Implementierung eines Tools für die referentielle Kompression von DNS-Sequenzen

Svenja Mehringer

October 27, 2015

Contents

1 Zusammenfassung

abstract...

2 Einleitung

Genomische DNS ist heutzutage Gegenstand vieler Forschungsarbeiten. Ein Genom umfasst das gesamte Erbgut eines Organismus' in Form einer einzelnen, fortlaufenden Sequenz der vier Basen Adenin, Cytosin, Guanin und Thymin. Die Sequenz kann als Text gespeichert und analysiert, wobei die Basen durch ihren jeweiligen Anfangsbuchstaben abgekürzt werden.

Die typische Form der Speicherung einer solchen DNS-Sequenz, ist das textbasierte FASTA Format. Es speichert die Sequenz, sowie einen vorangestellten Namen oder Kommentar, zeichenweise im ASCII-Code ab. Ein ASCII-kodierter Buchstabe verbraucht ein Byte. Vernachlässigt man den geringen Einfluss des Sequenznamen, ist die Größe der gespeicherten Datei damit proportional zur Länge der Sequenz. Am Beispiel eines Humangenoms mit einer Gesamtlänge von ca. 3 Milliarden Basenpaaren (haploid) ergibt sich eine Dateigröße von rund 3GB (Gigabyte). Untersucht man in einer Studie 1,000 Genome von Patienten, führen dadurch allein die reinen Sequenzdaten ohne Qualitätsinformation zu einem Speicheraufkommen von 3TB (Terrabyte). Obwohl bis jetzt erst wenige Studien eine solche Vielzahl an Sequenzierungen realisieren konnten, werden in Zukunft immer mehr Projekte wie das *1000 Genome Preject*[Consortium, 2012] erwartet. Der Grund dafür ist, dass es mit Hilfe von Next-Generation-Sequencing (NGS) in den

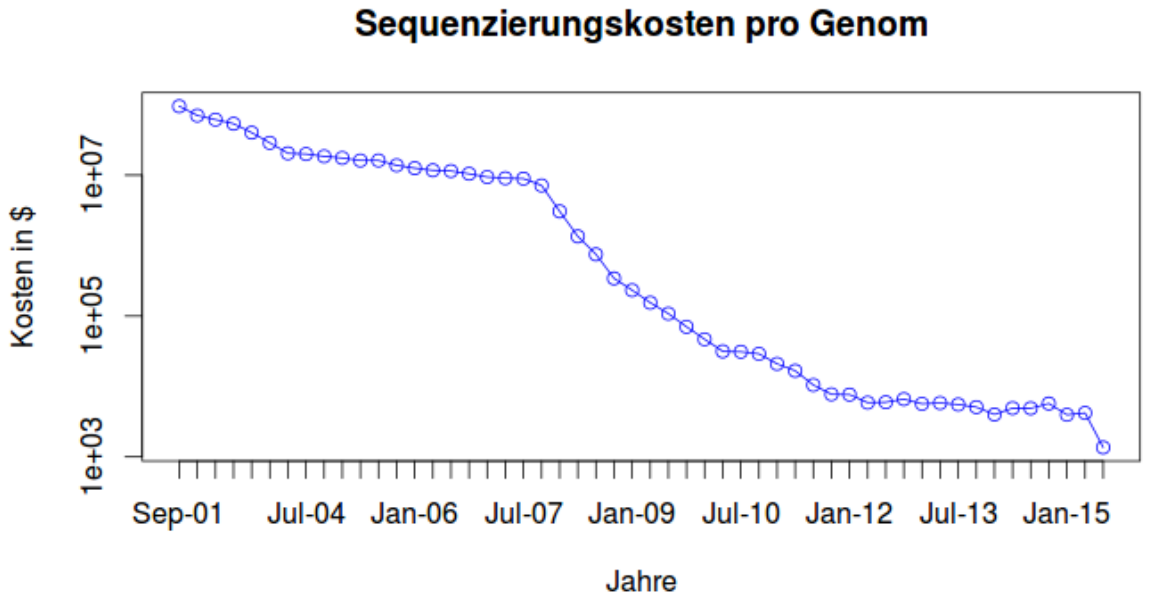

Figure 1: Trend der Genomesequenzierungskosten (NHGRI). Der starke Einbruch der Kosten im Jahr 2007 wird mit dem Übergang von Sanger-bastierten Sequenzierung zu 'second generation' oder 'next-generation'-Technologien erklärt [NHGRI, 2015].

letzten Jahren für viele Firmen und Institutionen immer erschwinglicher geworden ist, DNS-Proben zu sequenzieren (Grafik 1).

Zwar sinken ebenfalls die Kosten für die Speicherung, allerdings ist der Fortschritt in der Sequenzierungstechnologie dem der Hardware Industrie weit voraus [NHGRI, 2015]. Das Resultat sind immer größere Mengen an produzierten Daten. Sie stellen nicht nur für die langfristige und sichere Speicherung eine Herausforderung dar, sondern auch für deren Transfer und Analyse. Um diesen Problemen entgegenzuwirken, ist ein vielversprechender Lösungsansatz die Kompression der Daten. Da Sequenzdaten generell textbasiert gespeichert werden, finden die Grundlagen der Kompression von Texten Anwendung.

Ziel der Kompression ist es, eine Datei in einer Art und weise zu Verändern, bzw. zu Kodieren, sodass die dabei entstehende Version weniger Speicherplatz (in Byte gemessen) als vorher benötigt. Es wird grundsätzlich zwischen verlustfreier und verlustreicher Kompression unterschieden. Da bei DNS-Sequenzen schon kleinste Unterschiede von großer Bedeutung sein können (z.B. ein Single Nucleid Polymorphism (SNP)), wird eine verlustreiche Kompression vermieden und soll in dieser Arbeit nicht weiter thematisiert werden. Weiterhin kategorisiert man Algorithmen nach bit-manipulierender, statistischer und Wörterbuch-basierter Kompression.

Die Idee der Bit-Manipulation ist es zwei oder mehr Zeichen als mit Hilfe von Bit-Kodierung in einem einzigen Byte zu speichern. Durch verschiedene Kombinationen und limitierte Alphabete können so Kompressionsraten von 1:4 erreicht werden [Rani, 2015]. Statistische Methoden entwickeln Modelle, welche das nächste Zeichen voraussagen oder die abgeleiteten Häufigkeiten für eine sinnvolle Kodierung verwenden. Wörterbuch-basierte Methoden hingegen benutzen Substitution, um wiederholt auftretende Teilsequenzen durch einen Zeiger auf einen entsprechenden Wörterbucheintrag zu ersetzen.

Einer der am häufigsten als Basis benutzte Wörterbuch-basierten Algorithmen ist der Lempel-Ziv-Welch-Algorithmus (LZW) [Ziv and Lempel, 1977]. Er komprimiert verlustfrei jedes Format, da das dynamisches Wörterbuch erst bei Start des Verfahrens aufgebaut wird. Eine weitere, sehr wichtige Methode, ist die Huffman-Kodierung [Huffman, 2006]. Sie weist einer festen Anzahl von Symbolen eine Bit-Codierung (Codewort) mit variabler Länge zu. Dabei bekommen häufig auftretende Symbole die kürzesten Codewörter, um so die Eingabe mit optimaler (kleinster) Länge zu kodieren. Oft werden beide Algorithmen kombiniert ([Jean, 1992]).

Im Zuge der immer größeren Nachfrage nach speicherplatzeffizienten Formaten für DNS, wurde noch eine weitere Art der Kompression entwickelt.

