Sequenzvergleiche ohne Alignments durch lokales Dekodieren von Sequenzen

Christian Otto

(13. Juli 2007)

Inhaltsverzeichnis

1 Grundidee

Normalerweise werden bei Sequenzvergleichen multiple Alignments eingesetzt. Diese weisen jedoch neben der schlechten Laufzeit einige Nachteile auf: Sie beruhen auf Editieroperation (Deletion, Substitution, Insertion), wobei z.B. Deletionen/Insertionen nicht den regulären evolutionären Modellen folgen und somit evolutionär schlecht bewertet sind.
Beim multiplen Alignment werden mehrdeutige Regionen (in Bezug auf die Editieroperationen) gelöscht.
→ Dies führt zu Informationsverlust, da z.B. Permutationen und Inversionen dadurch vernachlässigt werden, obwohl sie evolutionär bedeutsam sind.

Um solche Regionen zu berücksichtigen, wird manuelles Alignment (Expert Editing) durchgeführt, welches jedoch zu einem großen Zeitaufwand vor allem bei langen Sequenzen und zu Problemen bei der Reproduzierbarkeit führt. Dabei wird zusätzliche Information verwendet, um auch Regionen zu berücksichtigen, die mehrfach aufeinanderfolgende Deletionen/Insertionen beinhalten, aber trotzdem evolutionär bedeutsam sind und im Alignment auftauchen sollten. Somit kann ein multiples Alignment verbessert werden (um z.B. ein lokales Minimum zu verlassen).

Idee der Sequenzvergleiche ohne Alignments:

Intuitiver Ansatz: Bei den Sequenzvergleichen ohne Alignments ist der intuitive Ansatz, dass man die Nukleotidhäufigkeiten der Sequenzen vergleicht, da Sequenzen mit signifikant unterschiedlichen Nukleotidzusammensetzungen nicht nah verwandt sein können.
Problematisch ist, dass es auch nicht nah verwandte Sequenzen mit identischen Nukleotidhäufigkeiten existieren.

Nächster Schritt: Man vergleicht N-Wörter (Teilsequenzen der Länge N) zwischen Sequenzen, um daraus Ähnlichkeiten zu berechnen. Die evolutionäre Idee ist, dass man somit homologe Bereiche identifiziert und dass Sequenzen mit vielen gemeinsamen homologen Blöcken nah verwandt sind.
Problematisch hierbei ist, dass die N-Wörter entweder identisch oder unterschiedlich sind. Dies kann kompensiert werden, indem man auch nicht perfekt identische N-Wörter betrachtet, wobei die im Vortrag vorgestellten Modelle und Algorithmen dies noch nicht berücksichtigen. Obwohl dann zusätzlich die Frage entsteht, wie man nicht perfekt identische N-Wörter mit einem Score bewertet und diesen einbezieht bzw. ab welchem Ähnlichkeitsgrad es überhaupt sinnvoll ist von homologen Blöcken zu sprechen.

Lokales N-Dekodieren:

Um evolutionäre Informationen aus mehrdeutigen Regionen (laut Alignment) zu gewinnnen und nicht auf manuelles Alignment angewiesen zu sein, wurde die Idee vom lokalen Dekodieren vom Grad N von Sequenzen entwickelt:
Diese beruht auf Vergleichen zwischen überlappenden Wörtern der Länge N. Für jede Sequenzposition i gibt es somit N überlappende Wörter, wobei i dann jeweils an Position $1, \ldots, N$ dieser Wörter stehen kann.
Übereinstimmungen von solchen Teilwörtern zwischen verschiedenen Sequenzen entsprechen konzeptionell homologen Blöcken zwischen diesen Sequenzen. Man kann nun auf Basis dieser Idee Sequenzvergleiche durchführen und Distanzen zwischen Sequenzen berechnen.

2 Naiver Algorithmus

2.1 Definitionen

Definition der Relation $\overset{k}{\smile}_N$:

Sei s eine Sequenz und $N \in \mathbb{N}, N > 0$.
Dann ist die Relation $(\overset{k}{\smile}_N)_{-1 \leq k < N}$ für alle Paare von Positionen (i, j) von s definiert durch:

- $i \overset{-1}{\smile}_N j \Leftrightarrow i = j$
- Für $0 \leq k < N$:
  $i \overset{k}{\smile}_N j \Leftrightarrow \exists l, 0 \leq l \leq k$: $s_{[i-l,i-l+N-1]} = s_{[j-l,j-l+N-1]}$.

