Bioinformatik

Substitutionsmatrizen

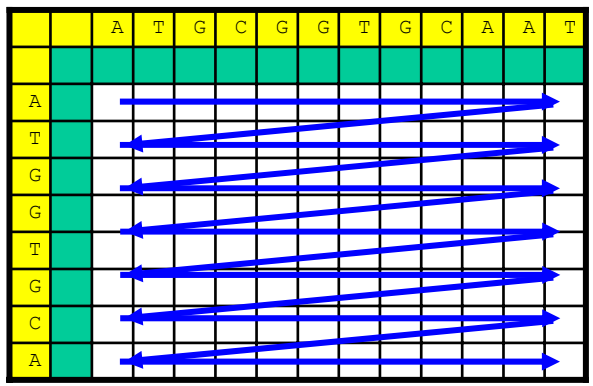
Exklusionsmethoden

Ulf Leser

Wissensmanagement in der Bioinformatik

Editabstand in O(n) Space

- Editabstand kann man leicht in linearem Platz berechnen



- Für Zeile i+1 sind nur Werte von Zeile i sowie Spalte 0 notwendig
  - Berechnung des Editabstand also in O(n) Space möglich
- Aber: Keine Berechnung des tatsächlichen Alignments mehr möglich (kein Traceback)

Strings und reverse Strings

- Definition
  - *Mit A^r bezeichnen wir das Reverse eines Strings A*
  - *$A^r[1..i]$ bezeichnet entsprechend die ersten i Zeichen von A^r*
  - *Für Strings A,B sei $v^r(i,j)=sim(A^r[1..i], B^r[1..j])$*
- Bemerkung
  - Offensichtlich gilt : $v^r(i,j)=sim(A[n-i..n],B[m-j..m])$
  - Berechnung von v^r kann exakt wie die Berechnung von v erfolgen

```
A  ............ATGCGGT
B  ..............GGTCGTAG

Ar TGGCGTA............
Br GATGCTGG..............
```

Problemhalbierung

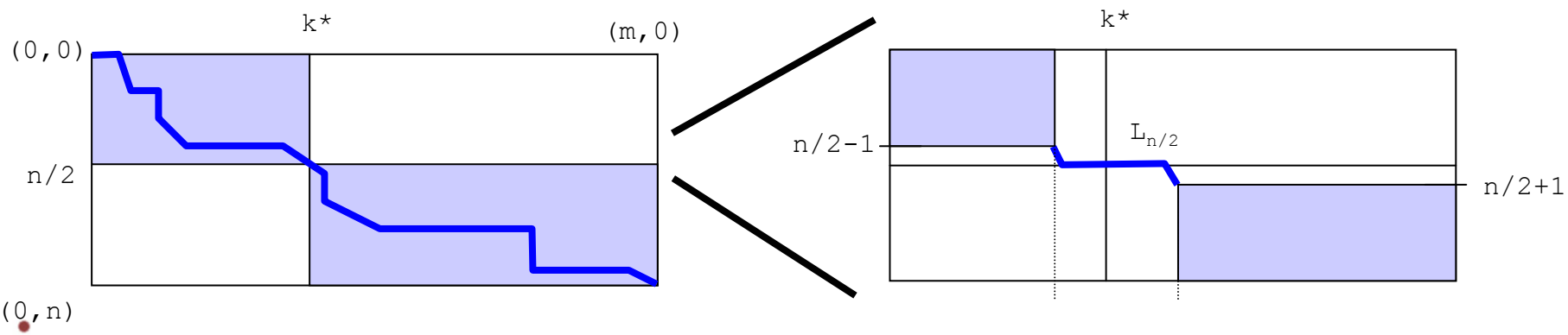
- Lemma.
  Gegeben A,B

$$v(n,m) = \max_{0 \le k \le m} \left(v(n/2,k) + v^r(n/2, m-k) \right)$$

- Beweisidee / Intuition
  - Wir alignieren
    - A[1..n/2] mit B[1..k] vorwärts
    - A[n/2+1..n] mit B[k+1..m] rückwärts
  - Durch das laufende k erwischen wir auf alle Fälle den optimalen Pfad durch die Matrix
- Bemerkung
  - Das Problem wird also bzgl. n (=|A|) halbiert

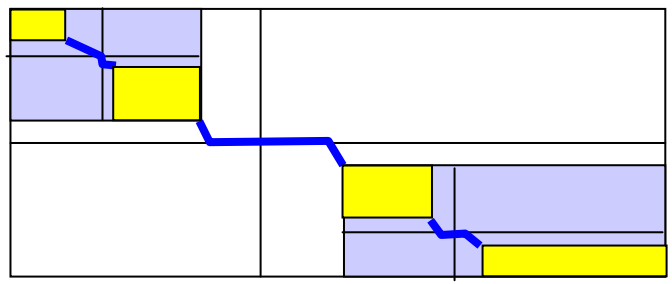
Teilpfade

- Definition
  - *Sei k* das k für das das Maximum v(n,m) erreicht wird*
  - *Sei L der Pfad von (0,0) bis (n,m)*
  - *Sei $L_{n/2}$ der Pfad zwischen dem letzten Knoten in der Zeile n/2-1 und dem ersten Knoten in n/2+1*
- Lemma
  - *L und $L_{n/2}$ müssen k* enthalten*
- Beweisidee: L muss irgendwo die Zeile n/2 passieren



