Effiziente Vorverarbeitung kurzer Sequenzstücke für die effiziente Filterung beim Vergleich von Proteinsequenzen

Stefan Kurtz

24. Mai 2023

Zum Vergleich großer Mengen von Proteinsequenzen werden typischerweise Filtertechniken verwendet. Diese Filtertechniken sollen möglichst effizient solche Paare von Proteinsequenzen ausschließen, für die es kein signifikantes lokales Alignment gibt. Alle anderen Sequenzpaare werden dann im Weiteren genauer, z.B. durch aufwändige Verfahren der dynamischen Programmierung untersucht. Solche Filtertechniken, basierend auf q-meren und k-Umgebungen werden seit mehr als 20 Jahren in Programmen wie z.B. Blastp oder MMseqs2 genutzt. Dabei ist q einen Längenwert und k ein minimaler Score-Wert, den Paare von q-mere erreichen müssen, damit sie im Rahmen der Filtertechniken als sogenannte Seeds betrachtet werden. Die Filtertechniken kann man durch die Wahl der Parameter q und k variieren, ebenso wie durch Bedingungen, die verschiedene Seeds relativ zueinander erfüllen müssen.

Der Filter von Blastp verwendet im Standardfall die Blosum62-Scorematrix mit $q = 3$ und $k = 22$, und verlangt zwei Blast-Hits mit einem maximalen Abstand auf der gleichen Diagonale [1]. Liegt ein solches Paar von Blast-hits vor, erfolgt ein Ungapped Alignment von der Mitte zwischen den Blast-hits in beide Richtungen.

Ein alternativer und sehr effizienter Ansatz wird in der MMseqs2-Software realisiert [2]. Hier wird $q = 7$ verwendet und k so gewählt, dass die k-Umgebung eines einzelnen q-mers 600–60 000 Elemente umfasst. Es werden Diagonalen bestimmt, in der zwei Blast-hits direkt aufeinander folgen. Gibt es in einem Paar von zwei Protein-Sequenzen ein solche Diagonale, erfolgt zunächst ein Erweiterung ohne Gaps und falls diese bestimmte Kriterien erfüllt, wird auch eine Erweiterung mit Gaps durchgeführt.

Sei $\mathcal{A}$ ein geordnetes Alphabet und $\sigma : \mathcal{A} \times \mathcal{A} \to \mathbb{R}$ eine Score-Funktion. Für $u, v \in \mathcal{A}^*$ mit $|u| = |v|$ definieren wir $\sigma(u, v) = \sum_{i=0}^{|u|-1} \sigma(u[i], v[i])$.

Um die Umgebung für ein 7-mer effizient berechnen zu können, wird für alle $q \in \{2, 3\}$

und alle $u \in \mathcal{A}^q$ die Liste $\mathsf{ST}[u] = [(v, \sigma(u, v)) \mid v \in \mathcal{A}^q]$ berechnet und nach den Scores absteigend sortiert.

Man kann ST als Scorematrix über dem Alphabet $\mathcal{A}^q$ auffassen.

Die Umgebung eines 7-mers u wird durch die Bildung des kartesischen Produktes

$$\mathsf{ST}[u[0 \ldots 2]] \times \mathsf{ST}[u[3 \ldots 4]] \times \mathsf{ST}[u[5 \ldots 6]] \tag{1}$$

aufgezählt. Durch die Sortierung der Scores müssen nicht alle Elemente von (**??**) aufgezählt werden. Die Aufzählung in einer Komponente kann abgebrochen werden, sobald das erste Element auftritt, dessen Score zu klein ist, um den Mindestscore zu erreichen.

Diese Idee der Zerlegung von u ergibt sich aus der detaillierten Analyse des Source Codes von MMseqs2. Sie wurde aber bisher meines Wissens nach nicht detailliert beschrieben.[1]

Sei $\varphi : \{0, 1, \ldots, q-1\} \to \{0, 1, \ldots, q-1\}$ eine bijektive Abbildung. Für eine Sequenz der Länge q definieren wir $\varphi(u) = u[\varphi(0)]u[\varphi(1)] \ldots u[\varphi(q-1)]$, d.h. φ permutiert die Buchstaben von u.

Hier sind wir an einer speziellen Permutation φ_u interessiert, so dass $u[\varphi_u(i)] \leq u[\varphi_u(i+1)]$ für alle i, $0 \leq i \leq q-2$, wobei $\leq$ die Ordnung der Zeichen des Alphabets $\mathcal{A}$ ist. φ_u ist also eine Permutation, die die Zeichen von u in sortierter Reihenfolge liefert. Wir sprechen hier von sortierten q-meren. Sei φ_u^{-1} die Umkehrabbildung von φ_u. Damit gilt $\varphi_u^{-1}(\varphi_u(i)) = i$ für alle i, $0 \leq i \leq q-1$ und es folgt $\varphi_u^{-1}(\varphi_u(u)) = u$.

