

# Bioinformatik

Exklusionsmethoden  
BLAST

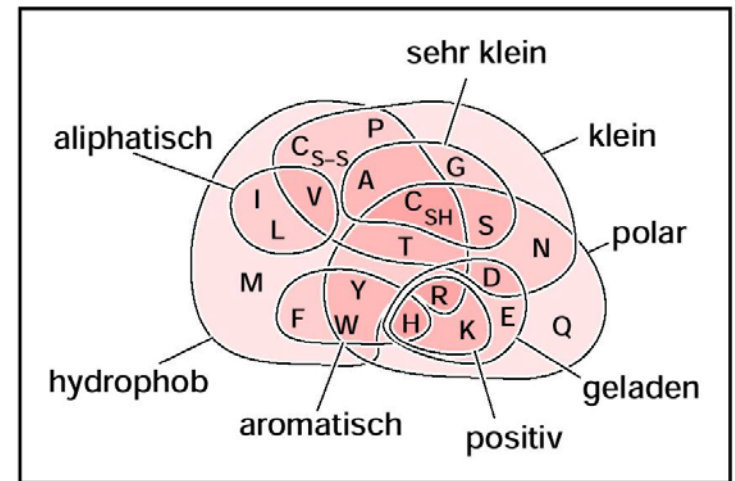
Ulf Leser

Wissensmanagement in der  
Bioinformatik



# Hintergrund

- Schon öfters angesprochen ...
  - Ähnlichkeitsmatrizen, Substitutionsmatrizen, Scorefunktionen, ...
- Ersetzung einer Base/Aminosäure durch eine andere hat **unterschiedliche Bedeutung**
  - Aminosäuren
    - Ersetzung mit „sehr ähnlichen“ Aminosäuren ändert Proteinstruktur in der Regel kaum
    - Ersetzung mit „wenig ähnlichen“ Aminosäuren kann Struktur vollkommen ändern



# Ist das notwendig?

---

Code	Häufigkeit	Mutierbarkeit
L	0.091	54
A	0.077	100
G	0.074	50
S	0.069	117
V	0.066	98
E	0.062	77
K	0.059	72
T	0.059	107
I	0.053	103
D	0.052	86
P	0.051	58
R	0.051	83
N	0.043	104
Q	0.041	84
F	0.040	51
Y	0.032	50
M	0.024	93
H	0.023	91
C	0.020	44
W	0.014	25

- Häufigkeiten der Ersetzung einer Aminosäure durch irgendeine andere AA im Verhältnis zu allen Ersetzungen
- Alanin (A) willkürlich als 100% gesetzt
- **Keine Gleichverteilung**
- Mutationen sind mehr oder weniger erfolgreich, je nachdem, welche AA ersetzt wird
  - Besser: Mutationen werden durch Selektion mehr oder weniger geduldet
  - Tryptophan (W) sehr selten (25)
  - Serin (S) sehr häufig (117)

# Woher nehmen?

---

- Wie kann man sinnvolle Werte für die Matrix bestimmen?
  - Wir wollen **Ähnlichkeit der biologischen Bedeutung** messen
  - Für Aminosäuren benötigt man ~200 (verschiedene) Werte
- Möglichkeit 1: Chemische Eigenschaften
  - Ladung, Größe, Polarität, ...
  - Viele Faktoren mit unklaren Gewichten
  - Wie soll man das durch **ein Bewertungsschema** ausdrücken?
  - Keine Verwendung in der Praxis
- Möglichkeit 2: Beobachtung
  - **Beobachtung der Evolution** statt analytischer Vorhersage
  - Lernen aus Beispielen, also „tatsächlich“ vorgekommener Mutationen
  - Benötigt Beispieldaten: Paare von homologen Sequenzen

# PAM als Sequenzabstand

---

- Definition

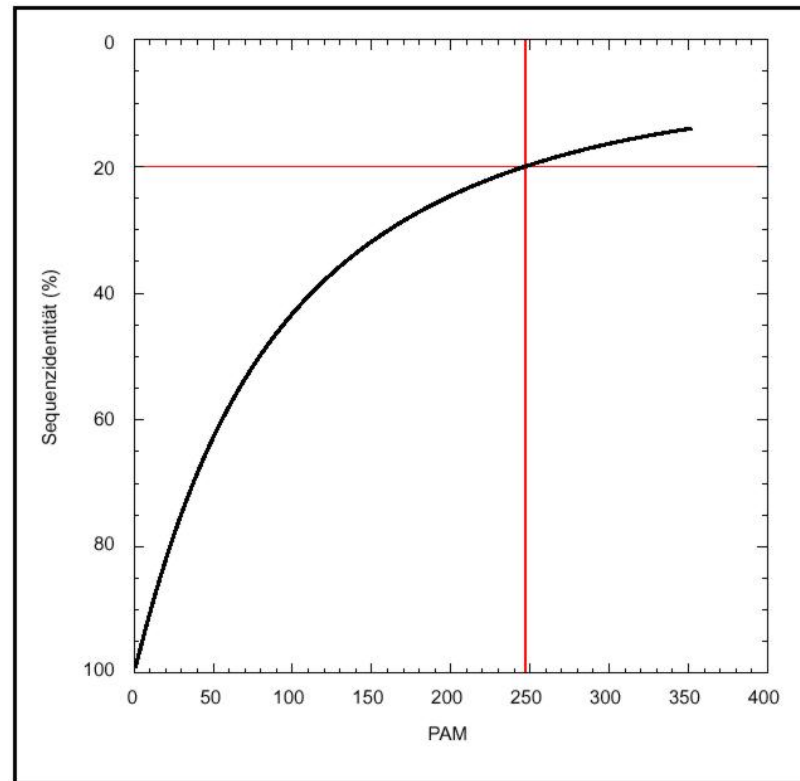
*Seien  $S_1$  und  $S_2$  zwei Proteinsequenzen mit  $|S_1|=|S_2|$ .  $S_1$  und  $S_2$  heißen  $x$  PAM entfernt, wenn  $S_1$  in  $S_2$  überführt wurde mit  $x$  Punktmutationen pro 100 Aminosäuren*

- Eigenschaften

- PAM beachtet keine Inserts und Deletions
- $x$  schätzt man aus den Unterschieden von  $S_1$  und  $S_2$ 
  - Kann man analytisch berechnen oder durch Simulation bestimmen
- 50 PAM Abstand heißt nicht 50 Veränderungen pro 100 Aminosäuren

