Seminar Bioinformatik im SS 1998

Thema : Pattern Identifikation in Biosequenzen
von : Christopher Kohlhaas
am : 9.6.1998

Einführung

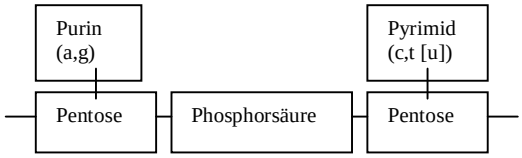
Dieser Vortrag handelt über die Ansätze für die systematische Entdeckung bestimmter Muster in Biosequenzen. Ich werde zuerst kurz auf die allgemeine Problematik eingehen und dann einen ersten Ansatz (den Bottom-Up) vertiefen. Für diesem Ansatz werde ich zuerst einige Formalismen aufstellen und ein ziemlich einfache Fitnessfunktion (für die Güte der Muster) erklären.
Dann werde ich einen ersten Algorithmus angeben und etwas über die Berechnung der vorher angegebenen Fitnessfunktion (statistische Signifikanz von Mustern) sagen.
Einen weiterer Ansatz (den Top-Down) zum Auffinden von Mustern in Biosequenzen werde ich kurz erläutern. Im Anschluß vergleiche ich die dann noch die beiden Ansätze.

Allgemeines über die zu untersuchenden Sequenzen (Bio-Sequenzen):

Die Biosequenzen sind bestimmte biologische Makromoleküle wie DNS/RNS oder Proteine (oder Teile dieser Moleküle). Sequenzen sind eine lange unverzweigte Kette einzelner kleiner Bausteine. Wir betrachten die Ketten im folgenden auch als Strings über einem Alphabet.

Da wäre die DNS (engl. DNA):

Die Desoxyribo- nucleinsäuren besteht aus einer abwechselnden Folge von Phosphorsäure und d-2-Desoxy-Ribose (Pentose) an der jeweils einer der vier Basen Cytosin, Thymin (Pyrimidinbasen), Adenin und Guanin (Pyrimidinbasen) anliegen. Diese vier Bausteine sind das Alphabet des genetischen Codes



In der Abfolge der Basen sind die Informationen für denn Aufbau der Eiweiße codiert. Die 46 Chromosomen des Menschen enthalten mehr als 10 Milliarden Basen. (200 Millionen Basen pro Chromoson)

Die Proteine:

Proteine bestehen aus einer Kette von über 100 α Aminocarbonsäuren verbunden durch Peptid Bindungen C-N (Bei weniger säuren spricht man von PolyPeptiden). Beim Bau der Proteine sind im wesentlichen 20 verschiedene α Aminocarbonsäuren beteiligt.

Die RNS (engl. RNA):

Der Ribo- nucleinsäuren-strang ist ähnlich der DNS. Statt der Base Thymin tritt hier Uracil auf und D-Ribose dient statt Desoxy-Ribose als Bindeglied. An der Übertragung der in der DNS codierten Informationen für den Bau der Porteine (Transciption) sind verschiedene Arten von RNS beteiligt. Sie übersetzten die Informationen der DNS in die entsprechenden Aminosäuren.

Dabei ergeben jeweils ein festes Triple von Basen in der DNS (4^3 =64 verschiedene Triple) eine Aminosäure in einem Protein. Da es nur 20 Aminosäuren in den Proteinen vorkommen, sind die Aminosäuren durch mehrere Basen-Triple festgelegt.

Datenbanken und Analyse der Daten:

Es ist mittlerweile billig die vorhandenen DNS Stränge und Proteine z.B. mit Hilfe der in den vergangenen beiden Vorträgen behandelten Methoden (Fragment Assembly) zu bestimmen. Deswegen gibt es viele Datenbanken mit einer recht großen (und schnell wachsenden) Anzahl Sequenzen. Sequenzen sind Ausschnitte aus den DNS Strängen oder Proteinen.

- z.B. die SWISS-PROT protein sequence database mit 49.000 Protein- Sequenzen und 17 Millionen Amino-Säuren Bauteine, das entspricht 10 – 200 Bausteinen pro Protein.
- Oder die DNA Datenbank EMBL nucleotide sequence database. Sie enthält über 180.000 Sequenzen (200 Millionen Basen)

Schwieriger ist die Analyse dieser Daten. Ziel dieser Analyse ist es, möglichst viel der biologische Bedeutung aus der Codierung durch Basen oder Aminosäuren herauszubekommen:
Wie haben sich die Moleküle im Laufe der Zeit verändert, entwickelt ? (Evolution der Moleküle). Oder noch wichtiger, wie lassen sich Rückschlüsse auf die Funktion und dreidimensionale Struktur der Protein ziehen.

