Teil 2

BWBBLE-Algorithmus

BWBBLE-Algorithmus

- Neue Methode zur Kompression von Genomsammlungen
  - Geeignet zur Verwaltung von Variationen
- Design eines neuen Alignment Algorithmus (Burrows–Wheeler transform)
  - Mapped Sequenzen eines neuen Genoms
  - Auf eine beliebig große Menge von Genomen
  - Genauigkeit sehr hoch
  - Keine Verfälschung durch Grundgenomableitung

Konzept

- Ersetzen des 4er ATCG-Nucleotidcodes
- Varianten (Insertionen und Deletionen) werden als „Bubbles" dargestellt
- Danach verwenden des Burrow Wheeler Transformation
- BWT wird jedoch um Varianten(IUPAC) erweitert und mit Graycode effizient umgesetzt
- Exaktes Matching und Inexaktes Matching

IUPAC statt ATCG

- Beispielgenome

AACTGGTAT••TTTTA

ACCGGGTATATTTTTA

AACGGG •••••TTTTA

IUPAC statt ATCG

- Variationen der Basen an einzelnen Stellen

AACTGGTAT••TTTTA

ACCGGGTATATTTTTA

AACGGG •••••TTTTA

Erweitertes IUPAC-Alphabet

- Erweiterung des 4er ACTG-Nucleotidcodes auf ein 16 IUPAC-Alphabet

Base	IUPAC	Base	IUPAC
-	#	A|C	M
A	A	A|C|G|T	N
C|G|T	B	A|G	R
C	C	C|G	S
A|G|T	D	T	T
G	G	A|C|G	V
A|C|T	H	A|T	W
G|T	K	C|T	Y

- Beim Matchen eines Reads können durch die neuen Buchstaben auch verschiedene Basen an die selbe Stelle gematcht werden

IUPAC statt ATCG

- Variationen der Basen an einzelnen Stellen

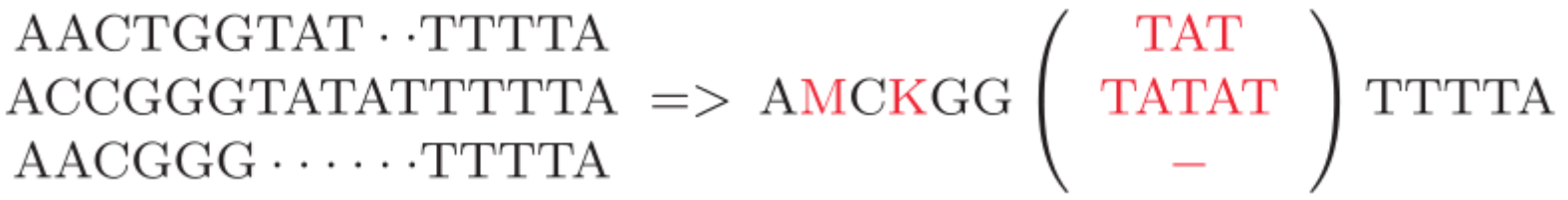
AACTGG(TAT••)TTTTA

ACCGGG(TATAT)TTTTA => AMCKGG...

AACGGG(•••••)TTTTA

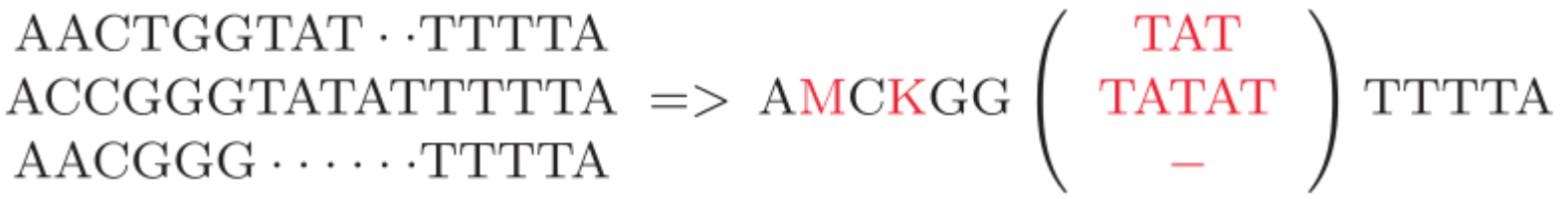
Variationen in der Länge

- Variationen durch Insertionen und Deletionen sind auch möglich (**Indels**)
- Variationen an einer Stelle durch Indels kann visuell als „Bubble“(Blase) dargestellt werden
- Bestehend aus multiplen Branches (Verzweigungen)