Die *referentielle Kompression* geht auf die spezifischen Eigenschaften von von DNS

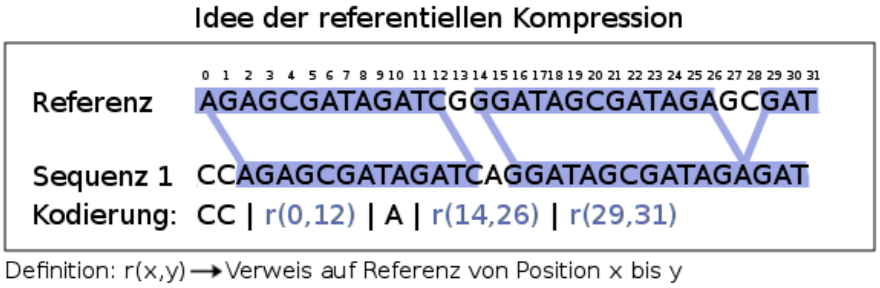

Figure 2: Idee der referentiellen Kompression.

ein. Ähnlich der Wörterbuch-basierten Variante, werden bei der referentiellen Kompression ebenfalls Zeiger verwendet. Sie verweisen allerdings auf Teile einer Referenzsequenz anstatt eines Wörterbucheintrags. Man macht sich hierbei die Tatsache zu Nutze, dass, besonders innerhalb einer Spezies, die Ähnlichkeit zwischen Sequenzen sehr hoch ist. Beispielweise unterscheiden sich zwei zufällige Humangenome im Durchschnitt um nur ca. 0,1% [Venter et al., 2001]. Folglich kann man 99,9% der Sequenz als Zeiger auf eine Referenz kodieren und sollte somit eine sehr hohe Kompression erreichen können. Der Nachteil dieses Verfahrens ist die starke Abhängigkeit von der verwendeten Referenz. Vergleicht man ein Humangenom mit z.B. einem Mausgenom würde man nur eine geringe Kompression erreichen können. [Wandelt et al., 2014] schätzt die Länge von gleichen Teilsequenzen zwischen Maus und Mensch auf nur ca. 20-25 Basen, wodurch eine Kodierung ähnlich der Grafik 2, mehr Speicherplatz als vorher verbrauchen würde. Folglich ist die referentielle Kompression weitestgehend für sogenannte Resequenzierungs-Projekte (*resequencing*) geeignet, welche sich auf die Sequenzierung von Genomen innerhalb einer Spezies konzentrieren.

Es existieren bereits einige Algorithmen und Programme mit verschiedenen Ansätzen für die referentiellen Kompression von DNS. Methoden die auf die Kompression von Rohdaten in Form von *reads* abzielen ([Deorowicz and Grabowski, 2011a]) werden vernachlässigt, da sie trotz ähnlicher Problemstellung nur sehr gering auf das Ziel dieser Arbeit übertragbar sind. Der Fokus liegt auf Algorithmen für die Kompression von ganzen Genomen bzw. Genomsammlungen. Erste erfolgreiche Resultate konnten 2008 von [Christley et al., 2009] verzeichnet werden: Das James Watson Genom wurde von 3.1 GB auf lediglich 4.1 MB komprimiert. Allerdings wurde hierbei das Wissen von genauen Unterschieden, namentlich SNP's und kurze Indel's, zum Referenzgenom vorausgesetzt. Das heißt, die Eingabe für den Algorithmus ist das durch *Variant Calling* vorprozessierte VCF-Format (*Varaint Calling Format*). Ebenfalls auf VCF-Basis erzielte

[Deorowicz et al., 2013] die bisher höchste Kompressionsrate von ca. 1:15000, auf den Daten des *1000 Genome Projects*.

Das Problem des Variant Calling ist jedoch äußerst schwierig, denn es erfordert ein perfektes globales Alignment zum entsprechenden Referenzgenom. Häufig kann diese nötige Voraussetzung nicht erfüllt werden. Aus diesem Grund muss zu den Algorithmen, welche auf reinen Sequenzdaten (FASTA-Format oder ähnliches) arbeiten können, unterschieden werden. Eine Übersicht dieser Algorithmen bis 2013 ist in Tabelle 1 dargstellt [Deorowicz and Grabowski, 2013].

Table 1: Gekürzte Originaltabelle [Deorowicz and Grabowski, 2013] zu bestehenden Kompressionsalgorithmen.

Name	**Compression Speed**	**Compression Ratio**	**Methods**
gzip	low	very low	LZ, Huf
bzip2	low	very low	BWT, Huf
7z	low	high	LZ, AC
GReEn	high	high	M.Models, AC
GRS	moderate	high	LCS, Huf
RLZ	moderate	high	LZ, Gol
ABRC	high	very high	LZ, Huf
GDC	high	very high	LZ, Huf

Abkürzungen: AC-arithmetic coding, Golomb (a statistical coding method), Huf-Huffman, LCS-longest common subsequence, LZ-an algorithm of the Lempel-Ziv familiy, M.models-markov models.

Die Programme gzip, bzip2 und 7z (Referenzen notwendig?), sind allgmeine, textbasierte Kompressoren, deren Kompression, unter Ausnahme von 7z, sehr gering eingeschätzt wird. Tatsächlich erzielt das sehr verbreitete Programm gzip lediglich eine Rate von ca. 1:4 (???nachgucken referenz). Dagegen erzielen die speziell für genomische DNS entworfenen Programme deutlich bessere Ergebnisse. Die zwei Algorithmen GRS [Wang and Zhang, 2011] und GreEn [Pinho et al., 2012] verzeichnen Kompressionsraten für einzelne Humangenome von ca. 1:200. Während GRS als Basis das Prinzip der *Longest Common Subsequence* mit anschließender Huffman-Codierung benutzt, verwendet GreEn einen statistischen Ansatz mit Hilfe Markov-Modellen. Ein Ansatz mittels der Lempel-Ziv Methode LZ77 wird von [Kuruppu et al., 2011b] vorgeschlagen (RLZ) und erreicht ähnliche Kompressionraten von 1:250. Ein weiterer Kompressionsalgorithmus (ABRC) wurde von Wandelt und Leser angeboten, der einen komprimierten Suffixbaum für die Suche von Gemeinsamkeiten blockweise aufbaut [Wandelt and Leser, 2012]. Durch anschließende Lempel-Ziv Transformation und Huffman-Kodierung erreicht das Programm ABRC Kompressionsraten von 1:400 auf Daten des *1000 Genome Projects*. Eine noch höhere Kopres-

sionsrate erreichte der Algorithmus GDC (Rate ca. 1:600) bzw. GDC Ultra (Rate ca. 1:2000) [Deorowicz and Grabowski, 2011b], gefolgt von GDC 2 (Rate ca. 1:9000) [Deorowicz et al., 2015]. Ähnlich dem Programm RLZ basiert der GDC/GDC-Ultra/GDC 2 Algorithmus auf LZ77, allerdings mit einem wichtigen Unterschied zu anderen Methoden. Alle vorherigen Methoden komprimieren auch bei einer Sammlung von Genomen jedes einzelne unabhängig. Im Gegensatz dazu, werden bei GDC mehrere Sequenzen als Referenz genutzt. Die zusätzliche Zusammenführung von Informationen zwischen den zu komprimierenden Sequenzen, stellt sich als sehr vielversprechende Idee da.

Über die Anstrengungen zur Reduzierung des Speicherplatzes und des Datentransfers hinaus, muss auch die zeitaufwändige Analyse solcher Datenmengen beachtet werden. Alle bisher angesprochenen Algorithmen, dienen lediglich der Kompression von Daten hinsichtlich ihrer Speicherung. Will man die Daten analysieren, müssen sie wieder dekodiert werden. Die Sequenzen bzw. Genome würden zudem einzeln und nacheinander untersucht werden. Man verwirft dabei die vorher zur Kompression benutzte Information der hohen Ähnlichkeit zwischen Sequenzen, was zur überflüssigen Analyse von redundanten Teilsequenzen führt. Es liegt folglich nahe, die referentielle Kompression auch auf die Analyse zu erweitern. Diese Problemstellung wird von [Loh et al., 2012] als *compressive genomics* benannt.

Die erste Möglichkeit, verschiedene sequentielle Algorithmen auf mehreren Sequenzen gleichzeitig laufen zu lassen, wird von [Rahn et al., 2014] vorgestellt. Durch eine unterliegende Datenstruktur, als *journaled string tree* (JST) bezeichnet, werden Sequenzen als eine Menge von Varianten, basierend auf einem gemeinsamen Koordinatensystem repräsentiert. Eine einzelne Sequenz kann wiederum aus ihren zugehörigen Varianten, als referentiell komprimierter String (*journaled string*), generiert werden. Durch die referentielle Kompression von Sequenzen zu Varianten, werden gemeinsame Bereiche nur einmal analysiert und die Laufzeit wird um den Faktor 115 Beschleunigt [Rahn et al., 2014].

