 / 46.8	95.9 / 42.3	91.0 / 88.7	97.8 / 97.8
ONT 1D	N. meningitidis	14.4 / 4.2	96.9 / 94.3	74.8 / 51.1	98.6 / 97.7	99.5 / 96.5	0.0 / 0.0	73.6 / 32.4	75.2 / 33.9	99.3 / 96.2	98.6 / 97.6
ONT 1D	E. coli	17.5 / 4.5	97.9 / 95.5	75.0 / 53.1	99.2 / 98.7	100.0 / 95.4	0.0 / 0.0	71.6 / 31.0	73.1 / 32.3	100.0 / 95.4	99.2 / 98.7
ONT 1D	S. cerevisiae	10.9 / 3.0	95.4 / 94.4	69.9 / 47.4	95.5 / 95.2	98.8 / 89.6	0.0 / 0.0	60.8 / 23.1	60.2 / 23.7	98.6 / 89.3	95.6 / 95.2
ONT 1D	C. elegans	13.0 / 3.3	97.2 / 97.1	57.1 / 37.7	95.1 / 94.9	94.5 / 76.2	0.0 / 0.0	60.8 / 23.4	58.7 / 21.3	94.4 / 76.1	95.1 / 94.9
ONT 1D	H. sapiens (chr3)	7.7 / 1.6	96.4 / 95.7	47.3 / 32.5	96.0 / 95.5	81.9 / 66.5	0.0 / 0.0	57.3 / 22.4	51.0 / 7.7	81.8 / 66.4	96.0 / 95.5
ONT 1D	H. sapiens	8.6 / 1.3	-	23.1 / 19.1	94.1 / 94.0	53.5 / 39.7	0.0 / 0.0	59.6 / 23.0	44.8 / 1.3	53.1 / 39.4	94.1 / 94.0

Table B.2: Precision/Recall in reconstructing the correct alignment of synthetic reads.

Datatype	Genome	BLASR	BLAST	BWA-MEM	GraphMap	LAST	DALIGNER-default	DALIGNER-k10	DALIGNER-k9	marginAlign	marginAlign w/ GraphMap
Illumina	N. meningitidis	95.4 / 95.3	97.6 / 97.6	98.2 / 98.2	97.7 / 97.6	97.4 / 97.4	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	94.8 / 94.8	97.7 / 97.6
Illumina	E. coli	95.7 / 95.6	99.0 / 99.0	98.8 / 98.8	98.5 / 98.2	98.9 / 98.9	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	98.6 / 98.6	98.5 / 98.2
Illumina	S. cerevisiae	93.7 / 93.6	96.5 / 96.3	96.0 / 96.0	96.0 / 95.8	96.3 / 96.1	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	96.0 / 95.8	96.0 / 95.8
Illumina	C. elegans	94.3 / 94.2	98.1 / 97.6	98.0 / 98.0	97.2 / 96.4	98.2 / 98.1	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	92.5 / 92.4	97.2 / 96.4
Illumina	H. sapiens (chr3)	96.8 / 96.8	99.0 / 99.0	99.0 / 99.0	98.4 / 98.1	98.9 / 98.9	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	84.8 / 84.8	98.4 / 98.1
Illumina	H. sapiens	95.5 / 95.4	-	98.3 / 98.3	97.7 / 97.4	98.7 / 98.5	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	92.3 / 92.1	97.7 / 97.4
PacBio	N. meningitidis	10.1 / 10.0	78.8 / 78.8	80.7 / 80.1	78.3 / 78.3	80.7 / 80.7	81.9 / 23.6	81.3 / 39.8	81.2 / 39.6	85.1 / 85.0	84.8 / 84.8
PacBio	E. coli	8.0 / 7.9	79.9 / 79.8	81.8 / 81.4	79.4 / 79.3	81.8 / 81.7	83.1 / 20.5	82.1 / 39.7	82.0 / 39.7	85.9 / 85.8	85.7 / 85.7
PacBio	S. cerevisiae	11.5 / 11.4	78.5 / 78.4	80.3 / 80.0	77.5 / 77.5	80.2 / 80.1	81.6 / 22.2	80.9 / 39.7	80.9 / 39.7	84.1 / 84.0	84.0 / 84.0
PacBio	C. elegans	8.0 / 7.9	77.3 / 77.2	79.1 / 78.5	77.1 / 77.1	79.0 / 78.9	80.3 / 19.5	79.3 / 38.8	79.3 / 39.4	83.5 / 83.4	84.0 / 84.0
PacBio	H. sapiens (chr3)	7.3 / 7.2	78.2 / 78.2	79.1 / 78.5	77.3 / 77.3	80.0 / 79.8	81.1 / 21.5	80.4 / 39.2	80.4 / 39.2	81.5 / 81.3	83.9 / 83.9
PacBio	H. sapiens	6.5 / 6.3	-	79.5 / 78.5	76.9 / 76.9	78.9 / 78.3	81.3 / 25.0	80.3 / 38.6	80.5 / 39.7	82.3 / 81.7	83.4 / 83.4
ONT 2D	N. meningitidis	4.8 / 2.0	80.7 / 80.6	81.9 / 80.1	79.4 / 79.4	81.7 / 81.6	90.5 / 2.9	81.0 / 38.6	81.0 / 38.6	84.4 / 84.4	84.3 / 84.3
ONT 2D	E. coli	7.1 / 2.9	81.4 / 81.4	82.7 / 80.4	80.0 / 80.0	82.5 / 82.2	91.2 / 2.5	81.9 / 38.5	81.9 / 38.5	85.0 / 84.7	84.7 / 84.7
ONT 2D	S. cerevisiae	5.8 / 2.4	79.9 / 79.8	81.2 / 78.9	77.1 / 77.0	81.0 / 80.9	90.7 / 3.2	81.5 / 34.2	81.4 / 34.4	83.5 / 83.4	82.0 / 81.9
ONT 2D	C. elegans	5.8 / 2.6	79.1 / 79.0	80.4 / 77.2	77.7 / 77.7	80.1 / 79.4	88.9 / 2.1	80.2 / 35.9	80.3 / 35.4	83.2 / 82.5	82.8 / 82.8
ONT 2D	H. sapiens (chr3)	7.8 / 3.1	79.7 / 79.4	80.5 / 76.1	77.9 / 77.9	80.0 / 78.3	89.8 / 2.9	81.1 / 35.1	81.1 / 32.4	82.6 / 80.8	82.9 / 82.9
ONT 2D	H. sapiens	4.9 / 2.1	-	79.3 / 69.9	77.4 / 77.4	79.6 / 75.0	75.1 / 0.1	80.9 / 35.0	81.3 / 34.1	81.9 / 77.3	82.4 / 82.3
ONT 1D	N. meningitidis	0.0 / 0.0	73.3 / 72.0	74.8 / 48.8	72.7 / 72.5	74.4 / 73.7	0.0 / 0.0	74.6 / 26.4	74.5 / 27.0	79.2 / 77.1	78.7 / 77.3
ONT 1D	E. coli	0.0 / 0.0	74.5 / 73.3	76.4 / 47.7	74.0 / 74.0	75.6 / 74.5	0.0 / 0.0	75.8 / 25.5	75.7 / 26.1	80.2 / 79.0	79.9 / 79.8
ONT 1D	S. cerevisiae	0.0 / 0.0	72.7 / 71.5	73.9 / 46.8	70.2 / 70.1	73.8 / 70.8	0.0 / 0.0	75.0 / 17.8	74.9 / 18.6	78.6 / 74.5	77.0 / 76.1
ONT 1D	C. elegans	0.0 / 0.0	71.1 / 70.6	71.5 / 37.2	67.8 / 67.8	72.2 / 63.6	0.0 / 0.0	73.3 / 19.0	73.2 / 18.5	77.5 / 68.4	74.1 / 74.1
ONT 1D	H. sapiens (chr3)	0.0 / 0.0	72.1 / 71.1	70.0 / 31.8	71.0 / 70.8	71.4 / 58.7	0.0 / 0.0	74.4 / 18.7	75.4 / 9.1	76.0 / 62.7	77.2 / 77.0
ONT 1D	H. sapiens	0.0 / 0.0	-	62.3 / 19.4	70.8 / 70.8	68.6 / 41.9	0.0 / 0.0	74.3 / 19.2	69.6 / 1.8	72.4 / 44.5	76.9 / 76.9

Table B.3: CPU time (in seconds) taken to execute each test on synthetic reads.

Datatype	Genome	BLASR	BLAST	BWA-MEM	GraphMap	LAST	DALIGNER-default	DALIGNER-k10	DALIGNER-k9	marginAlign	marginAlign w/ GraphMap
Illumina	N. meningitidis	2.6	22.4	0.1	4	0.9	1	1.8	2.7	48.8	51.8
Illumina	E. coli	2.6	5.1	0.1	6.4	0.5	1.5	1.5	2.6	92.6	101.8
Illumina	S. cerevisiae	4.1	16.2	0.1	15.4	0.7	4.9	4.6	8.2	62.5	60.9
Illumina	C. elegans	8.4	112.2	0.3	148.8	2.1	39.3	51.4	68.1	742.5	587.1
Illumina	H. sapiens (chr3)	11.5	1626.4	0.5	215.3	5.1	75.7	85.7	139.9	3639.2	3861.1
Illumina	H. sapiens	111.6	-	5.2	5767.6	9.7	1112.5	1199.9	1881.5	31207.5	11644.1
PacBio	N. meningitidis	26.9	464.1	45.6	25.8	43.8	2.5	267.9	4564.9	154.3	131.8
PacBio	E. coli	31	66.5	44.7	29.2	19.6	3	171	4850.3	173.9	183.3
PacBio	S. cerevisiae	44.1	380.7	50.9	45.3	36.9	6.6	1193.4	12129.3	212.4	153.2
PacBio	C. elegans	70.2	27366.7	49.7	334.5	152.4	42.4	2506.1	1831.7	1682.1	832.1
PacBio	H. sapiens (chr3)	82.5	104242.1	97.8	583	156.9	83.7	5289.7	552.4	4119.3	4471
PacBio	H. sapiens	205.4	-	55.1	14242.4	176.6	1146.5	59165	16353.4	49510	20623.4
ONT 2D	N. meningitidis	55.9	534.2	78.6	61.2	62.3	3.1	602.1	11211.9	254.8	251
ONT 2D	E. coli	57.6	77.9	68.4	64.3	32.4	3.8	409.2	13060.8	265.7	298.9
ONT 2D	S. cerevisiae	117.2	377.6	88.9	95.2	50.1	7.3	3857.7	35509.3	354.6	292.3
ONT 2D	C. elegans	119	15243.8	109.7	327	136.9	42.4	8987.7	3847.6	1560.4	939.7
ONT 2D	H. sapiens (chr3)	129.5	110670.2	167.9	534.5	175.3	94.3	12817.1	260.5	4602.7	4850
ONT 2D	H. sapiens	289.9	-	173.5	12224.3	205.8	1125.3	158245.8	28185.1	52659.5	20289.9
ONT 1D	N. meningitidis	50.4	105.7	108	58.2	34	2.4	385.1	6581.2	201.6	225.1
ONT 1D	E. coli	46.1	35.9	88.7	61	24.4	3.3	376.4	9240.9	238.6	283.6
ONT 1D	S. cerevisiae	96.8	177.8	120.7	88.9	33.5	6.7	2525.5	28763.4	241.2	264
ONT 1D	C. elegans	81.9	3727.2	77.8	262.6	56.9	39.1	6238.1	5086.8	891.3	852.4
ONT 1D	H. sapiens (chr3)	94.9	20591.8	82.8	413.4	44.6	75.9	14842.4	469.6	3530.9	4593.4
ONT 1D	H. sapiens	244.9	-	102.6	8995.6	62.5	1101.3	228589	40052.9	8717.3	17961.8

Table B.4: Maximum memory consumption (RSS; in *MB*) required to execute each test on synthetic reads.