Für alle Zahlen $-1 \leq k < N$ sei dann $\overset{k}{\smile}_N$ die transitive Hülle der Relation $\overset{k}{\smile}_N$.

Aus der Relation $\overset{k}{\smile}_N$ folgende Aussagen:

1. $i \overset{k-1}{\smile}_N j \Rightarrow i \overset{k}{\smile}_N j$ (wichtig für Iteration im Algorithmus)
2. $i \overset{k-1}{\smile}_N j \Rightarrow i+1 \overset{k}{\smile}_N j+1$ (wichtig für Nachfolgerbeziehung im Algorithmus)
3. $i \overset{k}{\smile}_N j \Rightarrow s_{[i,i-k+N-1]} = s_{[j,j-k+N-1]}$

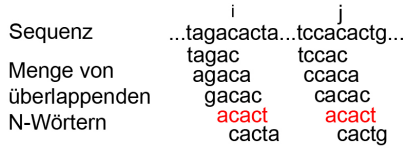
Für alle $-1 \leq k < N$ ist die Relation $\overset{k}{\smile}_N$ eine Äquivalenzrelation (unter der Annahme, dass $|s| \geq N$).
Δ_k sei die Menge der zur Relation $\overset{k}{\smile}_N$ gehörigen Äquivalenzklassen.

2.2 Lokales N-Dekodieren

Verwandtschaft zwischen Sequenzpositionen:

Zwei Sequenzpositionen a, b sind genau dann verwandt, wenn $a \overset{N-1}{\smile}_N b$ gilt
d.h. es gibt Positionen $k_0, \ldots, k_n$ mit $k_0 = a$, $k_n = b$ und $\forall i \in \{0, \ldots, n-1\} : k_i \overset{N-1}{\smile}_N k_{i+1}$. Dadurch wird sozusagen der transitive Abschluss dargestellt. Es gilt nun $i \overset{N-1}{\smile}_N j$, wenn ein überlappendes N-Wort an Position i und j übereinstimmt, wobei i und j in diesem Wort an der gleichen Position stehen.

Beispiel:



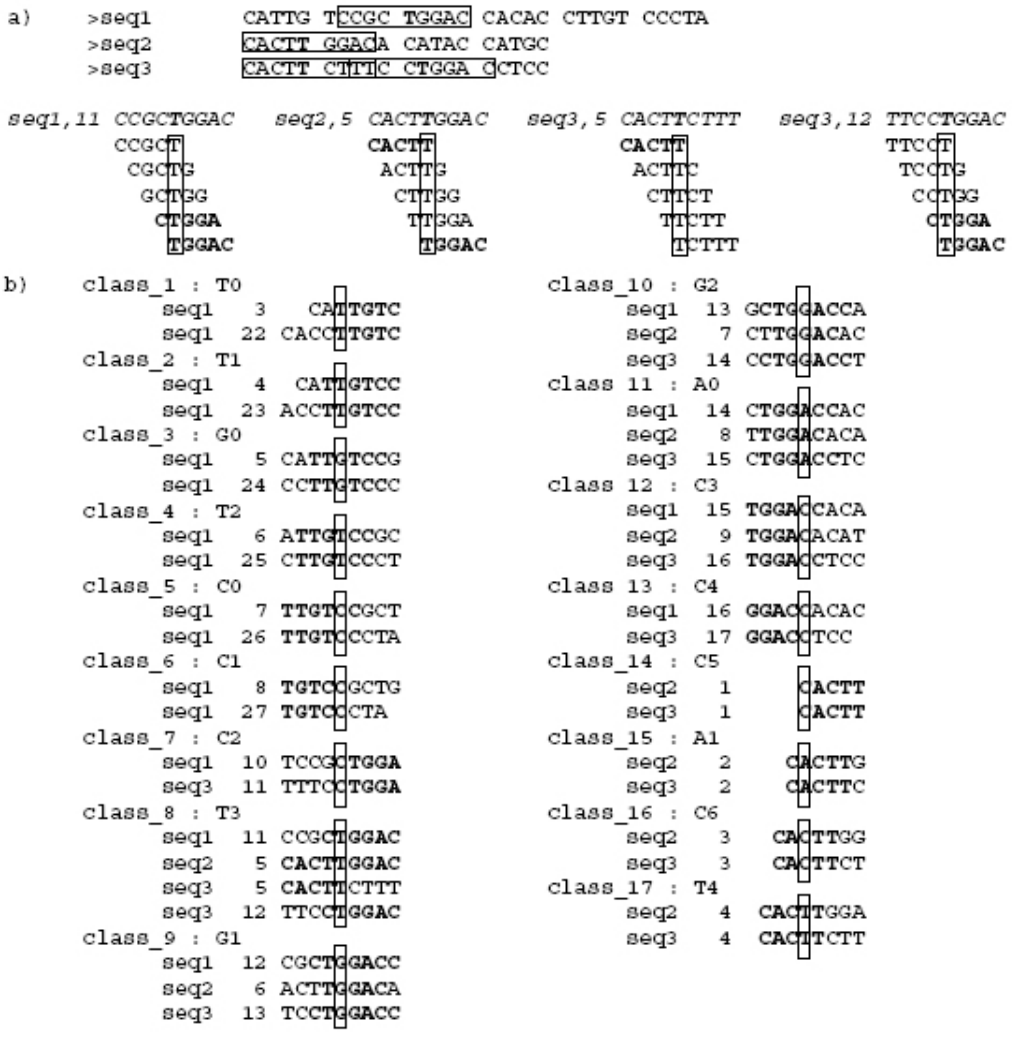
$$\Rightarrow i \overset{-1}{\not\sim}_5 j, i \overset{0}{\not\sim}_5 j, i \overset{1}{\sim}_5 j, i \overset{4}{\sim}_5 j$$