Die Funktion einzelner Sequenzen experimentell herauszufinden ist zu aufwendig. Eine allgemeine Lösung des Problems gibt es auch nicht. Deshalb versucht man durch systematischen Vergleiche der in den Datenbanken vorliegenden Sequenzen die Struktur und Funktion der Protein- und DNA Ketten herauszubekommen.

Eine Möglichkeit:
Die vorliegenden Sequenzen werden in Familien eingeteilt. Man geht dabei von der Annahme aus, daß aus einem ähnlicher Aufbau der DNS oder Protein-Ketten, ähnliche Funktion oder evolutionäre Verwandschaft folgt. Eine Familie ist dabei ein Menge von Sequenzen, die eine oder mehrere gemeinsame Eigenschaften besitzen.

z.B. die Eigenschaft : In jeder Sequenz der Familie sind Informationen über der Aufbau von Hormonen codiert.

Ich verweis auf den ersten Vortrag, in dem 2 oder mehrere Sequenzen durch Alignment miteinander verglichen werden. Insbesondere bei langen und sehr vielen Sequenzen hat diese Methode aber den Nachteil, daß die Rechenzeit zu schnell ansteigt (mehrdimensionale Matrix). Außerdem werden Eigenschaften wie:" *Die Sequenzen stimmen an drei (vielleicht markanten)Stellen ganz gut überein, an den restlichen Stellen unterscheiden Sie sich willkürlich* " nicht unbedingt erkannt.

Da wir die Sequenzen auch als einen langen String betrachten können, ist eine weitere Methode naheliegend. In diesem String wird nach bestimmten Mustern (z.B. bestimmte Zeichenfolgen) gesucht, die in allen (oder vielen) Sequenzen innerhalb einer Familie vorhanden sind.
Diese Muster werden im syntaktischen Zusammenhang mit den Stringsequenzen Patterns genannt. Es gibt verschiedene Arten von Patterns.

Bsp.1: Strings `CATATACACA` und `CGTATAGC` matchen das Pattern `TATA`

```
CATATACACA    , CGTATAGC
  TATA              TATA
```

Wie helfen uns diese Patterns Rückschlüsse auf die Funktion und Verwandschaft von Sequenzen zu finden ?

Wenn ein Pattern in einer Menge von Sequenzen entdeckt wird, von denen man schon weiß, daß sie zu einer Familie gehören (ähnliche Funktion oder ähnlicher Aufbau), dann ist es wahrscheinlich, daß dieses entdeckte Pattern wichtig für die gemeinsame Eigenschaft der Familie ist.
Oder wenn ein Pattern, das wir vorher schon einer bestimmten Familie von Sequenzen haben zuordnen können, ein neue Sequenz matched, dann ist anzunehmen, daß diese Sequenz auch zu der Familie gehört, obwohl wir die biologische Bedeutung der Sequenz noch nicht entschlüsselt haben.
So können wir gefundene Patterns benutzen um Sequenzen in Familien (Klassen) einzuteilen, sie zu klassifizieren und bestimmte Eigenschaften (Funktionen) einer unbekannte Sequenzen anhand der festgestellten Familienzugehörigkeit vorherzusagen.

Conservative und Classification functions

Das Problem beim Suchen von Patterns noch mal auf den Punkt gebracht:
Man hat eine große Menge von Sequenzen und eine kleine Teilmenge, von denen man die biologische Bedeutung kennt. Man versucht jetzt die Familienzugehörigkeit der restlichen Sequenzen festzustellen.

Klassifikation

Gegeben ist eine Menge von Sequenzen. Von einigen ist bekannt, daß sie zu einer bestimmten Familie gehören und von einigen anderen ist bekannt, daß sie nicht zu dieser Familie gehören. Von den meisten Sequenzen weiß man jedoch noch nicht, ob sie zur Familie gehören.
Gesucht: Eine Funktion die für eine unbekannte Sequenz mit hoher Wahrscheinlichkeit bestimmt, ob sie zur Familie gehört oder nicht.

Formal: Gegeben ist eine Menge von F Sequenzen. Diese Menge zerfällt in zwei disjunkte Teilmengen. Die Menge der Sequenzen, die zur Familie gehören, F_+ und die Menge der Sequenzen, die nicht zur Familie gehören F_-. Wir wissen von einer Teilmenge $S_+ \subset F_+$ von Sequenzen, daß sie zu dieser Familie gehören. Von einer Teilmenge $S_- \subset F_-$ wissen wir, daß sie nicht zu dieser Familie gehören. Gesucht ist jetzt eine Funktion : F -> {**true**,**false**, unknown }, die für alle Sequenzen aus S_+ true und für die restlichen Sequenzen aus F_+ mit hoher Wahrscheinlichkeit **true**, dagegen für alle Sequenzen aus S_- false und für die verbleibenden aus F_- auch mit hoher Wahrscheinlichkeit **false** zurückliefert.