Datatype	Genome	BLASR	BLAST	BWA-MEM	GraphMap	LAST	DALIGNER-default	DALIGNER-k10	DALIGNER-k9	marginAlign	marginAlign w/ GraphMap
Illumina	N. meningitidis	21	166	7	601	13	105	120	185	43	724
Illumina	E. coli	31	62	13	700	27	222	222	296	68	817
Illumina	S. cerevisiae	71	87	25	998	66	576	577	798	89	1070
Illumina	C. elegans	493	367	173	4613	535	4782	4782	5642	603	4667
Illumina	H. sapiens (chr3)	981	1848	336	8502	1130	9447	9447	11364	1909	8510
Illumina	H. sapiens	14789	-	5172	117513	15333	136452	136452	159843	18587	117668
PacBio	N. meningitidis	75	117	124	766	23	182	540	1522	321	761
PacBio	E. coli	103	67	129	904	44	263	793	2465	328	893
PacBio	S. cerevisiae	147	117	138	1069	85	595	2570	6089	521	1100
PacBio	C. elegans	607	1062	261	4186	554	4801	14820	16209	775	4229
PacBio	H. sapiens (chr3)	1112	2509	437	7681	1158	9467	28307	18885	1910	7744
PacBio	H. sapiens	14881	-	5347	103539	15352	136687	398627	264832	18589	103403
ONT 2D	N. meningitidis	96	82	136	981	34	262	719	2224	443	1006
ONT 2D	E. coli	108	58	146	969	47	338	1133	3823	437	1026
ONT 2D	S. cerevisiae	187	108	159	1426	98	598	3828	9553	686	1397
ONT 2D	C. elegans	657	531	401	4338	566	4801	21178	20564	616	4356
ONT 2D	H. sapiens (chr3)	1145	2259	574	8109	1166	9468	40296	15449	1910	8150
ONT 2D	H. sapiens	14948	-	5337	101764	15357	136668	568361	231013	18586	102713
ONT 1D	N. meningitidis	52	64	422	2393	69	211	522	1666	503	982
ONT 1D	E. coli	69	61	286	1324	63	291	906	3001	509	1285
ONT 1D	S. cerevisiae	139	106	574	1680	105	581	3037	8254	608	1360
ONT 1D	C. elegans	575	334	395	4241	565	4786	18324	20044	604	4265
ONT 1D	H. sapiens (chr3)	1084	1272	512	8038	1161	9451	35127	17511	1911	7994
ONT 1D	H. sapiens	14949	-	5374	102547	15367	136510	499923	249343	18587	102723

Table B.5: Results of mapping on various real datasets, part 1 / 2.

Lambda R6								
Mapper	Avg. coverage	% bases mapped	% reads mapped	CPU time [s]	Memory [MB]	Mapped reads/sec	Mapped bases/sec	Mapped bases/MB
GraphMap	2552.8	68.1	73.3	2210.9	1725.0	17.4	65093.2	83428.6
LAST	1855.4	30.2	52.2	897.4	40.0	15.4	71039.4	1593819.7
marginAlign	1627.0	30.5	52.2	147653.4	9640.0	0.1	436.3	6682.1
BWA-MEM	792.3	13.0	33.5	974.6	1304.0	36.8	28252.3	21114.7
BLASR	31.4	0.7	9.2	591.4	436.0	6.3	2405.9	3263.3
DALIGNER	29.4	1.4	4.5	152.6	6659.0	6.8	19678.4	450.9
E. Coli K-12 R7.3								
Mapper	Avg. coverage	% bases mapped	% reads mapped	CPU time [s]	Memory [MB]	Mapped reads/sec	Mapped bases/sec	Mapped bases/MB
GraphMap	63.8	96.6	88.9	6116.9	3533.0	11.5	49225.9	85227.8
LAST	53.2	61.3	53.3	1673.9	83.0	17.0	114046.4	2300095.6
marginAlign	45.2	61.5	53.3	173759.9	22174.0	0.2	1102.3	8638.0
BWA-MEM	41.6	55.2	50.9	5390.0	1289.0	12.5	31900.0	133391.3
BLASR	13.3	18.9	21.9	2895.8	807.0	5.3	20347.9	73014.5
DALIGNER	32.0	24.0	33.2	11975.8	44103.0	0.8	6255.8	1698.7
E. Coli K-12 R7.0								
Mapper	Avg. coverage	% bases mapped	% reads mapped	CPU time [s]	Memory [MB]	Mapped reads/sec	Mapped bases/sec	Mapped bases/MB
GraphMap	134.5	96.7	94.6	14,762.2	5,724.0	7.5	43,777.3	112,902.0
LAST	116.5	61.4	66.0	3,672.5	69.0	14.3	111,847.6	5,953,084.7
marginAlign	97.2	61.8	66.0	236,950.2	48,751.0	0.3	1,742.2	8,467.9
BWA-MEM	90.4	54.1	63.8	12,335.8	1,703.0	8.0	29,294.7	212,198.5
BLASR	26.4	17.9	26.5	8,534.1	1,609.0	3.5	14,003.9	74,276.8
DALIGNER	56.2	19.4	36.5	14,213.8	65,662.0	1.1	9,121.4	1,974.5
S. Enterica Typhi								
Mapper	Avg. coverage	% bases mapped	% reads mapped	CPU time [s]	Memory [MB]	Mapped reads/sec	Mapped bases/sec	Mapped bases/MB
GraphMap	32.3	96.6	90.9	3,734.6	2,884.0	8.7	43,777.9	56,689.5
LAST	26.0	57.8	60.3	892.3	58.0	16.0	109,586.1	1,685,996.6
marginAlign	22.1	58.1	60.3	153,852.0	11,295.0	0.1	639.1	8,705.5
BWA-MEM	20.7	50.9	58.5	2,328.2	1,025.0	12.5	37,039.2	84,130.4
BLASR	6.9	18.6	25.9	1,723.8	510.0	4.9	18,258.4	61,714.5
DALIGNER	14.4	20.4	32.9	10,319.6	27,068.0	0.4	3,346.7	1,275.9
E. Coli UTI89								
Mapper	Avg. coverage	% bases mapped	% reads mapped	CPU time [s]	Memory [MB]	Mapped reads/sec	Mapped bases/sec	Mapped bases/MB
GraphMap	8.2	99.5	98.7	467.7	1,196.0	19.3	80,161.6	31,348.4
LAST	8.8	85.5	88.1	240.2	57.0	24.3	134,098.7	565,134.1
marginAlign	7.6	85.6	88.1	88,798.5	3,684.0	0.1	363.5	8,761.1
BWA-MEM	7.0	75.9	86.7	725.9	432.0	11.0	39,390.4	66,186.9
BLASR	3.0	35.6	50.6	329.0	205.0	13.9	40,757.5	65,409.6
DALIGNER	5.2	38.6	53.4	1,748.7	7,354.0	1.5	8,317.8	1,977.8

Table B.6: Results of mapping on various real datasets, part 2 / 2.

A. baylyi ADP1								
Mapper	Avg. coverage	% bases mapped	% reads mapped	CPU time [s]	Memory [MB]	Mapped reads/sec	Mapped bases/sec	Mapped bases/MB
GraphMap	53.8	96.5	60.3	4,970.2	4,781.0	13.3	39,802.3	41,377.6
LAST	48.0	60.1	25.2	1,254.4	97.0	8.7	98,277.1	1,270,904.8
marginAlign	37.4	60.0	25.1	156,477.2	13,066.0	0.1	785.7	9,409.3
BWA-MEM	36.8	55.1	24.2	7,235.2	4,238.0	8.8	15,602.5	26,636.7
BLASR	19.0	32.1	14.3	4,722.6	650.0	2.0	13,930.3	101,211.4
DALIGNER	26.6	21.8	13.4	39,480.4	26,417.0	0.1	1,132.6	1,692.7
B. fragilis BE1								
Mapper	Avg. coverage	% bases mapped	% reads mapped	CPU time [s]	Memory [MB]	Mapped reads/sec	Mapped bases/sec	Mapped bases/MB
GraphMap	27.8	99.5	99.1	1,877.7	1,528.0	11.6	74,701.3	91,799.5
LAST	32.0	90.7	87.9	944.0	62.0	12.9	135,524.7	2,063,424.1
marginAlign	25.8	90.8	87.9	153,445.5	12,068.0	0.1	834.8	10,615.2
BWA-MEM	26.1	90.9	87.9	2,774.2	1,019.0	7.7	46,221.3	125,834.7
BLASR	15.0	53.5	54.0	1,617.2	470.0	7.3	46,621.8	160,418.7
DALIGNER	28.9	58.7	84.3	10,137.9	26,990.0	0.7	8,160.5	3,065.2
E. coli MAP006-1								
Mapper	Avg. coverage	% bases mapped	% reads mapped	CPU time [s]	Memory [MB]	Mapped reads/sec	Mapped bases/sec	Mapped bases/MB
GraphMap	220.8	98.7	98.1	18,413.6	38,412.0	6.3	56,694.9	27,177.8
LAST	259.4	83.2	90.2	7,327.5	69.0	6.7	120,180.3	12,762,660.8
marginAlign	192.6	83.8	90.2	331,801.0	77,552.0	0.3	2,671.1	11,428.3
BWA-MEM	198.4	83.4	89.1	18,084.6	1,363.0	5.9	48,790.0	647,358.6
BLASR	153.3	67.0	69.4	17,523.0	3,305.0	4.6	40,469.7	214,569.5
DALIGNER	209.2	49.8	79.1	8,683.8	73,848.0	3.4	60,720.2	7,140.1

Table B.7: Impact of various stages of GraphMap on its precision and recall.

