Effiziente Algorithmen

Ausgewählte Algorithmen der Bioinformatik: Teil 1

Markus Leitner
(Martin Gruber, Gabriele Koller)

1

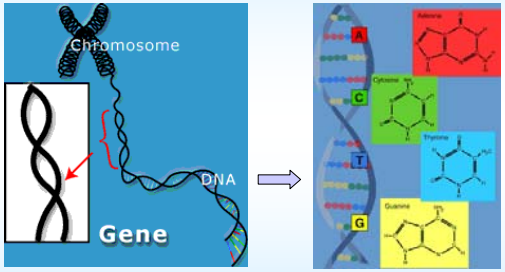
Ausgewählte Algorithmen der Bioinformatik

Zeitplan für Bioinformatik

17.11.2009	Einführung in die Molekularbiologie
	Paarweises Sequence Alignment
	Multiple Sequence Alignment (exakter Alg.)
24.11.2009	Multiple Sequence Alignment (Heuristiken)
	Maße für den Sequenzvergleich (Ähnlichkeit, Distanz)
	Phylogenetische Bäume

1. Einführung in die Molekularbiologie

Zusammenhang: Genom – Chromosom – Gen – DNA



DNA: Doppelhelix mit Nukleotiden aus Zucker, Phosphat und Basen (Adenin, Cytosin, Guanin, Thymin)

Was ist das Genom?

Als Genom bezeichnet man das gesamte genetische Material eines Lebewesens. Es besteht aus DNA (Desoxyribonukleinsäure).

Teil des Genoms sind die Chromosome (23 Chromosomenpaare beim Menschen), auf denen wiederum Gene liegen, die spezielle Merkmale eines Lebewesens prägen.

Jede Spezies hat ihr eigenes Genom. Innerhalb einer Spezies hat jedes Lebewesen sein individuelles Genom - seinen eigenen genetischen Fingerabdruck.

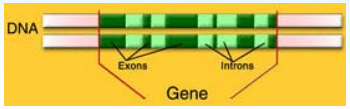
DNA ist ein großes Molekül und besteht aus Nukleotiden, die eine lange Kette bilden.

Ein Nukleotid besteht aus drei Teilen: einem Zuckermolekül, einem Phosphatmolekül und einer von vier Basen. Diese Basen, genannt Adenin, Cytosin, Guanin und Thymin (kurz A, C, G, T), enthalten die eigentliche genetische Information.

Die mittlerweile allseits bekannte Form der DNA wurde 1953 von James D. Watson und Francis Crick in Cambridge entdeckt: eine Doppelhelix, in der jeweils zwei Nukleotide über ihre Basen eine Sprosse bilden, wobei sich A-T und C-G verbinden. Dadurch gibt ein Strang die Anordnung des anderen vor, sie sind komplementär. Damit kann die DNA eindeutig durch einen String aus dem Alphabet A,C,G,T repräsentiert werden. Algorithmen für Strings sind daher in der Bioinformatik von großer Bedeutung. Die Reihenfolge der Basen steuert die RNA- und Proteinsynthese.

Gene und DNA/RNA

Genom: nur 5% Gene, bestehend aus Exons und Introns



Organismus	Basenpaare	Gene
Bäckerhefe	12 Mill.	6000
Fruchtfliege	137 Mill.	13.601
Mensch	3 Mrd.	~30.000

RNA: Uracil statt Thymin, Einzelstränge

4

Bestandteile der Gene

Das Genom enthält allerdings auch eine Menge „Müll“ - DNA, die keine Gene repräsentiert. Nur etwa 5% der DNA in einem Genom repräsentiert Gene, die Bedeutung des Restes ist noch unbekannt. Auch innerhalb eines Gens gibt es unterschiedliche Teile: Exons und Introns. Nur die Exons werden zur Proteinsynthese verwendet.

Details zur DNA

Die Länge eines DNA-Moleküls wird in der Einheit bp (Basenpaare) angegeben. Das gesamte Erbgut des Menschen besteht aus ca. 3 Gigabasenpaaren (3.000.000.000 bp), ein Chromosom aus etwa 100 Megabasenpaaren. Bei einfacheren Organismen wie der Bäckerhefe oder der Fruchtfliege sind es bedeutend weniger. Die Anzahl der menschlichen Gene wird derzeit von Experten auf ca. 30.000 eingeschätzt.

RNA versus DNA

Eine weitere Klasse der Nucleinsäuren sind die Ribonucleinsäuren (RNA), die nur aus Einzelsträngen besteht, in welchen Uracil (U) Thymin ersetzt. Diese spielen eine wichtige Rolle in der Proteinsynthese.

Proteine und Aminosäuren

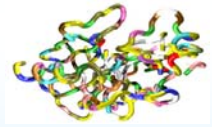
Proteine sind aus Aminosäuren aufgebaut

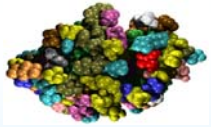
20 Aminosäuren: durch je Nukleotid-Triplets kodiert

Beispiel: GCA, GCC, GCG → Alanin

Primär- Sekundär- Tertiärstruktur

ADMVIKAPAG
AKVTKAPVAF
SHKGHASMDC





Proteinsynthese

- Transskription: DNA → mRNA
- Translation: mRNA → Aminosäuren → Protein

Was sind Proteine?

Proteine sind lange Moleküle, die aus Aminosäuren aufgebaut werden. Es gibt zwanzig Aminosäuren, die jeweils durch ein Nukleotid-Tripel (Codons) bestimmt werden.

Da es 64 (4^3) mögliche Codons für 20 Aminosäuren gibt, kann eine Aminosäure durch mehrere Codons bestimmt werden: GCA, GCC, GCG = Alanin.

Man bezeichnet die Kette der Aminosäuren als Primärstruktur des Proteins, die Kettenlänge liegt im Allgemeinen bei 150-300. Die zweidimensionale Faltung des Moleküls wird als Sekundärstruktur bezeichnet. Die Tertiärstruktur beschreibt die 3D-Struktur des Proteins und beeinflusst die Funktion des Proteins.

Ähnlichkeiten in der Primärstruktur implizieren häufig strukturelle oder funktionelle Ähnlichkeiten. Diese Gemeinsamkeiten sind die Grundlage für die Übertragbarkeit von Erkenntnissen aus der Genomanalyse von Modellorganismen auf das menschliche Genom. Die Ähnlichkeitsanalyse von zwei oder mehreren Sequenzen hat daher einen hohen Stellenwert in der Bioinformatik.

Welche Rolle spielt die DNA in der Erzeugung von Proteinen?

Die Reihenfolge der Basen auf der DNA steuert die Proteinsynthese. Der erste Schritt besteht in der Transkription eines Abschnittes, der ein Gen repräsentiert. Hier werden die beiden Stränge des DNA-Moleküls getrennt. Mittels des Enzyms RNA-Polymerase wird eine Messenger-RNA (mRNA) bestimmt, die zu dem als sinnvoll ausgewählten Einzelstrang komplementäre Basen bildet.