Ein Pattern nennt man diagnostisch für eine Familie, wenn es alle Sequenzen der Familie matched und alle (nicht nur die bekannten) Sequenzen, die nicht zu dieser Familie gehören nicht matched.

Konservierung

Gegeben sei eine Familie von Sequenzen. Im Gegensatz zu oben gibt es jetzt keine Sequenzen, von denn bekannt ist, daß sie nicht zur Familie gehören.
Gesucht sind Muster, die in allen Sequenzen der Familie vorkommen. Dieses Problem ist ähnlich dem obigen, nur daß keine negativ Beispiele vorkommen. Die Funktion soll für Sequenzen aus F_+ mit hoher Wahrscheinlichkeit **true** zurückliefert. Man nennt die Abschnitte einer Sequenz die für diese Funktion von Bedeutung sind, also genau die, welche in der Familie konserviert werden, konservierte Regionen.

Ein Problem bei obigen Aufgabenstellungen sind Fehler in den Eingabedaten. Die für eine Familie charakteristischen Muster stimmen nicht mit allen Sequenzen aus der Familie vollständig überein.
Man muß die obige formale Definition abschwächen:
f muß für fast alle Sequenzen aus S_+ true zurückliefern.

Eine Allgemein/e Formulierung der beiden Probleme:

Man versucht allgemeine Regeln für die ganze Menge aus der Teilmenge der bekannten "**Instanzen**" abzuleiten.
Die Patterns sind hier die allgemeinen Regeln und die Sequenzen die Instanzen. Regeln (Patterns) werden anhand von bekannten positiven Sequenzen (gehört zur Familie) und negativen Sequenzen (gehört nicht zur Familie) ermittelt.
Also quasi ein lernender Algorithmus mit den positiv und negativ Beispielen als Training Set.

Um klassifizierende Patterns zu bekommen, nimmt man aus den konservierten Pattern gerade diejenigen, die keinen der negativen Beispiele matchen.

Der Bottom-Up Algorithmus

Im folgenden werde ich einen Algorithmus zum Auffinden von Motiven oder anderen für Familien charakteristische Patterns angeben.

Welche Eigenschaften sollten solche Patterns haben ?

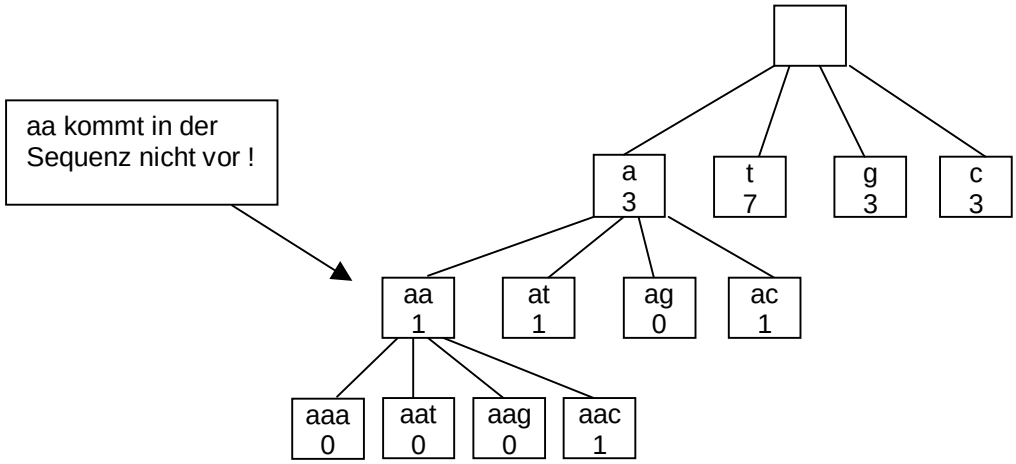
a) Sie sollten in möglichst vielen (fast allen) Sequenzen der Familie vorkommen.
b) Die Wahrscheinlichkeit, daß ein Pattern eine beliebige Sequenz matched muß möglichst gering sein.
c) Natürlich sollten diese Pattern auch eine biologisch sinnvolle Bedeutung haben. So kodieren zum Beispiel in DNA Sequenzen immer Triple ein Aminosäure. Die Länge solcher Patterns sollten dann ein Vielfaches von 3 sein.

Wie könnte ein Algorithmus aussehen ?

Die Sequenzen nach allen Pattern bis zu einer festgelegten Länge durchsuchen.

Beispiel über einem Alphabet mit vier Bausteinen

Beispiel 2:
Sequenzen: 1. TATGCTT 2. GAACTCTGT
Patterns sollen bis zu einer Länge von 3 gesucht werden.



Der Algorithmus fängt mit einfachen
Den Namen Bottom-Up kann man

Definitionen und Beispiele:

Patterns: Sei $\sum$ das Alphabet der Sequenzen
Dann ist $Q=Q_1 Q_2 ... Q_k$ $Q_i \in \bigcup_j b_j$ $b_j \in \sum$ ein Pattern der Länge k.

