BlastX: Ein Programm zur Suche von Reads in Proteindatenbanken

Annkatrin Bressin

SeqAn, Freie Universität Berlin, Takustr. 9, 14195 Berlin, Deutschland
E-mail: *AnnkatrinBressin@gmx.de*

ZUSAMMENFASSUNG

BlastX ist ein Read-Mapper, der Next-Generation-Sequencing Datensätze translatiert und zuordnet. Er ermöglicht es funktional ähnliche Proteine in Proteindatenbanken effizient zu finden. Im Unterschied zu anderen Mappern werden Indexstrukturen über die Read- und Proteindatenbanken aufgebaut und somit die Suche der lokalen Alignments beschleunigt. Durch die Reduzierung der Aminosäurenalphabete werden auch strukturell ähnliche Read Positionen gefunden. BlastX ist in C ++ implementiert und nutzt die SeqAn Bibliothek.

1 EINFÜHRUNG

Bei der Verarbeitung und Nutzung von NGS Datensätzen im Bereich der Metagenomik ist es von Interesse, dass Sequenzen einer Probe zu bekannten Proteinen mit ähnlichem Aufbau zugeordnet werden können.

Nukleotidsequenzanalysen vernachlässigen die Degeneration des genetischen Codes und damit wichtige Informationen über die kodierten Sequenzen. In der implementierten BlastX Variante werden aus diesem Grund die Aminosäuresequenzen der Reads nach den sechs möglichen Leserastern übersetzt und in einer Proteindatenbank gesucht. Um die umfangreichen Datenmengen auswerten zu können, ist es von großer Bedeutung die Suche der lokalen Alignments effizient zu gestalten. Deshalb werden Indizes über die Read- und Proteindatenbanken erstellt.

Ein weiterer Aspekt bei der Suche von gleichartigen Proteinen ist die Ähnlichkeit der Aminosäuren untereinander. Dabei sind z.B. Substitutionen einiger Aminosäuren weniger problematisch und führen zu keinen funktionellen Änderungen des Proteins. Um dies zu berücksichtigen wird das vorhandene 20 stellige Aminosäurenalphabet reduziert und strukturell ähnliche Aminosäuren zusammengefasst.

2 METHODEN

In der Implementierung sind ein Klusteralgorithmus zum Erstellen von Alphabeten, die Übersetztung der Reads und Proteine nach den sechs Leserastern, die indexbasierte Suche der lokalen Alignments und das Verifizieren mit Hilfe eines berechneten e-Values enthalten.(siehe Abbildung 1)

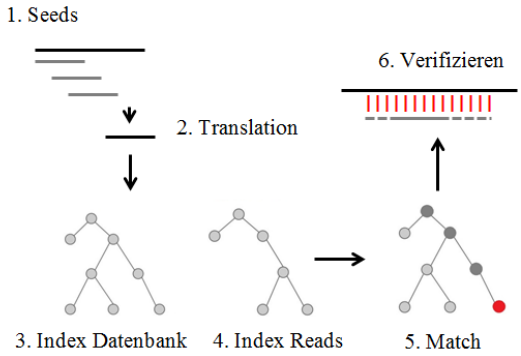

Abbildung 1. Schematischer Aufbau des Programms.

2.1 Alphabete

(siehe Marjan Faizi)

2.2 Translation Reads und Proteindatenbank

2.2.1 Translation der Reads Für die Übersetzung der Reads in *TRANSLATE_READS()* wird mittels der Funktion *hash()* jedes der $4^3 = 64$ möglichen Tripletts auf einen eindeutigen und nachvollziehbaren Wert zwischen 0 und 63 projeziert.

$$\textit{Beispiel: } hash(ACG) = 0 \cdot 4^2 + 1 \cdot 4^1 + 2 \cdot 4^0 = 6$$

Die Funktion *get_Amino_Acid_Pos()* enthält die Referenz zur Aminosäure und wird durch die Zahlen von 0 bis 19 und 20 als STOP-Triplett dargestellt und verarbeitet.

$$\textit{Beispiel: } get\_Amino\_Acid\_Pos(6) = 16 = T$$

Um ein reduziertes Alphabet zu erstellen, dienen die den Aminosäuren zugeordneten Zahlen als Adresse zur zugehörigen Gruppe im Alphabet.

Die Funktionalität, zur Erstellung des rückwärts verlaufenden komplementären Strangs, wird von der SeqAn Funktion *reverseComplement()* übernommen.

2.2.2 Translation der Proteindatenbank Bei der Übersetzung der Proteindatenbank *TRANSLATE_DATABASE()* sind die Aminosäuren schon bekannt. Diese dienen als direkte Adresse zur Zuordnung der

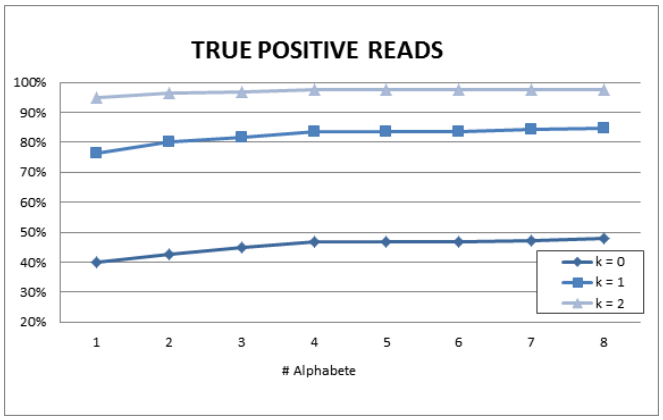

Abbildung 2. Richtig gefundene Reads in Prozent über der Anzahl der Alphabete auf einem Testdatensatz mit einer Fehlerrate von 10%.

jeweiligen Gruppe.

