Softwarelösungen

Kosteneffektive hybride Genomassemblierung mit LazyB

THOMAS GATTER, PETER F. STADLER
INSTITUT FÜR BIOINFORMATIK, UNIVERSITÄT LEIPZIG

Advances in genome sequencing have led to a paradigm shift where project costs are no longer limited by sequencing costs but rather by the computational problems associated with genome assembly. There is an urgent demand for more efficient and accurate methods, in particular for complex genomes. The combination of traditional second and emerging third generation sequencing offers unique benefits. Our own method LazyB enables the resource efficient assembly of low abundant datasets.

DOI: 10.1007/s12268-022-1762-1

Stetig sinkende Sequenzierungskosten haben die Produktion eines zunehmenden Stroms an Sequenzdaten ermöglicht, deren bioinformatische Verarbeitung eine immer größere Herausforderung darstellt. Die Assemblierung von Sequenzdaten hat sich in der praktischen Anwendung als ein schwieriges Problem herausgestellt. Projektkosten für die Assemblierung komplexer Genome werden nicht mehr nur durch die Sequenzierungskosten bestimmt, sondern maßgeblich durch die Kosten der bioinformatischen Verarbeitung.

Moderne Ansätze verbinden die kurzen aber nahezu fehlerfreien Reads der zweiten Generation von Sequenzierungsplattformen (typischerweise Illumina) mit den sehr langen, aber fehleranfälligen Reads der dritten Generation (Pacific BioSciences oder Oxford Nanopore). Die komplementären Eigenschaften beider Technologien eignen sich bestens, um die Qualität von Assemblies bedeutend zu verbessern. Im Folgenden werden die Reads der zweiten und dritten Generation respektive als Kurze Reads und Lange Reads abkürzend bezeichnet.

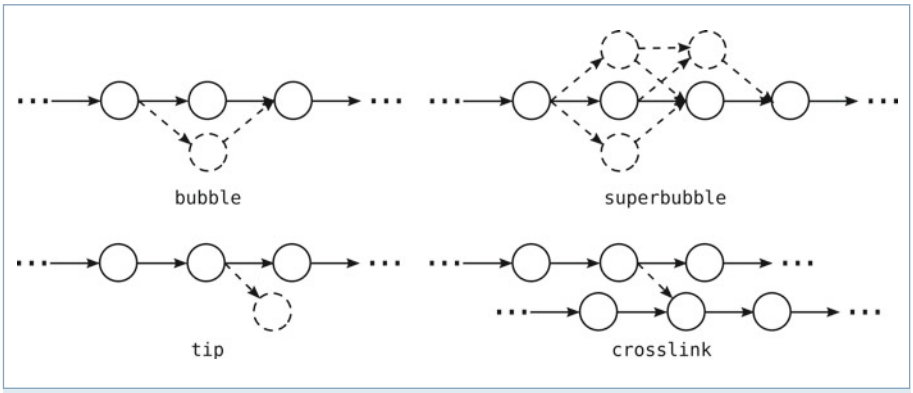
Klassische Ansätze zur Assemblierung von Genomen lassen sich in zwei Gruppen unterteilen. Das Overlap-Layout-Consensus(OLC)-Modell etabliert Überlappungen von Reads durch paarweise Vergleiche mittels einer Metrik für Sequenzähnlichkeit. Ein gemeinsames Layout wird aufgebaut und typischerweise in klassische Assemblierungsgraphen – wie Stringgraphen – überführt. Diese Klasse von Methoden ist äußerst flexibel in Bezug auf Readlängen und kompatiblen Fehlermodellen. Jedoch ist die Berechnung aller Überlappungen äußert kostenintensiv. Der Aufwand steigt beträchtlich mit zunehmenden Readlängen.