Q_i ist also eine Menge von Bausteinen (oder Buchstaben) über dem Alphabet $\sum$.
Eine Menge bestehend aus einem Baustein wird einfach mit dem Symbol für diesen Baustein beschrieben (single residue) Man nennt Q_i dann einen **festen Pattern Baustein** . Eine Menge mit mehreren Bausteinen wird durch $[b_1,b_2,...]$ beschrieben. Beinhaltet die Menge zwei Bausteine, spricht man auch von double residue oder ambige Bausteine. Ist die Menge das ganze Alphabet schreibt man ein Wildcard-Zeichen x (Platzhalter).

Beispiel 3: Pattern `Q = [AG]-xx-CT` , Sequenzen `1. TATGCTT 2. GAACTCTGT`

```
1. TATGCTT 2. GAACTCTGT
    AxxCT     GxxCT
                AxxCT
```

1. Ein d-Pattern ($2 \le d \le k$) hat an d Positionen **feste Bausteine** (oder Buchstaben), darunter auch die 1. und k-te, und an den restlichen Positionen Platzhalter. Wildcards werden also nicht mitgezählt.

2. Ein l-Segment einer Sequenz ist eine Teilstring $s_1..s_l$ der Länge l. s_i sind die einzelnen Zeichen des Segments. Ein Pattern Q der Länge k matched ein l-Segment, wenn k<=l und $s_i \in Q_i$ für alle $i \le k$.

3. Ein pattern-block $B_Q^{(l)}$ ist die Menge aller l-Segmente auf die Q paßt. Das universelle Pattern U matched auf alle Segmente (B_U der universelle Block beinhaltet alle Segmente).

4. Wir können B_Q auch als Kombination aus Elementarblöcken schreiben. $b_{i,r}$ ($i \le l$) ist dabei die Menge aller l-Segmente, die an der i-ten Position den Buchstaben r besitzen. $b_{i,r}$ ist unabhängig von Q. $$B_Q = \bigcap_{i=1}^{k} \left(\bigcup_{r \in Q_i} b_{i,r} \right)$$

$B_U^{(5)}$ in Beispiel 3

```
<1,1> TATGC       <2,1> GAACT
<1,2> ATGCT       <2,2> AACTC
<1,3> TGCTT       <2,3> ACTCT
<1,4> GCTTx       <2,4> CTCTG
<1,5> CTTxx       <2,5> TCTGT
<1,6> CTxxx       <2,6> CTGTx
<1,7> Cxxxx       <2,7> TGTxx
                  <2,8> GTxxx
                  <2,9> Txxxx
```

$B_Q^{(5)}$ in obigen Beispiel 3

```
<1,2> ATGCT       <2,1> GAACT
                  <2,3> ACTCT
```

Insgesamt kommt Q 3 mal in den beiden Eingabe-Sequenzen vor.

$N_k = \sum_{i=1}^{M} \max(0, L_i - k + 1)$ ist die Anzahl aller l-Segmente in M Sequenzen , die auf ein Pattern der Länge k (von der Länge her) passen könnten, daß heißt alle Segmente mit der Länge l>=k. L_i-k+1 ist dabei die Anzahl der unterschiedlichen Positionen in denen ein Pattern der Länge Q auf eine Sequenz

gelegt werden kann. max (0, ..) filtert Sequenzen deren Länge kleiner als k ist heraus. Dabei ist L_i die Länge der einzelnen Sequenzen. N_5 ist in dem Beispiel max(0,7-5+1)+max(0,9-5+1)=3+5=8 nämlich

<1,1>, <1,2>, <1,3> und <2,1>, <2,2>, <2,3>, <2,4>, <2,5>

Wir können B_Q auch als Kombinationen aus Elementarblöcken schreiben:
$B_{[AG]-xx-CT} = (b_{1,A} \cup b_{1,G}) \cap b_{4,C} \cap b_{5,T}$
Oder rekursiv:

$$B_Q = B_{Q'} \cap (\bigcup_{r \in Q_i} b_{i,r})$$

Q und Q' unterscheiden sich nur an Position i wo $Q_i' = \Sigma$ ist.

Die Pattern Häufigkeit (Fitnessfunktion)

Welche Anforderungen könnte man an ein Menge von Pattern stellen, um so die meisten unbedeutenden von den bedeutenderen zu trennen ?

1. Ein Pattern sollte mindestens C_{min} mal in den untersuchten Sequenzen vorkommen.

2. Ein Pattern wird nicht öfter als r-mal in einer Sequenz auftreten, sonst ist es unwahrscheinlich, daß charakteristische Informationen in diesem Motiv enthalten sind.

3. Die Wahrscheinlichkeit, daß das Pattern auch bei zufällige Segmente matched, sollte möglichst gering sein, damit es besonders charakteristisch für eine Familie wird.