Die m-RNA transportiert die Information dann in die Ribosomen der Zelle, wo zu jedem Nukleotid-Triplett unter Verwendung der r-RNA und der t-RNA die festgelegten Aminosäuren in der gegebenen Reihenfolge und damit das Protein hergestellt wird.

Genforschung

Ziele: besseres Verständnis des Genoms,
z.B. zur Diagnose bzw. Vorbeugung von genetisch beeinflussten Krankheiten

Methoden und Aufgaben:

- Sequenzierung eines Gens
- Lokalisierung der Gene innerhalb des Genoms
- Analyse der Funktion eines Gens
- Analyse der DNA außerhalb der Gene
- Beziehungen zwischen Genen
- Vergleich von Genomen

Ziele der Genforschung

Ziel der Genforschung ist im Wesentlichen ein besseres Verständnis des Genoms, insbesondere um genetisch beeinflusste Krankheiten des Menschen zu diagnostizieren und zu therapieren. Schritte auf dem Weg zur Erreichung dieser Ziele sind Bestimmung der Basenpaare der menschlichen DNA, Identifizierung sämtlicher Gene und Entwicklung von Methoden zur Analyse der Daten.

Aufgaben der Genforschung

Zu den Aufgaben der Genforschung gehören also die Sequenzierung eines Gens, die Lokalisierung der Gene innerhalb des Genoms, die Analyse der Funktion eines Gens oder der DNA außerhalb der Gene, die Analyse der Beziehungen zwischen Genen sowie der Vergleich von Genomen.

Die Sequenzierung des menschlichen Genoms wurde im Rahmen des Human-Genom-Projektes 2001 erfolgreich abgeschlossen, auch einige kleinere Gene wurden bereits vollständig erfasst. Vom vollständigen Verständnis des Genoms ist die Wissenschaft trotz allem noch weit entfernt.

Aufgaben der Bioinformatik

Unterstützung der Biologen in der Bearbeitung, Analyse und Interpretation großer Datenmengen

- Erstellung biologischer Datenbanken
- Methoden zum Vergleich und zur Funktionsvorhersage von Sequenzen
- Auffinden neuer Zusammenhänge in den Daten
- Simulation biologischer Prozesse und der an biologischen Systemen durchgeführten Experimente

Aufgaben der Bioinformatik

Die Aufgabe der Bioinformatik liegt darin, die Biologen in der Bearbeitung, Analyse und Interpretation großer Datenmengen zu unterstützen.

Dies geschieht durch Erstellung biologischer Datenbanken, Entwicklung von Methoden zum Vergleich und zur Funktionsvorhersage von Sequenzen, Auffinden neuer Information und neuer Zusammenhänge in den generierten Daten und in der Simulation biologischer Prozesse und der an biologischen Systemen durchgeführten Experimente.

Literatur

- João Setubal, João Meidanis: Introduction to Computational Molecular Biology, PWS Publishing Company, Boston, 1997.
- Dan Gusfield: Algorithms on Strings, Trees, and Sequences, Cambridge University Press, 1997.
- Georg Schnitger, Skriptum zur Vorlesung Algorithmen der Bioinformatik, Ludwig Maximilians Universität München, 2001.
- Volker Heun, Skriptum zur Vorlesung Algorithmische Bioinformatik I/II, TU München, 2009
  `http://www.bio.ifi.lmu.de/~heun/lecturenotes`

Literaturempfehlungen

Der zum Thema Bioinformatik gebrachte Stoff basiert größtenteils auf den oben genannten Bücher bzw. Skripten.

2. Vergleich von Sequenzen

Zweck:

- Suche in Datenbanken
- Neue Sequenzen mit bekannten vergleichen
- Auffinden von ähnlichen Sequenzstücken
- Basis für viele andere Berechnungen in der Bioinformatik (z.B. phylogenetische Bäume)

1. Vergleich zweier Sequenzen
2. Vergleich mehrerer Sequenzen
3. Maße für den Sequenzvergleich

Vergleich von Sequenzen

Häufig verwendete und sehr erfolgreiche Algorithmen im Bereich der Bioinformatik sind sogenannte Alignment-Algorithmen, um die von den Biologen gesammelten Sequenzdaten zu vergleichen und zu interpretieren. Die Fragestellungen sind vielfältig: schnelle Suche in riesigen Sequenzdatenbanken, Vergleich neuer Sequenzen mit bereits bekannten Sequenzen, Auffinden von ähnlichen Sequenzstücken in zwei oder mehreren Sequenzen und vieles mehr. Darüber hinaus ist der Sequenzvergleich die Basis für viele weitere Berechnungen in der Bioinformatik, z.B. für die später betrachteten phylogenetischen Bäume.

Wir wollen uns zuerst mit dem Fall beschäftigen, wo zwei Protein- oder DNA-Sequenzen miteinander verglichen werden (*pairwise sequence alignment*). Anschließend werden Algorithmen für den Fall mehrerer Sequenzen vorgestellt (*multiple sequence alignment*). Schließlich erfolgt eine Gegenüberstellung zweier Maße für den Sequenzvergleich: Ähnlichkeit und Distanz.

2.1 Vergleich zweier Sequenzen

Geg. DNA-Sequenzen: GACGGATTAG GATCGGAATAG

Mögliche Alignments:

```
GACGGATTAG-
GATCGGAATAG
```

```
GA-CGGATTAG
GATCGGAATAG
```

Gesucht: bestmögliches Alignment

- Globales Alignment: ganze Sequenzen
- Lokales und semiglobales Alignment

Vergleich zweier Sequenzen

In unserem Beispiel sind zwei DNA-Sequenzen gegeben. Es könnten aber genauso gut zwei Proteinsequenzen sein.

Ein Alignment dieser Sequenzen erhält man durch Übereinanderschreiben dieser Sequenzen unter eventueller Einfügung von Leerzeichen. Im Idealfall sind die Sequenzen so aligniert, dass gleiche oder ähnliche Sequenzstücke übereinander stehen.

Alignment (formale Definition):

Seien s und t zwei Strings über einem endlichen Alphabet Σ (ohne Leerzeichen „-“), wobei s die Länge m und t die Länge n hat. Ein Alignment von s und t sind die zwei Strings s' und t' mit der Länge k ($n,m \le k \le n+m$) über dem Alphabet $\Sigma \cup$ „-“ mit der Eigenschaft, dass für ein beliebiges i ($1 \le i \le m$) und j ($1 \le j \le n$) $s[i]$ oder $t[j] \in \Sigma$ ist und s' und t' ohne die Leerzeichen s bzw. t ergeben.

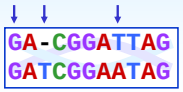
Unser Ziel ist es nun, unter der exponentiell großen Anzahl möglicher Alignments das beste zu finden.

Es können hierbei drei verschiedene Modelle von paarweisem Alignment unterschieden werden, die aber im Prinzip mit der gleichen algorithmischen Technik zu lösen sind.

Im Rahmen der Vorlesung werden wir uns ausschließlich dem globalen Alignment widmen, bei dem jeweils die ganze Sequenz involviert ist. Lokale und semiglobale Alignments sind Gegenstand eines Übungsbeispiels.