Rekursion

- Wo sind wir?
  - Wir halten die gewünschten Komplexitätsschranken
  - Wir haben das Mittelstück von L berechnet
- Damit können wir rekursiv absteigen
  - Löse rekursiv die Probleme
    - für A[1..n/2] / B[1..k*] und
    - für A[n/2+1..n] / B[k*+1..m]



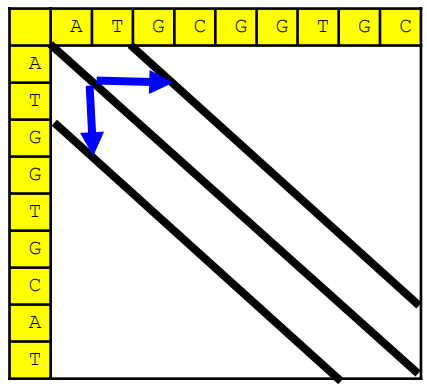
- Platzkomplexität steigt nicht
- Zeitkomplexität steigt auch nicht
  - Zeitbedarf steigt um Faktor 2
  - Beweis: Gusfield

K-Band Algorithmus

- Wir erlauben ein Abweichungen nur um +k/-k Schritte
- Algorithmus
  - Berechnet das beste globale Alignment innerhalb des Bandes der Breite 2*k

```
for i= 1 to n do
  for j= i-k to i+k do
    if (j<1) or (j>n) break;
    M[i,j]= M[i-1,j-1] + t(A[i],B[j]);
    if inband(i-1,j) then
      M[i,j]= max( M[i,j], M[i-1,j]+b);
    if inband(i,j-1) then
      M[i,j]= max( M[i,j], M[i,j-1]+b);
  end for;
end for;
return M[n,n]
```

Beispiel: k=2



Optimalität

- Theorem
  Gegeben Strings A, B mit |A|=|B|. Sei $d_K(A,B)$ der optimale K-Band Score für A und B. Wenn
  $d_K(A,B) \geq s*(n-k-1)+2b*(k+1)$, dann ist $d_K(A,B)=d(A,B)$.
- Beweis
  - Wenn das optimale Alignment im k-Band läuft, gilt auf alle Fälle $d_K=d(A,B)$
  - Wenn nicht, dann muss es irgendwo aus dem K-Band laufen. Im optimalen Fall haben wir dann n-k-1 Matches und dann k+1 Leerzeichen zum Verlassen des Bandes und weitere k+1 Leerzeichen, um am Ende noch (n,n) zu erreichen
  - Der bestmögliche Score außerhalb des K-Bandes ist also s*(n-k-1)+2b*(k+1)
  - Wenn also $d_k \geq s*(n-k-1)+2b*(k+1)$, muss das optimale Alignment im K-Band laufen und damit durch den K-Band Algorithmus gefunden werden
  - qed.

Iteratives K-Band

- Das können wir ausnutzen, um das optimale Alignment iterativ zu finden

```
k = 1;
while (true) do
        compute d_k;                    // Costs O(k*n)
        if d_k ≥ s(n-k-1)+2b(k+1) then
                return d_k;
        else
                k = 2*k;
        end if;
end while;
```

Zusammen ??? Gegen n? oder 0

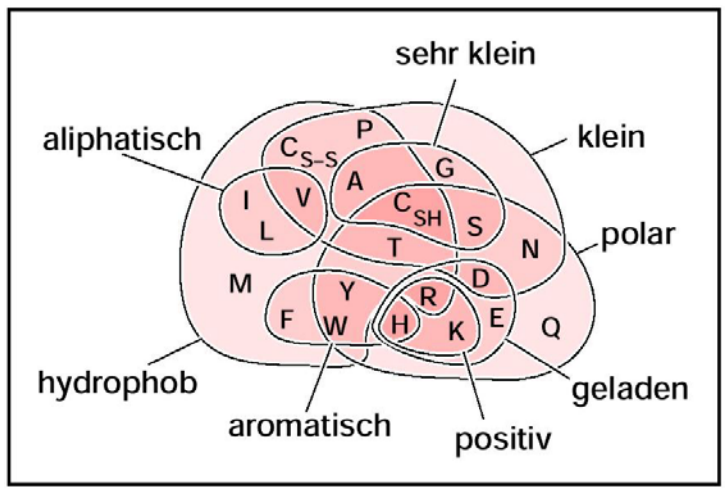
- K-Band Algorithmus hat Komplexität $O(sn^2-dn)$
- Setzen wir z.B. s=1
  - Dann kann d maximal n sein
  - Sehr ähnliche Sequenzen erreichen Wert nahe bei d
  - Dann läuft der Algorithmus auch sehr schnell
- K-Band also umso besser, je ähnlicher die Sequenzen sind
  - Gut, um das schnell festzustellen (Algorithmus mit kleinen k laufen lassen)
  - Schlecht, um irgendwelche Alignments zu berechnen