Diese Arbeit beschäftigt sich mit der Implementierung eines C++ Programms innerhalb der SeqAn Bibliothek [Döring et al., 2008], welches der verlustfreien referentiellen Kompression von DNS-Sequenzen dienen soll. Sie steht in engem Zusammenhang mit dem SeqAn Projekt des JST [Rahn et al., 2014] und ist auf dessen Kompatibilität abgestimmt.

Derzeit können zur Verwendung des JST nur Dateien im VCF-Format benutzt werden. Das Hauptinteresse liegt daher nicht nur in der bestmöglichen Speicherplatzreduzierung einer Datei, sondern in der möglichst effizienten Transformation und Speicherung der Eingabedaten (FASTA-Format o.ä.) in einem für den JST weiterverwendbares Format.

Der vorgestellte Algorithmus soll lediglich die Unterschiede zur Referenz kodieren und dabei die Idee der Zusammenführung von Informationen zwischen den zu komprimierenden Sequenzen von [Deorowicz and Grabowski, 2011b, Deorowicz et al., 2015] aufgreifen. Da viele Autoren auf die Abhängigkeit der Kompression von der verwendeten Referenzsequenz hindeuten [Deorowicz and Grabowski, 2013, Wandelt et al., 2014, Kuruppu et al., 2011a], ist eine Besonderheit gegenüber anderen Arbeiten, dass die verwendete Referenzsequenz variabel sein soll. Sie wird bei jeder Eingabe neu angelegt und kann fortlaufend verändert werden, mit dem Ziel, das Speicherformat weiter zu optimieren.

In der folgenden Sektion wird eine detaillierte Beschreibung des Algorithmus' und dessen Implementation gegeben. Anschließend werden Ergebnisse präsentiert, die bei Anwendung des Algorithmus' auf Realdaten gesammelt wurden. Nach der Diskussion der Ergebnisse, soll eine kurze Zusammenfassung und Einschätzung die Arbeit abschließen.

3 Methoden

3.1 Definitionen und allgemeine Notation

Ein *string* $s = s_0....s_{n-1}$, $s_i \in \sum$, ist eine Sequenz der Länge $|s| = n$, auf Basis des Alphabets $\sum$. Der leere *string* wird mit ϵ benannt. Ein Teil*string* von s, von Position i bis j, wird als $s[i,j[= s_i....s_{j-1}$ angegeben. Falls $i \geq j$, wird der ϵ zurückgegeben. Der Spezialfall $s[i, i+1[$ wird zu $s[i]$ abgekürzt.

Ein *journaled string* gleicht einem normalen *string* und wird als solcher behandelt. Der Unterschied liegt nur in der Implementation. Ein *journaled string* besitzt einen Zeiger auf eine Referenzsequenz, sowie eine Menge an Veränderungen bzw. Varianten. Varianten werden mittels eines Binären Baums (*journal tree*) gespeichert [Rahn et al., 2014].

Ein *journal entry* ist ein Eintrag im *journal tree* eines *journaled string* der eine Veränderung der Referenzsequenz beschreibt. Veränderungen bzw. Varianten können eine Insertion (Hinzufügen von Zeichen) oder eine Deletion (Löschen von Zeichen) beschreiben [Rahn et al., 2014].

Ein *seed* ist ein Tupel (b_r, b_s, e_r, e_s) und beschreibt einen gemeinsamen Teilstring von zwei Strings r und s. Es gilt: $r[b_r, e_r] = s[b_s, e_s]$. Die Länge $|seed|$ wird definiert als $b_r - e_r = b_s - e_s$.

Die Menge $S(p)$ enthält alle *seeds* die an Position p in der Query-Sequenz s bei der

Suche gegen eine Referenz r gefunden wurden: $seed = (b_r, b_s, e_r, e_s) \in S(p)$ falls $b_s = p$.

Ein Δ-*Event* $e = (pos, type, cov, ins, del)$ repräsentiert eine Veränderung einer Referenzsequenz r and der Position pos. Es kann einen von vier Typen annehmen ($type$): SNP, Deletion(DEL), Insertion(INS) und strukturelle Variation(SV). cov ist die *coverage* des Events (siehe Definition von *coverage*). ins bezeichnet einen String der an Stelle pos eingefügt wird und del die Länge des Teilstrings $r[pos, pos + del]$ der gelöscht wird.

Die *coverage* eines Δ-Events e bezeichnet die Information, auf wie vielen und welchen von n Eingabesequenzen e vorkommt. Hierfür wird ein binärer *string* cov der Länge n benutzt, für den gilt: $cov[i] = 1$ falls e auf Sequenz i zu finden ist, $cov[i] = 0$ sonst. Die Negation von cov wird wie folgt definiert: $\neg cov = \neg cov[i] \; \forall \; i = 0..n$

Eine *dependent region* ist eine Gruppe von Δ-Events, in der jedes Δ-Event von mindestens einem anderen Δ-Event der Gruppe abhängig ist. Definition von Abhängigkeit: $e_1 = (pos_1, type_1, seq_1, ins_1, del_1)$, $e_2 = (pos_2, type_2, seq_2, ins_2, del_2)$, o.B.d.A gilt $pos_1 \leq pos_2$. Dann ist e_1 abhängig von e_2 falls: $(pos_1 + del) > pos_2$.

3.2 Konzept

Der vorgestellte Algorithmus wird grundlegend in zwei Abschnitte unterteilt: Die vorläufige Kompression und deren Erweiterung.

Zunächst werden die zu komprimierenden Eingabesequenzen im FASTA-Format eingelesen und anschließend gegen eine Referenz aligniert. Aus dem resultierenden Alignment werden nun die Unterschiede zwischen der Sequenz und ihrer Referenz als Δ-Events gespeichert. Das Ergebnis ist eine referentielle Kompression der Eingabe in Form der Kodierung von Sequenzen in Unterschieden (bzw. Varianten).

Um die Kompression zu verbessern, werden im Anschluss zusätzliche Erweiterungen vorgenommen. Hierbei werden zwei Ansätze verfolgt: (1) Die Verbesserung der Speichereffizienz und (2) die Verringerung des potentiellen Analyseaufwands. Ersteres wird durch die Reduzierung der Anzahl an Δ-Events erreicht. Der zweite Ansatz bezieht sich auf die Integration des hier vorgestellten Algorithmus' in die Arbeit von [Rahn et al., 2014]: Die komprimierten Daten könnten weiterführend in einen JST überführt werden. Mit Hilfe des JST's können Sequenzen in Form von *journal strings* effizient untersucht werden. Die Schnelligkeit des Algorithmus' hängt dabei von der Anzahl an *journal entries* ab. Ziel ist es daher, durch Manipulation der Referenzsequenz, die Anzahl an potentiellen *journal entries* zu verringern.