Datatype	Genome	All Stages	Fixed seed k=13	w/o L1 filtering	w/o LCSk w/ L1	w/o LCSk w/o L1	4-1-4-1-4 seed only	6-1-6 seed only
Illumina	N. meningitidis	97.7 / 97.6	97.9 / 97.9	97.6 / 97.6	88.6 / 86.9	94.4 / 93.8	97.3 / 97.2	97.8 / 97.8
Illumina	E. coli	98.5 / 98.2	98.2 / 98.2	98.6 / 98.6	95.6 / 95.1	97.8 / 97.3	98.3 / 98.0	97.8 / 97.8
Illumina	S. cerevisiae	96.0 / 95.8	95.9 / 95.9	96.3 / 96.3	91.2 / 90.6	94.9 / 94.6	96.2 / 96.0	96.1 / 96.0
Illumina	C. elegans	97.2 / 96.4	97.1 / 97.1	97.8 / 97.8	82.2 / 78.5	88.2 / 86.1	97.1 / 96.6	97.2 / 97.0
Illumina	H. sapiens (chr3)	98.4 / 98.1	98.1 / 98.1	99.0 / 99.0	88.3 / 86.6	93.7 / 93.3	98.5 / 98.3	98.2 / 98.1
PacBio	N. meningitidis	99.1 / 99.1	99.0 / 99.0	99.1 / 99.1	76.6 / 61.2	64.4 / 52.0	99.2 / 99.2	99.0 / 99.0
PacBio	E. coli	99.7 / 99.7	99.7 / 99.7	99.9 / 99.9	82.8 / 67.4	71.9 / 59.5	99.7 / 99.7	99.7 / 99.7
PacBio	S. cerevisiae	98.2 / 98.2	97.9 / 97.9	98.8 / 98.8	72.1 / 57.1	61.6 / 49.8	98.1 / 98.1	98.0 / 98.0
PacBio	C. elegans	99.2 / 99.2	99.3 / 99.3	99.4 / 99.4	66.9 / 48.5	55.3 / 41.5	99.4 / 99.4	99.2 / 99.2
PacBio	H. sapiens (chr3)	98.7 / 98.7	98.5 / 98.5	98.8 / 98.8	66.7 / 51.1	57.1 / 44.5	98.8 / 98.8	98.5 / 98.5
ONT 2D	N. meningitidis	99.6 / 99.6	98.6 / 98.2	99.5 / 99.5	67.7 / 50.1	51.6 / 39.7	99.7 / 99.7	99.7 / 99.6
ONT 2D	E. coli	99.8 / 99.8	98.5 / 97.7	99.7 / 99.7	72.4 / 55.4	57.0 / 44.5	99.7 / 99.7	99.5 / 99.5
ONT 2D	S. cerevisiae	98.2 / 98.1	95.9 / 95.5	98.1 / 98.1	65.9 / 51.5	51.9 / 41.3	98.0 / 97.9	98.3 / 98.1
ONT 2D	C. elegans	99.0 / 99.0	95.6 / 95.5	98.8 / 98.8	59.2 / 41.9	43.6 / 32.9	98.7 / 98.7	98.6 / 98.6
ONT 2D	H. sapiens (chr3)	98.9 / 98.9	94.6 / 93.2	98.4 / 98.4	60.9 / 44.0	46.1 / 34.7	98.2 / 98.2	97.9 / 97.6
ONT 1D	N. meningitidis	98.6 / 97.7	88.2 / 82.5	97.4 / 97.2	76.2 / 60.5	61.8 / 50.4	98.1 / 97.0	96.5 / 95.4
ONT 1D	E. coli	99.2 / 98.7	90.4 / 84.5	98.3 / 98.1	77.0 / 61.1	64.4 / 52.5	98.5 / 97.5	96.8 / 95.9
ONT 1D	S. cerevisiae	95.5 / 95.2	80.5 / 77.4	95.3 / 95.3	66.0 / 50.9	55.5 / 44.2	94.8 / 94.2	92.5 / 91.5
ONT 1D	C. elegans	95.1 / 94.9	71.4 / 70.2	94.0 / 93.9	55.3 / 42.1	44.2 / 35.0	92.4 / 91.8	89.7 / 89.5
ONT 1D	H. sapiens (chr3)	96.0 / 95.5	72.7 / 68.4	95.4 / 95.1	63.2 / 50.1	50.4 / 41.6	94.2 / 93.4	90.7 / 89.6

Table B.8: Parameters used for generating simulated ONT reads

	2D reads	1D reads
Accuracy mean	0.69	0.59
Accuracy std	0.09	0.05
Accuracy min	0.40	0.40
Length mean	5600	4400
Length std	3500	3900
Length min	100	50
Length max	100000	100000
Error types ratio (mismatch:insertion:deletion)	55:17:28	51:11:38

Table B.9: Precision and recall of alignment for GraphMap using various read alignment settings

	Myers bit vector (default)	Gotoh	Anchored Alignment
N. meningitidis	79/79; 73/73	82/82; 75/73	80/79; 73/72
E. coli	80/80; 74/74	83/83; 76/76	80/80; 74/73
S. cerevisiae	77/77; 70/70	80/80; 72/72	79/77; 72/70
C. elegans	78/78; 68/68	81/81; 70/70	78/77; 71/67
H. sapiens chr 3	78/78; 71/71	81/81; 73/73	78/77; 71/70

Table B.10: Scalability as a function of read length and error rate. Data is in the format: CPU time [s] / Memory [MB].

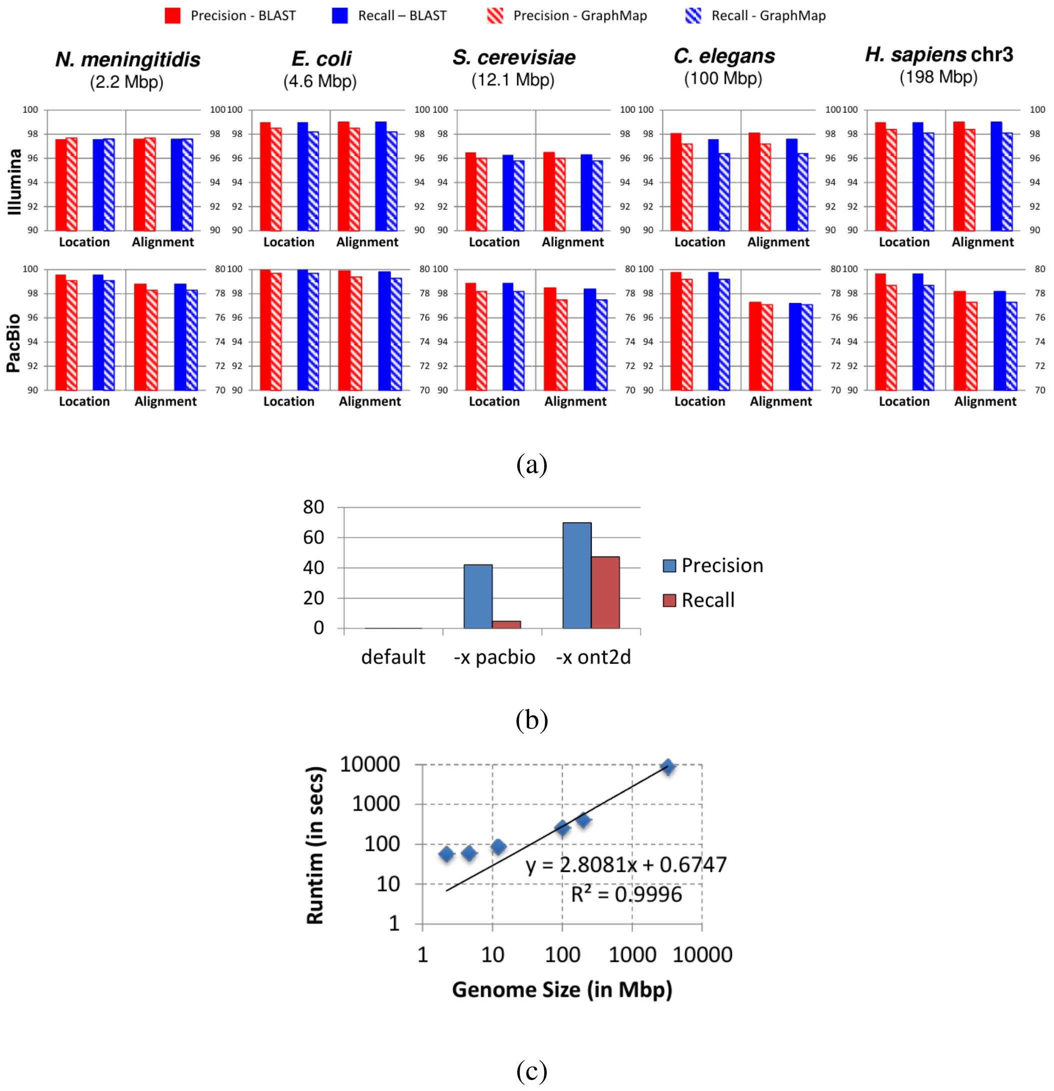
	Average read length				
Error rate	1000bp	2000bp	3000bp	4000bp	5000bp
0.05	130.7 / 952	210.7 / 960	278.5 / 972	349.5 / 992	457.9 / 1006
0.10	125.1 / 951	196.5 / 960	273.6 / 972	358.3 / 990	454 / 1006
0.15	119.9 / 951	195.9 / 959	257 / 972	365.1 / 989	461.4 / 1011
0.20	114.8 / 951	199.3 / 960	270.5 / 972	348.8 / 991	460.1 / 1008
0.25	108.7 / 951	196.7 / 960	271.9 / 972	358.1 / 991	485.2 / 1012

Table B.11: Testing for reference bias in GraphMap alignments

	SNP Errors (per Mbp)	Insertion Errors (per Mbp)	Deletion Errors (per Mbp)
BLASR	0.1 (0.02/0.1)	3.1 (0.04/3.1)	4.0 (0.3/3.7)
BWA-MEM	0.2 (0.1/0.1)	2.5 (0.04/2.5)	5.3 (1.7/3.6)
DALIGNER	0.4 (0.3/0.1)	1.2 (0.04/1.1)	6.9 (4.1/2.7)
GraphMap	0.2 (0.03/0.1)	3.3 (0.05/3.2)	4.1 (0.3/3.8)
LAST	1.7 (1.5/0.2)	3.9 (0.05/3.8)	4.6 (0.2/4.4)
marginAlign	0.1 (0.03/0.1)	2.0 (0.02/2.0)	4.4 (1.4/3.0)

Table B.12: Speed comparison across mappers on real datasets

	Lambda phage	E. coli R7.3	E. coli R7.0	E. coli UTI89	S. enterica Typhi
GraphMap	65	49	44	80	44
LAST	71	114	112	134	110
BWA-MEM	28	32	29	39	37
BLASR	2	20	14	41	18
marginAlign	0.4	1	2	0.4	0.7
DALIGNER	20	6	9	8	3

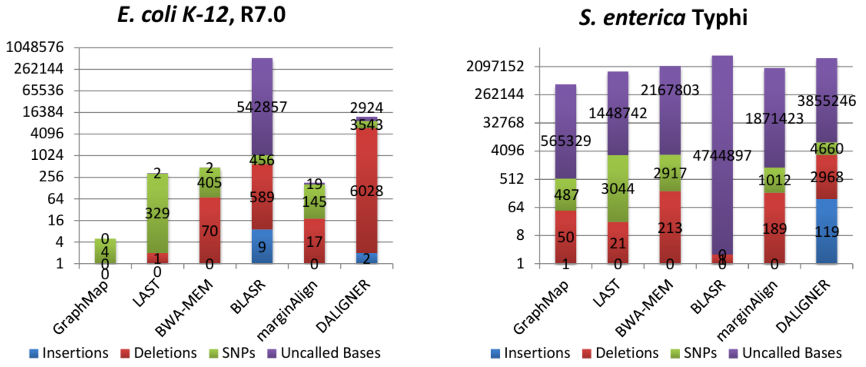


(a)

(b)

(c)

Figure B.1: Performance evaluation on synthetic datasets. *a*) GraphMap compared to BLAST on synthetic Illumina and PacBio reads *b*) BWA-MEM location results with different settings (*S. cerevisiae* genome; 1D reads) *c*) Runtime scalability for GraphMap (1D reads).



	E. coli K-12 R7.0	*S. enterica* Typhi	*E. coli* UTI89	*A. baylyi* ADP1	*B. fragilis* BE1
GraphMap	**97%, 135**	**97%, 32**	**99%,** 8	**97%, 54**	**99%,** 25
LAST	66%, 117	58%, 26	85%, **9**	60%, 48	91%, **29**
BWA-MEM	64%, 90	51%, 21	76%, 7	55%, 37	91%, 23
BLASR	27%, 26	19%, 7	36%, 3	32%, 11	53%, 5
marginAlign	66%, 97	58%, 22	86%, 8	60%, 37	91%, 22
DALIGNER	19%, 56	20%, 14	39%, 5	22%, 23	59%, 26

Figure B.2: Consensus calling errors and uncalled bases using MinION datasets and different mappers. Note that in the case of the *S. enterica* Typhi dataset, some of the observed variants (typically a few hundred SNPs and a handful of indels) could be true variants from the *S. enterica* Typhi Ty2 strain that was used as reference. Percentage of bases mapped (B%) and average coverage (C) of the genome is reported in the table below (in the format: B%, C; maximum values in each column are bolded).