Inhalt dieser Vorlesung

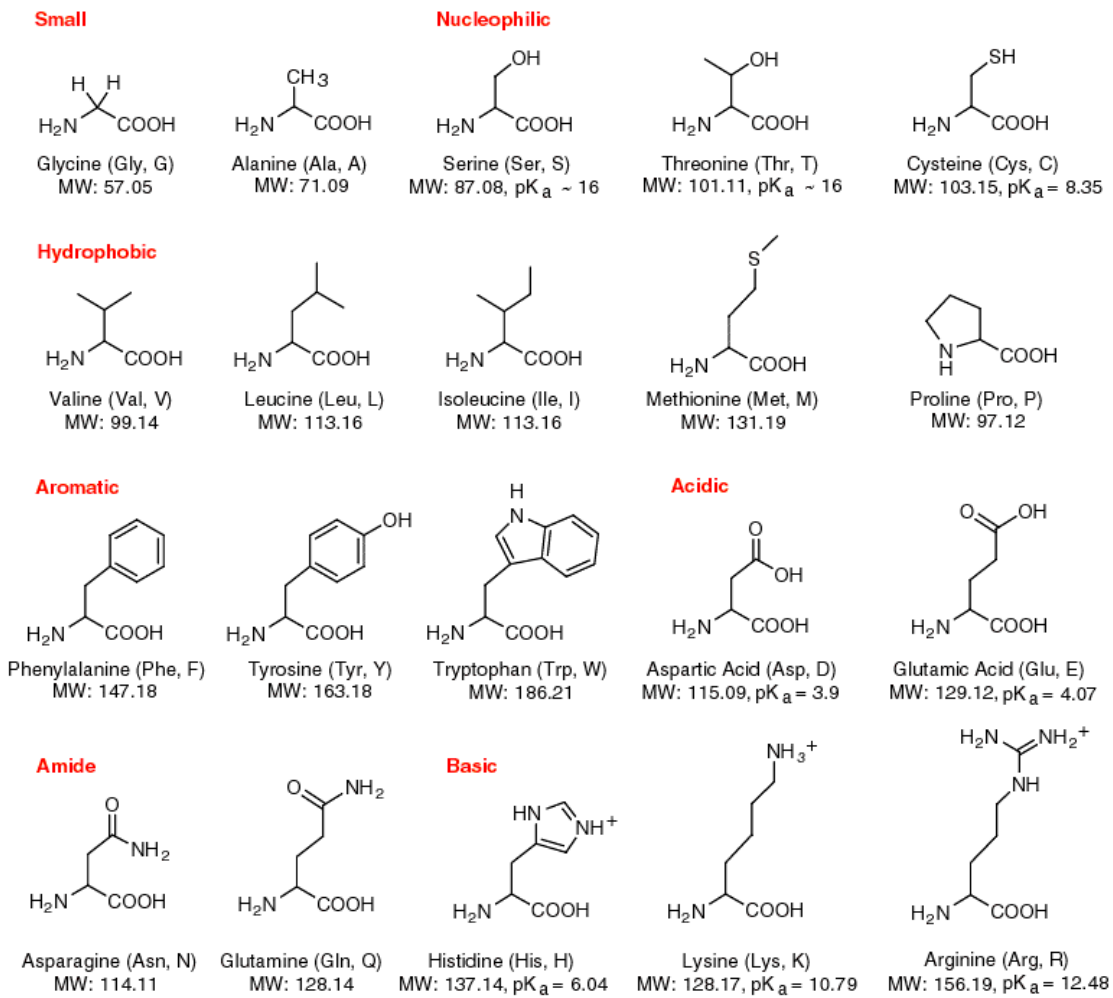
- Substitutionsmatrizen
  - PAM: Point-Accepted Mutations
  - BLOSUM
- Exklusionsmethoden
  - Auf dem Weg zu schnelleren Alignments

Hintergrund

- Schon öfters angesprochen ...
  - Ähnlichkeitsmatrizen, Substitutionsmatrizen, Scorefunktionen, ...
- Ersetzung einer Base/Aminosäure durch eine andere hat unterschiedliche Bedeutung
  - Basen: Auswirkungen auf kodiertes Protein nicht gleichverteilt über die drei Codon-Positionen
  - Aminosäuren
    - Ersetzung mit „sehr ähnlichen" Aminosäuren ändert Proteinstruktur kaum
    - Ersetzung mit „wenig ähnlichen" Aminosäuren kann Struktur vollkommen ändern