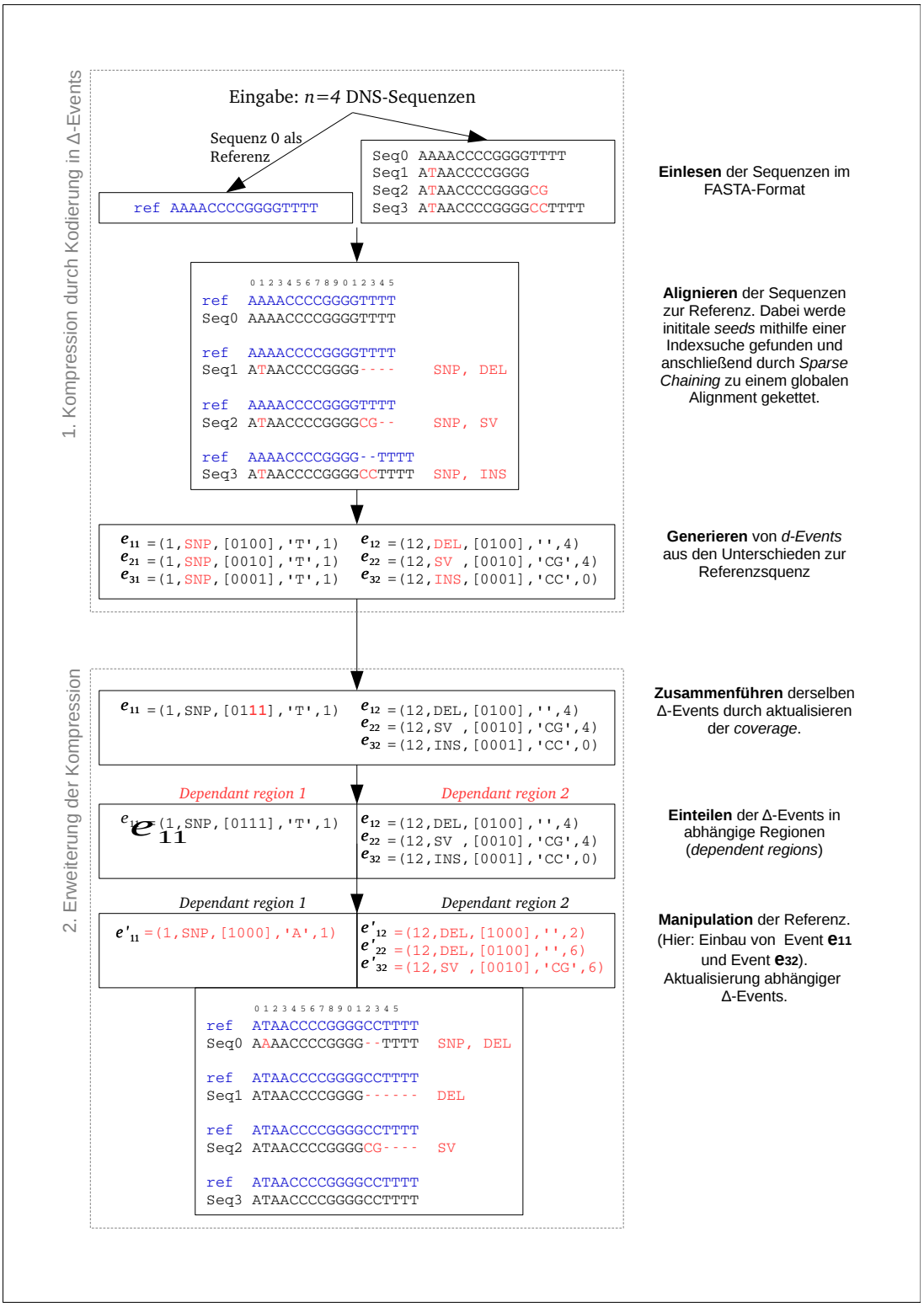

Figure 3: Konzept des Kompressionsalgorithmus' anhand eines Beispiels.

Zur Visualisierung des Algorithmus' ist ein Beispiel mit Zwischenergebnissen in Grafik 3 dargelegt.

3.3 Implementierung der Kompression

3.3.1 Indexsuche

Die Suche nach Gemeinsamkeiten zwischen einer Sequenz und ihrer Referenz ist eines der Schlüsselprobleme bei der referentiellen Kompression. In dem hier vorgestellten Algorithmus wird dazu zunächst ein q-Gram-Index über der Referenz aufgebaut. Die Wahl des Parameters q wird dem Benutzer überlassen. Allerdings wird empfohlen, das q bei genomischer DNS über 15 liegen sollte, damit zufällige q-mere vermieden werden [Wandelt et al., 2014]. Da bei großen Werten für q, auf einem Alphabet $\sum$, die Anzahl an möglichen q-meren exponentiell steigt ($|\sum|^q$), wird ab einem Wert von

$$q > log_{|\sum|}(RAM\ Speicher\ in\ Byte) \tag{1}$$

das *OpenAdressing()* (SeqAn [Döring et al., 2008]) verwendet. Beim *OprenAdressing()* werden nur solche q-mere gespeichert, die auch tatsächlich in der indizierten Sequenz zu finden sind.

Wird jede der m Eingabesequenzen, mit einer jeweiligen Durchschnittslänge n, sequentiell nach gemeinsamen q-meren durchsucht, ergibt sich trotz Zugriff auf den q-Gram-Index in $O(1)$ eine Laufzeit von $O(n * m)$. Aus diesem Grund wird die Suche um einige Faktoren erweitert, die auf die spezifischen Eigenschaften von DNS eingehen.

3.3.2 Verbesserung der Suche

Die Indexsuche wird um folgende vier Punkte erweitert:

(1) *Ignorieren von Repeats.* Genomische DNS enthält meist sehr viele repititive Abschnitte. Umfasst ein q-gram einen Repeat würde er die Suche enorm verlangsamen, denn pro q-gram in der durchsuchten Sequenz würden viele tausende in der Referenz gefunden werden. Solche q-gramme werden als Treffer ignoriert, indem jeder gefundene seed mittels seines Vorgängers auf einen Repeat getestet wird (siehe formale Definition in Sektion 3.1). Ähnlich verhält es sich auch, falls das DNS-Alphabet um den Buchstaben 'N' erweitert ist und ein q-gram ausschließlich aus 'N' besteht. Sie werden ebenfalls ignoriert.

(2) *Iterativer Schritt*(Grafik 4). Die Schnelligkeit kann auf kostet der Sensitivität erhöht werden, wenn der Benutzer einen relativ hohen Wert für q wählt. Um dennoch eine ausreichende Genauigkeit zu gewährleisten, können im Programmaufruf ein oder mehrere

iterative Schritte mit niedrigeren Werten für q spezifiziert werden. Der Algorithmus durchläuft dann eine initiale Suche mit dem ersten q-Wert, speichert die resultierende globale Kette und starten dann in den Bereichen zwischen den ausgewählten *seeds* der Kette einen erneute Suche mit dem nächsten Wert für q. Diese Methode ist der des LAGAN Algorithmus für globale Alignments nachempfunden [Brudno et al., 2003].

(3) *Initiale Suche mit dynamischen Offset.* Diese Erweiterung basiert auf der Erwartung, dass bei sehr ähnlichen Sequenzen lange gleiche Abschnitte zwischen einer Sequenz und ihrer Referenz existieren. Anstatt eine sequentielle Suche (Offset=1 oder q) durchzuführen, wird in jedem Schritt der Suche ein neuer Offset bestimmt. Hierfür wird an einer Position p in der Sequenz jeder gefundene $seed_i(p)$ (jedes initiale q-gram) zunächst erweitert. Die Erweiterung, bzw. Verlängerung des *seeds*, erfolgt solange in beide Richtungen, bis ein *mismatch* (Unterschied) auftritt. Anschließend wird jedem Seed ein Score

$$Score(seed) = |seed_i(p)| + diagonal(seed_i(p)) \tag{2}$$

zugewiesen, wobei $diagonal(seed_i(p))$ die Abweichung zur Hauptdiagonalen angibt. Die Länge des *seeds* mit dem besten Score wird als neuer Offset gesetzt und die Suche auf der Sequenz wird an Position $p + offset$ fortgeführt. Etwaige Verluste an Sensitivität können durch iterative Schritte wieder ausgeglichen werden.

(4) *Parallelisierung der initialen Suche.* Um die Suche abermals zu beschleunigen wird die initiale Suche parallelisiert. In Anlehnung an [Rani, 2015], wird die zu untersuchende Sequenz in Blöcke aufgeteilt. Jeder Block wird einzeln nach Gemeinsamkeiten untersucht aber im Gegensatz zu [Rani, 2015], werden die Blöcke vor der weiteren Analyse (und Kodierung) wieder zusammengeführt. Somit entsteht kein Informationsverlust und die Ergebnisse entsprechen denen ohne Parallelisierung.

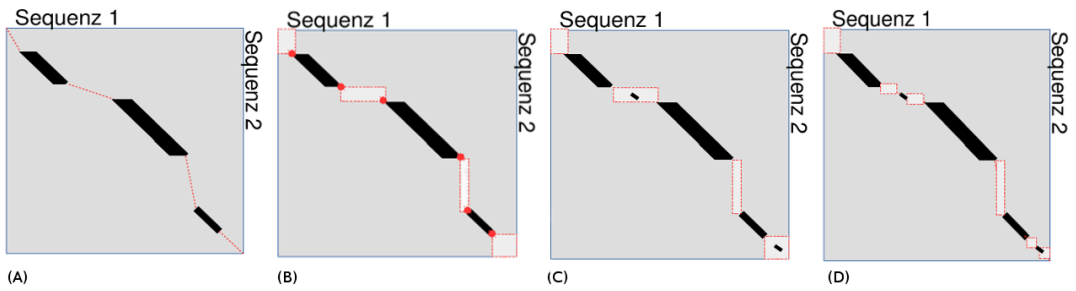

Figure 4: Iterative Schritte in der Indexsuche. In (A) sind in der initialen Suche drei große *seeds* gefunden und für die globale Kette selektiert worden. Wie in (B) dargestellt, ergeben sich dadurch Bereiche zwischen den *seeds*; Jeweils von der rechten unteren Ecke des vorhergehenden *seeds* zu linken oberen Ecke des darauffolgenden. In (C) sind dadurch kleinere *seeds* gefunden worden, die zu neuen Bereichen wie in (D) führen.