# PAM Abstand und Sequenzidentität

---



- Jenseits von PAM 250: Rauschen

# PAM Matrizen

---

- Definition

Seien  $(S_{1,1}, S_{2,1}), \dots, (S_{1,n}, S_{2,n})$  Paare von Sequenzen, die jeweils  $x$  PAM entfernt sind. Dann ist die **PAM- $x$  Matrix  $M_x$**  wie folgt definiert:

- Sei  $f(A_i)$  die absolute Häufigkeit der Aminosäure  $A_i$
- Seien alle Paare optimal aligniert
  - Sei  $S_{k,i}'$  das  $S_{k,i}$  mit den durch das Alignment eingefügten Leerzeichen
- Sei  $f(i,j)$  die **relative Übergangshäufigkeit** von  $A_i$  zu  $A_j$  in allen alignierten Paaren
  - Anzahl von Positionen  $k$  mit  $S_{1,z}'[k]=A_i$  und  $S_{2,z}'[k]=A_j$  oder umgekehrt
    - Übergang ist „richtungslos“:  $f(i,j) = f(j,i)$
    - Paare  $(A_x, -)$  werden ignoriert
  - $f(i,j)$  wird auf die Gesamtzahl aller Paare ohne INSDEL normiert
- Damit:

$$M_x(i, j) = \log \left( \frac{f(i, j)}{f(i) * f(j)} \right)$$

# Reale PAM Matrizen

---

- Annahme der „ **Molecular Clock Theory**“
  - Evolution verläuft gleichmäßig in der Zeit und in den Sequenzpositionen
  - Damit möglich: Hochrechnung der Matrixeinträge für große  $x$  aus der Matrix für kleine  $x$
- Vorgehen von Dayhoff et al.
  - Paare eng verwandter Sequenzen auswählen
    - >85% Identität, 34 Proteinfamilien
  - Manuell alignieren
  - PAM-1 Matrix  $M_1$  aus Häufigkeiten berechnen
  - PAM- $x$  Matrizen berechnen durch:  $M_x = (M_1)^x$

# Verwendung

---

- Welche PAM Matrix soll man zur Alignierung zweier Sequenzen verwenden?
  - Die, die dem PAM-Abstand der Sequenzen entspricht
  - Den kennt man aber nicht – schätzen
  - Schätzung benötigt Alignments
    - Zur Berechnung der Sequenzidentität
  - Alignments basieren auf Substitutionsmatrizen
  - **Henne – Ei Problem**
- Also
  - Iteratives Verfahren einsetzen und auf Konvergenz warten
  - Verschiedene Matrizen testen
  - Externes Wissen (Chemie, Domänen, etc.) hinzuziehen

# Vorarbeiten

---

- PROSITE

- Beschreibung funktionaler (=konservierter) Bereiche in [homologen Proteinsequenzen](#) durch reguläre Ausdrücke
- Expertenwissen - manuelle Pflege der Datenbank am EBI

- BLOCKS

- Alignierung von durch PROSITE Ausdrücke gematchten Sequenzen in [Multiple Alignments](#) (MSA)
  - Multiple Sequence Alignment (später mehr)
  - Heute: Verwendung weiterer Domänen aus PRINTS, PFAM, ...
- Ein [BLOCK](#) ist ein zusammenhängendes und hochgradig konserviertes Stück aus einem MSA

```
FMYMFYVPL PQ QVY
FYQDF VQLYP MFQV
FMY YUVQQP UMUQ
```

# Unterschiede PAM - BLOSUM

---

- BLOSUM verwendet nur hochkonservierte Bereiche, PAM komplette Alignments
- PAM rechnet große evolutionäre Abstände nur hoch, BLOSUM verwendet gezielt entfernte Sequenzen
- BLOSUM basiert auf deutlich mehr Sequenzen
- Heutige BLOSUM-Matrizen sind heuristisch verbessert
  - „Feedback-Schleife“: Mit initialer BLOSUM 62 Matrix erneute Alignierung
  - Bestimmung der BLOCKS verwendet BLOSUM Matrix
- BLOSUM-62 heute meist der Default in Alignmentprogrammen

# Inhalt dieser Vorlesung

---

- Exklusionsmethoden
  - Auf dem Weg zur Datenbanksuche
- BLAST: Basic Local Alignment Search Tool
  - Man kann Googlen und Blasten

# Exklusionsmethode BYP

---

- Alignment zweier Strings  $A, B$  dauert  $O(n \cdot m)$
- K-Band Algorithmus benötigt  $O(sn^2 - vn)$  für  $|A| = |B|$ 
  - Gutes Verfahren, wenn nur sehr ähnliche Sequenzen interessant sind
- Anderes Vorgehen zur Beschleunigung: Wir lassen nur **k Unterschiede** (Mismatch oder Leerzeichen) zu
- Definition
  - *Ein **k-Difference Vorkommen** von  $P$  in  $T$  ist ein Substring  $T'$  von  $T$ , der ein Alignment mit  $P$  mit Abstand höchstens  $k$  hat*
- Gesucht: Algorithmen, die alle **k-Difference Vorkommen** von  $P$  in  $T$  für geeignete  $k$  in  **$O(m)$  Laufzeit im Average Case** finden
  - Das ist weder echtes globales noch lokales Alignment, sondern approximative Substringsuche

# Exklusionsmethoden

---

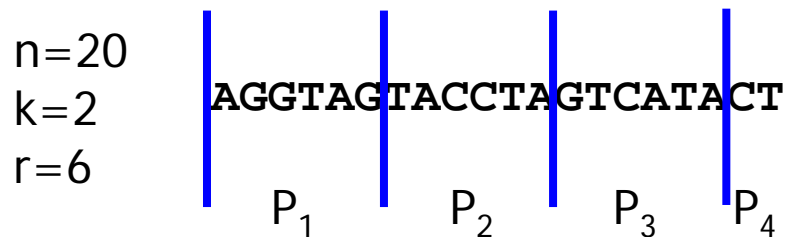
- Es gibt eine ganze Reihe Algorithmen, die das erreichen
- Wir sehen uns näher an
  - BYP: Baeza-Yates, Perleberg: „Fast and practical approximative string matching“, LNCS 664, 1992
- Grundaufbau
  - Wir suchen k-Difference Vorkommen von P in T
  - **1. Partition**: Partitioniere P geschickt in Teilstrings
  - **2. Search**: Suche Partitionen in T mit einem exakten Algorithmus
    - Partitionierung ist so gewählt, dass jedes potentielle k-Difference Vorkommen von P in T um die Fundstelle mindestens einer Partition liegen muss
    - Aber: Nicht jede Fundstelle ist Kern eines k-Difference Vorkommens
  - **3. Check**: Überprüfe alle Vorkommen von Partitionen von P in T
- Warum sparen wir?
  - Im **Average Case** müssen nur wenige Bereiche von T näher (und teuer) untersucht werden
  - Die anderen Bereiche überspringt man bei der (linearen) Search-Phase