Auf De-Bruijn-Graphen basierte Verfahren zerteilen Reads in Teilsequenzen einer festen Länge k. Diese k-mere dienen als Knoten in einem Assemblierungsgraphen. Kanten verbinden alle Knoten mit einem gemeinsamen (k-1)-mer. Unter idealen Bedingungen präsentiert dieser Graph einen Euler-Pfad pro Komponente respektiv für jedes Chromosom. Bedingt durch Sequenzierungsfehler gilt dies unter realistischen Bedingungen jedoch nicht.

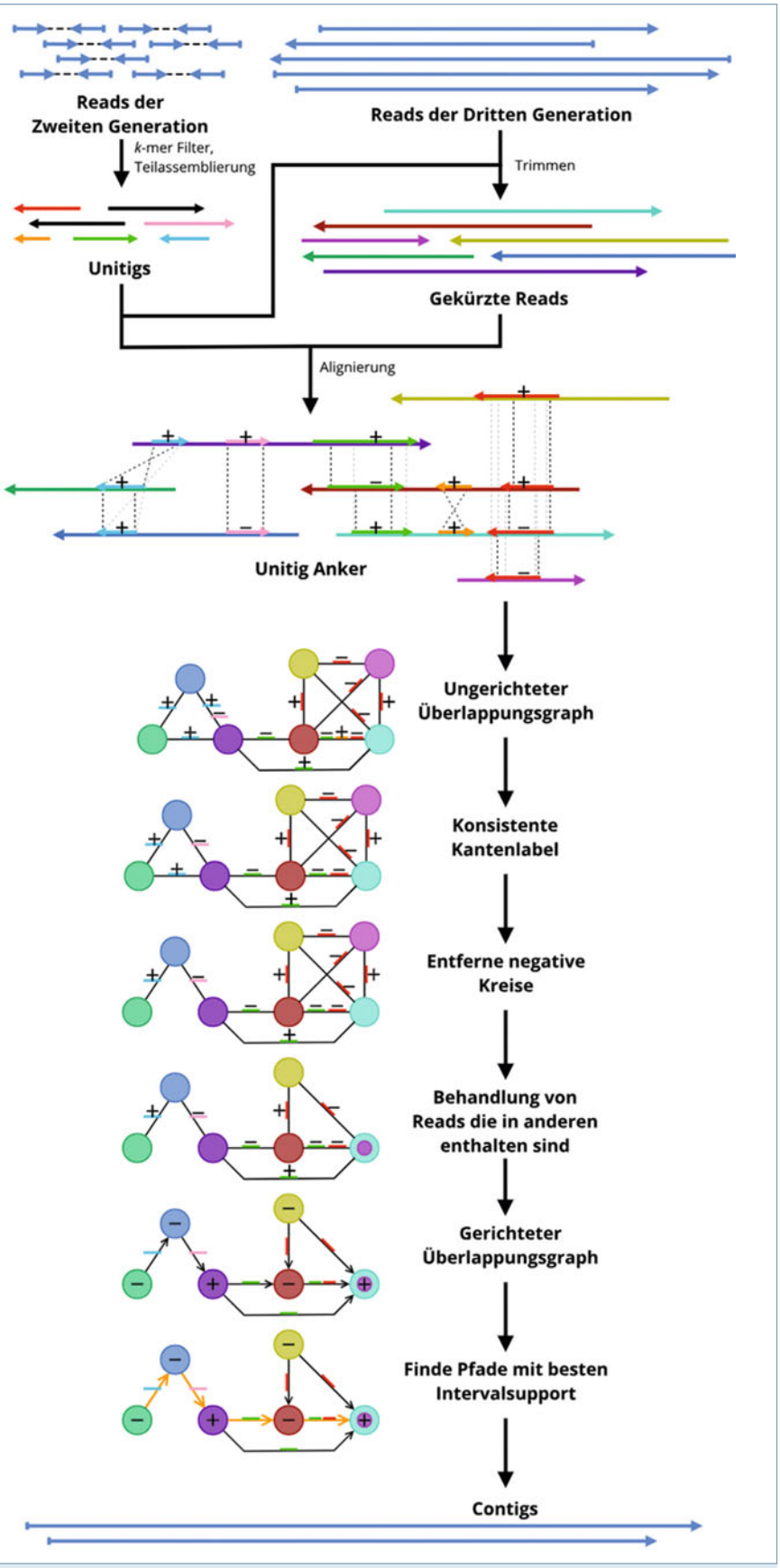
De-Bruijn-Graphen sind im Allgemeinen schneller und speicherplatzeffizienter als OLC-basierte Verfahren. Mit zunehmenden Read-Fehlerraten entstehen vermehrt fehlerhafte Pfade in De-Bruijn-Graphen – bedingt durch ebenso weniger akkurate k-mere, und die Qualität eines Assemblies sinkt beträchtlich. Lange Reads finden in De-Bruijn-Verfahren daher nur Anwendung, um Mehrdeutigkeiten in auf Kurzen Reads basierten Graphen aufzulösen. OLC-Strategien hingegen bieten ein breiteres Anwendungsspektrum für Assemblies mit Langen Reads. Im Fokus der Entwicklung stehen dabei Heuristiken, um Überlappungen auch ohne kostenintensive Jeden-gegen-Jeden Vergleiche zu bestimmen.