Ermittlung der Äquivalenzklassen:

a) Menge der N-Wörter jeder Sequenzposition ermitteln (siehe Beispiel oben) und miteinander zeilenweise vergleichen.
→ Man findet somit Relationen $i \overset{N-1}{\sim}_N j$ zwischen Sequenzpositionen i und j.

b) Transitiven Abschluss der Relation $\overset{N-1}{\sim}_N$ und Äquivalenzklassen bilden. Dazu fügt man nacheinander miteinander in Relation stehende Positionen in eine Menge und vereinigt Mengen, bei denen mindestens eine Sequenzposition gleich ist. Somit erhält man die Äquivalenzklassen Δ_{N-1}.

Beispiel:



Lokales N-Dekodieren:

c) Nun Äquivalenzklassen auf Zustände abbilden:
Dabei ist es in dem Paper üblich gewesen, dass einelementige Äquivalenzklassen auf das ursprüngliche Symbol (A, T, C, G) abgebildet werden. Diese Symbole werden im Folgenden nicht weiter betrachtet. Andere Äquivalenzklassen werden auch auf das ursprüngliche Symbol und einem Index abgebildet. Der Index wird pro Symbol einzeln gezählt und wird nach Vorkommen der Zeichen in einer Referenzsequenzfolge verwendet.
⇒ Dies wird als lokales N-Dekodieren der Sequenz bezeichnet (analog zu Hidden Markov Modell), da man Positionen einen Zustand zuordnet (nämlich die verwandtschaftliche Verbindung zu anderen Sequenzen), der normalerweise versteckt ist.

Beispiel:

c) >seq1 C A $T_0T_1G_0T_2C_0C_1G$ $C_2T_3G_1G_2A_0C_3C_4A$ C A C C $T_0T_1G_0T_2C_0C_1C$ T A
>seq2 $C_5A_1C_6T_4T_3G_1G_2A_0C_3A$ C A T A C C A T G C
>seq3 $C_5A_1C_6T_4T_3C$ T T T C $C_2T_3G_1G_2A_0C_3C_4T$ C C

2.3 Ermittlung der Distanzmatrix

d) Erstellen einer Liste mit Anzahl des Vorkommens jedes Zustandes (der einer Äquivalenzklasse entspricht) je Sequenz

Beispiel:

d)	seq1	seq2	seq3
T3	1	1	2
T4	0	1	1
A0	1	1	1
A1	0	1	1
G1	1	1	1
G2	1	1	1
C2	1	0	1
C3	1	1	1
C4	1	0	1
C5	0	1	1
C6	0	1	1

(T0, T1, T2, G0, C0 and C1 are
only repeated in seq1, two times
each)

Berechnung der Distanzmatrix:

Seien Zustände $x_1, \ldots, x_n$ mit $n \in \mathbb{N}$ gegeben.
Nun sei $|u|_{x_i}$ die Anzahl von Vorkommen des Zustandes x_i in der Sequenz u und $|u|$ die Länge der Sequenz u.

e) Berechnen der Ähnlichkeiten sim(u, v) zwischen Sequenzen u und v durch:

$$\mathrm{sim(u, v)} = \sum_{i=1}^{n} \min\left(|u|_{x_i}, |v|_{x_i}\right)$$

Dabei summiert man die minimale Anzahl der einzelnen Zustände in u und v auf.
Normalisierung und Berechnung der Distanzen durch:

$$\mathrm{dist(u, v)} = 1 - \frac{\mathrm{sim(u, v)}}{\min\left(|u|, |v|\right)}$$

Man normiert die Ähnlichkeiten zwischen den Sequenzen u und v, indem man durch die minimale Sequenzlänge von u und v teilt. Da die maximale Anzahl von Ähnlichkeiten der Sequenzlänge der kürzeren Sequenz entspricht, normiert man die Ähnlichkeit somit auf das Intervall $[0, 1]$. Zum Berechnen der Distanz muss lediglich noch 1-Ähnlichkeit gerechnet werden.

Beispiel:

d)

	seq1	seq2	seq3
T3	1	1	2
T4	0	1	1
A0	1	1	1
A1	0	1	1
G1	1	1	1
G2	1	1	1
C2	1	0	1
C3	1	1	1
C4	1	0	1
C5	0	1	1
C6	0	1	1

e) Similarities

	seq2	seq3
seq1	5	7
seq2		9

Distances

	seq2	seq3
seq1	0.75	0.65
seq2		0.55

⇒ Erstellung eines Verwandtschaftsbaumes möglich (z.B. Neighbor-Joining-Tree)

2.4 Aussagefähigkeit der Distanzen

Das Distanzmaß dist(u, v) ist zunächst nicht aussagekräftig bei einem neuen Verfahren, da man nicht weiß, bei welchem Wert von einem Zufallsbefund zu sprechen ist und ab wann eine konservierte Verwandtschaft vorliegen könnte. Dies gibt dann auch Auskunft darüber, wie man den Baum erstellt, da man Sequenzen mit niedriger Distanz erst weit unten aufsplittet, weil diese nah verwandt sind.
Idee: Vergleich mit Distanzwerten von randomisierten Sequenzen ⇒ Bootstrap (statistisches Verfahren)

Allgemeine Vorgehensweise bei Bootstrap:

1. Erstellen von Bootstrapsamples aus einer Stichprobe (Resampling)
2. Berechnung einer bestimmten Größe aus dem Sample (nicht unbedingt statistisch)
3. Iteration dieser Schritte (1000-10000 Iterationen sind sinnvoll)
   ⇒ Ergebnis: Verteilung der Größe aus den Samples

⇒ p-Wert Schätzung möglich

Anwendung des Bootstraps auf diesen Sachverhalt:

Sei Distanz dist(u, v) der Sequenzen u und v gegeben.

1. Zuerst u und v resamplen:
   Dazu für jede neue Sequenz aus der alten Sequenz mit Zurücklegen ziehen bis man eine neue Sequenz gleicher Länge erhält. Dabei kann es sein, dass manche Positionen aus der alten Sequenz gar nicht oder mehrfach gewählt werden.
   ⇒ neue Sequenzen u^*, v^*
2. dist(u^*, v^*) über N-lokales Dekodieren berechnen
3. mehrfach iterieren ⇒ Verteilung von dist erstellen

Wenn dist(u, v) nun extrem in der Verteilung liegt (z.B. in den 2.5% niedrigsten oder höchsten Werten), dann deutet dieser Distanzwert auf keinen Zufallsbefund hin sondern besitzt Aussagekraft (z.B. starke Verwandtschaft). Man schätzt somit quasi die Wahrscheinlichkeit eines bestimmten Distanzwertes des neuen Verfahrens ein. Man erhält eine Verteilung der Distanzwerte, wodurch man die Möglichkeit hat, einen p-Wert (wie oben bereits beschrieben) zu berechnen. Der p-Wert gibt dabei die Wahrscheinlichkeit an, einen solchen oder noch extremeren Wert unter der gegebenen Verteilung zu erhalten.