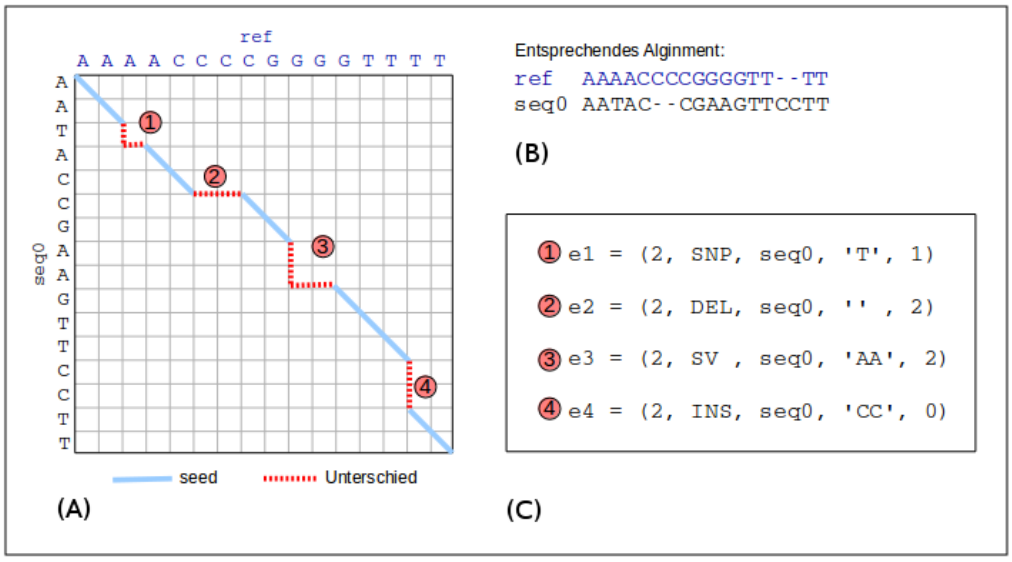

Figure 5: *Generierung von Δ-Events.* Abbildung (A) visualisiert das Ergebnis der Indexsuche. Die Bereiche zwischen den *seeds* bilden die Vorlage für die Δ-Events. (B) zeigt die entsprechende Alignment-Repräsentation und (C) die generierten Δ-Events.

3.3.3 Reduktion zu einer globalen Kette

Für die Reduktion einer Menge von *seeds* wird der *Sparse Chaining* Algorithmus von [Gusfield, 1997] verwendet. Er konstruiert in $O(r*log(r))$ eine optimale globale Kette von *seeds*, denen ein bestimmtes Gewicht (Score) zugewiesen wurde. Der Algorithmus ist bereits in der SeqAn-Bibliothek [Döring et al., 2008] enthalten und benutzt dort als Gewichtung die Länge des *seeds*. Die Gewichtung ist geeignet, da längere Gemeinsamkeiten später eine geringere Anzahl an Δ-Events bedeuten und somit zu bevorzugen sind.

Um die Laufzeit des Programmes zu verbessern, ist dem Benutzer möglich für den initialen Suchlauf (und nur für diesen) eine Mindestlänge der *seeds* zu spezifizieren. Liegt ein gefundener *seed* bei der Suche unterhalb dieser Länge wird er wieder verworfen. Mit dieser Einschränkung lässt sich die Anzahl an *seeds* kontrollieren, um den *Chaining*-Algorithmus zu entlasten. Auch hier wird ein drohender Verlust an Sensitivität, durch ausscheiden einiger mitunter wichtigen *seeds*, mit Hilfe des iterativen Schritts ausgeglichen.

3.3.4 Generierung von Δ-Events

Steht die globale Kette fest, repräsentiert sie ein globales Alignment zwischen der untersuchten Sequenz und der Referenz. Jeder *seed* beschreibt dabei eine gemeinsame Teilsequenz, wohingegen die Bereiche dazwischen Unterschiede bzw. Varianten darstellen. Jeder dieser Zwischenbereiche wird zu einem einzelnen Δ-Event, indem die zwei angrenzenden *seeds* mittels Manhattan-Distanz verbunden werden. Die Position eines Δ-Events wird

dabei als absolute Position in der Referenz gespeichert (Abbildung 5).

Formale Definition. Man betrachte die Analyse der i-ten Sequenz s (aus insgesamt n Sequenzen) gegen die Referenz r. $seed_1 = (b_r^1, b_s^1, e_r^1, e_s^1)$ und $seed_2 = (b_r^2, b_s^2, e_r^2, e_s^2)$ liegen hintereinander in der globalen Kette. O.B.d.A liegt $seed_1$ über und links von $seed_2$. Dann werden aus dem Bereich zwischen $seed_1$ und $seed_2$ folgende Attribute für das Event e

$$e = (pos,\ type,\ cov,\ ins,\ del) \tag{3}$$

definiert:

$$pos = e_r^1 \tag{4}$$

$$type = \begin{cases} SNP & falls\ |ins| = 1\ und\ del = 1, \\ DEL & falls\ |ins| = 0, \\ INS & falls\ del = 0, \\ SV & sonst \end{cases} \tag{5}$$

$$\forall j = 0..n \quad cov[j] = \begin{cases} 1 & falls\ j = i, \\ 0 & sonst \end{cases} \tag{6}$$

$$ins = s[e_s^1, b_s^2] \tag{7}$$

$$del = b_r^2 - e_r^1 \tag{8}$$

3.4 Erweiterung der Kompression

3.4.1 Verbesserung der Speichereffizienz

Die Idee beruht auf der Erwartung, dass auf verschiedenen Sequenzen die selben Variationen zu finden sind. Beispielweise gibt es häufig auftretende *Single Nucleotide Polymorphims* (SNP) innerhalb bestimmter menschlichen Population [Choudhury et al., 2014]. Anstatt folglich alle Variationen einzeln zu speichern, können gleiche zusammengefasst werden.

Hierzu werden alle gesammelten Δ-Events (aller Sequenzen) durchlaufen. Falls sich zwei Events in ihrer Position, ihres Typs und den Werten für *del* und *ins* exakt gleichen werden sie zusammengeführt. Es muss dabei lediglich die *coverage (cov)* des einen Events um die entsprechende Sequenzinformation des anderen aktualisiert werden. Letzteres Event kann nun gelöscht werden, da es redundante Informationen trägt. Der binäre *cov*-Vektor gewährleistet hierbei eine effiziente Speicherung und eine einfache Handhabung mittels logischer Operationen.

3.4.2 Verringerung des potentiellen Analyseaufwands

In der Analyse mittels eines *journal string tree* (JST) benötigt man für jede zu untersuchende Eingabesequenz einen *journal string*. Jeder *journal string* muss dabei auf die selbe Referenz verweisen und durch die Varianten zu jener Referenz (*journal entries*) seine entsprechende Ausgangssequenz darstellen. Beide Voraussetzungen sind beim Ergebnis des vorgestellten Algorithmus erfüllt: Die produzierten Δ-Events haben die selbe Referenz und gleichen in ihrer Struktur den Varianten eines *journal strings* (SNP, Deletion, Insertion und Strukturelle Variation). Es besteht folglich die Möglichkeit Δ-Events in *journal strings* zu überführen. Bei der Überführung kann festgehalten werden, dass pro Deletion ein zusätzlicher *journal entry* generiert werden würde und bei allen anderen Eventtypen zwei zusätzliche *journal entries* (siehe Zusatzmaterial für detaillierte Beschreibung). Wie bereits erwähnt, hängt die Effizienz der Analyse von der Gesamtanzahl an *journal entries* ab und das Ziel dieser Erweiterung ist es daher die Anzahl an potentiellen *journal entries* nach einer Überführung zu minimieren. Die Umsetzung erfolgt durch eine Manipulation der Referenzsequenz.