Bestehende Verfahren

Hybride Verfahren können für Kurze Reads zur effizienten und akkuraten Überlappung eingesetzt werden. Der Assembler **WENGAN** [1] führt zunächst ein traditionelles Assembly von ausschließlich Kurzen Reads durch. Aus Langen Reads werden künstliche *paired-end*-Reads generiert, die anschließend zum *scaffolding* des Assemblies Verwendung finden. Auch das Tool **HASLR** [2] assembliert



▲ **Abb. 1:** Beispiele für Graphdefekte in Assemblierungsgraphen. Fehlende oder falsche Überlappungen, z. B. aufgrund von Sequenzierungsfehlern, erzeugen alternative Pfade.



▲ **Abb. 2:** Ablaufdiagramm der LazyB-Methode. Fast perfekte Unitigs frei von repetitiven Elementen werden aus Kurzen Reads assembliert und gegen Lange Reads aligniert. Fehler in dem sich so ergebende Überlappungsgraphen werden schrittweise reduziert und Pfade als Contigs assembliert.

zunächst Kurze Reads zu Contigs, jedoch erst nach einem Filterschritt, der häufige *k*-mere, und damit repetitive Sequenzen, entfernt. Contigs werden gegen Lange Reads aligniert. Ein Assemblierungsgraph wird aufgebaut, indem Contigs als Knoten dienen, die mit einer Kante verbunden werden, falls sie auf mindestens einem gemeinsamen Langen Read liegen. Klassische Graphverarbeitungsstrategien können zum Assembly Anwendung finden. **DBG2OLC** [3] kehrt diese Graphdefinition um und setzt Lange Reads als Knoten ein, die mit Kanten verbunden werden, wenn auf beiden Partnern mindestens ein gemeinsames Contig aligniert wurde. Obwohl das Ablaufprotokoll explizit von der Verwendung von fehleranfälligen Assemblierungsschritten wie *scaffolding*, *gap closing* und *repeat-resolving* abrät, enthalten derart erzeugte Graphen dennoch zahlreiche falsch positive Kanten. Zur Korrektur dieser Fehler sowie für die Berechnung eines Konsenses zwischen allen Mappings ist eine hohe Read-Abdeckung (*coverage*) notwendig. Dies steht im Widerspruch zu den Bedürfnissen vieler Anwender, die Kosten für zusätzliche Sequenzdaten und Rechenaufwand aufbringen müssen.