Abschätzen der Stabilität des Baumes:

Wenn man die Distanzmatrix zwischen den ursprünglichen Sequenzen erstellt hat, kann man neben der Aussagekraft der einzelnen Distanzen zusätzlich die Aussagekraft des aus der Distanzmatrix erstellten Baumes abschätzen. Dies funktioniert, indem man den Bootstrap-Algorithmus etwas abändert:

1. Resampling auf alle Sequenzen anwenden
2. Distanzmatrix berechnen und Baum erstellen
3. Iteration
   ⇒ Aus den Bäumen kann man mit dem Programm Phylib einen Konsensusbaum erstellen, der Bootstrap-Werte bei Aufspaltungen enthält [3]. Diese sagen aus, in wieviel Prozent aller Bootstrap-Bäumen, diese Teiltopologie vorhanden war.
   ⇒ Man erhält dadurch Auskunft über die Stabilität und Aussagekraft des Baumes.

3 Verbesserungen

3.1 Verbesserung der Komplexität

Der naive Algorithmus hat eine hohe Komplexität in CPU-Zeit.
Das Problem dabei ist die Berechnung der Äquivalenzklassen Δ_{N-1}.
⇒ Verbesserung auf lineare Komplexität in Zeit und Speicher möglich.

Dazu weitere Definitionen:

- $\nabla_k \subset \Delta_k$ ist definiert durch:
  $\delta \in \nabla_k \Leftrightarrow \delta \in \Delta_k, \exists$ i,j $\in \delta$ mit $(i+1) \overset{k}{\not\sim}_N (j+1)$
- Die Abbildung F_k:$\Delta_k \rightarrow \Delta_{k+1}$ ist definiert durch:
  $F_k(\gamma) = \delta \Leftrightarrow \gamma \in \Delta_k, \delta \in \Delta_{k+1}, \forall i \in \gamma : i+1 \in \delta$
  Man nennt δ dann k-Nachfolger von γ.

Funktionsweise des verbesserten Algorithmus:

1. Ermittlung von Δ_{-1}, Δ_0, ∇_0 und F_{-1}
2. Iterative Berechnung von Δ_k, ∇_k und F_{k-1} aus Δ_{k-1}, ∇_{k-1} und F_{k-2} für $0 < k < N-1$
   ⇒ Δ_{N-1}

Ermittlung von Δ_0, ∇_0 und F_{-1}:

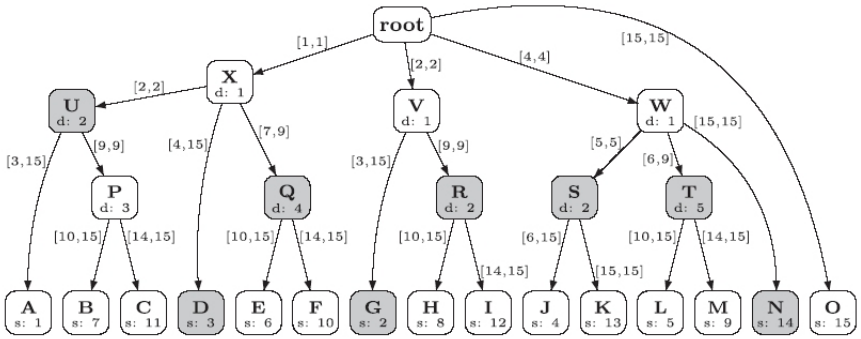
Gegeben sei die Sequenz s.

Beispiel: (N=2)

```
positions : 1 2 3 4 5 6 7 8 9 10 11 12 13 14
    s     = a g a c c a a g c  a  a  g  c  c
```