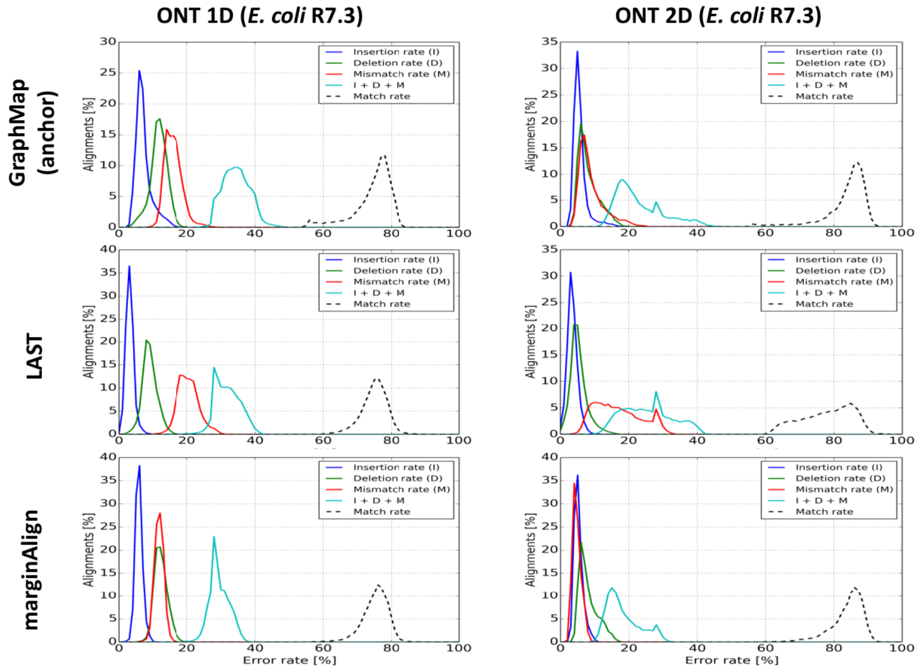


Figure B.3: Error rate distributions estimated using different aligners for ONT data.

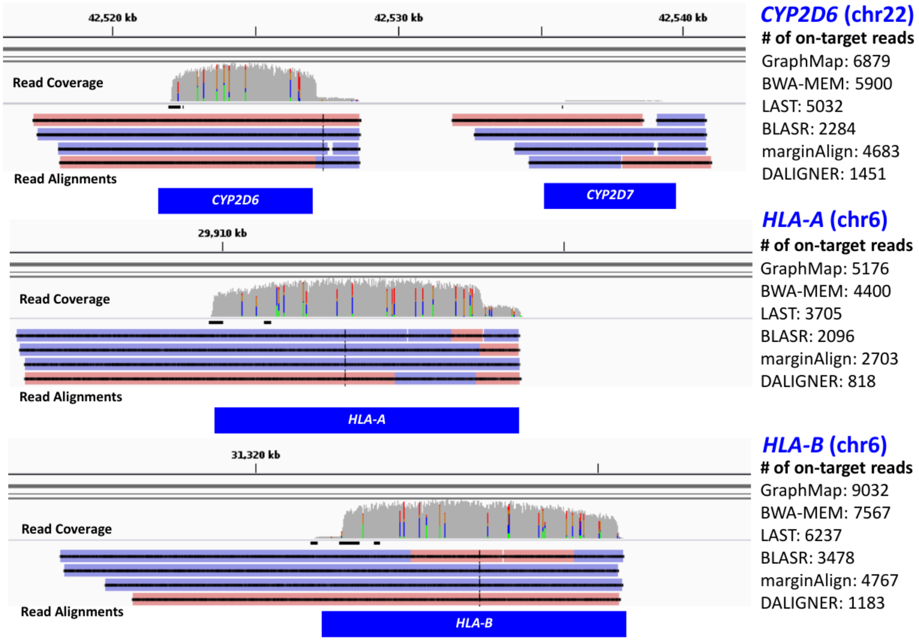


Figure B.4: Mapping of targeted sequencing reads from Ammar *et al.* Figures show a IGV browser view of GraphMap mappings to the targeted regions. Note that CYP2D6 has an orthologous gene CYP2D7 that is adjacent to it with 94% identity and yet has very few reads mapped to it.

B.6). The use of marginAlign with LAST did not improve its sensitivity significantly for these datasets. BWA-MEM results were frequently comparable to that of LAST while DALIGNER and BLASR had lower sensitivity in several datasets (Tables B.5 and B.6). Two of the datasets (*E. coli* UTI89 and *B. fragilis* BE1) contain only high quality 2D reads and associated 1D reads, and thus they only test mappers on a small, high-quality subset of the data. GraphMap was seen to provide a $10-15\%$ increase in sensitivity for such reads. On the full datasets, GraphMap typically provided a 50% improvement in mapped bases compared to LAST. The datasets *A. baylyi* ADP1 and *B. fragilis* BE1 were recently published and provide a more current perspective on GraphMap's utility for all data and high-quality 2D data, respectively. On a recent MinION MkI dataset (*E. coli* MAP006-1), GraphMap provided an 18% improvement in mapped bases compared to other mappers (Tables B.5 and B.6).

Bibliography

[1] Byrd, A. L., Perez-Rogers, J. F., Manimaran, S., Castro-Nallar, E., Toma, I., McCaffrey, T., Siegel, M., Benson, G., Crandall, K. a., Johnson, W. E., "Clinical PathoScope: rapid alignment and filtration for accurate pathogen identification in clinical samples using unassembled sequencing data", BMC Bioinformatics, Vol. 15, No. 1, 2014, str. 262, available at: http://www.biomedcentral.com/1471-2105/15/262

[2] Greninger, A. L., Naccache, S. N., Federman, S., Yu, G., Mbala, P., Bres, V., Stryke, D., Bouquet, J., Somasekar, S., Linnen, J. M., Dodd, R., Mulembakani, P., Schneider, B. S., Muyembe-Tamfum, J.-J., Stramer, S. L., Chiu, C. Y., "Rapid metagenomic identification of viral pathogens in clinical samples by real-time nanopore sequencing analysis.", Genome medicine, Vol. 7, No. 1, 2015, str. 99, available at: http://www.ncbi.nlm.nih.gov/pubmed/26416663

[3] Nagarajan, N., Pop, M., "Sequence assembly demystified.", Nature reviews. Genetics, Vol. 14, No. 3, 2013, str. 157–67, available at: http://www.ncbi.nlm.nih.gov/pubmed/23358380

[4] Miller, J. M., Malenfant, R. M., Moore, S. S., Coltman, D. W., "Short reads, circular genome: Skimming solid sequence to construct the bighorn sheep mitochondrial genome", Journal of Heredity, Vol. 103, No. 1, 2012, str. 140–146.

[5] 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, No. 5, 2012, str. 434–9, available at: http://www.ncbi.nlm.nih.gov/pubmed/22522955

[6] Pettersson, E., Lundeberg, J., Ahmadian, A., "Generations of sequencing technologies", Genomics, Vol. 93, No. 2, 2009, str. 105–111, available at: http://dx.doi.org/10.1016/j.ygeno.2008.10.003

[7] Schirmer, M., Ijaz, U. Z., D'Amore, R., Hall, N., Sloan, W. T., Quince, C., "Insight into biases and sequencing errors for amplicon sequencing with the Illumina MiSeq platform", Nucleic Acids Research, Vol. 43, No. 6, 2015.

[8] Koren, S., Schatz, M. C., Walenz, B. P., Martin, J., Howard, J. T., Ganapathy, G., Wang, Z., Rasko, D. A., McCombie, W. R., Jarvis, E. D., Adam M Phillippy, "Hybrid error correction and de novo assembly of single-molecule sequencing reads.", Nature biotechnology, Vol. 30, No. 7, 2012, str. 693–700, available at: http://dx.doi.org/10.1038/nbt.2280

[9] Chin, C.-S., Alexander, D. H., Marks, P., Klammer, A. A., Drake, J., Heiner, C., Clum, A., Copeland, A., Huddleston, J., Eichler, E. E., Turner, S. W., Korlach, J., "Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data", Nature Methods, Vol. 10, No. 6, 2013, str. 563–569, available at: http://www.nature.com/doifinder/10.1038/nmeth.2474

[10] Chaisson, M. J., Tesler, G., "Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR): application and theory.", BMC bioinformatics, Vol. 13, 2012, str. 238, available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3572422{&}tool=pmcentrez{&}rendertype=abstract

[11] Li, H., "Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM", arXiv preprint arXiv, Vol. 00, No. 00, 2013, str. 3, available at: http://arxiv.org/abs/1303.3997

[12] Laver, T., Harrison, J., O'Neill, P. A., Moore, K., Farbos, A., Paszkiewicz, K., Studholme, D. J., "Assessing the performance of the Oxford Nanopore Technologies MinION", Biomolecular Detection and Quantification, Vol. 3, 2015, str. 1–8, available at: http://linkinghub.elsevier.com/retrieve/pii/S2214753515000224

[13] Ip, C. L., Loose, M., Tyson, J. R., de Cesare, M., Brown, B. L., Jain, M., Leggett, R. M., Eccles, D. A., Zalunin, V., Urban, J. M., Piazza, P., Bowden, R. J., Paten, B., Mwaigwisya, S., Batty, E. M., Simpson, J. T., Snutch, T. P., Birney, E., Buck, D., Goodwin, S., Jansen, H. J., O'Grady, J., Olsen, H. E., "MinION Analysis and Reference Consortium: Phase 1 data release and analysis", F1000Research, Vol. 4, No. 1075, 2015, str. 1–35, available at: http://f1000research.com/articles/4-1075/v1

[14] Sovic, I., Sikic, M., Wilm, A., Fenlon, S. N., Chen, S., Nagarajan, N., "Fast and sensitive mapping of error-prone nanopore sequencing reads with GraphMap", bioRxiv, Vol. 7, apr 2015, str. 020719, available at: http://biorxiv.org/content/early/2015/06/10/020719.abstract

[15] Jain, M., Fiddes, I. T., Miga, K. H., Olsen, H. E., Paten, B., Akeson, M., "Improved data analysis for the MinION nanopore sequencer", Nature Methods, Vol. 12, No. 4, 2015, str. 351–356, available at: http://www.nature.com/doifinder/10.1038/nmeth.3290

[16] Loman, N. J., Quick, J., Simpson, J. T., "A complete bacterial genome assembled de novo using only nanopore sequencing data", Nature Methods, Vol. 12, No. 8, 2015, str. 733–735, available at: http://www.nature.com/doifinder/10.1038/nmeth.3444

[17] Taudien, S., Ebersberger, I., Glöckner, G., Platzer, M., "Should the draft chimpanzee sequence be finished?", str. 122–125, 2006.