Bubble

- Sequenzvarianten (Varianten der Sequenzlängen) werden durch „Bubbles“ dargestellt
- Jede Bubble enthält alle Variationen als Branches
- Jede Branch steht für eine Genomvariation



Bubble

- Bubbles werden „entrollt“ hinten an das Genom ankopiert und mit einem Übergangsbereich der Länge R links und rechts versehen

Burrow-Wheeler-Transformation (BWT)

Beispielcode RWYAYA

pos	
0	RWYAYA$
1	WYAYA$R
2	YAYA$RW
3	AYA$RWY
4	YA$RWYA
5	A$RWYAY
6	$RWYAYA

i	*SA[i]*	*BWT[i]*
0	6	$RWYAY **A**
1	4	YA$RWY **A**
2	2	YAYA$R **W**
3	0	RWYAYA **$**
4	1	WYAYA$ **R**
5	5	A$RWYA **Y**
6	3	AYA$RW **Y**

- Permutationen des Eingabezeichen
- Lexikalische Ordnung

[1] Ferragina and Manzini (2000)

Burrow-Wheeler-Transformation (BWT)

Beispielcode RWYAYA

pos	
0	RWYAYA$
1	WYAYA$R
2	YAYA$RW
3	AYA$RWY
4	YA$RWYA
5	A$RWYAY
6	$RWYAYA

i	*SA[i]*	*BWT[i]*
0	6	$RWYAY **A**
1	4	YA$RWY **A**
2	2	YAYA$R **W**
3	0	RWYAYA **$**
4	1	WYAYA$ **R**
5	5	A$RWYA **Y**
6	3	AYA$RW **Y**

Backward-Suche mit Hilfe des Suffix-Arrays ermöglicht O(|P|) Zeit für die Suche [1]

[1] Ferragina and Manzini (2000)

Exaktes Matching

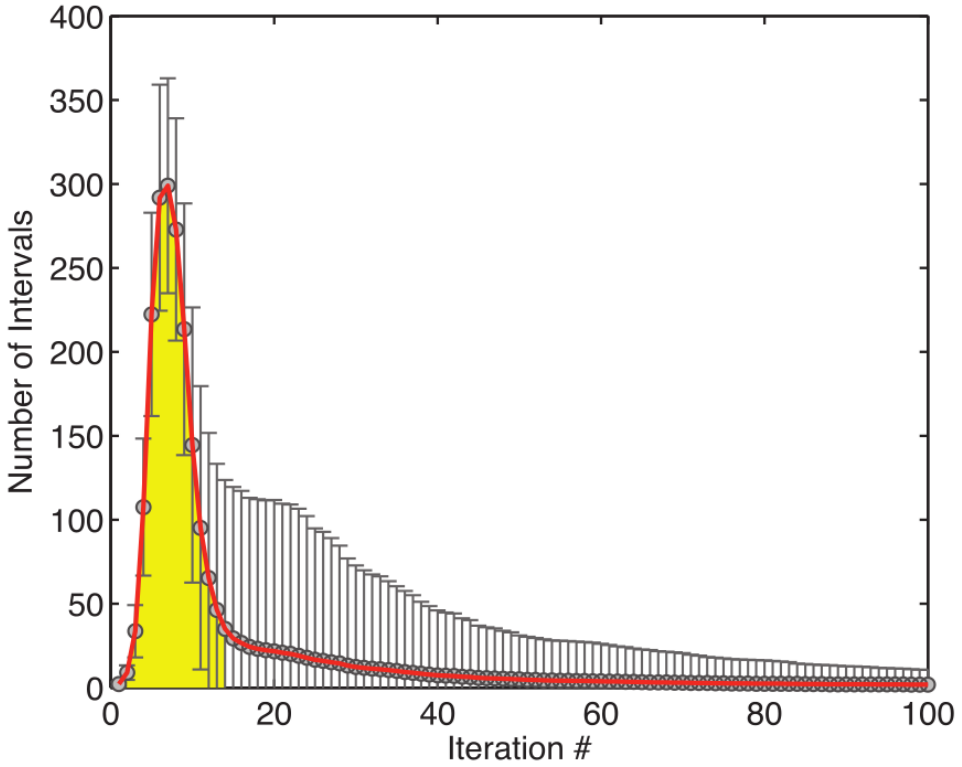
- Formel für exaktes Matching
- BWT gilt noch für das erweiterte Alphabet
- Iterative Backward-Suche funktioniert auch für unsere Multi-genome