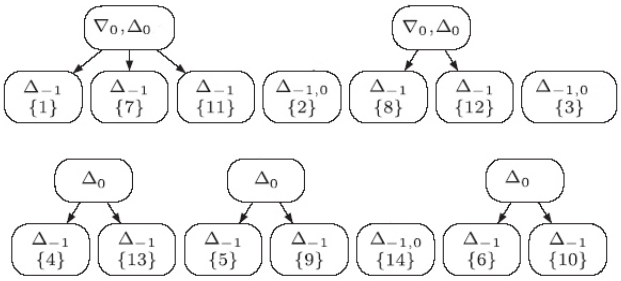
1. Erstellung des Suffixbaumes aus der Sequenz s:
   Ein Suffixbaum T für einen String s mit m Symbolen ist ein gerichteter Baum mit m Blättern. Dabei sind die Blätter jeweils die Sequenzpositionen von s. Der Baum T enthält alle Suffixe von S, wobei mehrfach auftretende Teilstrings nur einmal in T enthalten sind. Somit werden gleiche Präfixe zusammengefasst und innere Knoten sind gemeinsame Substrings der darunterliegenden Suffixe.
   Beschriftung: alle Knoten sind mit einem Label versehen; d bezeichnet die gemeinsame Präfixlänge bei einem inneren Knoten; die gerichteten Pfade sind mit dem Substring von s markiert, welcher identisch ist (meist erstes Auftreten in Sequenz ist Referenz); die Blattknoten sind mit der Sequenzposition bezeichnet, für welche der Pfad von der Wurzel zum Blatt den Suffix von s ab dieser Position darstellt.

Beispiel:



2. Ablesen von Δ_0 und ∇_0:
   Dazu werden die Kenntnisse aus dem Suffixbaum (siehe 1.) genutzt. Man sucht alle Knoten, die von der Wurzel über einen Pfad der Länge größer gleich N erreicht werden (dabei die jeweils die ersten). Als Länge des Pfades wird hier die Summe der Längen der Substring auf den Kanten genommen. Alle Blätter, die unterhalb dieser Knoten liegen (evtl. nur der Knoten selbst, wenn dieser ein Blatt ist), sind dann in einer Äquivalenzklasse von $\overset{0}{\smile}_N$. Die Menge aus diesen Klassen ist dann Δ_0.
   Alle Blätter unterhalb von Knoten, die von der Wurzel über einen Pfad der Länge genau N erreicht werden können, bilden jeweils Mengen, deren Vereinigung die Menge ∇_0 ergibt.

Beispiel:



3. Erstellen der Funktion F_{-1}:
   Aus den Mengen Δ_0 und ∇_0 kann die Funktion F_{-1} erstellt werden. Dabei ist für ein Element $\delta \in \Delta_{-1}$ ($\delta = \{i\}$ laut Definition) $F(\delta) = \gamma \in \Delta_0 \Leftrightarrow i + 1 \in \gamma$.

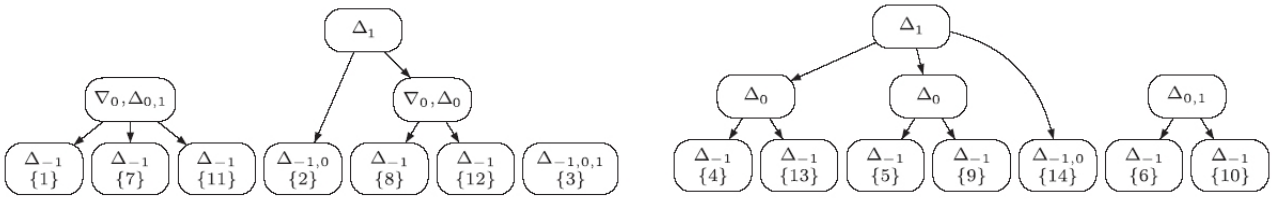
4. Iteration (grobe Vorgehensweise, einige Details wurden vernachlässigt):
   (a) Für jedes Element δ aus ∇_{k-1} ein neues Element ω in ∇_k erstellen, $F(\delta) = \omega$.
   (b) Solange $F^{-1}(\omega) \cap \nabla_{k-1} \neq \emptyset, \gamma \in F^{-1}(\omega) \cup \nabla_{k-1}$:
   Kinderelemente von ω sind Nachfolger der Kinder von γ.
   Nachfolger der Eltern der Vorgänger der Kinder von ω ist ω. $\Rightarrow$ ggf. Iteration
   (c) Falls bei $k = 1$ kein Nachfolger von einem Element aus Δ_0 gesetzt ist, wird der Nachfolger auf den Nachfolger von einem Kind des Elementes gesetzt.
   (d) Iteration mit $k = k + 1$ solange $k < N - 1$ gilt.

Bemerkungen zum Algorithmus:

Der Algorithmus beruft sich auf einige theoretischen Aussagen zu Δ, ∇ und F. Genaueres dazu (inklusive der Beweise) findet sich in dem Paper [2].

