SeqDPT - Sequencing Data Postprocessing Toolbox

Sebastian Roskosch

Freie Universität Berlin
E-mail: *serosko@zedat.fu-berlin.de*

ZUSAMMENFASSUNG

SeqDPT - Sequencing Data Postprocessing Toolbox - ist ein auf der C++ Softwarebibliothek SeqAn basierendes Programm für das Postprocessing von Next-Generation Sequencing Daten. Es umfasst die Funktionen Barcode Demultiplexing, Adapter Trimming und Low-Quality-Tail Removal. Dieses Application Note stellt die grundlegende Funktionsweise und den Umfang des Toolkits vor, wobei der Fokus jedoch auf der Methodik des Barcode-Demulitplexings liegen wird.

1 EINFÜHRUNG

Mit dem Gebrauch der Next-Generation Sequencing Methoden ist es möglich geworden, große Mengen an Sequenzierdaten zu erzeugen. Die gewonnenen Daten sind jedoch oft fehlerbehaftet oder durch Tags und Sequenzfragemente, welche für den Sequenziervorgang hinzugefügt wurde, kontaminiert, weshalb sie der Nachbearbeitung bedürfen, bevor sie zur weiteren Verwendung freigegeben werden können. Die mit am häufigsten anzutreffenden Probleme sind zwei: Die Kontamination der gelesenen Sequenzen durch die Adapter-Sequenzen und das Abfallen der Qualität der Reads zum Ende der Fragmente hin. Die Adapter-Sequenzen, welche sich bei paired-end Daten am 3'-Enden der Reads befinden können, müssen erkannt und entfernt werden, während für jeden Read individuell bestimmt werden muss, bis zu welcher Qualität man den gelesesenen Nukleotiden Glauben schenken kann, und ab wo der Read abgeschnitten wird. Eine weitere Aufgabe kommt hinzu, wenn mehrere Proben oder Proben aus verschiedenen Quellen zeitgleich sequenziert werden sollen. Um die Kosten möglichst gering zu halten, ist es gängige Praxis, eine einzelne Flow-Cell mit Proben aus unterschiedlichen Quellen (z.B. von verschiedenen Individuen, aus verschiedenen Geweben) zu beladen. Um nach der Sequenzierung festellen zu können, welcher Read zu welcher Quelle gehört, ist es notwendig die Fragmente mit so genannten Barcode-Sequenzen zu versehen. Diese werden entweder inline, d.h. als fester Bestandteil des Reads gelesesen, oder aber multiplex, also als zusätzliche, getrennte Information separat zu jedem Read. Steuern lässt sich dies durch die Auswahl der Sequenzier-Primer. Um die Reads nun wieder ihren jeweiligen Quellen zuordnen zu können, müssen die Barcodes erkannt (und im inline Fall abgetrennt), sowie die Reads zu Gruppen zusammengefasst werden. Bei allen drei der genannten Vorgängen kommt es zudem auf Grund der großen Menge an zu verarbeitenden Daten besonders darauf an, Rechenzeit und Speicherplatz möglichst effizient zu nutzen. [1]

2 METHODEN

SeqDPT wurde in C++ mithilfe der Softwarebibliothek SeqAn [2] implementiert. Das Programm lässt sich am übersichtlichsten in die Teile "Eingabe", "Barcode Demultiplexing", "Adapter Trimming", "Low-Quality-Tail Removal" und "Ausgabe" gliedern, wobei hier die Abschnitte "Adapter Trimming" und "Low-Quality-Tail Removal" nicht behandelt werden. Das Barcode Demultiplexing gliedert sich wiederum in zwei Abschnitte, das exakte Matching und das approximative Matching, welche unabhängig von der Art der Reads (paired-end oder single-end) und der Art der Barcodes (inline oder multiplex) sind.

2.1 Eingabe

SeqDPT akzeptiert Read-Daten im FastA und FastQ Format. Die genutzen Barcodes müssen ebenfalls in einer FastA-Datei bereitgestellt werden, wobei jedem Barcode eine ID vorangehen muss, welche später für den Namen der Gruppe verwendet wird. Werden multiplex Barcodes gebraucht, so wird die Datei mit den multiplex Barcodes als FastA oder FastaQ eingelesen. Gleiches gilt für die Adapter-Sequenzen, sofern diese Vorliegen und das Adapter-Trimming durchgeführt werden soll. Weitere Paramter (Art des Barcode-Clippings, Mindestlänge und Qualität von Reads, Anzahl der auf einmal zu ladenden Reads etc.) lassen sich über die Kommandozeile an den Argument-Parser weitergeben, welcher die nötigen Dateien lädt und anhand der Parameter die entsprechenden Programmstufen initiiert. Es ist auch möglich, die Dateien im komprimierten gzip-Format bereitzustellen. Die standardmäßig eingestellte Anzahl der in einem einzelnen Durchlauf behandelten Reads ist 1000 (2000 bei paired-end Daten), kann aber vom Benutzer frei verändert werden.

2.2 Barcode Demultiplexing

Im Barcode Matching wird ermittelt, welche Sequenzen den Barcode Gruppen zuzuordnen sind. Nach dem Matching erfolgt das Clipping, in welchem die Barcodes aus den Sequenzen entfernt werden, was im Fall von multiplex Barcodes nicht notwendig ist. Den letzten Schritt stellt die Gruppierung der Reads zur weiteren Bearbeitung oder Ausgabe dar. Da die Barcodes sich in jedem Fall in den ersten x Nukleotiden der Reads befinden, ist es

zudem nicht notwendig, die gesamte Sequenz zu durchsuchen. Die Länge der benötigten Prefices (und später auch die Zahl der abzutrennenden Nukleotide) wird automatisch aus den bereitgestellten Barcodes hergeleitet.