# Partitionierung

---

- Lemma

*Sei  $T'$  ein  $k$ -Difference Vorkommen von  $P$  in  $T$ . Sei  $Z = \{P_1, \dots, P_{k+2}\}$  eine Zerlegung von  $P$  in  $k+1$  Teilstrings der Länge  $r = \text{floor}(n/(k+1))$  und einen Reststring. Dann muss  $T'$  mindestens eine Partition von  $P$  der Länge  $r$  exakt enthalten*



# Beobachtung

---

- Lemma

*Sei  $T'$  ein  $k$ -Difference Vorkommen von  $P$  in  $T$ . Sei  $P$  partitioniert wie beschrieben. Dann **muss  $T'$  mindestens eine Partition von  $P$  der Länge  $r$  exakt enthalten***

- Beweis

- Nehmen wir an, dass  $T'$  existiert und dass  $P$  und  $T'$  optimal aligniert sind
- Jede der  $k+1$  Partition  $P_i$  von  $P$  der Länge  $r$  aligniert mit einem Substring  $T'_i$  von  $T'$
- Wenn in jedem dieser  $k+1$  Teilalignments mindestens ein Unterschied wäre, dann wären im Alignment  $P$  mit  $T'$  **mindestens  $k+1$  Unterschiede**
- Dann wäre  $T'$  kein  $k$ -Difference Vorkommen
- Also muss mindestens eines der Teilalignments fehlerfrei sein
- qed.

- Bemerkungen

- Die letzte Partition ignorieren wir
- $r$  ist genau so gewählt, dass **mindestens** ein Teilstück fehlerfrei sein muss

# BYP - Search

---

- Wir suchen alle exakten Vorkommen von den  $k+1$  ersten Partitionen von  $P$  in  $T$ 
  - Wenn wir eine finden, **kann** dort ein  $k$ -Difference Vorkommen von  $P$  in  $T$  liegen, muss aber nicht
- Wie machen wir das geschickt?
- Möglichkeit 1: **Suffixbäume**
  - Baue einen Suffixbaum für  $T$  in  $O(m)$
  - Durchsuche den Baum mit allen Partitionen
  - Zusammen:  $O(m+n)$  (Eigentlich:  $O(m+n+k^*)$ ,  $k^*$  Anzahl Vorkommen)
  - Aber – dafür brauchen wir viel Platz
- Möglichkeit 2: **Aho-Corasick** Algorithmus
  - Baue einen Keyword Tree für die  $k+1$  Partitionen
  - Durchsuche damit  $T$  in  $O(m)$
  - Zusammen:  $O(n+m)$
- In jedem Fall in  $O(n+m)$  möglich

# BYP - Check

---

- Sei  $i$  die Startposition einer Partition  $P'$  von  $P$  in  $T$
- Wir müssen testen, ob in  $T$  um  $i$  herum ein  $k$ -Difference Vorkommen von  $P$  liegt
- Ein solches Vorkommen kann im schlimmsten Fall  $n+k$  Zeichen lang sein
  - Und muss  $P'$  enthalten
- Also alignieren wir  $T[i-n-k .. i+n+k]$  mit  $P$
- Komplexität:  $O(n^2)$

# Alles zusammen

---

- Kosten
  - Partition ist umsonst
  - Search ist  $O(n+m)$
  - Check ist  $O(n^2)$  für jedes Vorkommen einer Partition
- Welche Average Case Komplexität ergibt sich daraus?
  - Dazu betrachten wir nur den letzten Schritt – die anderen halten ja ungefähr unsere Zielkomplexität  $O(m)$
- Annahmen
  - Sei  $T$  eine zufällige Sequenz über einem Alphabet der Größe  $s$ 
    - Alle Zeichen tauchen mit der selben Wahrscheinlichkeit auf
  - Schwierigkeit jeder Average Case Analyse – was ist ein Average Case?
- Folgerung
  - Die Wahrscheinlichkeit, dass ein konkreter String der Länge  $r$  an einer Position  $i$  in  $T$  beginnt ist  $1/s^r$

# BYP – Komplexität

---

- Theorem  
*Für Strings  $P$  und  $T$  mit  $|P|=n$  und  $|T|=m$  und  $k \sim n/\log_s(n)$  findet der BYP Algorithmus im *Average Case* alle  $k$ -Difference Vorkommen von  $P$  in  $T$  in  $O(m)$* 
  - *Average Case: Zufällige, zeichenweise gleichverteilte Sequenzen und Pattern*
- Beweis
  - Sei  $P'$  eine der  $k+1$  Partitionen von  $P$  der Länge  $r$ 
    - $T$  enthält  $m$  **potentielle Startpositionen** von  $P'$
    - Damit ist  $P'$  im Schnitt  $m/s^r$  Mal in  $T$  enthalten
  - Über alle  $k+1$  Partitionen erwarten wir damit  $(k+1) \cdot m/s^r$  Vorkommen einer Partition in  $T$
  - Für jede brauchen wir  $O(n^2)$ ; die Gesamtlaufzeit wollen wir aber in  $O(c \cdot m)$  halten ( $c$ : beliebige Konstante). Also

$$\frac{mn^2(k+1)}{s^r} < cm$$

# BYP – Komplexität Fortsetzung

---

- Beweis Fortsetzung

- $k$  ist immer kleiner als  $n$ ; damit suchen wir den Punkt mit

$$\frac{mn^3}{s^r} = cm$$

- Es folgt  $s^r = n^3/c$  und damit  $r = \log_s(n^3) - \log_s(c)$
- Da  $r = \text{floor}(n/(k+1))$  gewählt wurde, können wir einsetzen

$$\frac{n}{k+1} = \log_s n^3$$

- Auflösen nach  $k$  ergibt

$$k \approx \frac{n}{\log_s n^3} \approx \frac{n}{3 \log_s n}$$

- q.e.d.