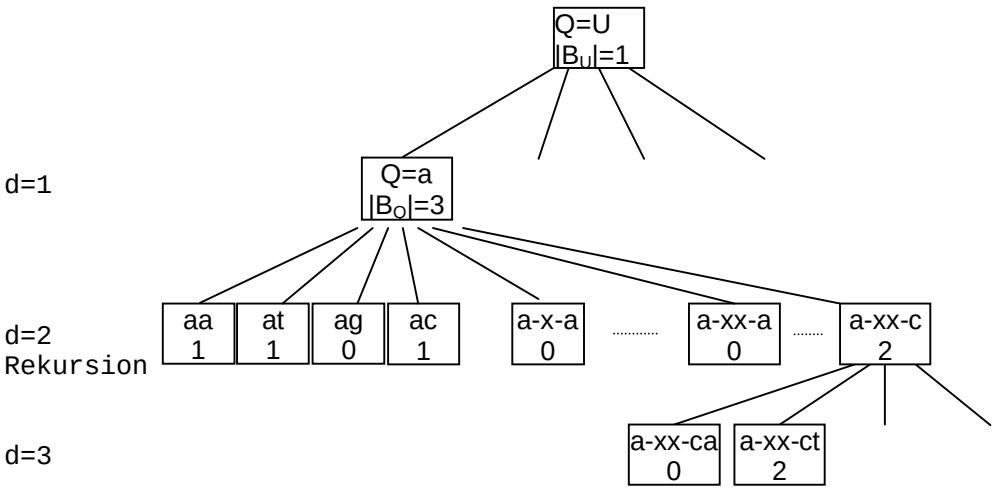
Gesucht ist eine Art Fitnessfunktion, die zu einem Pattern eine Zahl zurückliefert, die Aufschluß über die Güte eines Patterns gibt.
zum 3. Punkt.: Wir suchen eine Funktion, die uns für ein Pattern Q das in den N_k Segmenten C mal entdeckt wurde die Wahrscheinlichkeit zurückliefert, daß dieses Pattern N_k zufällige Segmente C mal oder öfter matched:
$P(N_k,C,Q)$ Wir nennen diesen Wert die **Pattern Wahrscheinlichkeit** für Q.
Je unwahrscheinlicher dieser Wert für ein gefundenes Pattern, desto statistisch signifikanter (s.u.) wird das Pattern. Weiter unten wird erklärt, wie man diesen Wert berechnet.

Der Algorithmus

Ziel des Algorithmus: Am Ende sollen die Patterns mit d<=d_{max} , die mindestens c_{min} mal vorkommen, nach der Pattern-Wahrscheinlichkeit geordnet in einem Min/Max Heap stehen.
Der Vorteil bei einem Heap ist: Einfügen in den Heap und Entfernen des Maximums geht in O (log n) Schritten. Hierbei ist n ist die Größe des Heaps. Es gibt jetzt die Möglichkeiten, den Baum in einer Tiefen und in einer Breitensuche abzuarbeiten. Die Breitensuche hat aber den Nachteil, daß man sich zu viele Patterns merken muß (Auch die, mit einem niedrigen Fitnesswert, der könnte nämlich mit einem Anhängsel steigen) .
Der Hauptroutine werden die Elementarblöcke übergeben, außerdem eine leere Heapstruktur, auf der die gefundenen Patterns gespeichert werden sollen.

```
procedure search (block b[][], modifies heap h)
  element r;
  pattern Q:=U;
  for r∈ ∑                    // Beginne mit dem ersten Buchstaben des Patterns
    Q1:={r};                  //  und durchlaufe das ganze Alphabet
    dpfs(b,b1,r, Q,  1, 2, H) //  in dieser Prozedur werden alle Patterns, die mit dem
                              //  Buchstaben r beginnen durchsucht.
  rof
end search
```

Baum für Beispiel 3 min C_{min}=2



maximale Tiefe, da k≤l

```
procedure search (block b[][], modifies heap h)
  element r;
  pattern Q:=U;
  for r∈ ∑                    // Beginne mit dem ersten Buchstaben des Patterns
    Q1:={r};                  //  und durchlaufe das ganze Alphabet
    dpfs(b,b1,r, Q,  1, 2, H) //  in dieser Prozedur werden alle Patterns, die mit dem
                              //  Buchstaben r beginnen durchsucht.
  rof
end search

procedure dpfs(elemtar block b[][],B', pattern Q, integer d,i, modifies
heap H)
  element r;
  real key;
  block B;
  integer k, C;
  for k = i .. l              //  Durchlaufe die noch nicht festgelegten Positionen
    for r∈∑
      B:=B' ∩ bk,r;           //  Das Zeichen r wird an Position k festgelegt
      C := |B|;               //  Wie oft kommt Q in den Sequenzen vor
      if C ≥ Cmin             //  Das Pattern muß mindesten Cmin  mal vorkommen
        Qk := {r};
        key:=P(Nk,C,Q )       //  Berechne die Pattern -Wahrscheinlichkeit
        insertheap (Q, key, H); //  Patterns, mit kleiner Wahrscheinlichkeit
                              //  werden nahe dem Heap-Kopf plaziert

                              //  Bei einer bestimmten Patternlaenge abbrechen
        if d<dmax             //  Achtung: ! Wildcards werden zur Länge nicht mitgezählt
          dpfs(b, B, Q, d+1,k+1,H)
        fi;
      fi;
    rof;
    Qk := ∑                   //  Wenn alle Zeichen einzeln durchprobiert wurden, dann
                              //  setzte Wildcard
  rof;
end dpfs;
```