Ähnlichkeit zweier Sequenzen

Geg. Alignment:



Einfache Bewertung:

Match: 1 z.B. G mit G
Mismatch: -1 z.B. T mit A
Space: -2 z.B. - mit T

Gesamtwert: 9·1 + 1·(-1) + 1·(-2) = 6
Ähnlichkeit (*Similarity*): Wert des besten Alignments

Bewertung eines Alignments zwischen zwei Sequenzen

Um das bestmögliche Alignment zu finden, benötigen wir zuerst ein Maß, um die Güte eines Alignments beurteilen zu können. Ein solches Maß ist die Ähnlichkeit (*Similarity*).

Für ein gegebenes Alignment zweier Sequenzen kann ein Wert berechnet werden. Dabei wird jede Spalte des Alignments in Abhängigkeit ihrer Inhalte bewertet. Dabei gilt generell: gleiche Symbole (*Matches*) werden honoriert, verschiedene Symbole (*Mismatches*) und Leerzeichen (*Spaces*) bestraft. Die Spaltenwerte werden schließlich aufsummiert.

Obige, sehr einfache Bewertung wird oft in der Praxis verwendet, obwohl insbesondere für Proteine unterschiedliche Bewertungsschemen existieren, die unter anderem Abstufungen bei der Bewertung von *Mismatches* vorsehen.

Das gegebene Alignment erreicht einen Gesamtwert von 6.

Optimales paarweises globales Alignment

Das beste Alignment ist jenes mit dem höchsten Gesamtwert, dieser ist ein Maß für die Ähnlichkeit zwischen den beiden Sequenzen.

Wie können wir nun die Ähnlichkeit zweier Sequenzen und in weiterer Folge das optimale (globale) Alignment zweier Sequenzen finden?

Lösungsansatz mittels DP

Geg. Sequenzen *s* (mit Länge *m*) und *t* (mit Länge *n*) über ein Alphabet Σ

Idee: Berechne alle Ähnlichkeitwerte zwischen Präfixen der beiden Sequenzen → Matrix

Es gibt nur 3 Arten, für *i*,*j* ein Alignment zu erhalten:
(zwei Leerzeichen in einer Spalte sind nicht erlaubt)

- ...- mit ...*t*[*j*] ↔ s[1..*i*] mit *t*[1..*j*-1] oder
- ...*s*[*i*] mit ...*t*[*j*] ↔ s[1..*i*-1] mit *t*[1..*j*-1] oder
- ...*s*[*i*] mit ...- ↔ s[1..*i*-1] mit *t*[1..*j*]

Berechnung der Ähnlichkeit zweier Sequenzen mittels Dynamischer Programmierung

Die Berechnung der Ähnlichkeit und des optimalen Alignments zwischen zwei Sequenzen erfolgt effizient mittels Dynamischer Programmierung (DP). Dabei werden alle Ähnlichkeiten zwischen beliebigen Präfixen der beiden Sequenzen bestimmt. Daraus wird die Lösung für die gesamten Sequenzen aufgebaut.

Problemdefinition:

Gegeben seien zwei Sequenzen *s* mit einer Länge von *m* und *t* mit einer Länge von *n*.
Es gibt *m*+1 mögliche Präfixe von *s* und *n*+1 mögliche Präfixe von *t* einschließlich des leeren Strings. Daher reicht für die Berechnung eine (*m*+1) x (*n*+1)-Matrix, wo jeder Eintrag (*i*,*j*) die Ähnlichkeit zwischen s[1..*i*] und *t*[1..*j*] angibt.

Um an einer Stelle *i* ($1 \le i \le m$), *j* ($1 \le j \le n$) ein Alignment des Präfixes zu erhalten, gibt es nur drei Fälle:

- t[j] wird mit einem Leerzeichen aligniert,
- s[i] wird mit t[j] aligniert oder
- s[i] wird mit Leerzeichen aligniert

Zwei Leerzeichen in einer Spalte sind ja nicht erlaubt.

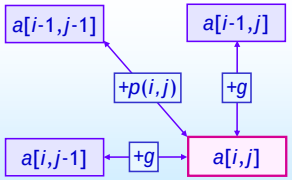
Sequenzvergleich > zwei Sequenzen > global

Algorithmus *Similarity*

Input : Sequenzen *s* und *t*
Output: Ähnlichkeit zwischen *s* und *t*
m ← |*s*|; *n* ← |*t*|;
for i ← 0 **to** *m* **do**
  a[*i*,0] ← *i*·*g*;
for *j* ← 0 **to** n **do**
  a[0,*j*] ← *j*·*g*;
for *i* ← 1 **to** *m* **do**
  for j ← 1 **to** n **do**
    a[*i*,*j*] ← max(*a*[*i*-1,*j*]+*g*,
      a[*i*-1,*j*-1]+*p*(*i*,*j*),
      a[*i*,*j*-1] + g);
return *a*[*m*,*n*];

Match/Mismatch:
p(*i*,*j*) = 1 falls s[*i*]=*t*[*j*]
-1 falls s[*i*]<>*t*[*j*]

Space: g = -2



13

Grundlegender Algorithmus für die Berechnung der Ähnlichkeit zweier Sequenzen

(Erster wichtiger Algorithmus: Needleman und Wunsch, 1970)

Gegeben seien zwei Sequenzen *s* und *t*, gesucht ist die Ähnlichkeit zwischen *s* und *t*.

Die erste Spalte und die erste Zeile werden mit Vielfachen der *Space-Penalty* initialisiert.

Danach werden die übrigen Matrixfelder in einer doppelt geschachtelten Schleife aufgefüllt. Dabei wird der maximale Wert eingetragen, der sich aus dem Vergleich der vorher angesprochenen drei Varianten zur Bildung eines Alignments ergibt:

- Wert des direkten oberen Nachbarn *a*[*i*-1,*j*] + *Space-Penalty g* (Leerzeichen in *t*)
- Wert des linken oberen Nachbarn *a*[*i*-1,*j*-1] + Funktionswert der *Match/Mismatch*-Funktion *p*(*i*,*j*) (beide Zeichen alignieren)
- Wert des direkten linken Nachbarn *a*[*i*,*j*-1] + *Space-Penalty g* (Leerzeichen in *s*)

Die *Match/Mismatch*-Funktion ergibt in unserem Fall 1 bei Übereinstimmung der Symbole an den Stellen *s*[*i*] und *t*[*j*], bei Nicht-Übereinstimmung -1. Die *Space-Penalty g* = -2.

Am Ende retourniert der Algorithmus den Wert des Feldes *a*[*m*,*n*] als Ähnlichkeit der gegebenen Sequenzen.

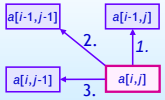
Paarweises globales Alignment

Um nun das oder die optimalen Alignments zu finden, muss man sich zusätzlich den Weg zum besten Ergebnis merken, und auf diesem dann schrittweise zurückgehen und dabei das Alignment von hinten aufbauen.