Unitigs als besondere Anker

Um den Rechenaufwand für Assemblies bedeutend zu senken, aber dennoch adäquate Ergebnisse zu erzeugen, bedienen wir uns dem Konzept von Unitigs [4]. Obwohl De-Bruijn- und OLC-Verfahren konzeptionelle Unterschiede aufweisen, manifestieren sich Fehler nahezu identisch (**Abb. 1**). Sequenzierungsfehler nahe der Enden von Reads erzeugen divergierend Pfade, die aufgrund fehlender Überlappung oder *k*-mere nicht weiter geführt werden können, und damit in kurzen Sackgassen enden. Diese Art von Fehlern bezeichnet man als *tips*. Fehler in der Mitte von Reads verhindern hingegen nicht die Überlappung an deren Start- und Endpunkten. So entstehen parallellaufende Pfade, die im Hauptpfad beginnen und enden. Derartige *bubbles* können auch durch echte biologische Variation wie Einzelnukleotid-Polymorphismen entstehen. Überlagern sich verschiedene Fehlerquellen, aggregieren sich mehrere *bubbles* in eine *superbubble*.

Insbesondere in OLC-Verfahren können weitere komplexe Fehlerstrukturen entstehen. Von besonderem Interesse sind dabei *crosslinks*, die zwei echte Contig-Pfade miteinander verbinden. Der Begriff Unitig beschreibt in diesem Zusammenhang Pfade,

die mathematisch beweisbar in allen möglichen Assemblies vorkommen. In ihrer einfachsten Interpretation beschreiben Unitigs so Pfade im Graphen, deren Knoten ausschließlich eine eingehende und eine ausgehende Kante aufweisen, also genau nicht über Fehlerstrukturen hinweg laufen. Vor einem Assembly können Reads mit repetitiven Elementen z. B. über *k*-mer-Filter reduziert werden.

Wendet man eine derartige Vorverarbeitung an und setzt eine Mindestlänge von wenigen 100 Basenpaaren voraus, stellen Unitigs sowohl einzigartige als auch nahezu fehlerfreie Abschnitte dar.