Der Aufwand des obigen Algorithmuses steigt im Worst-Case exponentiell mit der Maximallänge der zu suchenden Patterns. Für c_{min}=0 geht der Baum alle Äste bis zur maximalen Rekursionstiefe ab. (Also Σ^d Patterns) Für höheres c_{min} werden viele Äste schon vorher abgeschnitten (pruning the search tree). Allerdings muß dann mit steigender Anzahl der Segmente auch c_{min} angehoben werden, was vielleicht einige signifikante Patterns rauswerfen könnte.
Man könnte bei schlechten Pattern-Wahrscheinlichkeit aufhören, aber auch hier werden vielleicht einige signifikante Pattern rausfallen.

Wichtig: Der Algorithmus erkennt keine ambigen Bausteine (double residue). Dazu müßten die Zeilen

```
    for r∈∑
      B':=B ∩ bk,r;            //  Das Zeichen r wird an Position k festgelegt
```

abgeändert werden zu

```
    for r1∈∑
      for r2∈∑
        B':=B ∩ (bk,r1∪ bk,r2);       //  Das Zeichen r wird an Position k festgelegt
```

Der Aufwand würde sich nochmal quadratisch erhöhen.

Das Null Modell

1. Annahme des Null Modells: Die Segmente sind unabhängig voneinander. Die Annahmen ist nicht richtig, denn die untersuchten Segmente überlappen ja. Da aber die Anzahl der Segmente, im Vergleich zu den Überlappungen eines einzelnen Segmentes, bei den untersuchten Datenmengen gering ist, können wir trotzdem wie wir später sehen werden den errechneten Wert als obere Schranke nehmen. Den bei abhängigen Ereignissen ist die resultierende Wahrscheinlichkeit immer ≤ der Summe der Einzelwahrscheinlichkeiten.
Mit dieser Annahme ist das Ereignis, daß Q C oder mehr Segmente matched, binomial verteilt: Denn wir führen denselben Versuch N_k mal durch, jedesmal mit der Wahrscheinlichkeit p_Q, daß ein Pattern eines der Segmente matched. Wir erhalten für die N_k Segmente folgende Formel für die **Pattern Wahrscheinlichkeit**:

$$P(N_k, C, p_Q) = \sum_{i=C}^{N_k} \left(\frac{N_k}{i} \right) p_Q^i (1 - p_Q)^{N_k - i}$$

p_Q ist dabei die Wahrscheinlichkei, mit der das Pattern Q ein Segment matched.
Wie berechnet man p_Q:

2. Annahme des Null Modells:

Die Positionen in den Segmenten sind unabhängig voneinander (Die Wahrscheinlichkeit, daß ein bestimmter Buchstabe an Position i steht ist unabhängig von dem Buchstaben an Position i+1).

Da in den untersuchten Sequenzen, je nach Typ DNA, RNA oder Protein, die einzelnen Bausteine nicht unbedingt gleichverteilt sind, bestimmt man für die zu untersuchenden Sequenzen vorher die Häufigkeit der einzelnen Buchstaben. f_r sei die Häufigkeit (oder Frequenz) mit der der Buchstabe r in den zu untersuchenden Segmenten vorkommt. Man erhält diesen Wert in dem man einfach das Vorkommen des Buchstaben r in allen Segmenten durch die Anzahl aller Buchstaben teilt.

p_Q die erwartete Häufigkeit mit der das Pattern Q ein Segment matched ist dann:

$$p_Q = \prod_{i=1}^{k} \left(\sum_{r \in Q_i} f_r \right)$$

oder rekursiv:

$p_Q = p_{Q'} \bullet \sum_{r \in Q_i} f_r$ Q und Q' unterscheiden sich nur an Position i wo $Q_i' = \Sigma$.