Sequenzvergleich > zwei Sequenzen > global

Algorithmus *Align*

Input: Gefüllte Matrix *a*, Indizes *i*, *j*
Output: Optimales Alignment zwischen *s* und *t* in *sa* und *ta*
if *i* = 0 **and** *j* = 0 **then**
  len ← 0;
else if *i* > 0 **and** *a*[*i*,*j*]=*a*[*i*-1,*j*]+*g* **then**
  Align(*i*-1,*j*,*len*);
  len ← *len* + *1*; *sa*[*len*] ← s[*i*]; *ta*[*len*] ← **-**;
else if *i* > 0 **and** *j* > 0 **and** *a*[*i*,*j*]=*a*[*i*-1,*j*-1]+*p*(*i*,*j*) **then**
  Align(*i*-1,*j*-1,*len*);
  len ← *len* + *1*; *sa*[*len*] ← s[*i*]; *ta*[*len*] ← *t*[*j*];
else
  Align(*i*,*j*-1,*len*);
  len ← *len* + *1*; *sa*[*len*] ← **-**; *ta*[*len*] ← *t*[*j*] ;



14

Algorithmus für die Berechnung eines optimalen paarweisen globalen Alignments

Für eine gegebene Matrix können wir eines der optimalen Alignments einfach berechnen, indem wir vom rechten untersten Matrixfeld die Berechnung „rückwärts“ ausführen.

Der Algorithmus berechnet rekursiv die Spalten des optimalen Alignments, wobei bei den Indizes *i*=0 und *j*=0 abgebrochen und der Zähler *len* mit 0 initialisiert wird. Nach dem rekursiven Aufruf wird der Zähler *len* um eins erhöht, und die jeweiligen Symbole *s*[*i*], *t*[*j*] oder Leerzeichen werden in die aktuelle Spalte *len* des Alignments geschrieben.

Der Aufruf erfolgt mittels *Align*(*m*,*n*,*len*).

Dieser Algorithmus liefert das „oberste“ Ergebnis, d.h. wenn möglich verfolgt man primär den vertikalen Weg, erst sekundär den diagonalen und ganz zuletzt den horizontalen.

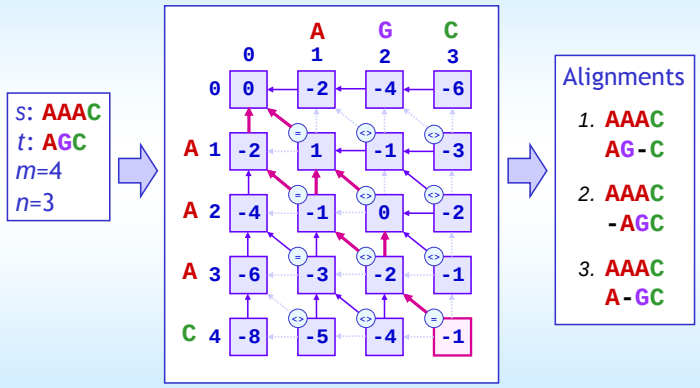
Durch Umkehrung der Abfragereihenfolge erhält man das „unterste“ Alignment. Alle Spalten, die sowohl im obersten als auch im untersten Alignment enthalten sind, sind in jedem optimalen Alignment enthalten.

Alternativ zu dieser Vorgehensweise kann man beim Berechnen der Ähnlichkeit Zeiger auf den jeweiligen Vorgänger setzen und diese dann rückverfolgen.

Berechnung aller optimalen paarweisen globalen Alignments

Durch eine Modifikation des Algorithmus kann man alle optimalen Alignments ausgeben; diese Anzahl kann jedoch sehr groß werden.

Bsp: Globales paarweises Alignment



Beispiel zum paarweisen globalen Alignment:

Gegeben seien die beiden Sequenzen s = „AAAC“ und t = „AGC“.

Dazu benötigen wir eine 5x4-Matrix. Die erste Spalte und die erste Zeile werden mit Vielfachen der Space-Penalty g (in unserem Fall -2) gefüllt.

Danach füllen wir schrittweise die restlichen Matrixfelder von links nach rechts und von oben nach unten. Wir merken uns dabei für jedes Feld, den bestmöglichen Wert und aus welchen Richtungen wir zu diesem Wert gekommen sind.

Am Ende der Berechnungen können wir im Feld [4,3] die Ähnlichkeit der beiden Sequenzen ablesen (= -1).

Finden des/der optimalen Alignments

Nun gehen wir schrittweise wieder zurück, um die Alignments, die diesen Ähnlichkeitswert ergeben, von rechts nach links aufzubauen. Wir folgen dabei den fettgedruckten Pfeilen. Ausgehen vom Eintrag (i,j) haben wir max. drei Möglichkeiten:

⇧ - Spalte enthält $s[i]$ und Leerzeichen in t

⇖ - Spalte enthält $s[i]$ und $t[j]$

⇦ - Spalte enthält Leerzeichen in s und mit $t[j]$

Die „Pfeile“ müssen nicht explizit programmiert werden, sondern es reicht wie gesagt ein einfacher Test, um den nächsten zu besuchenden Eintrag zu bestimmen.

In unserem Beispiel gibt es drei beste Alignments. Das oberste ist „AAAC“ und „AG-C“ (dieses wird auch vom vorgestellten Algorithmus *Align* gefunden), das unterste „AAAC“ und „-AGC“. Weiters finden wir noch „AAAC“ und „A-GC“.

Algorithmenanalyse

Similarity berechnen:

- Laufzeitanalyse:
  - Initialisierung 1. Spalte: O(m)
  - Initialisierung 1. Zeile: O(n)
  - Restmatrix füllen: O(mn)
  - ⇒ Gesamtaufwand: O(mn)
- Speicherplatz: O(mn)

Optimales Alignment erzeugen:

- für Alignment der Länge *len*: O(*len*) = O(m+n)

Gibt es schnellere/speichersparendere Algorithmen?

Analyse der Algorithmen *Similarity* und *Align*

Algorithmus *Similarity*: Die ersten beiden For-Schleifen benötigen O(m) bzw. O(n) Zeit. Die beiden verschachtelten For-Schleifen zum Auffüllen der restlichen Matrix benötigen O(mn) Zeit. Da dieser Term die beiden ersten dominiert ist der Gesamtaufwand für die Laufzeit O(mn).

Der benötigte Speicherplatz ist ebenfalls proportional zur Größe der Matrix, und damit O(mn).

Für Sequenzen mit annähernd gleicher Länge n ist der Aufwand somit quadratisch.

Algorithmus *Align*: Der Aufbau des Alignments erfordert - bei gegebener Matrix - einen zeitlichen Aufwand von O(*len*), wobei *len* die Länge des resultierenden Alignments und somit O(m+n) ist.

Das Finden eines globalen Alignments zweier Sequenzen ist also mit quadratischem Zeit- und Speicheraufwand möglich.

Können die Laufzeit oder der Speicherverbrauch verringert werden?

Es ist kein Algorithmus bekannt, der ebenso allgemein ist, dabei aber eine geringere asymptotische Laufzeit aufweist. Für Spezialfälle gibt es aber schnellere Algorithmen. Der Speicherplatz kann auf O(n) reduziert werden, indem nur ein Vektor im Speicher gehalten wird. Dies geschieht aber zu Lasten der Laufzeit, die sich in etwa verdoppelt. Das Alignment ist dann mit einer Divide&Conquer-Strategie zu finden.