[18] Pop, M., "Genome assembly reborn: Recent computational challenges", Briefings in Bioinformatics, Vol. 10, No. 4, 2009, str. 354–366, available at: http://bib.oxfordjournals.org/cgi/doi/10.1093/bib/bbp026

[19] Arner, E., Solving Repeat Problems in Shotgun Sequencing, 2006, available at: http://diss.kib.ki.se/2006/91-7140-996-3/thesis.pdf$\delimiter"026E30F$npapers2://publication/uuid/57797D1C-2D32-4C2F-A37B-7BCCC3D86C50

[20] Koren, S., Harhay, G. P., Smith, T. P. L., Bono, J. L., Harhay, D. M., Mcvey, S. D., Radune, D., Bergman, N. H., Phillippy, A. M., "Reducing assembly complexity of microbial genomes with single-molecule sequencing.", Genome biology, Vol. 14, No. 9, 2013, str. R101, available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=4053942{&}tool=pmcentrez{&}rendertype=abstract

[21] Biosciences, P., "PacBio read lengths", available at: http://www.pacb.com/smrt-science/smrt-sequencing/read-lengths/ 2016.

[22] Loose, M., Malla, S., Stout, M., "Real time selective sequencing using nanopore technology.", bioRxiv, 2016, str. 038760, available at: http://biorxiv.org/content/early/2016/02/03/038760.abstract

[23] Bao, S., Jiang, R., Kwan, W., Wang, B., Ma, X., Song, Y.-Q., "Evaluation of next-generation sequencing software in mapping and assembly.", Journal of human genetics, Vol. 56, No. May, 2011, str. 1–9, available at: http://www.ncbi.nlm.nih.gov/pubmed/21677664

[24] Pevsner, J., Bioinformatics and Functional Genomics, 2nd Ed., 2009, available at: http://www.amazon.com/Bioinformatics-Functional-Genomics-Edition-Jonathan/dp/B004KPVA46?SubscriptionId=1V7VTJ4HA4MFT9XBJ1R2{&}tag=mekentosjcom-20{&}linkCode=xm2{&}camp=2025{&}creative=165953{&}creativeASIN=B004KPVA46$\delimiter"026E30F$npapers2://publication/uuid/7BBC0F38-C8AF-4355-A

[25] Gotoh, O., "Journal of Molecular Biology_1982_Gotoh_An improved algorithm for matching biological sequences.pdf", Journal of Molecular Biology, Vol. 162, No. 3, 1982, str. 705–708.

[26] Needleman, S. B., Wunsch, C. D., "A general method applicable to the search for similiarities in the amino acid sequence of two proteins", Journal of molecular biology, Vol. 48, No. 3, 1970, str. 443–453.

[27] Smith, T. F., Waterman, M. S., "Identification of Common Molecular Subsequences", J. Mol. Biol., Vol. 147, 1981, str. 195–197.

[28] Lipman, D., Pearson, W., "Rapid and sensitive protein similarity searches", Science, Vol. 227, 1985, str. 1435–1441, available at: http://www.sciencemag.org/content/227/4693/1435.short

[29] Pearson, W. R., Lipman, D. J., "Improved tools for biological sequence comparison.", Proceedings of the National Academy of Sciences of the United States of America, Vol. 85, No. 8, 1988, str. 2444–8, available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=280013{&}tool=pmcentrez{&}rendertype=abstract

[30] Altschul, S. F., Gish, W., Miller, W., Myers, E. W., Lipman, D. J., "Basic local alignment search tool.", Journal of molecular biology, Vol. 215, No. 3, 1990, str. 403–10, available at: http://www.sciencedirect.com/science/article/pii/S0022283605803602

[31] Kent, W. J., "BLAT—The BLAST-Like Alignment Tool", Genome Research, Vol. 12, No. 4, mar 2002, str. 656–664, available at: http://www.genome.org/cgi/doi/10.1101/gr.229202

[32] Kielbasa, S. M., Wan, R., Sato, K., Horton, P., Frith, M. C., "Adaptive seeds tame genomic sequence comparison", Genome Research, Vol. 21, No. 3, 2011, str. 487–493.

[33] Sankoff, D., "Matching sequences under deletion-insertion constraints.", Proceedings of the National Academy of Sciences of the United States of America, Vol. 69, No. 1, 1972, str. 4–6.

[34] Sellers, P. H., "The theory and computation of evolutionary distances: Pattern recognition", Journal of Algorithms, Vol. 1, No. 4, 1980, str. 359–373.

[35] Hirschberg, D. S., "A linear space algorithm for computing maximal common subsequences", Communications of the ACM, Vol. 18, No. 6, 1975, str. 341–343.

[36] Fickett, J. W., "Fast optimal alignment.", Nucleic acids research, Vol. 12, No. 1 Pt 1, 1984, str. 175–179.

[37] Ukkonen, E., “Algorithms for approximate string matching”, Information and Control, Vol. 64, No. 1-3, 1985, str. 100–118.

[38] Myers, G., “A Fast Bit-Vector Algorithm for Approximate String Matching Based on Dynamic Programming”, Journal of the ACM, Vol. 46, No. 3, 1999, str. 395–415.

[39] Rognes, T., Seeberg, E., “Six-fold speed-up of Smith-Waterman sequence database searches using parallel processing on common microprocessors.”, Bioinformatics (Oxford, England), Vol. 16, No. 8, aug 2000, str. 699–706, available at: http://www.ncbi.nlm.nih.gov/pubmed/11099256

[40] Rognes, T., “Faster Smith-Waterman database searches with inter-sequence SIMD parallelisation.”, BMC bioinformatics, Vol. 12, No. 1, 2011, str. 221, available at: http://www.biomedcentral.com/1471-2105/12/221

[41] Farrar, M., “Striped Smith-Waterman speeds database searches six times over other SIMD implementations”, Bioinformatics, Vol. 23, No. 2, 2007, str. 156–161.

[42] Zhao, M., Lee, W. P., Garrison, E. P., Marth, G. T., “SSW library: An SIMD Smith-Waterman C/C++ library for use in genomic applications”, PLoS ONE, Vol. 8, No. 12, 2013, str. 1–7.

[43] Korpar, M., ??iki??, M., “SW#-GPU-enabled exact alignments on genome scale”, Bioinformatics, Vol. 29, No. 19, 2013, str. 2494–2495.

[44] Korpar, M., Šošić, M., Blažeka, D., Šikić, M., “SW#db: GPU-accelerated exact sequence similarity database search”, PLoS ONE, Vol. 10, No. 12, 2015.

[45] Liu, Y., Wirawan, A., Schmidt, B., “CUDASW++ 3.0: accelerating Smith-Waterman protein database search by coupling CPU and GPU SIMD instructions.”, BMC bioinformatics, Vol. 14, 2013, str. 117, available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3637623{&}tool=pmcentrez{&}rendertype=abstract

[46] Klus, P., Lam, S., Lyberg, D., Cheung, M. S., Pullan, G., McFarlane, I., Yeo, G. S., Lam, B. Y., “BarraCUDA - a fast short read sequence aligner using graphics processing units.”, BMC research notes, Vol. 5, 2012, str. 27, available at: http://www.scopus.com/inward/record.url?eid=2-s2.0-84855722896{&}partnerID=tZOtx3y1

[47] Hasan, L., Kentie, M., Al-Ars, Z., “DOPA: GPU-based protein alignment using database and memory access optimizations.”, BMC research notes, Vol. 4, No. 1, 2011, str. 261, available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3166271{&}tool=pmcentrez{&}rendertype=abstract

[48] li Shah, H. A., Hasan, L., Ahmad, N., “An optimized and low-cost FPGA-based DNA sequence alignment–a step towards personal genomics”, Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Annual Conference, Vol. 2013, 2013, str. 2696–2699.

[49] Hall, A., “Short-Read DNA Sequence Alignment with Custom Designed FPGA-based Hardware by A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF”, No. November, 2010.

[50] Altschul, S. F., Madden, T. L., Schäffer, A. A., Zhang, J., Zhang, Z., Miller, W., Lipman, D. J., “Gapped BLAST and PSI-BLAST: A new generation of protein database search programs”, str. 3389–3402, 1997.

[51] Johnson, M., Zaretskaya, I., Raytselis, Y., Merezhuk, Y., McGinnis, S., Madden, T. L., “NCBI BLAST: a better web interface.”, Nucleic acids research, Vol. 36, No. Web Server issue, 2008.

[52] Pearson, W. R., BLAST and FASTA Similarity Searching for Multiple Sequence Alignment. Totowa, NJ: Humana Press, 2014, str. 75–101, available at: http://dx.doi.org/10.1007/978-1-62703-646-7{_}5

[53] Gumbel, E. J., “Statistics of extremes”, Columbia University Press, New York, 1958.

[54] Karlin, S., Altschul, S. F., “Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes.”, Proceedings of the National Academy of Sciences of the United States of America, Vol. 87, No. 6, 1990, str. 2264–2268.

[55] Schwartz, S., Kent, W. J., Smit, A., Zhang, Z., Baertsch, R., Hardison, R. C., Haussler, D., Miller, W., “Human-mouse alignments with BLASTZ.”, Genome research, Vol. 13, No. 1, jan 2003, str. 103–107, available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=430961{&}tool=pmcentrez{&}rendertype=abstract

[56] Ma, B., Tromp, J., Li, M., “PatternHunter: faster and more sensitive homology search.”, Bioinformatics (Oxford, England), Vol. 18, No. 3, 2002, str. 440–445.

[57] Delcher, A. L., Kasif, S., Fleischmann, R. D., Peterson, J., White, O., Salzberg, S. L., “Alignment of whole genomes”, Nucleic Acids Research, Vol. 27, No. 11, 1999, str. 2369–2376.

[58] Kurtz, S., Phillippy, A., Delcher, A. L., Smoot, M., Shumway, M., Antonescu, C., Salzberg, S. L., "Versatile and open software for comparing large genomes.", Genome biology, Vol. 5, No. 2, 2004, str. R12, available at: http://genomebiology.com/2004/5/2/R12$\delimiter"026E30F$nhttp://www.ncbi.nlm.nih.gov/pubmed/14759262$\delimiter"026E30F$nhttp://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC395750

[59] "Novocraft.com: Novoalign short read mapper (http://www.novocraft.com/main/downloadpage.php)", available at: http://www.novocraft.com/main/downloadpage.php

[60] Langmead, B., Salzberg, S. L., "Fast gapped-read alignment with Bowtie 2", Nat Methods, Vol. 9, No. 4, 2012, str. 357–359, available at: http://dx.doi.org/10.1038/nmeth.1923

[61] Li, R., Yu, C., Li, Y., Lam, T. W., Yiu, S. M., Kristiansen, K., Wang, J., "SOAP2: An improved ultrafast tool for short read alignment", Bioinformatics, Vol. 25, No. 15, 2009, str. 1966–1967.

[62] Rumble, S. M., Lacroute, P., Dalca, A. V., Fiume, M., Sidow, A., Brudno, M., "SHRiMP: Accurate mapping of short color-space reads", PLoS Computational Biology, Vol. 5, No. 5, 2009.

[63] Harris, R., "Improved pairwise alignment of genomic DNA", Doktorski rad, The Pennsylvania State University, 2007.

[64] Li, H., Durbin, R., "Fast and accurate short read alignment with Burrows-Wheeler transform", Bioinformatics, Vol. 25, No. 14, 2009, str. 1754–1760.

[65] Ashton, P. M., Nair, S., Dallman, T., Rubino, S., Rabsch, W., Mwaigwisya, S., Wain, J., O'Grady, J., "MinION nanopore sequencing identifies the position and structure of a bacterial antibiotic resistance island", Nature Biotechnology, Vol. 33, No. 3, 2014, str. 296–300, available at: http://www.nature.com/doifinder/10.1038/nbt.3103

[66] Wilm, A., Aw, P. P. K., Bertrand, D., Yeo, G. H. T., Ong, S. H., Wong, C. H., Khor, C. C., Petric, R., Hibberd, M. L., Nagarajan, N., "LoFreq: A sequence-quality aware, ultra-sensitive variant caller for uncovering cell-population heterogeneity from high-throughput sequencing datasets", Nucleic Acids Research, Vol. 40, No. 22, 2012, str. 11 189–11 201, available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3526318{&}tool=pmcentrez{&}rendertype=abstract

[67] Wang, Y., Yang, Q., Wang, Z., "The evolution of nanopore sequencing", str. 449, 2015.