# Zusammenfassung

---

- Durch zusätzliche Annahmen können wir schnellere Algorithmen bauen
  - Sehr ähnliche Sequenzen: K-Band Algorithmus
  - k-Differences: BYP Algorithmus
- BYP implementiert ein **Datenbanksuchproblem**:  
T viel größer (oder viel mehr) als P
- Grundidee der Exklusionsmethoden wird in den heuristischen DB-Suchverfahren verallgemeinert
  - Keine Fehlergrenze vorgeben
  - Dafür auch nicht unbedingt alle Matches finden
  - Dafür noch viel schneller sein

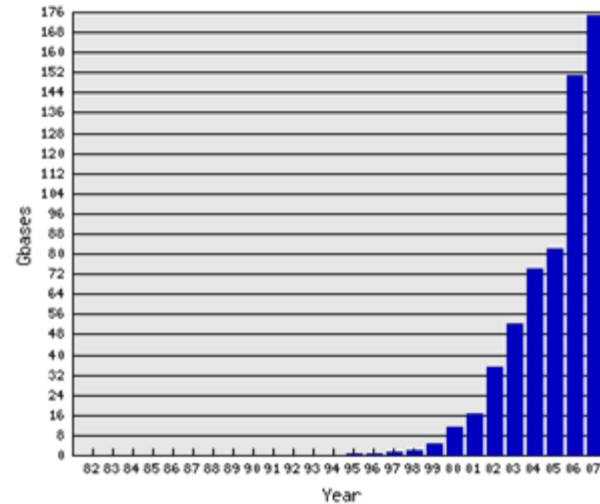
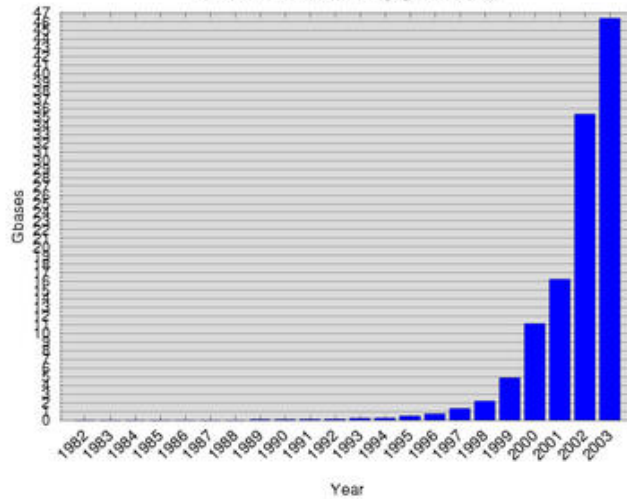
# Inhalt dieser Vorlesung

---

- Heuristische Algorithmen zum lokalen Alignment
  - BLAST
  - BLAT (später)

# Heuristische Alignierung

EMBL Database Growth  
total nucleotides (gigabases)



- Annotation neuer Sequenzen basiert auf Suche nach homologen Sequenzen in [Sequenzdatenbanken](#)
- Datenmenge wächst **exponentiell** – selbst lineare Algorithmen sind zu langsam
- Gesucht sind schnelle Verfahren
  - Auch wenn wir dabei ein paar (schlechte?) Ergebnisse verlieren

# Suche in Datenbanken

---

- Gedankenkette
  - Gegeben Sequenz  $S$  und Datenbank mit Sequenzen  $S_1, \dots, S_n$
  - Gesucht: Welche Sequenzen in DB sind homolog zu  $S$ ?
    - Kann nicht beantwortet werden
  - Annäherungen
    - Welche Sequenzen in DB sind sehr ähnlich zu  $S$ ?
      - Was heißt ähnlich?
    - Welche Sequenzen haben einen **hohen Alignment-Score** zu  $S$ ?
      - Berechnung Alignmentsscore dauert zu lange
    - Näherung: Welche Sequenzen in DB haben **wahrscheinlich einen hohen Alignmentsscore** zu  $S$ ?
      - BLAST
  - Was heißt hier „wahrscheinlich“?

# Sensitivität und Spezifität

---

		Reality	
		+	-
Prediction	+	TruePositive (TP)	FalsePositive (FP)
	-	FalseNegative (FN)	TrueNegative (TN)

- **Spezifität** =  $TP / (TP + FP)$  (Precision)
  - Wie viele der Treffen des Verfahrens sind wirklich welche?
- **Sensitivität** =  $TP / (TP + FN)$  (Recall)
  - Wie viele der echten Treffer findet das Verfahren?
- Oftmals **eine Balance**
  - Algorithmen berechnen einen Score pro Sequenz
  - Hoher Score – Positiv; Niedriger Score – Negativ
  - Wenn Score mit Wahrscheinlichkeit für korrekte Klassifikation korreliert, folgt daraus
    - Ergebnismenge klein: SP=hoch, SE=klein
    - Ergebnismenge groß: SP=niedrig, SE=hoch



# Beispiel

---

- Gegeben
  - Sequenz S und Datenbank mit 10.000 Sequenzen
- Algorithmus findet
  - 15 Sequenzen homolog zu S
- In Wahrheit
  - 20 Sequenzen homolog zu S
  - 10 davon hat der Algorithmus gefunden

	Real: Positive	Real: Negative
Alg: Positive	TP = 10	FP = 5
Alg: Negative	FN = 10	TN = 9.975

- Spezifität =  $TP / (TP + FP) = 10 / 15 = 66\%$
- Sensitivität =  $TP / (TP + FN) = 10 / 20 = 50\%$

# BLAST

---

- Altschul, Gish, Miller, Myers, Lipman: „Basic Local Alignment Search Tool“, J Mol Bio, 1990.
  - Heuristischer Suchalgorithmus
  - Datenbanksuche mit lokalem Alignment
  - Sehr schnell, findet aber nicht alle optimalen Alignments
- **\*\*Die\*\* Erfolgsgeschichte der Bioinformatik**
  - Für Biologen teilweise äquivalent zu „Bioinformatik“
  - Eingesetzt auf NCBI/EBI Server
  - Software frei erhältlich
- Diverse Weiterentwicklungen
  - Gapped-BLAST und PSI-BLAST (später), MegaBlast, BLAST-ALL, PATHBLAST, Name-BLAST, ...