Gap-Funktionen

Gap: Folge von $k>1$ Leerzeichen

bisher: lineare Gap-Funktion $w(k) = g \cdot k$

Alignment 1:

```
GACTGGAATG
G-C-T-A-TG
```

Alignment 2:

```
GACTGCAATG
G----CTATG
```

Überlegung:
Mutation, die Gap mit k Leerzeichen einfügt, ist wahrscheinlicher als Mutation, die k isolierte Leerzeichen einfügt.

17

Gap-Funktionen

Als *Gap* bezeichnet man eine Folge von $k>1$ Leerzeichen. Bisher haben wir alle Gaps in einem Alignment mit einer linearen Gap-Funktion bestraft: egal, ob sie einzeln oder gruppiert auftraten. Die beiden dargestellten Alignments erhalten demnach denselben Wert, da sie beide 5 Matches, 1 Mismatch und 4 Leerzeichen enthalten.

Es ist allerdings allgemein anerkannt, dass eine Mutation, die einen Gap mit k Leerzeichen einfügt, wahrscheinlicher ist als eine Mutation, die k isolierte Leerzeichen einfügt. Das liegt daran, dass ein Gap auf einen einzelnen Mutationsvorgang zurückgeführt werden kann, individuell auftretende Leerzeichen aber das Ergebnis mehrerer Mutationen sind.

Wir wollen uns nun Gap-Funktionen ansehen, die diesen Umstand berücksichtigen und *Alignment 2* einen höheren Wert zuweisen als *Alignment 1*.

Allgemeine Gap-Funktionen

allgemeine Gap-Funktion $w(k)$: nicht additiv

Zerlegung des Alignments in Blöcke → additiv

```
G | ACTG | C | - | A | A | T | G
G | ---- | C | T | – | A | T | G
```

Arten von Blöcken:

1. zwei alignierte Symbole
2. maximale Folge von Symbolen in t, die mit Leerzeichen in s aligniert sind
3. maximale Folge von Symbolen in s, die mit Leerzeichen in t aligniert sind

18

Allgemeine Gap-Funktion

Eine allgemeine Gap-Funktion w(k) ist in der Regel nicht additiv, d.h. die Summe der Werte für zwei Teile des Alignments ist ungleich dem Wert für das ganze Alignment. Die Bewertung wird jedoch (auf Blockebene) wieder additiv, wenn man das Alignment in sogenannte Blöcke zerlegt. Es gibt drei Arten von Blöcken:

1. Zwei alignierte Symbole.
2. Eine größtmögliche Folge von aufeinanderfolgenden Symbolen in t, die mit Leerzeichen in s aligniert sind.
3. Eine größtmögliche Folge von aufeinanderfolgenden Symbolen in s, die mit Leerzeichen in t aligniert sind.

DP-Algorithmus für allg. Gap-Funktionen

3 (m+1)x(n+1)-Matrizen: eine für jeden Blocktyp

- a: Alignments mit Symbol-Symbol-Spalte am Ende
- b: Alignments mit Endleerzeichen in s
- c: Alignments mit Endleerzeichen in t

- Blöcke Typ 2 (b) und 3 (c) dürfen *jeweils* nicht direkt aufeinander folgen
- Einträge in b und c hängen von mehr als einem früheren Wert ab
- Ergebnis: max{$a[m,n]$, $b[m,n]$, $c[m,n]$}

19

detaillierte Informationen → siehe nächste Folie

DP-Algorithmus für allg. Gap-Funktionen

3 (m+1)x(n+1)-Matrizen: eine für jeden Blocktyp

Initialisierung: $(0 \le i \le m,\ 0 \le j \le n)$

$a[0,0] = 0$, $b[0,j] = -w(j)$, $c[i,0] = -w(i)$, Rest: $-\infty$

Rekursion zur Matrix-Berechnung: $(1 \le i \le m,\ 1 \le j \le n)$

$$a[i,j] = p(i,j) + \max\{a[i-1,j-1],\ b[i-1,j-1],\ c[i-1,j-1]\}$$
$$b[i,j] = \max\{a[i,j-k] - w(k),\ c[i,j-k] - w(k)\} \text{ für } 1 \le k \le j$$
$$c[i,j] = \max\{a[i-k,j] - w(k),\ b[i-k,j] - w(k)\} \text{ für } 1 \le k \le i$$

Ergebnis: max{$a[m,n]$, $b[m,n]$, $c[m,n]$}

→ Laufzeit: $O(mn^2+m^2n)$

→ Speicher: $O(mn)$, aber 3 Matrizen

20

Berechnung des optimalen globalen Alignments mit einer allgemeinen Gap-Funktion

Zur Berechnung eines optimalen Alignments reicht es nun nicht mehr, nur die letzte Spalte zu betrachten, sondern es muss der letzte Block betrachtet werden. Außerdem muss die Reihenfolge, in der die Blöcke aufeinanderfolgen dürfen, beachtet werden: Blöcke vom Typ 2 bzw. 3 dürfen nicht unmittelbar aufeinander folgen.

Der DP-Algorithmus muss nun entsprechend angepasst werden. Für jedes Paar (i,j) muss der beste Wert der Präfixe gespeichert werden, der in einem bestimmten Blocktyp endet. Da wir drei Arten von Blöcken unterscheiden, benötigen wir drei (m+1) x (n+1)-Matrizen: a wird für Alignments mit Symbol-Symbol-Spalte am Ende, b für Alignments mit Endleerzeichen in s und c für Alignments mit Endleerzeichen in t verwendet.

Die erste Zeile bzw. Spalte wird wie folgt initialisiert: $a[0,0] = 0$, b: erste Zeile auf $-w(j)$, c: erste Spalte auf $-w(i)$, alle übrigen Matrix-Werte werden auf $-\infty$ gesetzt.

Je nachdem, auf welchen Blocktyp das optimale Alignment der Präfixe endet, wird nun Matrix a, b oder c aktualisiert. Die Matrix-Einträge werden wie oben angegeben berechnet.

Bemerkungen: Die Einträge in b und c hängen von mehr als einem früheren Wert ab. Weiters werden bei der Berechnung von b vorangehende b-Einträge nicht berücksichtigt, da sie nicht aufeinander folgen dürfen. Dasselbe gilt analog für c.

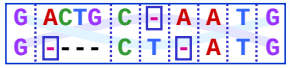
Am Ende gibt der maximale Wert aus $a[m,n]$, $b[m,n]$ und $c[m,n]$ die Ähnlichkeit an.

Die Laufzeit des Algorithmus ist $O(mn^2+m^2n)$ bzw. $O(n^3)$ für annähernd gleich lange Sequenzen. Der Beweis dafür findet sich in der Literatur (z.B. Setubal/Meidanis).

Das entsprechende optimale Alignment wird wieder durch Zurückverfolgen der Matrix-Einträge auf dem Weg zum optimalen Wert gefunden. Dabei merkt man sich jeweils, welche Matrix zuletzt benutzt wurde.