[68] Risse, J., Thomson, M., Patrick, S., Blakely, G., Koutsovoulos, G., Blaxter, M., Watson, M., “A single chromosome assembly of Bacteroides fragilis strain BE1 from Illumina and MinION nanopore sequencing data.”, GigaScience, Vol. 4, No. 1, 2015, str. 60, available at: http://gigascience.biomedcentral.com/articles/10.1186/s13742-015-0101-6

[69] Madoui, M.-A., Engelen, S., Cruaud, C., Belser, C., Bertrand, L., Alberti, A., Lemainque, A., Wincker, P., Aury, J.-M., “Genome assembly using Nanopore-guided long and error-free DNA reads”, BMC Genomics, Vol. 16, No. 1, 2015, str. 327, available at: http://www.biomedcentral.com/1471-2164/16/327

[70] Langmead, B., Salzberg, S. L., “Fast gapped-read alignment with Bowtie 2”, Nat Methods, Vol. 9, No. 4, 2012, str. 357–359, available at: http://dx.doi.org/10.1038/nmeth.1923

[71] Mikheyev, A. S., Tin, M. M. Y., “A first look at the Oxford Nanopore MinION sequencer”, Molecular Ecology Resources, Vol. 14, No. 6, 2014, str. 1097–1102, available at: http://www.ncbi.nlm.nih.gov/pubmed/25187008

[72] Miller, J. R., Koren, S., Sutton, G., “Assembly algorithm for next-generation sequencing data.”, Genomics, Vol. 95, No. 6, 2010, str. 315–327.

[73] Warren, R. L., Sutton, G. G., Jones, S. J. M., Holt, R. A., “Assembling millions of short DNA sequences using SSAKE”, Bioinformatics, Vol. 23, No. 4, 2007, str. 500–501, available at: http://bioinformatics.oxfordjournals.org/cgi/doi/10.1093/bioinformatics/btl629

[74] Jeck, W. R., Reinhardt, J. A., Baltrus, D. A., Hickenbotham, M. T., Magrini, V., Mardis, E. R., Dangl, J. L., Jones, C. D., “Extending assembly of short DNA sequences to handle error”, Bioinformatics, Vol. 23, No. 21, 2007, str. 2942–2944.

[75] Dohm, J. C., Lottaz, C., Borodina, T., Himmelbauer, H., “SHARCGS, a fast and highly accurate short-read assembly algorithm for de novo genomic sequencing”, Genome Research, Vol. 17, No. 11, 2007, str. 1697–1706.

[76] Myers, E. W., “Toward Simplifying and Accurately Formulating Fragment Assembly”, Journal of Computational Biology, Vol. 2, No. 2, 1995, str. 275–290, available at: http://www.liebertonline.com/doi/abs/10.1089/cmb.1995.2.275

[77] Medvedev, P., Georgiou, K., Myers, G., Brudno, M., “Computability of Models for Sequence Assembly”, Gene, Vol. 4645, 2007, str. 289–301, available at: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=pubmed{&}cmd=Retrieve{&}dopt=

AbstractPlus{&}list{_}uids=7496691639575447408related:cAeJt1OZCWgJ$\delimiter"026E30F$nhttp://www.springerlink.com/index/H711368771048H21.pdf

[78] Lin, Y., Li, J., Shen, H., Zhang, L., Papasian, C. J., Deng, H. W., “Comparative studies of de novo assembly tools for next-generation sequencing technologies”, Bioinformatics, Vol. 27, No. 15, 2011, str. 2031–2037.

[79] Davendra, D., Traveling Salesman Problem, Theory and Applications, 2010, available at: http://www.intechopen.com/books/traveling-salesman-problem-theory-and-applications

[80] Myers, E. W., “The fragment assembly string graph”, Bioinformatics, Vol. 21, No. SUPPL. 2, 2005, str. ii79–ii85, available at: http://bioinformatics.oxfordjournals.org/cgi/doi/10.1093/bioinformatics/bti1114

[81] Li, H., “Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences”, arXiv, No. March, 2015, str. 1–7, available at: http://arxiv.org/abs/1512.01801

[82] 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., 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, str. 2196–2204, available at: http://www.ncbi.nlm.nih.gov/pubmed/10731133

[83] Canu, “Canu assembler”, available at: https://github.com/marbl/canu 2016.

[84] Margulies, M., Egholm, M., Altman, W. E., Attiya, S., Bader, J. S., Bemben, L. a., Berka, J., Braverman, M. S., Chen, Y.-J., Chen, Z., Dewell, S. B., Du, L., Fierro, J. M., Gomes, X. V., Godwin, B. C., He, W., Helgesen, S., Ho, C. H., Ho, C. H., Irzyk, G. P., Jando, S. C., Alenquer, M. L. I., Jarvie, T. P., Jirage, K. B., Kim, J.-B., Knight, J. R., Lanza, J. R., Leamon, J. H., Lefkowitz, S. M., Lei, M., Li, J., Lohman, K. L., Lu, H., Makhijani, V. B., McDade, K. E., McKenna, M. P., Myers, E. W., Nickerson, E., Nobile, J. R., Plant, R., Puc, B. P., Ronan, M. T., Roth, G. T., Sarkis, G. J., Simons, J. F., Simpson, J. W., Srinivasan, M., Tartaro, K. R., Tomasz, A., Vogt, K. a., Volkmer, G. a., Wang, S. H., Wang, Y., Weiner, M. P., Yu, P., Begley, R. F., Rothberg, J. M., “Genome sequencing in microfabricated high-density picolitre reactors.”, Nature, Vol. 437, No. 7057, 2005, str. 376–80, available at: http://www.ncbi.nlm.nih.gov/pubmed/16056220

[85] Hernandez, D., François, P., Farinelli, L., Østerås, M., Schrenzel, J., “De novo bacterial genome sequencing: Millions of very short reads assembled on a desktop computer”, Genome Research, Vol. 18, No. 5, 2008, str. 802–809.

[86] Hossain, M. S., Azimi, N., Skiena, S., “Crystallizing short-read assemblies around seeds.”, BMC bioinformatics, Vol. 10 Suppl 1, 2009, str. S16.

[87] Sommer, D. D., Delcher, A. L., Salzberg, S. L., Pop, M., “Minimus: a fast, lightweight genome assembler.”, BMC bioinformatics, Vol. 8, No. 1, 2007, str. 64, available at: http://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-8-64

[88] Simpson, J. T., Durbin, R., “Efficient de novo assembly of large genomes using compressed data structures”, Genome Research, Vol. 22, No. 3, 2012, str. 549–556, available at: http://genome.cshlp.org/cgi/doi/10.1101/gr.126953.111

[89] Li, H., “Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences”, Bioinformatics, mar 2016, available at: http://bioinformatics.oxfordjournals.org/content/early/2016/03/18/bioinformatics.btw152.abstract

[90] Chin, C.-s., Peluso, P., Sedlazeck, F. J., Nattestad, M., Concepcion, G. T., Dunn, C., Malley, R. O., Figueroa-balderas, R., Morales-cruz, A., Grant, R., Delledonne, M., Luo, C., Ecker, J. R., Cantu, D., Rank, D. R., “Phased Diploid Genome Assembly with Single Molecule Real-Time Sequencing”, 2016.

[91] Lőve, K., “Otto-von-Guericke-Universit at Magdeburg”, Doktorski rad, Otto-von-Guericke-Universit¨ at Magdeburg, 2010.

[92] Idury, R. M., Waterman, M. S., “A New Algorithm for DNA Sequence Assembly”, Journal of Computational Biology, Vol. 2, No. 2, 1995, str. 291–306, available at: http://online.liebertpub.com/doi/abs/10.1089/cmb.1995.2.291

[93] Pevzner, P. A., Tang, H., Waterman, M. S., “An Eulerian path approach to DNA fragment assembly.”, Proceedings of the National Academy of Sciences of the United States of America, Vol. 98, No. 17, 2001, str. 9748–53, available at: http://www.pnas.org/content/98/17/9748.abstract

[94] Bankevich, A., Pevzner, P. A., “TruSPAdes: barcode assembly of TruSeq synthetic long reads.”, Nature methods, Vol. 13, No. 3, 2016, str. 248–50, available at: http://www.nature.com/doifinder/10.1038/nmeth.3737$\delimiter"026E30F$nhttp://www.ncbi.nlm.nih.gov/pubmed/26828418

[95] Zerbino, D. R., Birney, E., “Velvet: Algorithms for de novo short read assembly using de Bruijn graphs”, Genome Research, Vol. 18, No. 5, 2008, str. 821–829, available at: http://www.ncbi.nlm.nih.gov/pubmed/18349386

[96] Butler, J., MacCallum, I., Kleber, M., Shlyakhter, I. A., Belmonte, M. K., Lander, E. S., Nusbaum, C., Jaffe, D. B., “ALLPATHS: De novo assembly of whole-genome shotgun microreads”, Genome Research, Vol. 18, No. 5, 2008, str. 810–820.

[97] Gnerre, S., Maccallum, I., Przybylski, D., Ribeiro, F. J., Burton, J. N., Walker, B. J., Sharpe, T., Hall, G., Shea, T. P., Sykes, S., Berlin, A. M., Aird, D., Costello, M., Daza, R., Williams, L., Nicol, R., Gnirke, A., Nusbaum, C., Lander, E. S., Jaffe, D. B., “High-quality draft assemblies of mammalian genomes from massively parallel sequence data.”, Proceedings of the National Academy of Sciences of the United States of America, Vol. 108, No. 4, 2011, str. 1513–8, available at: http://www.pnas.org/content/108/4/1513.abstract

[98] Simpson, J. T., Wong, K., Jackman, S. D., Schein, J. E., Jones, S. J. M., Birol, I., “ABySS: A parallel assembler for short read sequence data”, Genome Research, Vol. 19, No. 6, 2009, str. 1117–1123.