Zur Verdeutlichung soll folgendes Beispiel dienen: Angenommen die ersten acht von zehn Sequenzen unterscheiden sich zur Referenz r durch den selben SNP, $A \rightarrow C$, an Position 16. Nach Zusammenführung der einzelnen SNP-Events bliebe das Δ-Event $e = (16,\ SNP,\ cov = 1111111100,\ 'C',\ 1)$. Pro Sequenz würden zwei *journal entries* für den jeweiligen *journal string* benötigt werden: $8 * 2 = 16$ *journal entries*. Man kann nun eine Veränderung der Referenz an Position 16 von A auf C in Erwägung ziehen. Durch diese Veränderung würde aus e das Event $\tilde{e} = (16,\ SNP,\ cov = 0000000011,\ 'A',\ 1)$ entstehen, welches zwar den selben Speicherplatz benötigt, bei einer Überführung aber nur $2 * 2 = 4$ *journal entries* erfordern würde.

Um diese Idee generisch auf alle Δ-Events anwenden zu können, müssen diese zuerst in Gruppen, sogenannte *dependent regions* (siehe Defintionen), eingeteilt werden. Diese Einteilung ist notwendig, da bei einer Änderung der Referenz bezüglich e, die von e abhängigen Events gesondert betrachtet und geändert werden müssen. Eine fehlerfreie Kodierung ist sonst nicht mehr gewährleistet.

Die Zuteilung verläuft wie folgt: Gegeben sei Δ-Event e_j in einer aufsteigend sortierten Liste von Δ-Events, sowie $e_{j-i} \in G$. Ist e_j abhängig von e_{j-1}, wird e_j zu G hinzugefügt. Andernfalls ist G vollständig und es wird eine neue Gruppe mit e_j als erstem Element initiiert.

Alle weiteren Schritte werden gruppenweise durchgeführt.

Zunächst wird für alle Δ-Events einer Gruppe ein Score berechnet. Der Score eines Events e wird definiert als die Anzahl an *journal entries*, welche durch den Einbau von e in die Referenz hinzukommen oder wegfallen würden. Ein negativer Score drückt daher aus, dass eine Änderung der Referenz durch den Einbau von e von Vorteil wäre, da die Anzahl an *journal entries* verringert werden würde.

Das Δ-Event mit dem besten (am meisten negativen) Score wird nun in die Referenz eingebaut. Der Einbau in die Referenz bedeutet, dass die Änderungen, die e beschreibt, in die Referenzsequenz r übernommen werden. Es wird die Teilsequenz $r[pos_e,\ pos_e + del_e]$ aus r gelöscht und die Teilsequenz ins_e an Position pos_e eingefügt. Aufgrund der Änderung der Referenz sind nun die übrigen Kodierungen der Sequenzen falsch und müssen aktualisiert werden. Bei einer Aktualisierung wird zwischen Sequenzen, welche abhängige Events zu e besitzen und denen die unabhängig sind, unterschieden.

Für die unabhängigen oder nur indirekt abhängigen Sequenzen muss lediglich ein neues Event $\tilde{e}$ erstellt werden. $\tilde{e}$ kann als gegenteiliges Event zu e gesehen werden und wird definiert als:

$$\tilde{e} = (pos_e,\ \neg type_e,\ \neg cov_e,\ r[pos_e,\ pos_e + del_e],\ |ins_e|) \tag{9}$$

wobei

$$\neg type_e = \begin{cases} DEL & falls\ type_e = INS, \\ INS & falls\ type_e = DEL, \\ type_e & sonst \end{cases} \tag{10}$$

Für Sequenzen mit von e abhängigen Events muss jedes Event einzeln aktualisiert werden. Gegeben das in die Referenz einzubauende Event e und ein von e abhängiges Event e', dann wird e' wie folgt aktualisiert:
Falls $pos_e \leq pos_{e'}$

$$k_e = (pos_{e'},\ \ *type,\ cov_{e'}, ...) \tag{11}$$

Falls $pos_e \geq pos_{e'}$

$$k_e = (pos_{e'},\ \ *type,\ cov_{e'}, ...) \tag{12}$$

* wobei jeweils *type* anschließend wie Gleichung (6) ermittelt wird.

Nach erfolgreiche Manipulation der Referenz werden alle Schritte für die aktualisierten Events der Gruppe wiederholt. Falls kein Δ-Event mehr für vorteilhaft befunden wird, wird mit der nächsten Gruppe fortgefahren.

4 Ergebnisse

Die Auswertung wurde auf einem internen Server mit Linux 3.13.0 System mit .... two Intel®Core™i5-3317U CPU's at 1.70GHz (a total of 2 physical and 2 virtual cores) and 5.8GB of RAM.

Das Programm wurde auf Realdaten getestet. Der erste Datensatz besteht aus bis zu 200 Versionen des menschlichen Chromosoms 22 erhalten vom *1000 Genome Project* [Consortium, 2012]. Alle Varianten der 1000 Individuen sind einer VCF-Datei gespeichert und mussten für die Analyse zurück in einzelne FASTA-Dateien überführt werden. Hierfür wurde vcf-subset aus dem Paket vcf-tools benutzt, um den VCF-Datensatz auf einzelne Individuen zu reduzieren. Anschließend wurde mittels GATK -Genome..bla.bla aus dem VCF-Format und der im Projekt angegebenen Referenz die entsprechende FASTA-Datei generiert. Der zweite Datensatz besteht aus 20 E.Coli Genomen bereitgestellt durch EcoliWiki.net [PortEco, 2010].

Der Kompressionsalgorithmus besitzt folgende Parametereinstellungen:

Kürzel	Parameter	Beschreibung
-h	*- -help*	Displays this help message.
-v	*- -version*	Display version information.
-i	*- -input_file*	Path to the input file. Valid filetype: FASTA.
-q	*- -q_gram*	Size of the q-gram(-index) when finding seeds. Define multiple times for iterative search.
-t	*- -threads*	The number of threads that can be used.
-mss	*- -maximum_seed_size*	The threshold in the first seeding step for the maximum seed size.
-ev	*- -evaluation*	If set, this will trigger an evaluation of the performed algorithm.

Table 2: *Parameter des SeqAn Tools für referentielle Kompression.*

4.1 Evaluation der Indexsuche

Das durch die Indexsuche produzierte Alignment ist kein optimales globales Alignment aber es sollte sich diesem idealerweise annähern.

Abbildung 6 zeigt den Vergleich von im Voraus bekannten VCF-Einträgen zu den bei der Indexsuche gefundenen *seeds*. Es ist eine starke Korrelation der jeweiligen Graphen erkennbar. Die Ergebnisse in Abbildung 6 wurden mit dem Parametern $q = 20$ (initial)

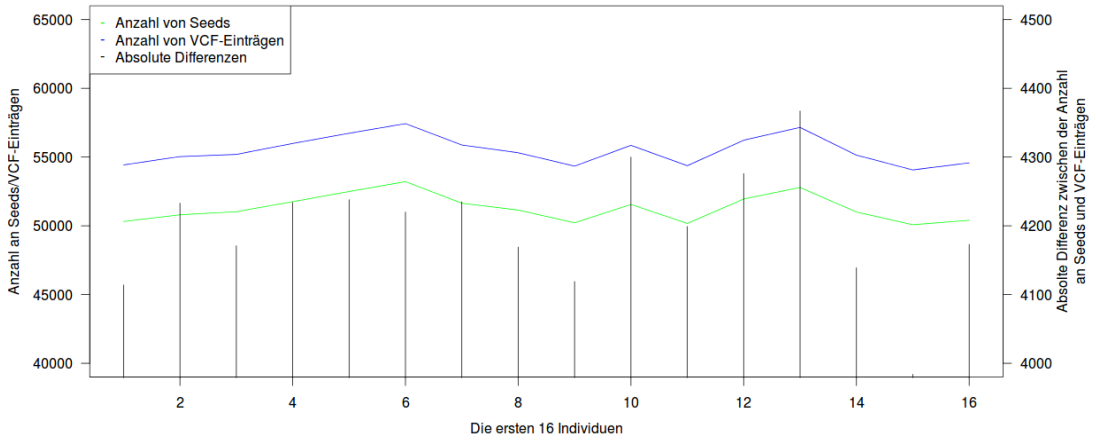

Figure 6: *Vergleich der Anzahl von Seeds gegen die zugrunde liegenden VCF-Einträge.* Der blaue Graph zeigt die Anzahl an Varianten der Referenzdatei des 1000 Genome Project. Der grüne Graph zeigt die entsprechende Anzahl an gefundenen Seeds als Ergebnis der Indexsuche. Die Indexsuche wurde mit dem Parameter $q = 20$ und im Iterativen Schritt mit $q = 15$ auf 16 Kernen durchgeführt, mit einer initialen Mindestgröße für Seeds von 100 . Die hinzugefügten Balken zeigen die absolute Differenz zwischen beiden Graphen.