Affine Gap-Funktionen

Etwas weniger allgemein, aber in $O(n^2)$?
Idee: 1. Leerzeichen höher bestrafen (h+g)



Affine Funktion: $w(k) = h + gk$, mit $w(0) = 0$.

→ 3 (m+1) x (n+1)-Matrizen

Affine Gap-Funktion

Der Nachteil der allgemeinen Gap-Funktion besteht also im größeren zeitlichen Aufwand und im größeren Speicherbedarf. Gibt es eine etwas weniger allgemeine Gap-Funktion, die einen Gap der Länge k weniger bestraft als k individuelle Leerzeichen und trotzdem in $O(n^2)$ läuft? Ja.

Die Idee ist folgende: das erste Leerzeichen eines Gaps erhält eine höhere Bestrafung (*gap opening penalty*) als folgende Leerzeichen, die lediglich mit einem niedrigeren Wert bestraft werden (*gap extension penalty*). Formell resultiert das in einer Funktion der Form $w(k) = h + gk$, wobei $w(0) = 0$.

Diese Funktion erfüllt unsere Bedingung und ist sub-additiv.

Wir können diese Gap-Funktion ebenfalls in den bereits bekannten DP-Algorithmus einbauen. Es werden wieder drei (m+1) x (n+1)-Matrizen benötigt, um zwischen dem ersten Leerzeichen und Folgeleerzeichen unterscheiden zu können: a wird für Alignments mit Symbol-Symbol-Spalte am Ende, b für Alignments mit einem Endleerzeichen in s und c für Alignments mit einem Endleerzeichen in t verwendet.

DP-Algorithmus für affine Gap-Funktionen

Unterschiede zu allgemeiner Gap-Funktion:

- Initialisierung
- Einträge in b und c hängen nur mehr von direkten Vorgängern ab

Ergebnis: $\max\{a[m,n], b[m,n], c[m,n]\}$

→ Laufzeit: $O(mn)$
→ Speicher: $O(mn)$, aber 3 Matrizen

detaillierte Informationen → siehe nächste Folie

DP-Algorithmus für affine Gap-Funktionen

Initialisierung:

$a[0,0] = 0,\ a[i,0] = -\infty,\ a[0,j] = -\infty$ $(1 \le i \le m,\ 1 \le j \le n)$

$b[i,0] = -\infty,\ b[0,j] = -(h+gj)$ $(1 \le i \le m,\ 0 \le j \le n)$

$c[i,0] = -(h+gi),\ c[0,j] = -\infty$ $(0 \le i \le m,\ 1 \le j \le n)$

Rekursion zur Matrix-Berechnung: $(1 \le i \le m,\ 1 \le j \le n)$

$$a[i,j] = p(i,j) + \max\{a[i-1,j-1], b[i-1,j-1], c[i-1,j-1]\}$$
$$b[i,j] = \max\{a[i,j-1] - (h+g), b[i,j-1] - g, c[i,j-1] - (h+g)\}$$
$$c[i,j] = \max\{a[i-1,j] - (h+g), b[i-1,j] - (h+g), c[i-1,j] - g\}$$

Ergebnis: $\max\{a[m,n], b[m,n], c[m,n]\}$

→ Laufzeit: $O(mn)$

→ Speicher: $O(mn)$, aber 3 Matrizen

23

Berechnung des optimalen globalen Alignments mit einer affinen Gap-Funktion

Die erste Zeile bzw. Spalte wird wie folgt initialisiert:

a: $a[0,0] = 0$, der Rest der ersten Zeile und Spalte von a wird auf $-\infty$ gesetzt (hier wird nur der Fall betrachtet, dass zwei Symbole in der letzten Spalte stehen);

b: erste Zeile auf $-(h+gj)$, erste Spalte auf $-\infty$, um konsistent mit der Definition zu sein: es gibt genau ein Alignment zwischen der leeren Sequenz $s[1..0]$ und $t[1..j]$, aber es gibt kein Alignment zwischen $s[1..i]$ und der leeren Sequenz $t[1..0]$, das mit einem Leerzeichen in $s[i]$ endet);

c: erste Spalte auf $-(h+gi)$, erste Zeile auf $-\infty$ (Erklärung analog zu b).

Die weiteren Matrix-Einträge werden wie oben angegeben berechnet.

Bemerkungen: Der Eintrag $b[i,j]$ enthält in der letzten Spalte ein Leerzeichen. Es muss überprüft werden, ob es ein erstes oder ein Folgeleerzeichen ist, und es entsprechend bestrafen. Dazu betrachtet man die Alignments der Präfixe $s[1..i]$ und $t[1..j-1]$. Jene in $a[i,j-1]$ und $c[i,j-1]$ enden nicht auf ein Leerzeichen in s, daher muss das Leerezeichen in diesen Fällen als erstes Leerzeichen $(h+g)$ bewertet werden. Dagegen endet $b[i,j-1]$ bereits auf ein Leerzeichen in s, das aktuelle Leerzeichen erhält daher lediglich den niedrigeren Wert für ein Folgeleerzeichen (g). Die Einträge in c werden analog berechnet.

Am Ende gibt der maximale Wert aus $a[m,n]$, $b[m,n]$ und $c[m,n]$ die Ähnlichkeit an.

Die Laufzeit des hier beschriebenen Algorithmus ist $O(mn)$ bzw. $O(n^2)$ für annähernd gleich lange Sequenzen. Es wurden aber auch Algorithmen in $O(n)$ entwickelt. Allerdings werden auch für affine Gap-Penalties drei Matrizen benötigt.

Das entsprechende Alignment wird wie bei der allgemeinen Gap-Funktion gefunden.

2.2 Vergleich mehrerer Sequenzen

Ziel: optimales multiples Alignment von k>2 Sequenzen

Anwendungsgebiete:

- Phylogenetische Analyse: Darstellung von Abstammungsbeziehungen auf Basis der Distanz
- Identifizierung von Motiven und Profilen: durch die Evolution erhalten gebliebene Merkmale
- Struktur- und Funktionsvorhersage

24

Vergleich mehrerer Sequenzen

Bisher behandelten wir nur den Vergleich zweier Sequenzen und paarweise Alignments. Nun verallgemeinern wir das Problem auf den Vergleich von k>2 Sequenzen. Unser Ziel ist es nun, ein optimales multiples Alignment für k Sequenzen zu erzeugen.

Anwendungsgebiete

Phylogenetische Analyse: Mit phylogenetischen Bäumen können evolutionäre Beziehungen zwischen verschiedenen Organismen aufgezeigt werden. Dazu wird aus dem multiplen Alignment ein Maß für die phylogenetischen Distanzen abgeleitet. Diese Distanzen werden dann zur Erstellung der Bäume herangezogen. Mehr dazu im nächsten Kapitel.