Im verbesserten Algorithmus wird immer nur eine Sequenz s betrachtet. Um mehrere Sequenzen zu vergleichen, kann dies mit einer einfachen Veränderung geschehen, indem man vorher die Sequenzen zu

Beispiel:



einer Sequenz zusammenfasst, wobei zwischen den Sequenzen jeweils ein einzigartig paarweise verschiedenes Trennsymbol eingefügt wird. Somit wird gewährleistet, dass nur die einzelnen Sequenzen verglichen werden und es zu keinen Übereinstimmungen von N-Wörtern kommt, die über diese Grenzen hinweg gehen.

Algorithmus:

```
 1: init Δ_{-1} ; {create a node for each position of s}
 2: Δ_0 and ∇_0 ← suffix tree of s$ ;
 3: F_* ← F_{-1} ;
 4: for k = 1 to N − 1 do
 5:    ∇_k ← ∅ ; S_done ← ∅ ;
 6:    while ∇_{k−1} ⊈ S_done do
 7:       Let δ ∈ ∇_{k−1} and δ ∉ S_done ;
 8:       Create a new node ω with no child;
 9:       ∇_k ← ∇_k ∪ ω ;
10:       F_*(δ) ← ω ; {implicitly F_*^{-1}(ω) ← F_*^{-1}(ω) ∪ {δ}}
11:       while F_*^{-1}(ω) ∩ ∇_{k−1} ⊈ S_done do
12:          Let γ ∈ F_*^{-1}(ω) ∩ ∇_{k−1} and γ ∉ S_done ;
13:          S_done ← S_done ∪ {γ} ;
14:          for all β ∈ CHILDREN(γ) do
15:             CHILDREN(ω) ← CHILDREN(ω) ∪ F_*(β) ;
16:             for all α ∈ F_*^{-1}(F_*(β)) do
17:                if α ∈ Δ_{k−1} then
18:                   F_*(α) ← ω ;
19:                else
20:                   F_*(ANCESTOR(α)) ← ω ;
21:                end if
22:             end for
23:          end for
24:       end while
25:    end while
26:    if k = 1 then
27:       for all γ ∈ Δ_0 with F_*(γ) not set do
28:          Let β ∈ CHILDREN(γ) ;
29:          F_*(γ) ← F_*(β) ;
30:       end for
31:    end if
32: end for
```

3.2 Erweiterung des evolutionären Modells

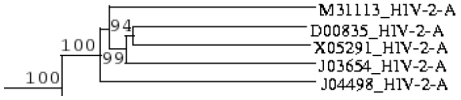
Optionen zur Erweiterung:

- Möglichkeit auch nicht perfekte Übereinstimmungen zu berücksichtigen
- Inversionen berücksichtigen:
  Invertierte Strings in Menge von überlappenden N-Wörtern übernehmen
- weitere Möglichkeiten wären denkbar (ggf. Permutationen betrachten),
  aber: Rechenzeit wird dadurch erhöht

4 Leistung bei HIV/SIV Subtyping

Zunächst wurde das N-lokale Dekodieren auf 70 HIV/SIV Genome angewendet, wobei 4 davon unvollständig und die anderen komplett waren. Es ergaben sich sehr gute Übereinstimmungen der Topologie des resultierenden Baumes mit existierendem Wissen. Der Baum beinhaltete auch bekannte evolutionär signifikante Ereignisse der Entwicklung der HIV/SIV Subgruppen. Jedoch ist die Topologie des Baumes abhängig von der Wahl des Parameters N, wobei für N=13-29 die Topologie gefunden wurde, welche identisch zu dem HIV/SIV Sequenz Compendium 2000 ist [4]. Man kann dadurch feststellen, dass durch geeignete Wahl von N sehr gute Ergebnisse bei den Sequenzvergleichen erzielt werden können, wenn es auf komplette Genome angewendet wird.

Ausschnitt: (mit N=15)



Der Ausschnitt enthält die Bezeichnungen der einzelnen Subgruppen und einen Bootstrap-Wert für jede Verzweigung, wobei nur Werte $> 50\%$ aufgeführt werden.