Table 3: *Übersicht der Kompressionsergebnisse von Chromosom 22.*

#Seq	Rohdaten	Kompression auf Datei-Ebene			Kompression auf JS-Ebene		
	[MB]	Vorher[MB]	Nachher[MB]	**Ratio**	Vorher[MB]	Nachher[MB]	**Ratio**
10	573.72	7.24	2.29	**250.16**	12.93	7.28	**78.82**
50	2659.97	36.08	7.02	**378.94**	64.19	34.83	**76.38**
100	5215.63	71.68	12.77	**408.56**	127.49	69.03	**75.55**
200	10431.27	144.08	24.69	**422.50**	256.47	139.53	**74.76**

Anmerkung. Die Bezeichnung *Vorher* bezieht sich auf das Ergebnis vor den in Sektion 3.4 beschriebenen Verbesserungen: Auf Datei-Ebene vor der Zusammenführung von Events und auf Journaled String (JS)-Ebene vor der Prozessierung von Events. Das Verhältnis (Ratio) bezieht sich jeweils auf die *'Nachher'*-Werte zu den entsprechenden Rohdaten.

und $q = 15$ (iterativer Schritt) durchgeführt. Das bedeutet, dass Varianten die weniger als 15 Basen von einander entfernt liegen, nicht separat erkannt werden und folglich als eine einzelne große Variante gespeichert werden. Im Durchschnitt der sechzehn untersuchten Individuen, haben 3792 Varianten einen geringeren Abstand als 15 Basen zu Ihrem Vorgänger. Diese Zahl liegt nahe der absoluten Differenzen zwischen der Anzahl an VCF-Einträgen und der Anzahl von *seeds*.

4.2 Leistung beim menschlichen Chromosoms 22

Der Kompressionsalgorithmus wurde auf einer verschiedenen Anzahl an menschlichen Chromosomen 22 getestet. Für alle 4 Datensätze (Anzahl 10, 50, 100 und 200 Sequenzen)

Table 4: *Übersicht der Performance.*

#Chr 22	Rohdaten [MB]	Laufzeitanalyse ohne Proz.[MB/s]	Laufzeitanalyse mit Proz.[MB/s]	Speicherverbrauch [MB]
10	573.72	7.39	2.68	4.72
50	2659.97	8.71	5.31	9.84
100	5215.63	9.12	6.28	16.49
200	10431.27	8.95	6.86	30.32

Anmerkung. Ohne/mit Proz. = ohne/mit nachfolgender Prozessierung von Events.

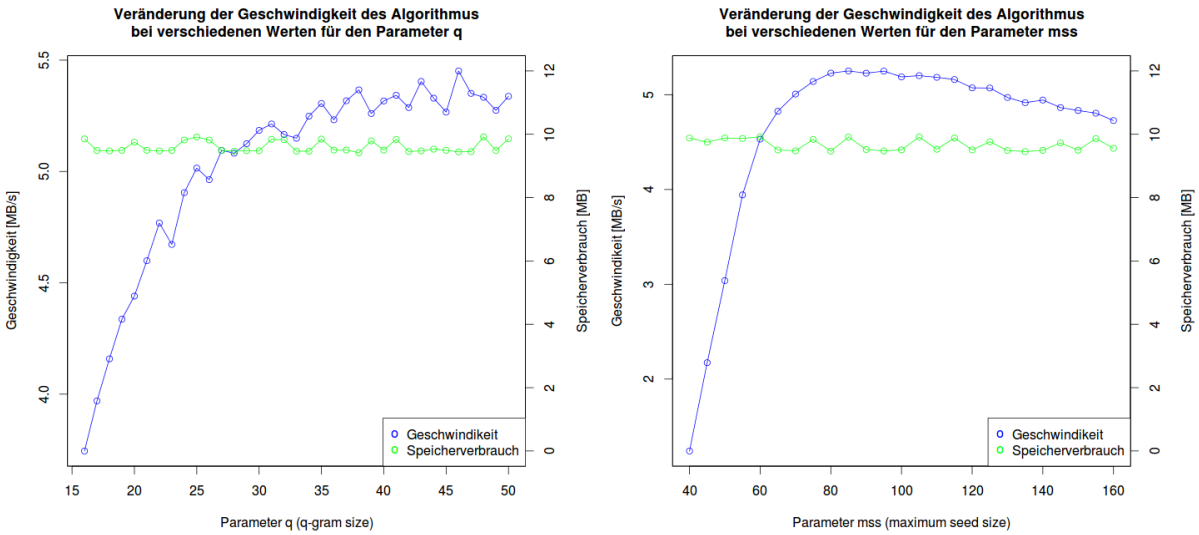

Figure 7: *Einfluss der Parameter q (q-gram Größe) und mss (Maximale seed Größe).* Die Untersuchungen wurden mit einer Anzahl von 50 Chromosomen 22 durchgeführt.

wurden folgende Parametereinstellungen gewählt:

```
1   $ compressFASTA -i input.fa -q 35 -q 15 -mss 100 -t 16 (-ev)
```

Auf den Parameter -*ev* wurde zur Laufzeit- und Arbeitsspeicheranalyse verzichtet.

Die resultierenden Ergebnisse hinsichtlich der Kompression sind in Tabelle 3 aufgelistet. Es wird zwischen dem Speicheraufwand bei der Kompression auf Datei-Ebene und der Kompression auf *Journaled String*-Ebene unterschieden.

Die Kompression auf Datei-Ebene zielt auf die Reduzierung des Speicheraufwands einer gespeicherten Ergebnisdatei ab. Die generierten (vorher) und zusammengeführten (nachher) Δ-Events können als solche in einer Datei gespeichert werden. Bisher wurde noch kein entsprechendes Ausgabeformat erstellt, weswegen die angegeben Werte hypo-

thetischen Berechnungen entsprechen. Dabei werden für die Attribute *pos*, *type* und *del* jeweils ein *unsigned integer* (4 Byte) und für den Bitvektor *cov* eine ungefähre Größe von #Sequenzen/8 Byte (8 Bit pro Byte) berechnet. Der Wert für *ins* wird bei der Ausgabe des Programms für jede Sequenz angegeben (vorrausgesetzt *-ev* wurde spezifiziert). Je höher die Anzahl an zu komprimierenden Sequenzen, desto bessere Kompressionsverhältnisse werden erreicht. Vergleichbare Ergebnisse liefern die Algorithmen ABRC [Wandelt and Leser, 2012] und GDC [Deorowicz and Grabowski, 2011b] die ebenfalls auf dem Datensatz des *1000 Genome Projects* arbeiten. ABRC erreicht auf 1000 Sequenzen im Durchschnitt ein Kompressionsverhältnis von 1:397 [Wandelt and Leser, 2012], während GDC bei 1092 Chromosomen 22 ein Verhältnis von 1:823 erzielt (durchschnittlich 1:627) [Deorowicz et al., 2015].

Die Kompression auf Ebene der Journaled String erzielte geringere Verhältnisse. Dabei ist es wichtig anzumerken, dass die Zusammenführung von Events keine Auswirkungen auf die Anzahl an Journal Entries hat, da jede Sequenz individuell und unabhängig aufgebaut werden muss. Zur Berechnung des Speicheraufwands eines Journaled Strings .....

Die Performance des Algorithmus' ist in Tabelle 4 zu sehen. Die Laufzeit wird dabei in bedeutendem Maße von der Wahl an Parametern beeinflusst (siehe Grafik7).