Unterschiedliche Aminosäuren



Substitutionsmatrizen

- Bewertung individueller Substitutionen
- Alignmentalgorithmus ist davon unberührt, nur Parameter ändern sich
- Das Ergebnis kann sich aber vollkommen ändern
- Beispiele: Blosum62, Identitätsmatrix

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0	-2	-1
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3	-1	0
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3	3	0
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3	4	1
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1	-3	-3
Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2	0	3
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2	1	4
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3	-1	-2
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3	0	0
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3	-3	-3
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1	-4	-3
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2	0	1
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1	-3	-1
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1	-3	-3
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2	-2	-1
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2	0	0
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0	-1	-1
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3	-4	-3
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1	-3	-2
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4	-3	-2
B	-2	-1	3	4	-3	0	1	-1	0	-3	-4	0	-3	-3	-2	0	-1	-4	-3	-3	4	1
Z	-1	0	0	1	-3	3	4	-2	0	-3	-3	1	-1	-3	-1	0	-1	-3	-2	-2	1	4

	A	C	G	T
A	5	-4	-4	-4
C	-4	5	-4	-4
G	-4	-4	5	-4
T	-4	-4	-4	5

Woher nehmen?

- Wie kann man sinnvolle Werte für die Matrix bestimmen?
  - Wir wollen Ähnlichkeit der biologischen Bedeutung messen
- 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 große Menge homologe Sequenzen

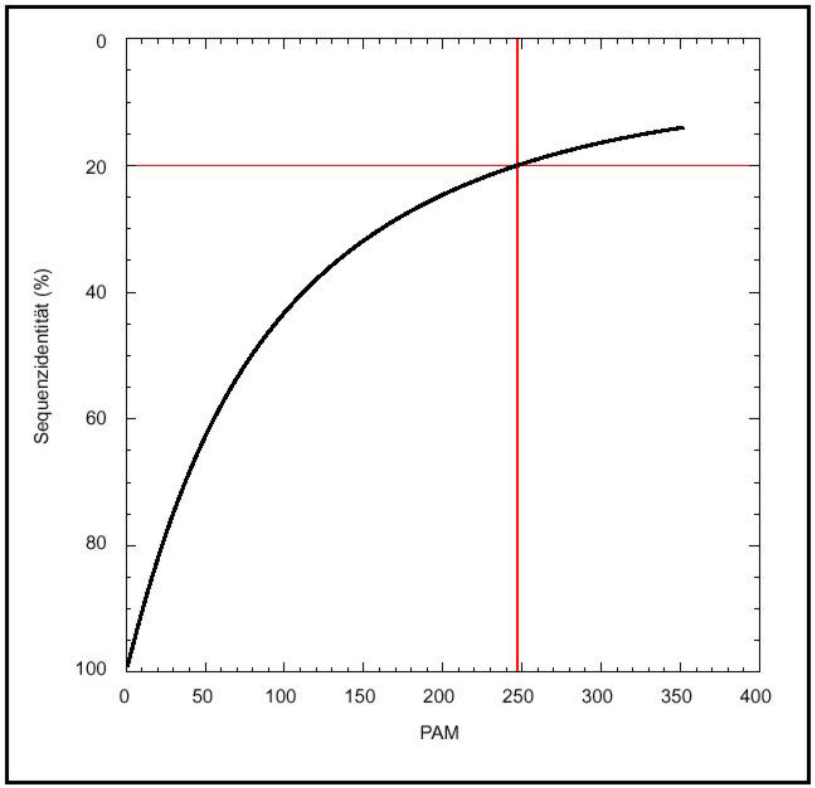
PAM: Point-Accepted Mutations

- Dayhoff et al., „A model of evolutionary change in proteins“, 1978
- PAM: Zwei Bedeutungen
  - 1 PAM – Einheit für den Abstand von Proteinsequenzen
  - PAM-X Matrix – Berechnete Substitutionsmatrix für zwei Sequenzen die X PAM entfernt sind

1. PAM als Sequenzabstand

- Definition
  Seien S_1 und S_2 zwei Proteinsequenzen. S_1 und S_2 heißen x PAM entfernt, wenn gilt
  - *S_1 wurde in S_2 überführt mit x Punktmutationen pro 100 Aminosäuren*
- Eigenschaften
  - PAM beachtet keine Inserts und Deletions
    - Großes Manko des PAM Ansatzes
  - Der echten PAM Abstand zweier Sequenzen ist nicht bestimmbar
    - Dazu hätte man ein paar Millionen Jahre zusehen müssen
  - Stattdessen: Schätzen des PAM Abstandes
    - Grundlage: durchschnittliche Veränderung nach X Mutationen
    - Ermittelbar z.B. durch Simulation
  - Vorsicht: 50 PAM heißt nicht 50 veränderte pro 100 Aminosäuren
    - Doppelmutationen, Rückmutationen, etc.
- Eigentlich besser
  - S_1, S_2 sind x PAM entfernt, wenn: *S_1 wurde am wahrscheinlichsten in S_2 überführt mit x Punktmutationen pro 100 Aminosäuren*