Identifizierung von Motiven und Profilen: multiple Alignments ermöglichen die Identifizierung von Motiven, die durch die Evolution erhalten blieben und eine wichige Rolle für die Struktur und Funktion einer Gruppe von verwandten Proteinen spielen. Innerhalb eines multiplen Alignments zeigen sich solche Strukturen oft als Spalten mit weniger Variation als die Umgebung. Zusammen mit experimentellen Daten sind sie gut geeignet, um Sequenzen mit unbekannter Funktion zu charakterisieren. Wenn das Motiv zu "versteckt" ist, kann man sich auch mit Profilen helfen, die die Eigenschaften einer Proteinfamilie Spalte für Spalte zusammenfassen (Profil: positions-spezifische Bewertungen für das Vorkommen einzelner Aminosäuren sowie Bewertungen für Einfügungen und Löschungen). Auf diese Art können auch entfernte Mitglieder einer Proteinfamilie identifiziert werden.

Struktur-Vorhersage: Ein weiteres Anwendungsgebiet ist die Vorhersage von Sekundär- und Tertiärstrukturen und in weiterer Folge von Funktionen von Proteinen. Mit Hilfe von multiplen Alignments erzielte Vorhersagen von Sekundärstrukturen sind weit verlässlicher (75%) als solche, die nur auf einer Sequenz basieren (60%).

Berechnung multipler Alignments

Schwieriges Problem:

1. Wahl der Sequenzen:
   Homologie (gemeinsamer Vorfahr) vorausgesetzt
   → Aufgabe der Biologen
2. Wahl der Bewertungsfunktion:
   Ideal: biologisch korrektes Alignment erzielt höchsten Wert
3. Wahl des Algorithmus: komplexe Aufgabe
   - exakte Verfahren
   - Heuristik: progressiv, iterativ

25

Berechnung multipler Alignments

Die Berechnung multipler Alignments stellt ein schwieriges Problem dar, das Ergebnis hängt von drei Faktoren ab: der Auswahl der Sequenzen, der Bewertungsfunktion und des Algorithmus, der die Bewertungsfunktion optimiert.

Ein Vergleich von Sequenzen ist meist nur sinnvoll, wenn *homologe* (d.h. Sequenzen mit gemeinsamem Vorfahren) Sequenzen miteinander verglichen werden. Für globale Methoden ist weiters notwendig, dass die Sequenzen über die ganze Länge verwandt sind. Andernfalls muss man auf Verfahren für den lokale Vergleich zurückgreifen. Die Wahl der richtigen Sequenzen ist aber die Aufgabe der Biologen.

Die Bewertungsfunktion soll so gewählt werden, dass ein biologisch korrektes Alignment einen möglichst hohen Wert erzielt. Dazu später.

Die Berechnung des mathematisch optimalen Alignments für mehrere Sequenzen ist ebenfalls eine extrem komplexe Aufgabe, auch wenn eine passende Menge von Sequenzen und die biologisch perfekte Zielfunktion verfügbar wären. Dazu stehen exakte Verfahren und verschiedene Heuristiken zur Auswahl.

Multiple Sequence Alignment (MSA)

Multiples Alignment von *k*>2 Sequenzen $s_1, \ldots, s_k$:

- alle Sequenzen durch ev. Einfügen von Leerzeichen auf dieselbe Länge erweitern
- keine Spalte darf nur aus Leerzeichen bestehen

Beispiel: Multiples Alignment von Proteinsequenzen

Sequenzen

```
1 PEAALYGRFTIKS
2 PEALNYGWYSSES
3 PEVIRMQDDNPFSFQS
4 PESLAYNKFSIKS
```

MSA

```
1 PEAALYGRFT---IKS
2 PEALNYGWY---SSES
3 PEVIRMQDDNPFSFQS
4 PESLAYNKF---SIKS
```

26

Multiples Sequence Alignment (MSA)

Ein multiples Alignment ist eine einfache Verallgemeinerung des paarweisen Alignments. Nun sind *k*>2 Sequenzen $s_1, \ldots, s_k$ mit Längen $|s_1|, \ldots, |s_k|$ durch eventuelles Einfügen von Leerzeichen an geeigneten Stellen auf eine gemeinsame Länge *len* zu erweitern, wobei keine Spalte nur Leerzeichen enthalten darf. Das Alignment hat damit eine Länge von mindestens der Länge der längsten gegebenen Sequenz und maximal der Summe der Längen aller gegebenen Sequenzen.

Da in der Praxis multiple Alignments von Proteinsequenzen gebräuchlicher sind, werden hier auch in den Beispielen Proteinsequenzen statt DNA-Sequenzen verwendet. Für die gegebenen 4 Sequenzen erhält man beispielsweise durch Einfügen von je 3 Leerzeichen in die Sequenzen 1, 2 und 4 das rechts abgebildete multiple Alignment.

Bewertung eines multiplen Alignments

Beispiel:

Multiples Alignment

```
1 PEAALYGRFT---IKS
2 PEALNYGWY---SSES
3 PEVIRMQDDNPFSFQS
```

Beschränkung auf additive Funktionen

Anforderungen:

- Spalten mit vielen gleichen Symbolen erhalten höheren Wert als solche mit unterschiedlichen oder Leerzeichen
- Reihenfolge der Sequenzen egal

Naheliegend: *k*-dimensionales Feld für jede Kombination (Platzbedarf steigt exponentiell mit *k*)

Bewertung eines multiplen Alignments

Die Bewertung eines multiplen Alignments gestaltet sich schwieriger als die eines paarweisen Alignments. Im Idealfall sollte alles Wissen über die Sequenzen in die Bewertungsfunktion einfließen. Diese Informationen sind aber selten verfügbar. Daher beschränkt man sich hier auf einfachere, rein additive Funktionen, d.h. der Wert des Alignments ist die Summe der Spaltenwerte. Wie soll nun eine Spalte bewertet werden, d.h. welche Anforderung stellen wir an die Bewertungsfunktion? Ziel ist, dass Spalten mit vielen gleichen oder ähnlichen Symbolen höhere Werte als Spalten mit vielen unterschiedlichen Symbolen oder Leerzeichen erhalten sollen. Weiters soll die Reihenfolge der Symbole in den Spalten keine Rolle spielen.

Die naheliegendste Variante wäre, alle möglichen Zeichenkombinationen in einem *k*-dimensionalen Feld aufzulisten und ihnen einen Wert zuzuweisen. Die Anzahl der Felder steigt aber exponentiell mit der Anzahl der Sequenzen und ist daher aus Platzgründen eher ungeeignet.

Sum-of-Pairs-Maß

SP-Score: Summe der paarweisen Alignment-Werte einer Spalte, wobei $p(-,-) = 0$

Beispiel:

Multiples Alignment

```
1 PEAALYGRFT---IKS
2 PEALNYGWY---SSES
3 PEVIRMQDDNPFSFQS
```

SP-Score(Spalte 3) = $p(A,A) + p(A,V) + p(A,V)$

SP-Score(Spalte 12) = $p(-,-) + p(-,F) + p(-,F)$

Sequenzen 1 und 2

```
1 PEAALYGRFT---IKS
2 PEALNYGWY---SSES
```

⇨ *Projektion*

```
1 PEAALYGRFT-IKS
2 PEALNYGWY-SSES
```


Sum-of-Pairs-Maß

Eine praktikable Lösung stellt dagegen die Sum-of-Pairs-Funktion dar, die die Anforderungen erfüllt. Der SP-Wert ist definiert als die Summe der paarweisen Werte aller Paare von Symbolen in der Spalte.