2.2.1 Exaktes Matching. Die erste Herangehensweise für das Barcode Matching besteht in einem schnellen und exakten Index basiertem Matching der Barcodes. Über die genutzten Barcodes wird einmalig ein Esa-Index (Enhanced Suffix Array Index) gebaut, mit dessen Hilfe alle gelesenen Reads schnell auf das Vorhandensein eines Barcodes geprüft werden können. Wurde eine Sequenz einem Barcode zugeordnet, so wird dies zunächst als Integer in einem Vektor vermerkt. Die Position innerhalb des Vektors entspricht hierbei der Position der Sequenz im aktuellen Bearbeitungssatz, während der Integer die Position des zugehörigen Barcodes angib, bzw. den Wert -1 annimmt, sofern kein passender Barcode gefunden wurde. Nun werden mit Hilfe der in dem Vektor enthaltenen Informationen die Sequenzen umgeordnet und in einem Vektor aus mehreren StringSets plaziert. Eine Spalte des Vektors, ein StringSet also, repräsentiert hier eine Gruppe von Reads, die zu ein und dem selben Barcode gehören. Welche Spalte hierbei welchem Barcode entspricht wird in einer separaten Map vermerkt. Diese Map und der Vektor können anschließend genutzt werden, um entweder die Ergebnisse direkt in Dateien schreiben zu lassen, wobei eine Datei alle Reads enthält, welche dem gleichen Barcode zugeordnet wurden - der Name der Datei enspricht der ID des Barcodes -, oder die Daten können an die Programmstufen zum Adapter Trimmung und/oder Low-Quality-Tail Removal weitergereicht werden.

2.2.2 Approximatives Matching Die zweite Herangehensweise besteht im approximativen Matching der Barcodes. Hierbei wird während des Matchings bis zu ein Mismatch erlaubt, Indels werden jedoch weiterhin ausgeschlossen. Der zur Anwendung kommende Algorithmus ist der DPSearch-Algorithmus, basierend auf dem Needleman-Wunsch-Sellers Algorithmus [3] . Da hier ein Fehler zugelassen wird und die Barcodes nicht im Vorfeld zu einem Index prozessiert werden können, ist diese Variante langsamer als das exakte Matching. Die weiteren oben beschriebenen Vorgänge und Abläufe bleiben grundsätzlich gleich.

2.3 Ausgabe

Die Ausgabe erfolgt im Format der Eingabe-Dateien, also als FastaA oder FastQ. Liegen die Eingabedateien komprimiert vor, so wird auch die Ausgabe komprimiert erstellt. Der Ausgabe-Pfad kann vom Benutzer frei gewählt werden.

3 ERGEBNISSE UND DISKUSSION

Die Tests erfolgten auf einem Windows 7 64 Bit System mit 8x3.6 GHz AMD Prozessor und 16 GB RAM. Bei den Testdaten handelte es sich um RNA-Seq Daten mit 41823304 Illumina Reads (als single-end) unterschiedlicher Länge im FastQ-Format, welche von einer Serial ATA 3.0 Gbit/s Festplatte geladen wurden. Für den paired-end Fall kam nochmal die gleiche Anzahl an backward-Reads hinzu. Die Länge der sechs verwendeten Barcodes betrug sechs. Den größten Zeitaufwand benötigen stets die I/O-Operationen, während das reine Demultiplexing in wesentlich weniger Rechenzeit ausgeführt wird. Dies spricht besonders dafür, die vom Programm angebotenen Pipeline zu nutzen und alle Arbeitsschritte hintereinander ausführen zu lassen, ohne I/O-Operationen unnötig zu wiederholen. Bei paired-end Reads erhöht sich die I/O-Zeit aufgrund der zusätzlich zu ladenden Daten weiter. Die Arbeit auf komprimierten Daten erhöht die I/O-Zeit etwa um den Faktor 3.

Tabelle 1. Laufzeiten der Testläufe

Reads	Fehler	Barcode	I/O-Zeit	Rechen-Zeit	Gesamt
single	exakt	inline	384 s	128 s	8,5 min
single	exakt	multiplex	297 s	189 s	8,1 min
single	approx.	inline	342 s	172 s	8,5 min
single	approx.	multiplex	362 s	254 s	10,3 min
paired	exakt	inline	948 s	307 s	20,9 min
paired	exakt	multiplex	1106 s	351 s	24,3 min
paired	approx.	inline	1031 s	382 s	23,6 min
paired	approx.	multiplex	1032 s	414 s	24,1 min

Das größte Verbesserungspotential liegt im Bereich der Ein- und Ausgabe. Eine Parallelisierung dieser Prozesse mit den Berechnungen oder der Prozessierung der diversen Datei-Streams dürfte zu einer erheblichen Zeitersparnis führen. Geringfügige Verbesserungen können eventuell auch im Bezug auf die angewandten Algortihmen gemacht werden, z.B. durch die Wahl eines anderen Algorithmus für das approximative Matching, welcher ebenfalls auf einem Index über den Barcodes beruht.

4 SCHLUSSFOLGERUNG

Trotz der noch vorhandenen Verbesserungsmöglichkeiten ist SeqDPT bereits dafür geeignet, große Mengen an Next-Generation Sequencing Daten schnell und flexibel zu verarbeiten.

5 REFERENZEN

[1] Dodt, M.; Roehr, J.T.; Ahmed, R.; Dieterich, C. FLEXBAR—Flexible Barcode and Adapter Processing for Next-Generation Sequencing Platforms. Biology 2012, 1.

[2] Döring, A.; Weese, D.; Rausch, T.; Reinert, K. SeqAn an efficient, generic C++ library for sequence analysis. BMC Bioinformatics 2008, 9, 11.

[3] Needleman S.B.; Wunsch, C.D. A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology 48, 1970.