PAM Abstand und Sequenzidentität



- Jenseits von PAM 250 ist nur noch Rauschen

2. PAM Matrizen - Grundidee

- Vorgehen
  Seien $(S_{1,1}, S_{2,1})$, ..., $(S_{1,n}, S_{2,n})$ Paare von Sequenzen die jeweils x PAM entfernt sind. Dann berechnet sich die PAM-x Matrix M_x wie folgt
  - Messe absolute Häufigkeit $f(A_i)$ für alle Aminosäuren A_i über alle Sequenzen, normiert auf Gesamtlänge aller Sequenzen
  - Aligniere alle Paare entsprechend der evolutionären Wahrheit
    - $S_{k,l}$‘ sei die Sequenz $S_{k,l}$ mit den durch das Alignment eingefügten Leerzeichen
  - Messe Übergangshäufigkeiten f(i,j) zwischen allen Paaren von Aminosäuren (A_i, A_j), normiert auf Gesamtzahl aller Paare
    - Anzahl von Positionen k mit $S_{1,z}$‘[k]=A_i und $S_{2,z}$‘[k]=A_j über alle Positionen k in allen Paaren
    - Paare (A_x, _) werden ignoriert
    - Übergang ist „richtungslos“; f(i,j) = f(j,i)
  - Berechne Matrixelemente

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

Erläuterung

- Typische Formel für Log-Odds Ratio
- Benutzung des Logarithmus zur Ersetzung von Multiplikation mit Addition
- Bruch
  - Normierung der Übergangshäufigkeit mit Wahrscheinlichkeit des zufälligen „Umkippens“

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

  - M(i,j) = 0 (Bruch = 1)
    - Keine Selektion - Anzahl Übergänge entspricht statistischer Erwartung
  - M(i,j) < 0 (Bruch < 1)
    - Negative Selektion – Übergang wird unterdrückt
  - M(i,j) > 0 (Bruch > 1)
    - Positive Selektion – Übergang wird bevorzugt

Beispiel

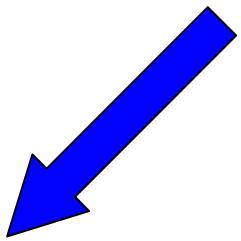
$S_{1,1}$: ACGGTGAC
$S_{2,1}$: AGG_TGCC

$S_{1,3}$: GTT_AGCTA
$S_{2,4}$: TTTCAG_TA

$S_{1,2}$: GGTCAA
$S_{2,2}$: AGTC_A

Absolute Häufigkeiten

A: 11/42	C: 8/42	G: 12/42	T: 11/42



Übergangshäufigkeiten

Substitutionsmatrix

	A	C	G	T
A	4/19	1/19	1/19	0/19
C		2/19	1/19	0/19
G			4/19	1/19
T				5/19

	A	C	G	T
A	0,48	0,02	-0,15	-
C		0,46	-0,01	-
G			0,41	-0,15
T				0,58

Probleme mit PAM Matrizen

- Es gibt keinen Algorithmus, der diese Matrizen berechnet
  - „Evolutionäre Wahrheit" ist nicht bekannt
    - Besonders bei großem Abstand ist Bestimmung eines „wahren" Alignments sehr schwierig und subjektiv

einfach

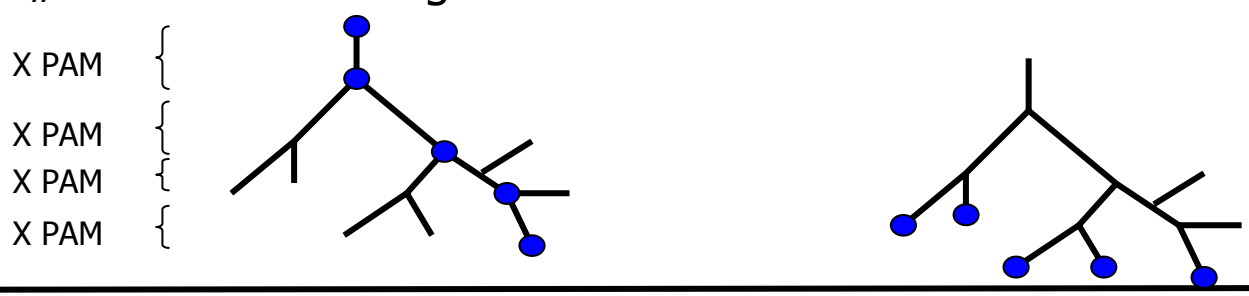
```
FMMIYVVYL    FMM_IYVVYL
FMMUIYVYL    FMMUIYV_YL
```

schwierig

```
FMMFYVVYL    _FMMFYVVYL
UFPHVYLYL    UFPHVYL_YQ

             FMMFYVVYL__
             __UFPHVYLYL
```

  - Wir haben keine Sequenzen, die x PAM entfernt sind
    - Die „wahre" Entfernung kann man immer nur schätzen



Reale PAM Matrizen

- 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 wie folgt berechnen: $M_n=(M_1)^n$
- Dem liegen viele Annahmen zugrunde
  - Evolutionary Clock Theory: Evolution verläuft gleichmäßig
    - in der Zeit und in den Sequenzpositionen
    - Proportionalität von Veränderungen
    - Hochrechnung langer Distanzen aus kurzen
  - Keine Insertions oder Deletions
  - Unabhängigkeit der Mutationswahrscheinlichkeit von der Position in der Sequenz (und der Nachbarschaft)

Ist das alles notwendig?