p_Q wäre für obiges Beispiel 3 mit f_a= 3 / (9 + 7) = 3/16, f_g= 3 / 16, f_t= 7 / 16, f_c= 3 / 16
also

```
     [ a      g  ] - x - x -   c   -   t
```

p_Q = `(3/16 + 3/16 ) * 1 * 1 * 3/16 * 7/16 = 126 / 4096 ≈ 0,0307617`

mit C=3 (das Pattern wurde 3 mal gefunden) ergibt sich

$$P(8,3, p_Q) = \sum_{i=3}^{8} \left(\frac{8}{i} \right) p_Q^i (1 - p_Q)^{8-i}$$

= 56 * 0,03 ^{3} * 0,97 ^{5} + 70 * 0,03 ^{4} * 0,97 ^{4} + 56 * 0,03 ^{5} * 0,97 ^{3} + 28* 0,03 ^{6} * 0,97 ^{2}
+ 8 * 0,03 ^{7} * 0,97 + 1 * 0,03 ^{8} * 1 ≈ 0,0015

Das ist die Wahrscheinlichkeit, daß Q mindestens 3 von 8 Segmenten bei obiger Verteilung der Buchstaben matched. Die Formel ist in diesem Fall nicht korrekt, denn die Voraussetzung für eine

gute Näherung (eine weitaus größere Menge unabhänger als abhängiger Segmente) ist nicht erfüllt, der wirkliche Wert ist noch kleiner.

Statistische Signifikanz

Wenn wir eine große Menge von Patterns durchsuchen (z.B. 1.000.000), dann haben ungefähr 1000 eine Pattern zufällig eine Wahrscheinlichkeit von weniger als 0,001.

Deshalb definiert man die statistische Signifikanz eines Patterns unabhängig von der Anzahl der durchsuchten Patterns wie folgt:
Def.: Die statistische Signifikanz eines Patterns Q ist durch den p-Wert gegeben (nicht zu verwechseln mit dem p_Q oder $P(N_k,..)$ oben. Dies ist die Wahrscheinlichkeit, daß unter allen durchsuchten Patterns (nicht nur q) im Null Modell mindestens eines genauso oder unwahrscheinlicher als Q ist.

Einfacher, aber nicht so aussagekräftig, ist es, die Pattern Wahrscheinlichkeit mit der Anzahl der untersuchten Patterns zu multiplizieren.

Wir testen jetzt mit zufälligen Sequenzen. Dazu ermitteln wir erst einmal die Häufigkeit der einzelnen Buchstabenin verschiedenen Sequenzen und die Sequenzlänge aus der Datenbank. Danach mischen wir die Bausteine, unter Berücksichtigung der Häufigkeiten zu zufälligen Sequenzen. Die Länge der Sequenzen entspricht dabei wieder den ursprünglichen Sequenzlängen.
Das ganze wird mal 10000 wiederholt und in jedem Durchlauf wird das Pattern mit der kleinsten Pattern Wahrscheinlichkeit gespeichert (Dieses Pattern muß mindestens c_{min} mal vorkommen). Wir betrachten damit insgesamt nur Pattern mit einer großen statistischen Signifikanz. Danach werden sie snach der errechneten Pattern-Wahrscheinlichkeit aufsteigend in einer "Hitliste" geordnet.
Wir ermitteln den empirischen p-Wert in dem wir den Rang (in der Hitliste der Patterns) durch die Anzahl der Durchläufe (10000) teilen.

Wir erhalten folgende Grafik:

\+ : Wenn wir den erwarteten p-Wert ausrechnen.
: Wenn wir einfach die pattern wahrscheinlichkeit mit der Anzahl der Patterns multiplizieren.

Wir sehen, daß der erwarteten p-Werte nur leicht über dem empirisch bestimmten p-Wert liegt. Unsere Annahmen sind also brauchbar.

Wenn wir jetzt auch noch Patterns mit ambigen Bausteinen erlauben erhalten wir :

Hier liegt der erwartete Wert nicht so dicht an den empirisch ermittleten. Das liegt daran, das die Überlappung bei Patterns mit ambigen Buchstaben viel größer ist. Ein Pattern mit einem ambigen Buchstaben kann man ja auch als eine Vereinigung von zwei Pattern, die sich bis auf eine Stelle vollständig überlappen, auffassen.

Gewichtung der Werte

Je nachdem, welche Patterns durchsucht werden ändert sich auch die statistische Signifikanz eines Patterns. Eine Suche, die auch ambige Bausteine zuläßt wird nicht so viele signifikante Patterns mit Einfach-Bausteinen entdecken, wie eine Suche, die keine ambigen Bausteine zuläßt, da es ja viel mehr Pattern durchsucht werden. Will man bestimmte Arten Pattern bevorzugt entdecken, muß man den p-Wert bei diesen Pattern einfach etwas absenken und den p-Wert bei den übrigen im entsprechenden Verhältnis anheben

Der Top-Down Ansatz

Ich will noch kurz eine weitere Methode zum Auffinden von Patterns in Biosequenzen erklären.
Der Ansatz der Top-Down Methode ist folgender:
Man sucht möglichst lange gemeinsame Teilstrings (enthalten keine Leerräume) oder Teilsequenzen in den Sequenzen.