# BLAST Varianten

---

- BLAST gibt es in verschiedenen Ausprägungen
  - Blastn : DNA-Anfrage/DNA-Datensammlung
  - Blastp : Protein-Anfrage/Protein-Datensammlung
  - Blastx : translatierte DNA-Anfrage/Protein-Daten
  - Tblastn : Protein-Anfrage/translatierte DNA-Daten
  - Tblastx: translatierte DNA-Anfrage/DNA-Daten
- Bei DNA/Proteinsuche immer Suche **mit allen sechs Reading Frames**
- Prinzip immer identisch
- Suche in Proteinen läuft etwas anders als in DNA
  - Später

# BLAST Parameter

---

- Zunächst
  - Suche nach guten lokalen Alignments in DNA Sequenzen
- Gegeben
  - Suchsequenz  $P$ , Datenbank  $DB = \{S_1, \dots, S_n\}$
  - Substitutionsmatrix  $M$
  - Parameter  $w$ : Länge der „Seeds“
  - Parameter  $t$ : Initialer Schwellwert
  - Parameter  $c$ : Gesamtschwellwert
    - Wird berechnet in Abhängigkeit von  $t$ ,  $M$ ,  $|DB|$ ,  $|P|$
  - Parameter  $v$ : Erwünschte Anzahl Treffer
    - Blast berechnet die  $v$  ähnlichsten Subsequenzen

# BLAST Schritt 1 und 2

---

- Schritt 1
  - Bestimme alle **Teilwörter**  $P_1, \dots, P_m$  der Länge  $w$  in  $P$ 
    - Mit Überlappung – keine Partitionierung
    - Wie viele gibt es?
- Schritt 2
  - Suche nach **Hits** von  $P_1, \dots, P_m$  in DB mit **Score über  $t$** 
    - Hits müssen nicht exakt sein
    - Vergleiche alle  $P_i$  mit allen Teilwörtern in DB der Länge  $w$  in  $T$ 
      - Keine INSDELS, Verwendung von  $M$
  - Durch den „unscharfen“ Hit mit Schwellwert  $t$ 
    - werden auch Hits gefunden, die keine perfekten Matches sind
    - werden **wenig aussichtsreiche Positionen** in DB ausgeschlossen
      - Annahme: Ein gutes lokales Alignment  $A$  in Sequenz  $S$  muss mindestens einen Hit enthalten
      - Es ist nicht notwendig, dass jedes Teilwort in  $A$  mit seinem Teilwortpartner in  $P$  einen Hit hat.

# BLAST Schritt 3

---

- Schritt 3
  - Für jeden Hit H zwischen DB-Sequenz S und  $P_i$
  - **Verlängere Bereich** um H sowohl in P als auch in S
    - Gapfree Alignment nach links und rechts wachsen
    - Solange, bis
      - Sequenz P oder S zu Ende ist, oder
      - Alignmentsscore fällt unter geschätzten Schwellwert  $c$ , oder
      - Alignmentsscore fällt „signifikant“ unter bisherige  $v$  beste Treffer
        - » „Signifikant“ heuristisch bestimmt
    - Ergibt „**Maximal Segment Pairs (MSP)**“
    - Die besten  $v$  MSP sind das Ergebnis

# Beispiel

$W=5, t=5, \text{Kosten: } M=+1, R=-3$   
 $P=ACGTGATA$   
 $S=GATTGACGTGACTGCAAGTGATACTATAT$

Schritt 1  
Teilwörter



$P_1=ACGTG$   
 $P_2=CGTGA$   
 $P_3=GTGAT$   
 $P_4=TGATA$

$GATTGACGTGACTGCAAGTGATACTATAT$   
 $GATTGACGTGACTGCAAGTGATACTATAT$   
 $GATTGACGTGACTGCAAGTGATACTATAT$



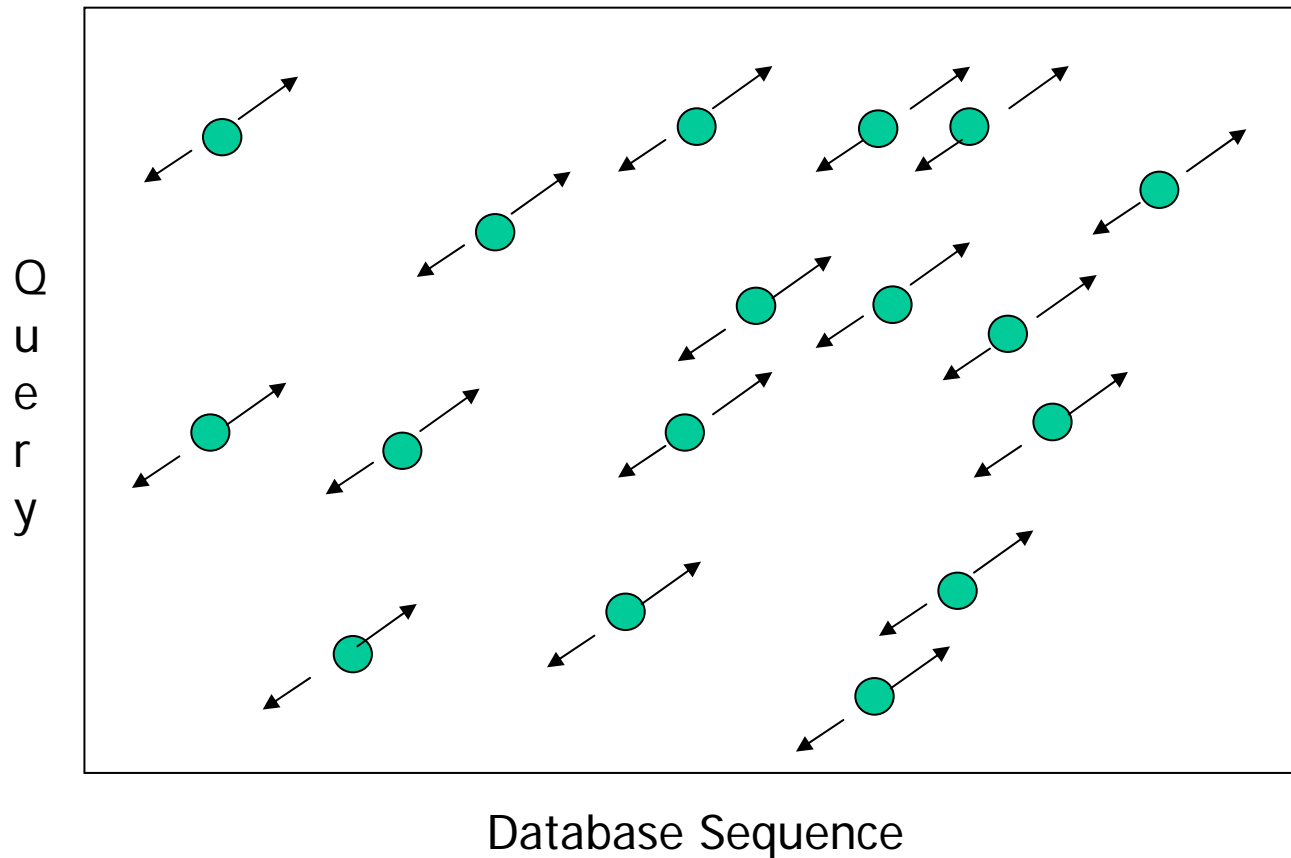
Schritt 2  
Hitsuche

Schritt 3  
Verlängerung



$GATTGACGTGACTGCAAGTGATACTATAT$	
$ACGTGATA$	5
$ACGTGATA$	$5+1=6$
$ACGTGATA$	$6-3=3$
...	...