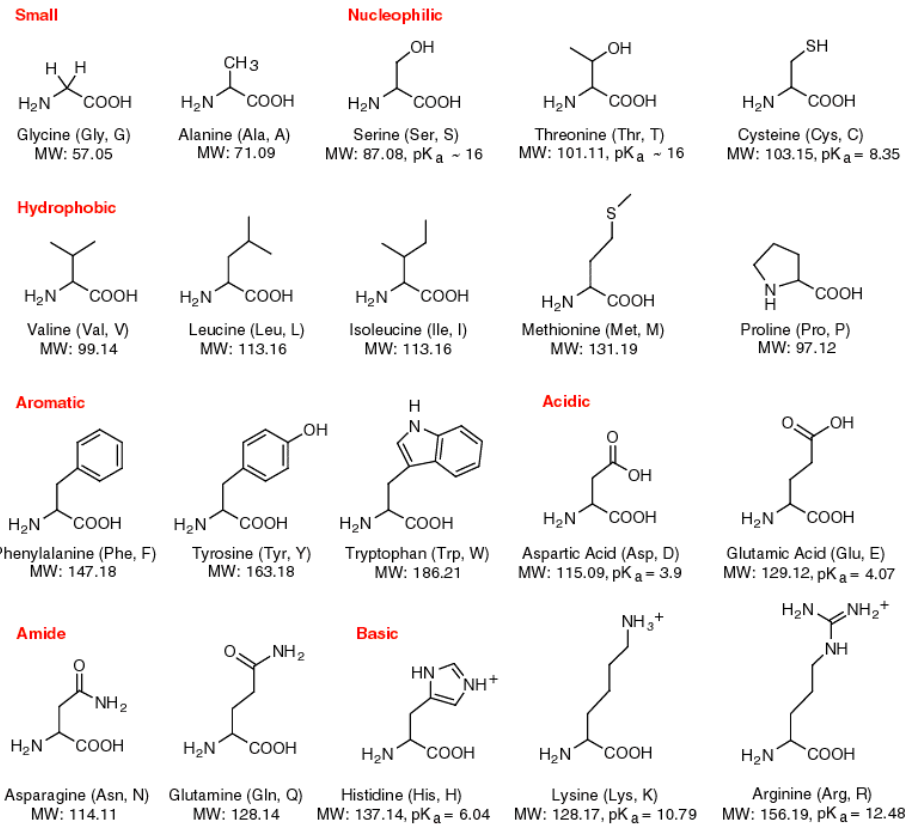
Code	Häufig-keit	Mutier-barkeit
L	0.091	54
A	0.077	100
G	0.074	50
S	0.069	117
V	0.066	98
E	0.062	77
K	0.059	72
T	0.059	107
I	0.053	103
D	0.052	86
P	0.051	58
R	0.051	83
N	0.043	104
Q	0.041	84
F	0.040	51
Y	0.032	50
M	0.024	93
H	0.023	91
C	0.020	44
W	0.014	25

- Häufigkeiten der einzelnen Aminosäuren im Verhältnis zur Gesamtzahl
- Häufigkeiten der Ersetzung einer Aminosäure im Verhältnis zu allen Ersetzungen
- Alanin (A) willkürlich als 100% gesetzt
- Keinesfalls Gleichverteilung
- Es gibt klar bevorzugte Mutationen
  - Besser: durch Selektion klar benachteiligte Mutationen
  - Tryptophan (W) sehr selten (25)
  - Serin (S) sehr häufig (117)

PAM 250

250 Multiplikationen der PAM-1 Matrix mit sich selber

	Cys	Gly	Pro	Ser	Ala	Thr	Asp	Glu	Asn	Gln	His	Lys	Arg	Val	Met	Ile	Leu	Phe	Tyr	Trp
Cys	12																			
Gly	-3	5																		
Pro	-3	-1	6																	
Ser	0	1	1	1																
Ala	-2	1	1	1	2															
Thr	-2	0	0	1	1	3														
Asp	-5	1	-1	0	0	0	4													
Glu	-5	0	-1	0	0	0	3	4												
Asn	-4	0	-1	1	0	0	2	1												
Gln	-5	-1	0	-1	0	-1	2	2												
His	-3	-2	0	-1	-1	-1	1	1												
Lys	-5	-2	-1	0	-1	0	0	0												
Arg	-4	-3	0	0	-2	-1	-1	-1												
Val	-2	-1	-1	-1	0	0	-2	-2	–											
Met	-5	-3	-2	-2	-1	-1	-3	-2												
Ile	-2	-3	-2	-1	-1	0	-2	-2	–											
Leu	-6	-4	-3	-3	-2	-2	-4	-3	–											
Phe	-4	-5	-5	-3	-4	-3	-6	-5	–											
Tyr	0	-5	-5	-3	-3	-3	-4	-4	–											
Trp	-8	-7	-6	-2	-6	-5	-7	-7	–											