Eine Teilsequenz $c_1..c_k$ aus zwei Sequenzen ist definiert durch: Es gibt $i_1 < ... < i_k$ und $j_1 < ... < j_k$, so daß $c_1..c_k = a_{i1}..a_{ik} = b_{i1}..b_{ik}$ gilt.

Gesucht ist die LCS (longest common subsequence), also die Teilsequenz mit dem größten Wert für k.

Beispiel:
Sequenzen `AWCDEFGHIJKLM` und `NEFGOPQAWRJKLS`

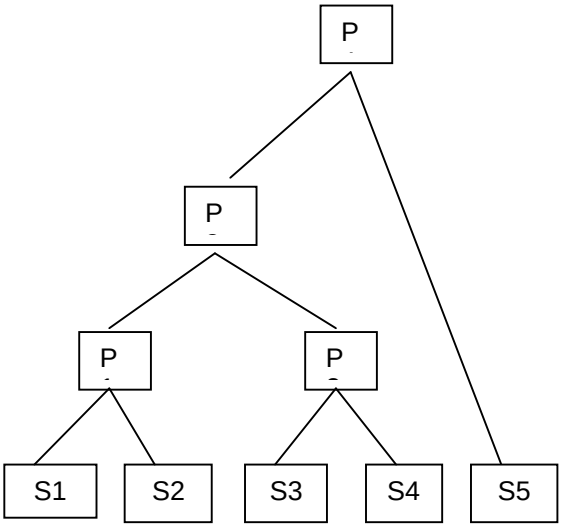
```
AWCDEFGHI    JKLM
    EFGxxxxxxJKL    CS
   NEFGOPQAWRJKLS
```

Das Problem den LCS zu finden ist eng mit dem Problem des besten Alignments aus dem ersten Vortrag verwandt. Auch hier kann man die Güte des Patterns durch eine Scoring Matrix beschrieben. Lücken (Wildcards) werden bestraft. Ambige Bausteine werden nicht so stark belohnt wie feste Bausteine.

Problem:

Wie findet man ein gutes Alignment für mehr als zwei Sequenzen.
Man versucht ja gemeinsame Pattern in einer ganzen Familie von Sequenzen zu finden. Ein Vergleich von mehreren Sequenzen ist zwar möglich, aber die Kosten dafür steigen sehr schnell an. Denn, dazu wird das Verfahren aus dem ersten Vortrag benutzt. Die Dimension der Matrix steigt mit der Anzahl der zu vergleichenden Strings. Die meisten Algorithmen benutzen deswegen immer nur paarweises Vergleichen. Es wird hier allerdings nicht unbedingt die longest common subsequence gefunden.

Ein Algorithmus um patterns in biosequenzen mit Hilfe von paarweisen Vergleichen zu bestimmen, wurde von Randall und Temple F.Smith entwickelt:
Beispiel Dendogramm:
Sequenzen S1,..S5



Aus den Sequenzen S1 und S2 wurde das Pattern P1 und aus den Sequenzen S3 und S4 wurde das Pattern P2 extrahiert. Nun werden die beiden Patterns miteinander verglichen und heraus kommt das Pattern P3. Dies ist nicht ganz einfach und ich verweise hier auf den Artikel *Automatic generation of primary sequence patterns from sets of related protein sequences* von Randall und Temple F.Smith. Pattern P3 wird dann noch mit der Sequenz S5 verglichen, die sich am meisten von allen anderen Sequenzen unterscheidet.
Hier wird auch der Name der Methode top down verständlich. Man versucht zuerst die LCS von je zwei Sequenzen zu bestimmen und verallgemeinert dann zu einem Pattern das wieder aus zwei LCS extrahiert wird.

Vergleich der beiden Anzätze

	Vorteile	Nachteile
Der Bottom-Up Algorithmus	- Findet alle Patterns einer vorgegebenen Länge.	- Laufzeit im Worst Case exponentiell mit der Pattern Länge.
Der Top-Down Algorithmus	- funktioniert auch bei langen Patterns - Findet gut die conserved Regions	- findet nicht unbedingt alle signifikanten Muster

Anlage: Einige Beispiele von gefundenen Patterns in DNS und Protein Datenbanken.

verwendete Literatur:

- A. Brazma, I. Jonassen, I. Eidhammer, D.Gilbert,
  Approaches to the automatic discovery of patterns in biosequenzen, Report No. 113 Dept. of Informatics, University of Bergen, Norway, 1995

- A.Neuwald, P.Green, *Detecting patterns in protein sequences, Journal of Molecular Biology, vol. 239 : 698-712 ,1994*

- Randalf und Temple F. Smith *Automatic generation of primary sequence patterns from sets of related protein sequences* Department of Biostatistics, Boston. 1989

- Lehrbuch *Organische Chemie* Schroedel Verlag 1981 (s.124-135)