$$\langle L(\alpha R), U(\alpha R)\rangle = \bigcup_{\forall \theta_\alpha \in \Theta_\alpha} [L(\theta_\alpha R), U(\theta_\alpha R)] \qquad (3)$$

$$L(\theta_\alpha R) = C(\theta_\alpha) + O(\theta_\alpha, L(R) - 1) + 1 \qquad (4)$$

$$U(\theta_\alpha R) = C(\theta_\alpha) + O(\theta_\alpha, U(R)) \qquad (5)$$

Exaktes Matching



- Meisten Intervalle werden in den 12–14 Iterationen der Backwardsuche erstellt
- Algorithmus lässt sich mit einem Preprocessing boosten
  - Vorberechnung der SA-Intervall-Substrings der Länge 12-14

Anpassung des BWT - Graycode

- Lexikalische Ordnung der IUPAC(Varianten)
- Ordnung ermöglicht weniger Suffix-Array-Intervalle
- Effiziente Abfrage

AMCKGG..

Gray code	Base	IUPAC	Gray code	Base	IUPAC
0000	-	#	1100	A|C	M
0001	T	T	1101	A|C|T	H
0011	G|T	K	1111	A|C|G|T	N
0010	G	G	1110	A|C|G	V
0110	C|G	S	1010	A|G	R
0111	C|G|T	B	1011	A|G|T	D
0101	C|T	Y	1001	A|T	W
0100	C	C	1000	A	A

Inexaktes Matching

- SA-Intervalle werden für alle IUPAC-Zeichen durchgeführt
- Weiterschreiten bei matches/mismatches
- Nicht für Deletionen
- Insertionen
  - Skippen der Leseposition
  - ohne Neuberechnugn der SA-intervalle


```
AlignRead(R, G, n)
    return InexactMatch(R, |R| − 1, n, 0, |G| − 1)

InexactMatch(R, i, n, L, U)
    if i < 0
        return {[L, U]}
    if n < 0
        return ∅
    S ← ∅
    // Insertions
    S ← S ∪ InexactMatch(R, i − 1, n − 1, L, U)
    for c ∈ IUPAC_ALPHABET
        L ← C(c) + O(c, L − 1) + 1
        U ← C(c) + O(c, U)                    } Berechnung SA
        if L ≤ U
            // Deletions
            S ← S ∪ InexactMatch(R, i, n − 1, L, U)
            if GrayCode[c] & GrayCode[R[i]] > 0
                // Match
                S ← S ∪ InexactMatch(R, i − 1, n, L, U)
            else
                // Mismatch
                S ← S ∪ InexactMatch(R, i − 1, n − 1, L, U)
    return S
```

Evaluation

- Speicherverbrauch
  - 16 n log n bits vs 4n log n bits (BWT + 16ner Alphabet)
  - Referenzgenom 2,8GB groß
  - Multigenom mit 3,2GB nicht allzuviel größer.

- Und die Laufzeitskalierung.
  - Suche in den 1000 Genomen dauert etwa 100 mal so lange
  - **Skalierung** mit mehr Genomen ist bei dem BWBBLE-Algoritmus deutlich besser als linear!

Evaluation - Laufzeit