Die Grundlage dieses Projektes ist die Eigenschaft

$$\sigma(u, v) = \sigma(\varphi_u(u), \varphi_u(v)) \tag{2}$$

für alle $v \in \mathcal{A}^q$. D.h. die Anwendung von φ_u auf u und auf v permutiert die Zeichen in der gleichen Weise, sowohl in u als auch in v. Dadurch bleibt der paarweise Score in den permutierten Sequenzen bzgl. σ unverändert.

Die Idee ist nun, $\mathsf{ST}[\varphi_u(u)]$ einmal zu berechnen und abzuspeichern. Für alle $u' \in \mathcal{A}^q$ mit $\varphi_u(u) = \varphi_{u'}(u')$ ergibt sich dann

$$\mathsf{ST}[u'] = [(\varphi_{u'}^{-1}(v), s) \mid (v, s) \in \mathsf{ST}[\varphi_u(u)]]. \tag{3}$$

Der Vorteil dieses Ansatzes besteht darin, dass die Anzahl der sortierten q-mere viel kleiner ist, als die Anzahl der q-mere.[2] Damit sollte sich der Zeit- und Speicherplatzaufwand für die Vorverarbeitung erheblich reduzieren lassen. Der zusätzliche Aufwand besteht in der Anwendung der Umkehrabbildung φ_u^{-1} auf v, siehe Gleichung (3). Falls sich dieser Aufwand als zu groß herausgestellt, wäre hier ggf. eine effiziente Berechnung mit Hilfe von SIMD-Instruktionen denkbar.

Die Aufgabe dieses B.Sc. Projektes besteht darin, die hier beschriebenen Ideen effizient in objekt-orientierter Form in C++ zu implementieren und auszuwerten. Dabei sollen soweit

[1] Hier prüfen, ob es inzwischen die Doktorarbeit von Martin Steinegger (vermutluch an der TU München) gibt. Hierin könnte das Verfahren beschrieben sein.

[2] Sortierte q-mere entsprechen Multisets. Hierzu bitte die Folien aus PfN1 zum Thema Multisets ansehen.

wie möglich bereits existierende Sourcen aus `github.com:stefan-kurtz/gttl.git` wiederverwendet werden. Die Implementierung könnte aus den folgenden Komponenten bestehen:

- Klassen zur Verwaltung von Spaced Seeds[3]
- Klassen zur Codierung von sortierten q-meren durch Integercodes
- Klassen zur Darstellung von Scorematrizen, die indiziert sind durch Integercodes sortierter q-mere u und die es erlauben, die q-mere v zusammen mit dem Score $\sigma(u, v)$ in sortierter Reihenfolge aufzuzählen.
- Klassen zur Verwaltung von Permutationen, die q-mere in sortierte q-mere transformieren (und umgekehrt).
- Klassen zu Aufzählen von q-meren für $q \in \{4, 5, 6, 7\}$ durch Bildung kartesischer Produkte, analog zu (1).

Die Implementierung soll generisch sein und durch Wahl entsprechender Template-Parameter z.B. für den Wert von q und die Score-Funktion, großes Optimierungspotential für den Compiler bieten. Dabei sollen die relevanten Daten kompakt dargestellt werden. So sollen nur $\lceil \log_2(r-1) \rceil$ Bits verwendet werden, um sortierte q-mere zu codieren, falls ihre Anzahl r ist. Ebenso erscheint es bei den nach Scores sortierten Listen von Paaren mit einem q-gram und einem Score sinnvoll, die aufeinanderfolgenden Scores durch Differenzen zu codieren. Hier könnte man ggf. unäre Codes verwenden.

Ein wichtiger Aspekt der Arbeit besteht in der systematischen Auswertung der Implementierung hinsichtlich Speicherplatz und Laufzeit für den relevanten Parameterraum.

Literatur

[1] S.F. Altschul, T.L. Madden, A.A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman. Gapped BLAST and PSI-BLAST: A New Generation of Protein Database Search Programs. *Nucleic Acids Research*, 25(17):3389–3402, 1997.

[2] Martin Steinegger and Johannes Söding. Mmseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nature biotechnology*, 35(11):1026, 2017.

[3] Tatsächlich werden in MMseqs2 nicht q-mere betrachtet, sondern Strings einer bestimmten Länge s, in denen $q < s$ Positionen für die Bewertung relevant sind. Dabei wird s Span genannt und q das Gewicht des Spaced Seeds. Damit werden $s - q$ Positionen von s nicht bei der Bewertung berücksichtigt.