[99] Luo, R., Liu, B., Xie, Y., Li, Z., Huang, W., Yuan, J., He, G., Chen, Y., Pan, Q., Liu, Y., Tang, J., Wu, G., Zhang, H., Shi, Y., Liu, Y., Yu, C., Wang, B., Lu, Y., Han, C., Cheung, D. W., Yiu, S.-M., Peng, S., Xiaoqian, Z., Liu, G., Liao, X., Li, Y., Yang, H., Wang, J., Lam, T.-W., Wang, J., “SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler.”, GigaScience, Vol. 1, No. 1, 2012, str. 18, available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3626529{&}tool=pmcentrez{&}rendertype=abstract

[100] Boisvert, S., Laviolette, F., Corbeil, J., “Ray: simultaneous assembly of reads from a mix of high-throughput sequencing technologies.”, Journal of computational biology : a journal of computational molecular cell biology, Vol. 17, No. 11, 2010, str. 1519–1533, available at: http://www.liebertonline.com/doi/abs/10.1089/cmb.2009.0238

[101] Chin, C.-S., Alexander, D. H., Marks, P., Klammer, A. A., Drake, J., Heiner, C., Clum, A., Copeland, A., Huddleston, J., Eichler, E. E., Turner, S. W., Korlach, J., “Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data”, Nature Methods, Vol. 10, No. 6, 2013, str. 563–569, available at: http://www.nature.com/doifinder/10.1038/nmeth.2474

[102] 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, str. 2818–2824, available at: http://bioinformatics.oxfordjournals.org/cgi/doi/10.1093/bioinformatics/btn548

[103] Berlin, K., Koren, S., Chin, C.-S., Drake, J. P., Landolin, J. M., Phillippy, A. M., "Assembling large genomes with single-molecule sequencing and locality-sensitive hashing.", Nature biotechnology, Vol. 33, No. 6, 2015, str. 623–630, available at: http://dx.doi.org/10.1038/nbt.3238

[104] Goldberg, S. M. D., Johnson, J., Busam, D., Feldblyum, T., Ferriera, S., Friedman, R., Halpern, A., Khouri, H., Kravitz, S. a., Lauro, F. M., Li, K., Rogers, Y.-H., Strausberg, R., Sutton, G., Tallon, L., Thomas, T., Venter, E., Frazier, M., Venter, J. C., "A Sanger/pyrosequencing hybrid approach for the generation of high-quality draft assemblies of marine microbial genomes.", Proceedings of the National Academy of Sciences of the United States of America, Vol. 103, No. 30, 2006, str. 11 240–5, available at: http://www.ncbi.nlm.nih.gov/pubmed/16840556

[105] Myers, G., "Efficient local alignment discovery amongst noisy long reads", in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 8701 LNBI, 2014, str. 52–67.

[106] Liao, Y.-C., Lin, S.-H., Lin, H.-H., "Completing bacterial genome assemblies: strategy and performance comparisons", Scientific Reports, Vol. 5, 2015, str. 8747, available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=4348652{&}tool=pmcentrez{&}rendertype=abstract

[107] Bankevich, A., Nurk, S., Antipov, D., Gurevich, A. a., Dvorkin, M., Kulikov, A. S., Lesin, V. M., Nikolenko, S. I., Pham, S., Prjibelski, A. D., Pyshkin, A. V., Sirotkin, A. V., Vyahhi, N., Tesler, G., Alekseyev, M. a., Pevzner, P. a., "SPAdes: A New Genome Assembly Algorithm and Its Applications to Single-Cell Sequencing", Journal of Computational Biology, Vol. 19, No. 5, 2012, str. 455–477.

[108] AMOS, "AMOS message format", available at: http://amos.sourceforge.net/wiki/index.php/Programmer's{_}guide{#}AMOS{_}messages{_}and{_}the{_}Perl{_}API

[109] Treangen, T. J., Sommer, D. D., Angly, F. E., Koren, S., Pop, M., Next generation sequence assembly with AMOS, 2011, No. SUPP.33.

[110] Melsted, P., "GFA format", available at: https://github.com/pmelsted/GFA-spec/blob/master/GFA-spec.md

[111] Wick, R. R., Schultz, M. B., Zobel, J., Holt, K. E., "Bandage: Interactive visualization of de novo genome assemblies", Bioinformatics, Vol. 31, No. 20, 2015, str. 3350–3352.

[112] WGS, “Overlap types”, available at: http://wgs-assembler.sourceforge.net/wiki/index.php/Overlaps

[113] Chaisson, M. J. P., Wilson, R. K., Eichler, E. E., “Genetic variation and the de novo assembly of human genomes”, Nature Reviews Genetics, Vol. 16, No. 11, 2015, str. 627–640, available at: http://dx.doi.org/10.1038/nrg3933

[114] Broder, a., “Identifying and filtering near-duplicate documents”, Combinatorial Pattern Matching, 2000, str. 1–10, available at: http://www.springerlink.com/index/KTN21YJUL3R379XY.pdf

[115] Roberts, M., Hayes, W., Hunt, B. R., Mount, S. M., Yorke, J. A., “Reducing storage requirements for biological sequence comparison”, Bioinformatics, Vol. 20, No. 18, 2004, str. 3363–3369.

[116] PacBio, “bas.h5 Reference Guide”, str. 1–7, available at: https://s3.amazonaws.com/files.pacb.com/software/instrument/2.0.0/bas.h5+Reference+Guide.pdf

[117] CHIN, C. S., Marks, P., Alexander, D., Klammer, A., Turner, S. W., Lengsfeld, C. S., Shoureshi, R. A., “Hierarchical genome assembly method using single long insert library”, Vol. 1, No. 19, 2014, available at: http://www.google.com/patents/US20140025312

[118] Carneiro, M. O., Russ, C., Ross, M. G., Gabriel, S. B., Nusbaum, C., Depristo, M. a., “Pacific biosciences sequencing technology for genotyping and variation discovery in human data”, BMC Genomics, Vol. 13, No. 1, 2012, str. 375, available at: BMCGenomics

[119] Modzelewski, M., Dojer, N., “MSARC: Multiple sequence alignment by residue clustering”, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 8126 LNBI, 2013, str. 259–272.

[120] Chatzou, M., Magis, C., Chang, J.-M., Kemena, C., Bussotti, G., Erb, I., Notredame, C., “Multiple sequence alignment modeling: methods and applications”, Briefings in Bioinformatics, No. October, 2015, str. 1–15, available at: http://bib.oxfordjournals.org/content/early/2015/11/27/bib.bbv099.abstract

[121] Lee, C., Grasso, C., Sharlow, M. F., “Multiple sequence alignment using partial order graphs”, Bioinformatics, Vol. 18, No. 3, 2002, str. 452–464, available at: http://bioinformatics.oxfordjournals.org/content/18/3/452

[122] Lee, C., “Generating consensus sequences from partial order multiple sequence alignment graphs”, Bioinformatics, Vol. 19, No. 8, 2003, str. 999–1008.

[123] Quick, J., Quinlan, A. R., Loman, N. J., “Erratum: A reference bacterial genome dataset generated on the MinIONTM portable single-molecule nanopore sequencer”, GigaScience, Vol. 4, No. 1, 2015, str. 6, available at: http://www.gigasciencejournal.com/content/4/1/6

[124] Loman, N. J., Quinlan, A. R., “Poretools: A toolkit for analyzing nanopore sequence data”, Bioinformatics, Vol. 30, No. 23, 2014, str. 3399–3401, available at: http://biorxiv.org/content/early/2014/07/23/007401.abstract

[125] Gurevich, A., Saveliev, V., Vyahhi, N., Tesler, G., “QUAST: Quality assessment tool for genome assemblies”, Bioinformatics, Vol. 29, No. 8, 2013, str. 1072–1075.

[126] Delcher, A. L., Salzberg, S. L., Phillippy, A. M., “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, str. Unit 10.3, available at: http://www.ncbi.nlm.nih.gov/pubmed/18428693

[127] Burkhardt, S., Kärkkäinen, J., “One-gapped q-gram filters for Levenshtein distance”, Combinatorial pattern matching, Vol. 14186, 2002, str. 225–234, available at: http://link.springer.com/chapter/10.1007/3-540-45452-7{_}19

[128] Li, M., Ma, B., Kisman, D., Tromp, J., “PatternHunter II: highly sensitive and fast homology search.”, Genome informatics. International Conference on Genome Informatics, Vol. 14, No. 03, 2003, str. 164–75, available at: papers3://publication/uuid/B9084C91-D38A-4297-863C-B1C29E604D18$\delimiter"026E30F$nhttp://www.worldscientific.com/doi/abs/10.1142/S0219720004000661$\delimiter"026E30F$nhttp://www.ncbi.nlm.nih.gov/pubmed/15706531

[129] Benson, G., Levy, A., Shalom, R., “Longest Common Subsequence in k-length substrings”, 2014, available at: http://arxiv.org/abs/1402.2097

[130] Pavetic, F., Zuzic, G., Sikic, M., “$LCSk$++: Practical similarity metric for long strings.”, CoRR, Vol. abs/1407.2, 2014, available at: http://arxiv.org/abs/1407.2407

[131] Ammar, R., Paton, T. A., Torti, D., Shlien, A., Bader, G. D., “Long read nanopore sequencing for detection of HLA and CYP2D6 variants and haplotypes.”, F1000Research, Vol. 4, No. 0, 2015, str. 17, available at: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=4392832{&}tool=pmcentrez{&}rendertype=abstract

[132] Huang, W., Li, L., Myers, J. R., Marth, G. T., “ART: A next-generation sequencing read simulator”, Bioinformatics, Vol. 28, No. 4, 2012, str. 593–594.

[133] Ono, Y., Asai, K., Hamada, M., “PBSIM: PacBio reads simulator - Toward accurate genome assembly”, Bioinformatics, Vol. 29, No. 1, 2013, str. 119–121.

[134] Goodwin, S., Gurtowski, J., Ethe-Sayers, S., Deshpande, P., Schatz, M., McCombie, W. R., “Oxford Nanopore Sequencing and de novo Assembly of a Eukaryotic Genome”, bioRxiv, 2015, str. 13490, available at: http://biorxiv.org/content/early/2015/01/06/013490.abstract

[135] Loman, N. J., Quick, J., Simpson, J. T., “A complete bacterial genome assembled de novo using only nanopore sequencing data”, Nature Methods, Vol. 12, No. 8, 2015, str. 733–735, available at: http://www.nature.com/doifinder/10.1038/nmeth.3444

[136] Zook, J. M., Chapman, B., Wang, J., Mittelman, D., Hofmann, O., Hide, W., Salit, M., “Integrating human sequence data sets provides a resource of benchmark SNP and indel genotype calls”, Nature Biotechnology, Vol. 32, No. 3, 2014, str. 246–251, available at: http://www.nature.com/nbt/journal/v32/n3/full/nbt.2835.html$\delimiter"026E30F$nhttp://www.nature.com/nbt/journal/v32/n3/pdf/nbt.2835.pdf

[137] Layer, R. M., Chiang, C., Quinlan, A. R., Hall, I. M., “LUMPY: a probabilistic framework for structural variant discovery.”, Genome biology, Vol. 15, No. 6, 2014, str. R84, available at: http://genomebiology.com/2014/15/6/R84

[138] Thorvaldsdóttir, H., Robinson, J. T., Mesirov, J. P., “Integrative Genomics Viewer (IGV): High-performance genomics data visualization and exploration”, Briefings in Bioinformatics, Vol. 14, No. 2, 2013, str. 178–192.

[139] Szalay, T., Golovchenko, J. A., “De novo sequencing and variant calling with nanopores using PoreSeq”, Nature Biotechnology, Vol. 33, No. September, 2015, str. 1–7, available at: http://www.nature.com/nbt/journal/vaop/ncurrent/full/nbt.3360.html{#}ref1$\delimiter"026E30F$nhttp://www.nature.com/doifinder/10.1038/nbt.3360

[140] Zook, J. M., Chapman, B., Wang, J., Mittelman, D., Hofmann, O., Hide, W., Salit, M., “Integrating human sequence data sets provides a resource of benchmark SNP and indel genotype calls”, Nature Biotechnology, Vol. 32, No. 3, 2014, str. 246–251, available at: http://www.nature.com/nbt/journal/v32/n3/full/nbt.2835.html$\delimiter"026E30F$nhttp://www.nature.com/nbt/journal/v32/n3/pdf/nbt.2835.pdf

[141] Patel, A., Schwab, R., Liu, Y. T., Bafna, V., “Amplification and thrifty single-molecule sequencing of recurrent somatic structural variations”, Genome Research, Vol. 24, No. 2, 2014, str. 318–328.