Verwendung

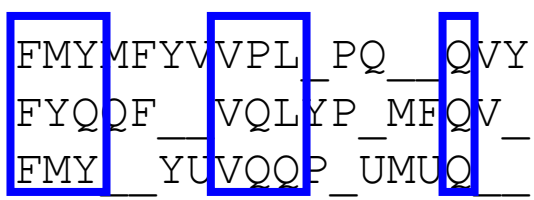
- Welche PAM Matrix soll man nun zur Alignierung zweier Sequenzen verwenden?
  - Die, die dem PAM-Abstand der Sequenzen entspricht
  - Den kennt man aber nicht – schätzen
  - Schätzung benötigt Alignments
    - Zur Berechnung der Sequenzidentität
  - Alignments basieren auf Substitutionsmatrizen
  - Henne – Ei Problem
- Also
  - Verschiedene Matrizen testen
  - Alignments prüfen
  - Externes Wissen (Chemie, Strukturen, etc.) hinzuziehen
  - Biologen fragen

BLOSUM Matrizen

- Hauptkritikpunkte am PAM Ansatz
  - Nur Verwendung sehr ähnlicher Sequenzen
    - Realistische Zahlen für evolutionär weiter entfernte Sequenzen?
  - Manuelles Alignment
    - Stimmt das?
  - PAM-x für hohe x vervielfältigen eventuelle Fehler in PAM-1
  - Schwierigkeit, das korrekt „x" für konkrete Alignmentaufgaben auszuwählen
- Anderer (neuerer) Ansatz: BLOSUM Matrizen
  - BLOcks SUbstitution Matrix
  - Basiert auf multiplen Alignments evolutionär entfernter, aber homologer Proteinsequenzen
  - Populärer als PAM Matrizen

BLOSUM Vorarbeiten

- PROSITE
  - Beschreibung identifizierender (funktionstragender?) Bereiche in homologen Proteinsequenzen durch reguläre Ausdrücke
  - Expertenwissen - manuelle Pflege der Datenbank am EBI
- BLOCKS
  - Alignierung der durch PROSITE Ausdrücke gematchten Sequenzen in Multiple Alignments (MSA)
    - Multiple Sequence Alignment – Dazu später mehr
    - BLOSUM heute: Verwendung weiterer Domänen aus PRINTS, PFAM, ...
  - Ein BLOCK ist zusammenhängendes Stück in einem MSA



BLOSUM Matrizen

- Berechnung der BLOSUM Matrizen verläuft identisch zur Berechnung der PAM-1 Matrix
  - Alle BLOCKS werden betrachtet
  - Absolute Häufigkeiten aller Aminosäuren
  - Häufigkeiten aller Übergänge in allen Paaren

$$M_1(i,j) = \log\left(\frac{f(i,j)}{f(i) * f(j)}\right)$$

- BLOSUM–x Matrizen
  - Bias in BLOCKS durch zu viele sehr ähnliche Sequenzen
  - Zur Berechnung der BLOSUM-x Matrix werden in jedem Block alle Sequenzen mit >x% Identität zu einer Sequenz zusammengefasst
  - Gänzlich andere Bedeutung des „x" als in PAM-x
  - Aber ähnliche Verwendung: x ~ evolutionärem Abstand

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
- Hochgradig heuristisches Feld
- BLOSUM-62 oft Default in Alignmentprogrammen

BLOSUM 45 Matrix

- Ausblendung Sequenzen >45% Sequenzidentität

	Gly	Pro	Asp	Glu	Asn	His	Gln	Lys	Arg	Ser	Thr	Ala	Met	Val	Ile	Leu	Phe	Tyr	Trp	Cys
Gly	7																			
Pro	-2	9																		
Asp	-1	-1	7																	
Glu	-2	0	2	6																
Asn	0	-2	2	0	6															
His	-2	-2	0	0	1	10														
Gln	-2	-1	0	2	0	1	6													
Lys	-2	-1	0	1	0	-1	1	5												
Arg	-2	-2	-1	0	0	0	1	3	7											
Ser	0	-1	0	0	1	-1	0	-1	-1	4										
Thr	-2	-1	-1	-1	0	-2	-1	-1	-1	2	5									
Ala	0	-1	-2	-1	-1	-2	-1	-1	-2	1	0	5								
Met	-2	-2	-3	-2	-2	0	0	-1	-1	-2	-1	-1	6							
Val	-3	-3	-3	-3	-3	-3	-3	-2	-2	-1	0	0	1	5						
Ile	-4	-2	-4	-3	-2	-3	-2	-3	-3	-2	-1	-1	2	3	5					
Leu	-3	-3	-3	-2	-3	-2	-2	-3	-2	-3	-1	-1	2	1	2	5				
Phe	-3	-3	-4	-3	-2	-2	-4	-3	-2	-2	-1	-2	0	0	0	1	8			
Tyr	-3	-3	-2	-2	-2	2	-1	-1	-1	-2	-1	-2	0	-1	0	0	3	8		
Trp	-2	-3	-4	-3	-4	-3	-2	-2	-2	-4	-3	-2	-2	-3	-2	-2	1	3	15	
Cys	-3	-4	-3	-3	-2	-3	-3	-3	-3	-1	-1	-1	-2	-1	-3	-2	-2	-3	-5	12