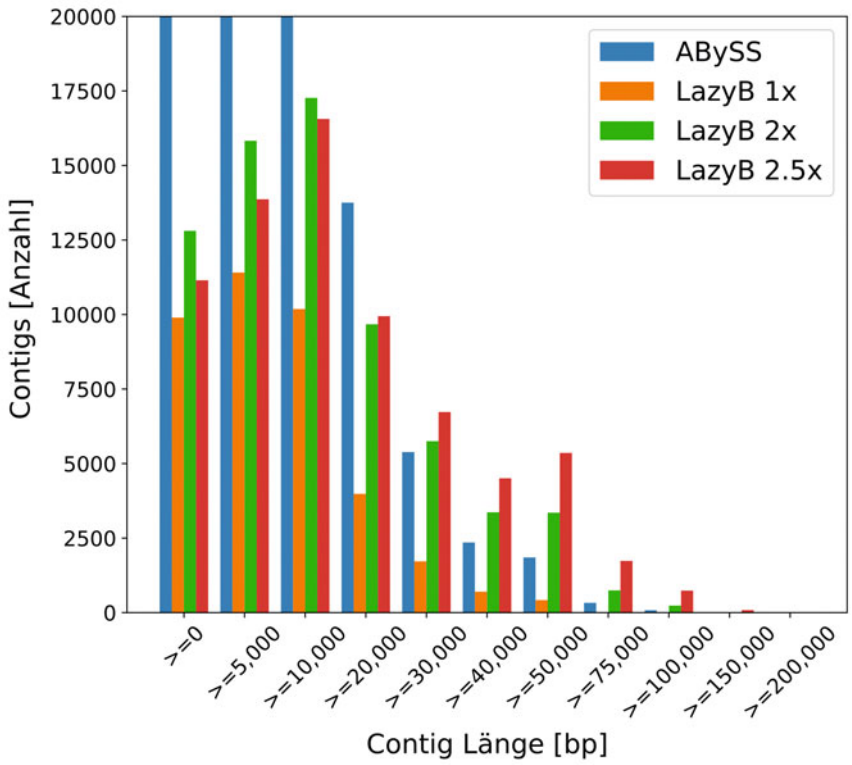
LazyB

Auch **LazyB** basiert auf einem OLC-Verfahren (**Abb. 2**). Kurze Reads werden nicht zu Contigs, sondern nach einem *k*-mer-Filterschritt mit dem Tool **AbySS** [5] zu nahezu fehlerfreien Unitigs ohne repetitive Elemente assembliert. Unitigs werden mit hoher Konfidenz gegen ein Set Langer Reads aligniert. Entsprechend definieren wir einen Assemblierungsgraphen wie folgt: Jeder Lange Read erzeugt einen Knoten. Jeder Unitig erzeugt Kanten von jedem assoziierten Knoten in seinem Map-Set zu jedem anderen Knoten in diesem. Diese Konstruktion erscheint initial identisch zu **DBG2OLC.** Die Wahl von Unitigs bedingt jedoch eine erheblich niedrigere Rate falsch positiver Elemente. Aufwendige Korrekturverfahren entfallen zugunsten einfacher Heuristiken und signifikant weniger Abdeckung von Langen Reads wird benötigt.

Da Reads von beiden DNA-Strängen erzeugt werden, können auch Unitigs direkt (+) oder als ihr reverses Komplement (−) auf diesen mappen. Pro Unitig und Kante wird nun die relative Orientierung als Produkt beider Richtungen vermerkt (+ = Unitigs mappen in gleicher Richtung, − = ein Unitig mapped indirekt und eines als reverses Komplement). Eine Qualitäts-Metrik ergibt sich aus der Länge der Region des Unitigs, der auf beide Langen Reads mapped. Mehrere Unitigs können die gleiche Kante bestimmen. Obwohl sowohl Unitigs als auch Mappings nahezu fehlerfrei konzipiert wurden, können Unitigs zueinander inkompatible Überlappungen bestimmen. Wir reduzieren diese wie folgt: Für jedes Read-Paar bestimmen wir das maximale Set kompatibler Unitigs, die diese Kante definieren als ein Maximales *chaining*. Die relative Orientierung auf Kanten bestimmt die relative Richtung eines Reads im Layout, also ob dieser direkt oder reverses Komplement in das finale Assembly eingefügt wird. Zyklen mit negativem Kantenprodukt, also einer ungeraden Zahl von „−“ Kanten, bedingen ein inkompatibles Layout, in dem ein Read in beiden Richtungen vorliegen müsste. Mithilfe einer Kirchhoff-Kreisbasis können wir derartige Kreise effizient enumerieren und entfernen. Der so verarbeitete Graph kann nun in ein gerichtetes Layout überführt werden. Eine zufälliger Read wird gewählt und in seiner natürlichen Orientierung angeordnet. Die Orientierungen weiterer Reads folgt aus der Traversierung des Graphen. Kanten werden so gerichtet, das sie in Richtung höherer Koordinaten in dieser Anordnung zeigen. Reads, die vollständig in einem anderen enthalten sind, können dabei entfernt werden, da diese uns keine zusätzlichen Informationen liefern können.

Das Ergebnis approximiert einen Intervallgraphen, in dem jeder Read ein Intervall in genomischen Koordinaten definiert, und jede Kante eine Überlappung der Intervalle. Jeder Pfad definiert nun einen potentielles Contig. Die Wahl längster Pfade allein begünstigt die Eingliederung von *crosslinks* und damit chimärer Contigs. Wir nutzen die Eigenschaften dieser Graphklasse, um stattdessen maximale plausible Pfade zu extrahieren. Genauer definiert die ausgehende Nachbarschaft jedes Knotens in Intervallgraphen einen Turniergraph. Wir schränken die Wahl maximaler Pfade so ein, dass in jeder Abzweigung die Kante, die in der maximalen Anzahl von Turnieren enthalten ist, gewählt wird. Für jeden Pfad wird nachfolgend eine Konsensus-Sequenz als das finale Assembly bestimmt. *Bubbles* und *tips* müssen nicht unmittelbar behandelt werden.