[142] Cao, M. D., Ganesamoorthy, D., Elliott, A., Zhang, H., Cooper, M., Coin, L., “Real-time strain typing and analysis of antibiotic resistance potential using Nanopore MinION sequencing”, bioRxiv, 2015, str. 019356, available at: http://biorxiv.org/content/early/2015/05/15/019356

[143] Milne, I., Stephen, G., Bayer, M., Cock, P. J. A., Pritchard, L., Cardle, L., Shawand, P. D., Marshall, D., “Using tablet for visual exploration of second-generation sequencing data”, Briefings in Bioinformatics, Vol. 14, No. 2, 2013, str. 193–202.

[144] Sović, I., Križanović, K., Skala, K., Šikić, M., “Evaluation of hybrid and non-hybrid methods for de novo assembly of nanopore reads.”, Bioinformatics (Oxford, England), 2016, str. 030437, available at: http://biorxiv.org/content/early/2015/11/13/030437.abstract$\delimiter"026E30F$nhttp://www.ncbi.nlm.nih.gov/pubmed/27162186

[145] Krumsiek, J., Arnold, R., Rattei, T., “Gepard: A rapid and sensitive tool for creating dotplots on genome scale”, Bioinformatics, Vol. 23, No. 8, 2007, str. 1026–1028.

Biography

Ivan Sović was born on 4th of October 1986 in Zagreb. In 2005 he enrolled the Faculty of Electrical Engineering and Computing of University of Zagreb. In 2008 he obtained his bachelor degree in Electrical engineering, module Electronic and Computer Engineering. The topic of his thesis was "Visualization of macromolecules". He enrolled in the masters degree programme in 2008 on the same faculty and module. In academic year of 2009/2010, Ivan received a Rector's award for his work titled "New robust method for QSAR analysis based on multivariate linear regression and the L_1 norm". In 2010, Ivan graduated and obtained the title of Master of Science in Electrical Engineering and Information Technology. The topic of his thesis was: "Protein Docking Tool: Visualization module".

Since December of 2010, Ivan is employed as a research assistant in Centre for Informatics and Computing of the Ruđer Bošković Institute (RBI) in Zagreb. In 2011 he enrolled in the Ph.D. programme at the Faculty of Electrical Engineering and Computing of University of Zagreb. During his work at the RBI, he participated and worked on national and international projects: "Methods for scientific visualization" (Ministry of science, education and sport of Republic of Croatia, project number: 098-098 2562-2567) and "E2LP: Embedded engineering learning platform" (EU FP7, project number: 317882). During the work on his thesis, Ivan was awarded a scholarship by the Agency for Science, Technology and Research (A*STAR) of Singapore, allowing him to spend one year abroad on the Genome Institute of Singapore in the period of May 2014 to May 2015.

Ivan published over 20 works, including 10 research papers in journals, 5 book chapters and 8 conference papers.

Publications

Journal papers

1. Sović, I., Križanović, K., Skala, K., Šikić, M., "*Evaluation of hybrid and non-hybrid methods for de novo assembly of nanopore reads*", Bioinformatics, 11 (2016).
2. Sović, I., Šikić, M., Wilm, A., Fenlon, S.N., Chen, S., Nagarajan, N., "*Fast and sensitive mapping of nanopore sequencing reads with GraphMap*", Nature Communications, 7

(2016).

3. Skala, K., Davidović, D., Afgan, E., Sović, I., Šojat, Z., "*Scalable Distributed Computing Hierarchy: Cloud, Fog and Dew Computing*", Open Journal of Cloud Computing (OJCC), 2 (2015), 1; pp. 16-24.
4. Skala, K., Davidović, D., Lipić, T., Sović, I., "*G-Phenomena as a Base of Scalable Distributed Computing —G-Phenomena in Moore's Law*", International Journal of Internet and Distributed Systems, 2 (2014) , 1; pp. 1-4.
5. Lučić, B., Sović, I., Batista, J., Skala, K., Plavšić, D., Vikić-Topić, D., Bešlo, D., Nikolić, S., Trinajstić, N., "*The Sum-Connectivity Index - An Additive Variant of the Randić Connectivity Index*", Current computer-aided drug design, 9 (2013), pp. 184-194.
6. Lučić, B., Sović, I., Bešlo, D., Plavšić, D., Vikić-Topić, D., Trinajstić, N., "*On the Novel Balaban-like and Balaban-Detour-like Molecular Descriptors*", International Journal of Chemical Modeling, 5 (2013), 2/3; pp. 277-294.
7. Skala, K., Lipić, T., Sović, I., Grubišić, I., Grbeša, I., "*Towards 3D thermal models standardisation for human body in motion*", Quantitative InfraRed Thermography Journal, Vol. 10, No. 2, pp. 207-221, 2013.
8. Skala, K., Lipić, T., Sović, I., Gjenero, L., Grubišić, I., "*4D Thermal Imaging System for Medical Applications*", Periodicum biologorum, 113 (2011) , 4; pp. 407-416.
9. Skala Kavanagh, H., Dubravić, A., Grazio, S., Lipić, T., Sović, I., "*Computer supported thermography monitoring of hand strength evaluation by electronic dynamometer*", Periodicum biologorum, 113 (2011), 4; pp. 433-437.
10. Sović, I., Lipić, T., Gjenero, L., Grubišić, I., Skala, K., "*Experimental verification of heat source parameter estimation from 3D thermograms*", Periodicum biologorum, 113 (2011), 4; pp. 417-423.

Book chapters

1. Medved Rogina, B., Skala, K., Škoda, P., Sović, I., Michieli, I., "*Exercises for Embedded Engineering Learning Platform*", Book title: "Embedded Engineering Education", Editors: Szewczyk, R., Kaštelan, I., Temerinac, M., Barak, M., Sruk, V., Cham: Springer International Publishing, 2016. pp. 45-59.
2. Šojat, Z., Skala, K., Medved Rogina, B., Škoda, P., Sović, I., "*Implementation of Advanced Historical Computer Architectures*", Book title: "Embedded Engineering Education", Editors: Szewczyk, R., Kaštelan, I., Temerinac, M., Barak, M., Sruk, V., Publisher Cham: Springer International Publishing, 2016. pp. 61-79.
3. Lučić, B., Sović, I., Trinajstić, N., "*On coding and ordering benzenoids and their Kekulé structures by using Kekulé index and some related codes*", Book title: "Ante Graovac – Life and Works", Editors: Gutman, I., Pokrić, B., Vukičević., Kragujevac, University of

Kragujevac and Faculty of Science Kragujevac, 2014. pp. 163-178.

4. Lučić, B., Sović, I., Trinajstić, N., "*The four connectivity matrices, their indices, polynomials and spectra*", Book title: "Advances in mathematical chemistry and applications", Editors: Basak, S.C., Restrepo, G., Villaveces, J.L., Sharjah, UAE: Bentham Science Publishers, 2014. pp. 76-91.
5. Lučić, B. Sović, I., Plavšić, D., Trinajstić, N., "*Harary Matrices: Definitions, Properties and Applications*", Book title: "Distance in Molecular Graphs – Applications", Editors: Gutman, I., Furtula, B., Kragujevac: University of Kragujevac and Faculty of Science Kragujevac, 2012. pp. 3-26.

Conference papers

1. Sović, I., Skala, K., Šikić, M., "*Approaches to DNA de novo Assembly*", Proceedings of the 36th International Convention Mipro 2013.
2. Sović, I., Šikić, M., Skala, K., "*Aspects of DNA Assembly Acceleration on Reconfigurable Platforms*", PROCEEDINGS IT SYSTEMS 2013, Editors: Žagar, M., Knezović, J., Mlinarić, H., Hofman, D, Kovač, M., Bol, Brač, Hrvatska: University of Zagreb, Faculty of Electrical Engineering and Computing, 2013. pp. 289-292.
3. Afgan, E., Skala, K., Davidović, D., Lipić, T., Sović, I., "*CloudMan as a tool execution framework for the cloud*", Proceedings of the 35th International Convention MIPRO, 2012, Editors: Biljanović, P., Skala, K., Zagreb, 2012. pp. 437-441.
4. Bojović, V., Sović, I., Bačić, A., Lučić, B., Skala, K., "*A novel tool/method for visualization of orientations of side chains relative to the protein's main chain*", Proceedings Vol. I. MEET&GVS 34rd International Convention MIPRO 2011, Editors: Biljanović, P., Skala, K., Zagreb: Croatian Society for Information and Communication Technology, Electronics and Microelectronics - MIPRO, 2011. pp. 273-276.
5. Grubišić, I., Gjenero, L., Lipić, T., Sović, I., Skala, T., "*Active 3D scanning based 3D thermography system and medical applications*", Proceedings Vol. I. MEET&GVS 34rd International Convention MIPRO 2011, Editors: Biljanović, P., Skala, K., Zagreb: Croatian Society for Information and Communication Technology, Electronics and Microelectronics - MIPRO, 2011. pp. 300-304.
6. Sović, I., Lipić, T., Gjenero, L., Grubišić, I., Skala, K., "*Heat source parameter estimation from scanned 3D thermal models*", Proceedings Vol. I. MEET&GVS 34rd International Convention MIPRO 2011, Editors: Biljanović, P., Skala, K., Zagreb: Croatian Society for Information and Communication Technology, Electronics and Microelectronics - MIPRO, 2011. pp. 283-287.
7. Pale, P., Sović, A., Sović, I., Jeren, B., "*Some aspects of student teamwork on practical assignments for complex control and measurement systems*", Proceedings of the 33th In-

ternational Convention MIPRO, 2010, Computers in education, Editors: Čičin-Šain, M., Uroda, I., Turčić Prstačić, I., Sluganović, I., Opatija: Mipro, 2010. pp. 815-820.

8. Sović, I., Antulov-Fantulin, N., Čanadi, I., Šikić, M., "*Parallel Protein Docking Tool*", Proceedings of the 33th International Convention MIPRO, 2010, Opatija, pp. 1333-1338.

Životopis

Ivan Sović rođen je 4. listopada 1986. godine u Zagrebu. 2005. godine upisao je Fakultet elektrotehnike i računarstva Sveučilišta u Zagrebu. Godine 2008. završava preddiplomski studij s odličnim uspjehom na smjeru Elektrotehnika, modul Elektroničko i računalno inženjerstvo, sa završnim radom na temu "Vizualizacija makromolekula", te odmah potom upisuje diplomski studij na istom fakultetu, s istim usmjerenjem. Za akademsku godinu 2009./2010. nagrađen je Rektorovom nagradom za rad "Nova robusna metoda za QSAR analizu temeljena na multivarijatnoj linearnoj regresiji i normi L_1". Godine 2010. završava diplomski studij, također s odličnim uspjehom, s diplomskim radom na temu "Alat za prianjanje proteina: modul za vizualizaciju".

Od prosinca 2010. zaposlen je kao znanstveni novak na Institutu Ruđer Bošković (IRB), u Centru za informatiku i računarstvo. 2011. godine upisuje doktorski studij na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu. Tijekom rada na IRB-u, sudjelovao je kao suradnik na projektima: "Metode znanstvene vizualizacije" (Ministarstvo znanosti, obrazovanja i sporta Republike Hrvatske, broj projekta: 098-098 2562-2567) i "E2LP: Embedded engineering learning platform" (EU FP7, broj projekta: 317882). Tijekom izrade doktorskog rada, proveo je studijsku godinu u inozemstvu, u razdoblju svibanj 2014. - svibanj 2015., na Genome Institute of Singapore.

Objavio je preko 20 radova, od čega je 10 u časopisima s međunarodnom recenzijom, 5 poglavlja u knjigama i 8 članaka na konferencijama s međunarodnom recenzijom.