4.1 Kurze Sequenzen: HIV/SIV *gag*, *pol*, *env* und *nef*

Nun wird das N-lokale Dekodieren auf kurze Sequenzen innerhalb der HIV/SIV Genome angewandt. Dabei sind *gag* (1473-1569 bp), *pol* (2993-3360 bp) *env* (2499-2658 bp) Gene, die Informationen über die strukturellen Proteine für neue Viruspartikel codieren. Das Gen *nef* (292-783 bp) hingegen codiert ein Protein, welches für die effiziente Vermehrung des Virus verantwortlich ist. Es wurden *gag*, *pol*, *env* und *nef* aus 66 HIV/SIV Sequenzen verwendet und die Ergebnisse des lokalen N-Dekodierens stimmen gut mit etablierten HIV/SIV Phylogeniebäumen für diese Regionen überein. Nur bei *nef* treten einige Unterschiede auf, welche möglicherweise darauf zurückzuführen sind, dass beim N-lokalen Dekodieren mehrdeutige Regionen im Gegensatz zum multiplen Alignment berücksichtigt werden.
Als empirisch gute Werte für N haben sich herausgestellt:
gag - N=11-23; *pol* - N=11-30; *env* - N=12-24 und *nef* - N=11-20

4.2 Non-coding Long Terminal Repeats (LTR)

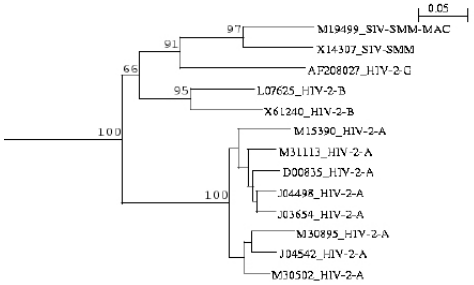
Die Enden von jedem HIV/SIV RNA-Strang enthalten eine RNA-Sequenz, die Long Terminal Repeat (LTR) genannt wird. Die Regionen in den LTR agieren als Schalter zur Kontrolle der Produktion neuer Viren und können sowohl von HIV als auch von der Wirtszelle angesteuert werden. Es wurden 43 non-coding LTR-Regionen untersucht, welche viele Duplikationen, Insertionen sowie Deletionen beinhalten und somit schwer mit herkömmlichen alignment-basierten Vergleichen analysiert werden können. Deshalb sind keine Referenzbäume bzw. Einteilungen konkret vorhanden, sodass versucht wurde, diese mit Hilfe von CLUSTAL-W sowie DIALIGN-2 zu erstellen. Jedoch brachte der Abgleich der gewonnenen Bäume mit bekanntem Wissen schlechte Ergebnisse, da CLUSTAL-W zu schlechter Trennung der HIV-1, HIV-2 und SIV-Sequenzen führte und DIALIGN-2 die Subtypen von HIV-1 schlecht auftrennte (z.b. HIV-1 N ist innerhalb von HIV-1 M). Die Performance von alignment-basierten Methoden war erwartungsgemäß schlecht.
Durch N-lokales Dekodieren mit N=10-21 wurden die HIV/SIV Subtypen gut voneinander getrennt, wobei jedoch Referenzen fehlen, um den erstellten Baum genauer und besser einzuschätzen.

5 Abschießende Bemerkungen

- Erweiterung auf komplexeres evolutionäres Modell wäre sinnvoll

- Wahl von Parameter N entscheidend für sinnvolle Ergebnisse:
  N=13-20 sind gute empirische Standardwerte,
  aber: weitere Untersuchungen notwendig

Ausschnitt aus dem Baum von N-lokalen Dekodieren mit N=11:



- Berücksichtigung auch von mehrdeutigen Regionen (z.b. LTR)
  aber: Problem der Validierung
- Implementation des verbesserten Algorithmus verfügbar [5]

Literatur

[1] Didier G, Debomy L, Pupin M, Laprevotte I,
Comparing sequences without using alignments: application to HIV/SIV subtyping, BMC Bioinformatics. 2007 Jan 02

[2] Didier G, Laprevotte I, Pupin M, Hénaut A,
Local decoding of sequences and alignment-free comparison,
J Comp Biol. 2006, 13: 1465-1476

[3] Felsenstein J,
PHYLIP (Phylogeny Inference Package) version 3.6,
Department of Genome Sciences University of Washington, Seattle. 2005

[4] *The 2000 HIV Sequence Compendium*,
http://www.hiv.lanl.gov/content/hiv-db/COMPENDIUM/2000/HIV12SIVcomplete.pdf

[5] *SCeNE - Prototypimplementierung des N-lokalen Dekodierens*,
http://iml.univ-mrs.fr/∼didier/laprevot/