Zum Vergleich mit anderen Kompressionsalgorithmen muss die Laufzeit ohne Prozessierung betrachtet werden, da andere Algorithmen keine weiterverarbeitende Datenstruktur zur Verfügung stellen. Algorithmen zur Kompression einzelner Sequenzen erreichen Geschwindigkeiten von ca. 1.2 MB/s (GRS [Wang and Zhang, 2011]) und 8.2 MB/s (GreEn [Pinho et al., 2012]). Die vorher angesprochenen Algorithmen erreichen sogar durchschnittlich 40 MB/s (ABRC [Wandelt and und ca. 73 MB/s (GDC [Deorowicz and Grabowski, 2011b]).

4.3 Leistung bei E.Coli Genomen

Für die Untersuchung an Sequenzen mit einer geringen Ähnlichkeit untereinander wurden 20 E.Coli Genome untersucht [PortEco, 2010]. Es wurden die selben Parameter wie bei der Auswertung der menschlichen Chromosomen benutzt. Die Ergebnisse für die Kompression und Performance sind in Tabelle 5 und Tabelle 6 dargelegt. Die Kompressionsverhältnisse sind deutlich geringer als beim menschlichen Chromosom. Auch die Verbesserungen durch Zusammenführung oder Prozessierung von Events zeigen nur wenig Erfolge bei der Speicherverminderung. Die Laufzeit wäre ohne die Prozessierung von Events ähnlich der des menschlichen Chromosoms, ist mit dieser allerdings sehr langsam.

Ein Vergleich der Anzahl an Δ-Events bzw *Journal Entries* pro MB an Eingabesequenz zeigt bei E.Coli sehr viel höhere Verhältnisse (siehe Tabelle 7).

Table 5: *Übersicht der Kompressionsergebnisse von E.Coli Genomen.*

#Seq	Rohdaten [MB]	Kompression auf Datei-Ebene			Kompression auf JS-Ebene		
		Vorher[MB]	Nachher[MB]	**Ratio**	Vorher[MB]	Nachher[MB]	**Ratio**
5	30.70	12.17	11.18	**2.75**	14.20	13.06	**2.35**
10	55.87	24.29	22.18	**2.52**	28.16	26.08	**2.14**
20	113.54	49.98	41.23	**2.75**	60.93	52.84	**2.15**

Anmerkung. Die Bezeichnung *Vorher* bezieht sich auf das Ergebnis vor den in Sektion 3.4 beschriebenen Verbesserungen: Auf Datei-Ebene vor der Zusammenführung von Events und auf Journaled String (JS)-Ebene vor der Prozessierung von Events. Das Verhältnis (Ratio) bezieht sich jeweils auf die *'Nachher'*-Werte zu den entsprechenden Rohdaten.

Table 6: *Übersicht der Performance.*

#Seq	Rohdaten [MB]	Laufzeitanalyse		Speicherverbrauch [MB]
		ohne Proz.[MB/s]	mit Proz.[MB/s]	
10	30.70	0.19	2.26	0.52
50	55.87	0.02	2.18	0.83
100	113.54	0.01	2.13	1.49

Anmerkung. Ohne/mit Proz. = ohne/mit nachfolgender Prozessierung von Events.

Table 7: *Vergleich der Anzahlen von Δ-Events bzw. Journal Entries.*

	E.Coli (10 *Sequenzen*) [Anzahl/MB]	menschl. Chr 22 (10 *Sequenzen*) [Anzahl/MB]
Δ-Events	2935.44	202.69
Journal Entries	9064.45	948.42

Schlechtere Ergebnisse bei Sequenzen mit geringerer Ähnlichkeit sind bei allen bisher angesprochenen Algorithmen zu beobachten.

5 Disskusion und Fazit

tralala

References

[Brudno et al., 2003] Brudno, M. et al. (2003). LAGAN and Multi-LAGAN: efficient tools for large-scale multiple alignment of genomic DNA. *Genome Res*, 13(4):721–731.

[Choudhury et al., 2014] Choudhury, A. et al. (2014). Population-specific common snps reflect demographic histories and highlight regions of genomic plasticity with functional relevance. *BMC Genomics*.

[Christley et al., 2009] Christley, S., Lu, Y., Li, C., and Xie, X. (2009). Human genomes as email attachments. *Bioinformatics*, 25(2):274–275.

[Consortium, 2012] Consortium, T. . G. P. (2012). An integrated map of genetic variation from 1,092 human genomes. *Nature*, 491(7422):56–65.

[Deorowicz et al., 2013] Deorowicz, S., Danek, A., and Grabowski, S. (2013). Genome compression: a novel approach for large collections. *Bioinformatics*, 29(20):2572–2578.

[Deorowicz et al., 2015] Deorowicz, S., Danek, A., and Niemiec, M. (2015). Gdc 2: Compression of large collections of genomes. *Cornell University Library arXiv:1503.01624*.

[Deorowicz and Grabowski, 2011a] Deorowicz, S. and Grabowski, S. (2011a). Compression of genomic sequences in FASTQ format. *Bioinformatics*, 27(6):860–862.

[Deorowicz and Grabowski, 2011b] Deorowicz, S. and Grabowski, S. (2011b). Robust relative compression of genomes with random access. *Bioinformatics*, 27(21):2979–2986.

[Deorowicz and Grabowski, 2013] Deorowicz, S. and Grabowski, S. (2013). Data compression for sequencing data. *Algorithms for Molecular Biology*, 8(1):25+.

[Döring et al., 2008] Döring, A., Weese, D., Rausch, T., and Reinert, K. (2008). SeqAn an efficient, generic C++ library for sequence analysis. *BMC Bioinformatics*, 9:11.

[Gusfield, 1997] Gusfield, D. (Janurary 1997). *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press.

[Huffman, 2006] Huffman, D. (2006). A method for the construction of minimum-redundancy codes. *Resonance*, 11(2):91–99.

[Jean, 1992] Jean (1992). gzip. Programa de computador.

[Kuruppu et al., 2011a] Kuruppu, S., Puglisi, S., and Zobel, J. (2011a). Reference sequence construction for relative compression of genomes. In Grossi, R., Sebastiani, F., and Silvestri, F., editors, *String Processing and Information Retrieval*, volume 7024 of *Lecture Notes in Computer Science*, pages 420–425. Springer Berlin Heidelberg.

[Kuruppu et al., 2011b] Kuruppu, S., Puglisi, S. J., and Zobel, J. (2011b). Optimized relative lempel-ziv compression of genomes. In *Proceedings of the Thirty-Fourth Australasian Computer Science Conference - Volume 113*, ACSC '11, pages 91–98, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

[Loh et al., 2012] Loh, P.-R., Baym, M., and Berger, B. (2012). Compressive genomics. *Nature Biotechnology*, 30(7):627–630.

[NHGRI, 2015] NHGRI (2015). Dna sequencing costs: Data from the nhgri genome sequencing program (gsp). Technical report, National Human Genome Research Institute.

[Pinho et al., 2012] Pinho, A. J., Pratas, D., and Garcia, S. P. (2012). GReEn: a tool for efficient compression of genome resequencing data. *Nucleic Acids Research*, 40(4):e27.

[PortEco, 2010] PortEco (2010). Ecoliwiki. http://ecoliwiki.net/colipedia/index.php/Sequenced_E._coli_Genomeshello.

[Rahn et al., 2014] Rahn, R., Weese, D., and Reinert, K. (2014). Journaled string tree - a scalable data structure for analyzing thousands of similar genomes on your laptop. *Bioinformatics*.

[Rani, 2015] Rani, M. M. S. (2015). a new referential method for compressing genomes. *International Journal of Computational Bioinformatics*.

[Venter et al., 2001] Venter, J. et al. (2001). The sequence of the human genome. *Science*, 291(5507):1304–1351.

[Wandelt et al., 2014] Wandelt, S., Bux, M., and Leser, U. (2014). Trends in genome compression. *Current Bioinformatics*, 9(3):315–326.

[Wandelt and Leser, 2012] Wandelt, S. and Leser, U. (2012). Adaptive efficient compression of genomes. *Algorithms for Molecular Biology*, 7(1):30+.

[Wang and Zhang, 2011] Wang, C. and Zhang, D. (2011). A novel compression tool for efficient storage of genome resequencing data. *Nucleic Acids Research*, 39(7):e45.

[Ziv and Lempel, 1977] Ziv, J. and Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE TRANSACTIONS ON INFORMATION THEORY*, 23(3):337–343.