Zusammenfassung

- Willkommen im Reich der Heuristiken
- Substitutionsmatrizen
  - Beachtet tatsächlich vorkommende evolutionäre Prozesse
  - Können nur angenähert werden
  - Welche Matrix ist denn nun besser?
  - Berechnung von Matrizen immer mit Hilfe „bekannter" echter Homologien – Henne-Ei Problem

Exklusionsmethode BYP

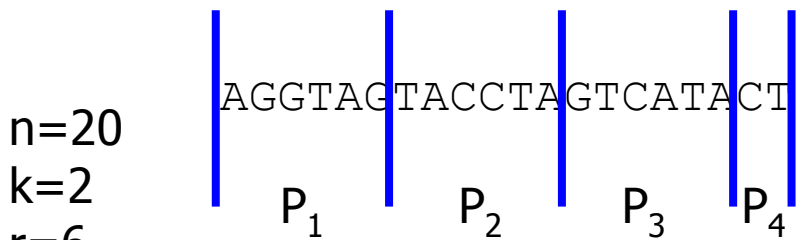
- Alignment zweier Strings A,B dauert O(n*m)
- K-Band Algorithmus benötigt $O(n^2\text{-}dn)$ für |A|=|B|
  - d ist tatsächlicher Editabstand von A,B
  - Gutes Verfahren, wenn nur binäre Entscheidung gefragt („Ähnlichkeit mindestens irgendein d_0“)
  - Sonst vorab kaum Aussage über erwartete Komplexität möglich
  - Komplexität bleibt im Bereich O(kn)
- Unser Ziel jetzt
  - Wir lassen nur k Unterschiede (Mismatch oder Leerzeichen) zu
  - Wir wollen Algorithmen, die alle k-Difference Vorkommen von P in T für geeignete k in O(m) Laufzeit im Average Case finden
    - Es gibt auch Algorithmen mit sublinearer Average Case Komplexität!
- 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*

Exklusionsmethoden

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

BYP Algorithmus - Partition

- Voraussetzung: höchstens k Differences
- Partition
  - Zerlege P gleichmäßig in Teilstrings der Länge r=floor(n/(k+1))
  - Damit haben wir k+1 Substrings von P der Länge r und einen kürzeren
    - Das sei der letzte



Beobachtung

- Lemma
  Sei T' ein k-Difference Vorkommen von P in T. Sei P partitioniert wie oben beschrieben. Dann muss T' mindestens eine Partition von P der Länge r exakt enthalten
- Beweis
  - Wir alignieren P und T' optimal
  - 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
  - Intuition: Wenn es nur k Unterschiede geben darf, dann müssen längere Teilstücke von P fehlerfrei alignieren (für k << |P|)
  - 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 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 i enthalten
- Also alignieren wir T[i-n-k .. i+n+k] mit P
- Komplexität: $O(n^2)$

Alles zusammen

- Partition-Phase ist praktisch umsonst
- Search-Phase ist O(n+m)
- Check-Phase ist $O(n^2)$ für jedes Vorkommen einer Partition von P in T
- 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
- 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~n/log$_s$(n) findet der BYP Algorithmus im Average Case alle k-Difference Vorkommen von P in T in O(m)
- Beweis
  - Sei p eine der k+1 Partitionen von p der Länge r
    - T enthält O(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 m(k+1)/s^r Vorkommen einer Partition in T
  - Für jede brauchen wir O(n^2); die Gesamtlaufzeit wollen wir aber in O(cm) halten (c: beliebige Konstante). Also

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

BYP – Komplexität Fortsetzung

- Beweis Fortsetzung
  - K sei 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=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
- Sehr gute Algorithmen erreichen sublineare Laufzeit für verschiedene Fehlergrenzen
  - Chang-Lawler; Myers Algorithmus (Celera!)
- Grundidee der Exklusionsmethoden wird in den heuristischen Verfahren verallgemeinert
  - Keine Fehlergrenze vorgeben
  - Dafür auch nicht unbedingt alle Matches finden
  - Dafür noch schneller laufen