# Veranschaulichung



- Original BLAST – kein Zusammenfügen mehrerer MSP
- Keine Beachtung von Gaps

# BLAST für Proteinsequenzen

---

- Proteinsequenzen haben größeres Alphabet
  - Schon viel kleinere Scores und **kürzere Matches** sind aussichtsreich
    - Und können signifikante Alignments ergeben
- Suche nach allen „t guten“ Teilwörtern **nicht sensitiv genug**
  - Beispiel: MASGTLVWG und MTSDTSVRG
  - 50% identisch, aber selbst bei  $w=2$  hätte kein Seed einen Hit mit positivem Score
    - Annahme: Einfaches Kostenmodell (M: +1, R: -1)
- Wenn man  $w$  aber zu klein macht, verliert man den Vorteil des 2-Phasen Ansatzes
- Stattdessen
  - Gegeben Seeds  $P_1, \dots, P_m$  der Länge  $w$  aus  $P$
  - Berechne aus den Seeds eine Menge  $Q = \{Q_1, \dots, Q_z\}$  von Teilwörtern der Länge  $w$  ( $z \gg m$ )
    - Für jedes  $P_i$  **permutiere alle Positionen** mit allen Aminosäuren
    - Alle Permutation mit Score zu  $P_i$  größer als  $t$  in  $Q$  aufnehmen



# Beispiel

W=2, t=8, M=PAM120  
 P=qlnfsagw  
 S=...



P<sub>1</sub>=ql  
 P<sub>2</sub>=ln  
 P<sub>3</sub>=nf  
 P<sub>4</sub>=fs  
 P<sub>5</sub>=sa  
 P<sub>6</sub>=ag  
 P<sub>7</sub>=gw



Teilwörter

Permutationen mit Score > t

ql:

ql, qm, hl, zl

ln:

ln, lb

nf:

nf, af, ny, df, qf, ef, gf, hf, kf, sf, tf, bf, zf

fs:

fs, fa, fn, fd, fg, fp, ft, fb, ys

sa:

no score > 8 (including "sa")

ag:

ag

gw:

gw, aw, rw, nw, dw, qw, ew, hw, iw, ...

A	3						
R	-3	6					
...							
P	1	-1	...	-5	6		
S	1	-1	...	-3	1	3	
T	1	-2	...	-4	-1	2	4
...							
	A	R	...	F	P	S	T

# Bemerkungen

---

- BLAST ist eine **Exklusionsmethode**
  - Bestimme und suche nach Seeds = minimale Alignments, die (höchstwahrscheinlich) im Kern jedes sehr guten lokalen Alignments stecken
  - Erweitere nur diese zu Alignments für die gesamte Suchsequenz (bzw. deren „lokalen“ Kern)
  - **Praktisch alle Heuristiken zur Sequenzsuche arbeiten nach diesem Muster**
    - Quasar, Biohunter, BLAT, FASTA, ...
    - Textsuche: Suche nach Wörtern mit  $d < t$ , Suche mit regulären Ausdrücke, ...
- Heuristik
  - BLAST findet i.d.R. **nicht alle hinreichend guten Alignments**
  - Grund: Keine INSDEL, Schwellwerte  $w$  und  $t$

# Defaulteinstellungen

---

- Proteine
  - BLOSUM62 Matrix
  - $w=4, t=14$
- DNA
  - Einheitsmatrix (Match +1, Mismatch -3)
  - $w=12, t=12$
  - Also Extension nur um exakte Matches von Teilwörtern der Länge 12

# BLAST Screenshot

NCBI results of BLAST

BLASTP 2.2.1 [Apr-13-2001]

**Reference:**  
Altschul, Stephen F., Thomas L. Madden, Alejandro A. Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997), "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs", Nucleic Acids Res. 25:3389-3402.

RID: 1011021848-13603-5892

**Query=** gi|2501594|sp|Q57997|Y577\_METJA PROTEIN MJ0577  
(162 letters)

**Database:** All non-redundant GenBank CDS translations+PDB+SwissProt+PIR+PRF  
846,869 sequences; 266,854,569 total letters

Sequences producing significant alignments:

			Score (bits)	E Value
<a href="#">gi 15668757 ref NP_247556.1</a>	(NC_000909)	conserved hypothet...	248	2e-65
<a href="#">gi 14590690 ref NP_142758.1</a>	(NC_000961)	hypothetical prote...	89	1e-17
<a href="#">gi 15679011 ref NP_276128.1</a>	(NC_000916)	conserved protein ...	76	1e-13
<a href="#">gi 15668711 ref NP_247510.1</a>	(NC_000909)	conserved hypothet...	76	1e-13
<a href="#">gi 15790518 ref NP_280342.1</a>	(NC_002607)	Vng1536c [Halobact...	72	1e-12
<a href="#">gi 15678181 ref NP_275296.1</a>	(NC_000916)	conserved protein ...	72	2e-12
<a href="#">gi 15678918 ref NP_276035.1</a>	(NC_000916)	conserved protein ...	71	3e-12
<a href="#">gi 15679076 ref NP_276193.1</a>	(NC_000916)	conserved protein ...	69	2e-11
<a href="#">gi 15790787 ref NP_280611.1</a>	(NC_002607)	Vng1898c [Halobact...	69	2e-11
<a href="#">gi 15887843 ref NP_353524.1</a>	(NC_003062)	AGR_C_878p [Agroba...	68	4e-11
<a href="#">gi 16120145 ref NP_395733.1</a>	(NC_002608)	Vng6205c [Halobact...	67	7e-11
<a href="#">gi 16080976 ref NP_391804.1</a>	(NC_000964)	similar to hypoth...	66	1e-10
<a href="#">gi 17934409 ref NP_531199.1</a>	(NC_003304)	conserved hypothet...	66	1e-10
<a href="#">gi 15790505 ref NP_280329.1</a>	(NC_002607)	Vng1518h [Halobact...	65	2e-10
<a href="#">gi 15791176 ref NP_281000.1</a>	(NC_002607)	Vng2386c [Halobact...	64	7e-10
<a href="#">gi 17544898 ref NP_518300.1</a>	(NC_003295)	CONSERVED HYPOTHET...	62	2e-09
<a href="#">gi 15790676 ref NP_280500.1</a>	(NC_002607)	Vng1752c [Halobact...	62	2e-09
<a href="#">gi 16330107 ref NP_440835.1</a>	(NC_000911)	unknown protein [S...	60	8e-09
<a href="#">gi 17546223 ref NP_519625.1</a>	(NC_003295)	CONSERVED HYPOTHET...	59	2e-08
<a href="#">gi 15794596 ref NP_284418.1</a>	(NC_003116)	conserved hypothet...	59	2e-08
<a href="#">gi 15677353 ref NP_274508.1</a>	(NC_003112)	conserved hypothet...	59	2e-08
<a href="#">gi 15921817 ref NP_377486.1</a>	(NC_003106)	157aa long conserv...	59	2e-08
<a href="#">gi 11499518 ref NP_070759.1</a>	(NC_000917)	conserved hypothet...	59	2e-08
<a href="#">gi 15221345 ref NP_176997.1</a>	(NC_003070)	unknown protein [A...	58	3e-08
<a href="#">gi 17544903 ref NP_518305.1</a>	(NC_003295)	CONSERVED HYPOTHET...	57	4e-08
<a href="#">gi 11499349 ref NP_070588.1</a>	(NC_000917)	conserved hypothet...	57	5e-08
<a href="#">gi 15790608 ref NP_280432.1</a>	(NC_002607)	Vng1658c [Halobact...	56	1e-07
<a href="#">gi 7262999 gb AAF44047.1 AF206717.1</a>	(AF206717)	hypothetical...	56	2e-07
<a href="#">gi 15899493 ref NP_344098.1</a>	(NC_002754)	Conserved hypothet...	56	2e-07
<a href="#">gi 15675620 ref NP_269794.1</a>	(NC_002737)	conserved hypothet...	55	2e-07
<a href="#">gi 17546078 ref NP_519480.1</a>	(NC_003295)	CONSERVED HYPOTHET...	55	2e-07

# Laufzeit

---

- Keine exakte Komplexität bekannt
- In der Regel sublinear
- Wie sucht man viele  $P_i$  in sehr vielen  $S$ ?
  - Hashing
    - Für festes  $w$ 
      - Berechne alle **möglichen Strings** der Länge  $w$  (Hashkey)
      - Sortiere alle Teilwörter von  $S$  der Länge  $w$  in Hashbuckets
      - Sehr viele Buckets, Speicherung der Startpositionen auf Platte
    - Für ein  $P_i$ 
      - Berechne alle **Teilwörter der Länge  $w$  mit Score über  $t$**  für  $P_i$ 
        - Für Proteine: Wie beschrieben; für DNA: auch mehrere Mutationen in einem Teilwort zulassen
      - Startpositionen findet man in den Hashbuckets
- Für „normal-seltene“ Teilwörter (keine Repeats etc.) ist das extrem schnell

# BLAST-2

---

- 7 Jahre nach der Originalveröffentlichung
  - Altschul, Madden, Schaffer, Zhang, Zhang, Miller, Lipman: „Gapped BLAST and PSI-BLAST: a new generation of protein database search programs“, NAR, 1997
- Zwei Verbesserungen
  - Performance verbessern
    - Denn: Sequenzdatenbanken wachsen schneller als Geschwindigkeit der Computer
  - Gaps beachten
    - Denn: Mehrere kurze Alignments mit Gaps werden von BLAST-1 übersehen, wenn keines davon signifikant bzgl.  $t$  ist
    - Zusammen können diese Alignments aber hochsignifikant sein
    - BLAST-1 liefert bei mehrere, nahe beieinander liegenden MSP auch mehrere Ergebnisse (statt einem größeren)

# Zwei-Hit-Strategie

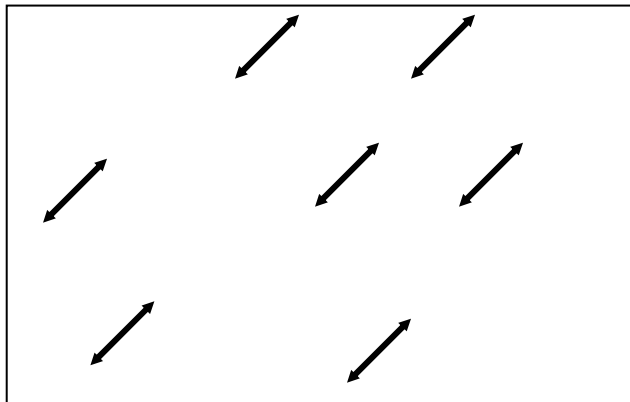
---

- Original: Alle Hits mit  $\text{Score} > t$  werden zu MSPs verlängert
  - Beobachtung: Extensionen fressen  $>90\%$  der gesamten Laufzeit
  - Aber: Interessante Alignments sind wesentlich länger als ein Seed
  - Man sollte also mehr verlangen als nur ein (kurzes) Seed, bevor man die Extension beginnt
  - Aber: Lange Seeds verringern die Sensitivität
- Neue Strategie
  - Extension erfolgt nur, wenn **zwei nicht-überlappende Hits auf einer Diagonale** mit Abstand  $< a$  gefunden wurden
    - Neuer Parameter:  $a$
  - Dadurch werden weniger Extensionen ausgeführt – großer Performancegewinn
  - Andererseits sinkt Sensitivität – deswegen mit kleineren  $t$  arbeiten
- Ergebnis
  - Performance verdoppelt bei gleichbleibender Sensitivität