Bereits eine extrem niedrige Abdeckung von 2x Langen Reads ist ausreichend, um den Zusammenhang eines bestehendes Assembly basierend auf ausschließlich Kurzen Reads maßgeblich zu verbessern (**Abb. 3**). Andere getestete Assembler haben in diesen Benchmarks entweder gar kein



▲ **Abb. 3:** Histogramm der Längenverteilung von korrekt assemblierten Contigs von LazyB mit 1x, 2x und 2,5x Read-Abdeckung Langer Reads und 43x Kurzer Reads sowie einem Assembly mit den identischen Kurzen Reads (ABySS). Bereits bei 2x Abdeckung kann LazyB deutlich mehr Contigs > 20 kbp erzeugen.

Glossar

Stringgraph: Assemblierungsgraphen in denen jeder Knoten eine Abzweigung zwischen verschiedenen Sequenzen und damit unterschiedlichen Assemblierungen darstellt. Kanten bezeichnen einzigartige Sequenzen zwischen Knotenpaaren.

de Bruijn-Graph: Assemblierungsgraphen in denen jeder Knoten eine einzigartige Teilsequenz fester Länge bezeichnet. Kanten verbinden Knoten die bis auf je die letzte und erste Base übereinstimmen.

Euler-Pfad: Ein Pfad der genau jeden Knoten eines Graphens einmal besucht.

Contig: Die Ausgabesequenz eines Assemblierungsprogramms. Teilsequenzen werden heuristisch zusammengefügt und können daher Fehler aufweisen.

Unitig: Nachweisbar nahezu fehlerfreie Teilsequenzen eines Assemblies.

Ergebnis erzeugt (z. B. **DBG2OLC** und **HASLR**) oder Korrekturmechanismen schlagen fehl und Contigs enthalten ein hohes Maß an Fehlern (z. B. **WENGAN**). Unsere Methode eignet sich damit sehr gut für z. B. explorative Assemblierung. Eine effiziente Implementierung von LazyB ist auf GitHub frei verfügbar. ■

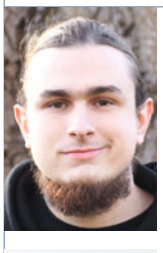
Literatur

[1] Di Genova A, Buena-Atienza E, Ossowski S, Sagot MF (2021) Efficient hybrid de novo assembly of human genomes with WENGAN. Nat Biotechnol 39: 422–430
[2] Haghshenas E, Asghari H, Stoye J et al. (2020) HASLR: fast hybrid assembly of long reads. iScience 23: 101389
[3] Ye C, Hill CM, Wu S et al. (2016). DBG2OLC: efficient assembly of large genomes using long erroneous reads of the third generation sequencing technologies. Sci Rep 6: 1–9
[4] Kececioglu JD, Myers EW (1995) Combinatorial algorithms for DNA sequence assembly. Algorithmica 13: 7–51.
[5] Simpson JT, Wong K, Jackman SD et al. (2009) ABySS: a parallel assembler for short read sequence data. Genome Res 19: 1117–1123

Funding note: Open Access funding enabled and organized by Projekt DEAL.

Korrespondenzadresse:
Prof. Dr. Peter F. Stadler
Professur für Bioinformatik
Institut für Informatik
Universität Leipzig
Härtelstraße 16-18
D-04107 Leipzig
stadler@bioinf.uni-leipzig.de

AUTOREN



Thomas Gatter
2010–2015 Studium Kognitive Informatik und Naturwissenschaftliche Informatik an der Universität Bielefeld. Seit **2015** am Lehrstuhl Bioinformatik der Universität Leipzig. Dort **2022** Promotion unter Anleitung von Prof. Dr. P. F. Stadler.

Peter F. Stadler
Bis **1990** Studium der Chemie, Mathematik, Physik und Astronomie an der Universität Wien, Österreich. Dort **1990** Promotion bei Dr. P. Schuster in Chemie. Seit **1994** externes Fakultätsmitglied des Santa Fe Institutes, Santa Fe, USA. **1997–2002** Außerordentlicher Universitätsprofessor an der Universität Wien. Seit **2002** Professor für Bioinformatik an der Universität Leipzig.