2.3 Lokale Alignments finden

Bei der Suche lokaler Alignments werden sowohl die Größe der zu suchenden Subsequenzen aus den Reads, als auch die Fehlertoleranz in der Subsequenz als Benutzerparameter variabel gehalten. Um die Laufzeit zu verringern werden in der Funktion *FIND_MATCHES()* Indezes über den Subsequenzen sowie der Proteindatenbank erstellt. Basierend auf einen Suffix Baum wird für die Reads der SeqAn-Index *IndexWotd* genutzt. Über der Proteindatenbank wird der *IndexSa* auf Grundlage eines Suffix Array verwendet.

Die Suche der Subsequenzen in der Datenbank ist mit einem spezialisierten Backtracking-Algorithmus der SeqAn-Klassen *Finder* und *Pattern* realisiert. Die Treffer werden mit der Funktion *finder()* aufgerufen und anschließend mit *append_to_match_found()* in einem Objekt der Klasse *Match_found* gespeichert.

Mit dem Befehl *get_read_position()* wird das Read zur gefundenen Subsequenz ermittelt.

2.4 Lokale Alignments verifizieren

Die vom Backtracking-Algorithmus gefundenen Teiltreffer werden der Reihe nach erweitert und bewertet. *VERIFY_ALL()* nutzt zur Erweiterung nicht die reduzierten Alphabete, sondern die orginalen Aminosäuren. Dabei wird ein Fenster mit der Größe des Reads über den lokalen Treffer und der Proteinstelle gelegt. Zur Erstellung eines globales Alignment wird die SeqAn Funktion *globalAlignment()* mit der Blosum80 Matrix verwendet. Um unnötige Durchläufe zu vermeiden speichert *known_position()* die Fensterintervalle.

3 RESULTATE UND DISKUSSION

BlastX wurde auf simulierten Datensätzen erfolgreich getestet. Das Simulations-Programm gibt zur Überprüfung und Kontrolle die Positionen der Reads in der Datenbank aus. Bei fehlerfreien Datensätzen werden mit einem Alphabet 100% der Reads zugeordnet. Für die Erstellung der in Abbildung 2 und Tabelle 1 verwendeten Testdatensätze wurde eine Mutationsrate von 10% genutzt.

Tabelle 1. Laufzeit in Abhängigkeit zur Anzahl der Reads, Anzahl der akzeptierten Fehler und zur Größe des Aminosäure Alphabetes

Anzahl Reads	Laufzeit		Aminosäuren	
	k=0	k=1	20 As	10 As
10000	26.027 s	238.009 s	26,11 s	21,45 s
20000	43.708 s	355.716 s	49,374 s	43,353 s
30000	70,98 s	/	77,14 s	68,279 s

3.1 Analsye zur Anzahl an Alphabeten

Um die Anzahl an richtigen Treffern zu erhöhen steht es dem Benutzer offen mehr als nur ein Alphabet zur Suche zu verwenden. Die Abbildung 2 zeigt die gefundenen TP Treffer in Prozent in Abhängigkeit zu der Anzahl der genutzten Alphabete. Dabei wird ersichtlich, dass die Trefferanzahl mit ein bis vier Alphabeten monoton steigt. Danach ist eine Hinzunahme von weiteren Alphabeten nicht empfehlenswert, da durchschnittlich nur noch 1 % an TPs gefunden werden und unnötige Laufzeitkosten entstehen.

3.2 Laufzeitanalyse

In Tabelle 1 sind Laufzeitversuche in Abhängigkeit zur Anzahl der Reads, der zugelassenen Missmatches und der Aminosäuregruppen abgebildet. Dabei ist das Zulassen von Missmatches sehr teuer und führt im Versuch zu einer zehnfachen Erhöhung der Laufzeit. Dieses Verhalten ist durch die ansteigende Trefferanzahl zu erklären. Die Nutzung von reduzierten Aminosäure Alphabeten hat wie in Tabelle 1 gezeigt keinen negativen Einfluss auf die Laufzeit, da gesetzte Readintervalle durch eine Kontrollfunktion nur einmal verifiziert werden.

4 ABSCHLUSS

Für die aktuelle Version 1 des Programms wird eine Alphabetanzahl von vier empfohlen und ein achtsamer Umgang mit dem Zulassen von Missmatches in den Subsequenzen.

LITERATUR

Brudno, M., Do, C. B., Cooper, G. M., Kim, M. F., Davydov, E., Program, N. I. S. C. C. S., Green, E. D., Sidow, A., and Batzoglou, S. (2003). LAGAN and Multi-LAGAN: efficient tools for large-scale multiple alignment of genomic DNA. *Genome Res*, **13**(4), 721–731.

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

Needleman, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, **48**(3), 443–453.

Rausch, T., Emde, A.-K., Weese, D., Dring, A., Notredame, C., and Reinert, K. (2008). Segment-based multiple sequence alignment. *Bioinformatics*, **24**(16), i187–i192.