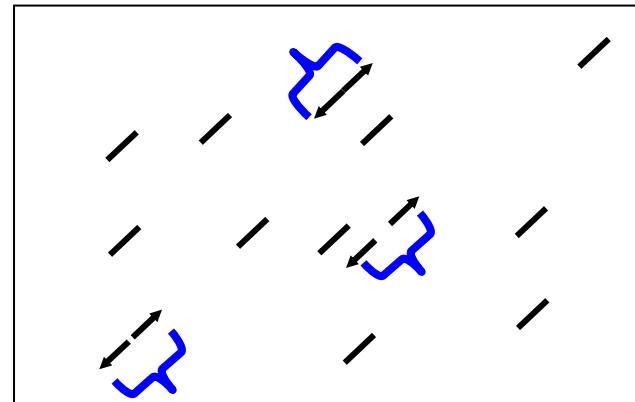
# Illustration

---

Blast-1



Blast-2



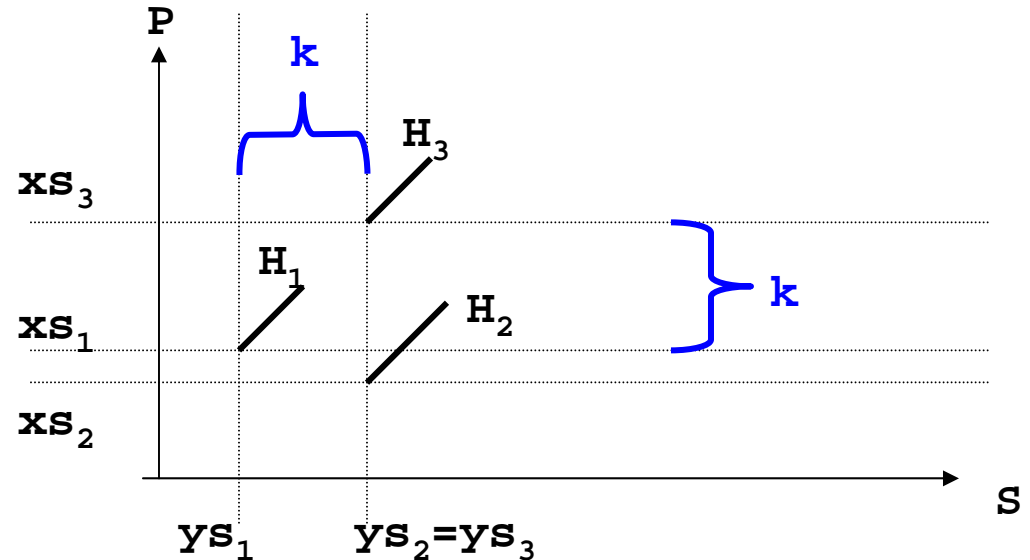
- Woher wissen wir, ob zwei Hits auf der selben Diagonale liegen?

# Berechnung der Diagonalen

## Diagonale eines Hits:

Subtraktion der Koordinaten

$$\begin{aligned}\text{diag}(H_1) &= \mathbf{x}s_1 - \mathbf{y}s_1 \\ \text{diag}(H_2) &= \mathbf{x}s_2 - \mathbf{y}s_2 \\ \text{diag}(H_3) &= \mathbf{x}s_3 - \mathbf{y}s_3 = \\ &= (\mathbf{x}s_1 + k) - (\mathbf{y}s_1 + k) \\ &= \text{d}(H_1)\end{aligned}$$



- Datenbank nach Hits durchsuchen
- Pro Hit merkt man sich die Diagonale
- Pro Diagonale merken wir uns das Ende des letzten Hits
- Wenn ein neuer Hit folgt, testen wir auf Abstand a
- Paarbildung während der (linearen) Hitsuche möglich

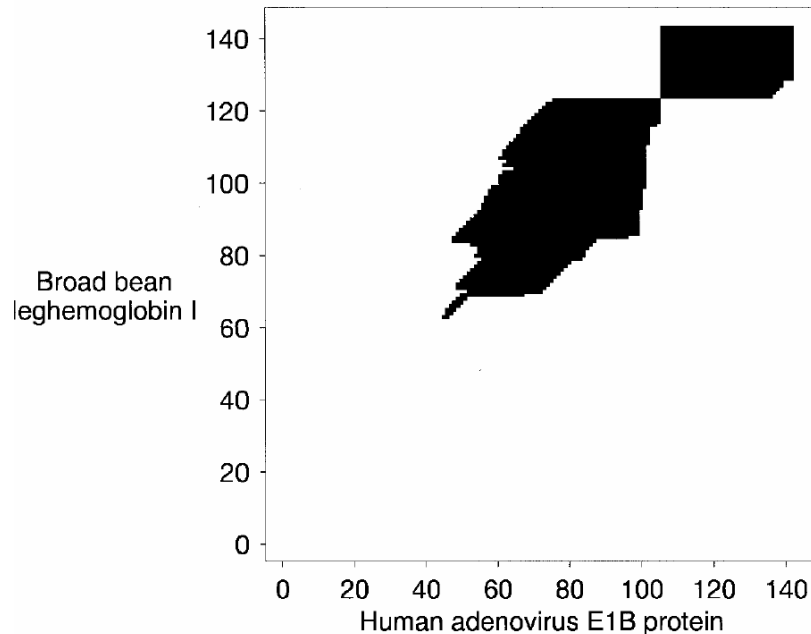
# Gaps in BLAST-2

---

- Original: Hits werden verlängert ohne Gaps
- BLAST-2: Gapped Alignment
  - Wenn zwei Hits  $H_1$ ,  $H_2$  in Abstand  $a$  gefunden werden, wird in dieser Diagonale ein **Smith-Waterman Alignment** berechnet
  - Dazu sucht man das Sequenzstück zwischen  $H_1$  und  $H_2$  der Länge  $w$  mit dem höchsten Score ohne Gaps
  - Von diesem „Seedpoint“ aus lokales Alignment berechnen
  - Da **SW sensitiver ist als Extensionen ohne Gaps**, kann man  $t$  wieder erhöhen (musste man wegen der zwei-Hits Strategie verringern)
- Weitere **Performanceverbesserung** trotz besserer Sensitivität
  - 500x langsamere Extension durch SW, aber 4000x weniger Extensionen durch Erhöhung von  $t$  von 11 auf 13

# Lokales Smith-Waterman

---



- Variante des Banded Alignment
- SW ausgehend vom Seed-Point **in beide Richtungen**
- Abbruch bei Unterschreiten bestimmter Schranken
  - Abhängig von bisherigen besten Treffern etc.

# Zusammenfassung

---

- Datenbanksuche nach ähnlichen Sequenzen braucht **sehr schnelle Algorithmen**
- BLAST ist Standard
  - Exklusionsmethode
  - Im Allgemeinen sublinear, aber benötigt sehr großen Hashindex
  - Schwerpunkt auf hoher Sensitivität – Falsch Positive Hits kann man in einem zweiten Schritt leicht rausfiltern
    - Denn Anzahl MSP  $\ll$  |DB|
  - **Trade-Off** zwischen hohen t/w Werten, Geschwindigkeit und Sensitivität