Bemerkenswert ist der Umstand, dass nun im Gegensatz zum paarweisen Alignment auch zwei Leerzeichen bei der Berechnung der paarweisen Werte auftreten können. Hier wird üblicherweise der Wert 0 zugewiesen, obwohl Leerzeichen normalerweise negativ bewertet werden. Allerdings ermöglicht diese Vorgangsweise, dass der Wert des multiplen Alignments auch aus der Summe der Bewertungen der induzierten paarweisen Alignments (= Projektionen des multiplen Alignments) berechnet werden kann.

Bewertungsmatrix für den Proteinvergleich

Bewertungsschema soll Ähnlichkeiten von Aminosäuren (Mutationswahrscheinlichkeit) reflektieren

PAM-Matrizen (Dayhoff, 1978):
auf Basis von akzeptierten Mutationen

Palette von PAM-Matrizen

- PAM40: kürzere, starke lokale Ähnlichkeiten
- PAM250: längere, schwächere lokale Ähnlichkeiten

BLOSUM-Serie (Henikoff und Henikoff, 1992)

Affine Gap-Funktion: *GOP* und *GEP*

29

Bewertungsmatrizen für den Proteinvergleich

Beim Vergleich von Proteinen reicht ein einfaches Bewertungsschema, wie es für den DNA-Vergleich vorgestellt (z.B. Match=1, Mismatch=-1, Space=-2) nicht aus. Die unterschiedlichen chemischen und physikalischen Eigenschaften der Aminosäuren, wie Größe oder die Fähigkeit, sich mit Wassermolekülen zu verbinden, beeinflussen die Häufigkeit von Substitutionen im Laufe der Evolution. Beispielsweise ist es wahrscheinlicher, dass Aminosäuren ähnlicher Größe einander ersetzen als solche mit stark unterschiedlicher Größe. Da hinter Proteinvergleichen oft evolutionäre Überlegungen stehen, ist es wichtig, dass ein Bewertungsschema diese Wahrscheinlichkeiten so gut wie möglich widerspiegelt.

Dazu wurden 1978 von Margaret Dayhoff und ihren Kollegen PAM-Matrizen entwickelt. PAM steht für *Point Accepted Mutations* oder *Percent of Accepted Mutations*. Es gibt unterschiedliche Varianten, z.B. PAM120, PAM250 etc., je nach evolutionärer Distanz. Mutationen mit hoher Wahrscheinlichkeit haben dabei hohe Werte, selten beobachtete bekommen niedrige Werte. Für unsere Zwecke ist es eher unwichtig, wie die PAM-Matrizen berechnet werden, wichtiger ist ihre Bedeutung für die Anwendung. Je höher die Zahl der PAM-Matrix, desto längere, aber schwächere lokale Ähnlichkeiten werden im Allgemeinen aufgefunden.

Eine Alternative dazu stellen die BLOSUM-Matrizen von Henikoff und Henikoff (1992) dar. Während die PAM-Matrizen aus globalen Alignments ähnlicher Sequenzen abgeleitet wurden, stammen die BLOSUM-Werte aus lokalen Alignments von entfernter verwandten Sequenzen. Auch hier gibt es eine ganze Palette zur Auswahl, die je nach Ähnlichkeit der Sequenzen Verwendung finden. Die BLOSUM-Matrizen sind i.A. besser für lokale Alignments geeignet.

Gaps werden oft mit der affinen Gap-Funktion berechnet. Dabei sind zwei Parameter zu wählen: *Gap Opening Penalty* (GOP: $h+g$ beim paarweisen Alignment) und *Gap Extension Penalty* (GEP: g beim paarweisen Alignment).

Beispiel: PAM-Matrix

PAM10-Matrix für die 20 Aminosäuren

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	7	-10	-7	-6	-10	-7	-5	-4	-11	-8	-9	-10	-8	-12	-4	-3	-3	-20	-11	-5
R	-10	9	-9	-17	-11	-4	-15	-13	-4	-8	-12	-2	-7	-12	-7	-6	-10	-5	-14	-11
N	-7	-9	9	-1	-17	-7	-5	-6	-2	-8	-10	-4	-15	-12	-9	-2	-5	-11	-7	-12
D	-6	-17	-1	8	-21	-6	0	-6	-7	-11	-19	-8	-17	-21	-12	-7	-8	-21	-17	-11
C	-10	-11	-17	-21	10	-20	-20	-13	-10	-9	-21	-20	-20	-19	-11	-6	-11	-22	-7	-9
Q	-7	-4	-7	-6	-20	9	-1	-10	-2	-11	-8	-6	-7	-19	-6	-8	-9	-19	-18	-10
E	-5	-15	-5	0	-20	-1	8	-7	-9	-8	-13	-7	-10	-20	-9	-7	-9	-23	-11	-10
G	-4	-13	-6	-6	-13	-10	-7	7	-13	-17	-14	-10	-12	-12	-10	-4	-10	-21	-20	-9
H	-11	-4	-2	-7	-10	-2	-9	-13	10	-13	-9	-10	-17	-9	-7	-9	-11	-10	-6	-9
I	-8	-8	-8	-11	-9	-11	-8	-17	-13	9	-4	-9	-3	-5	-12	-10	-5	-20	-9	-1
L	-9	-12	-10	-19	-21	-8	-13	-14	-9	-4	7	-11	-2	-5	-10	-12	-10	-9	-10	-5
K	-10	-2	-4	-8	-20	-6	-7	-10	-10	-9	-11	7	-4	-20	-10	-7	-6	-18	-12	-13
M	-8	-7	-15	-17	-20	-7	-10	-12	-17	-3	-2	-4	12	-7	-11	-8	-7	-19	-17	-4
F	-12	-12	-12	-21	-19	-19	-20	-12	-9	-5	-5	-20	-7	9	-13	-9	-12	-7	-1	-12
P	-4	-7	-9	-12	-11	-6	-9	-10	-7	-12	-10	-10	-11	-13	8	-4	-7	-20	-20	-9
S	-3	-6	-2	-7	-6	-8	-7	-4	-9	-10	-12	-7	-8	-9	-4	7	-2	-8	-10	-10
T	-3	-10	-5	-8	-11	-9	-9	-10	-11	-5	-10	-6	-7	-12	-7	-2	8	-19	-9	-6
W	-20	-5	-11	-21	-22	-19	-23	-21	-10	-20	-9	-18	-19	-7	-20	-8	-19	13	-8	-22
Y	-11	-14	-7	-17	-7	-18	-11	-20	-6	-9	-10	-12	-17	-1	-20	-10	-9	-8	10	-10
V	-5	-11	-12	-11	-9	-10	-10	-9	-9	-1	-5	-13	-4	-12	-9	-10	-6	-22	-10	8

30

Beispiel: PAM-Matrix

Das Bild stellt eine PAM10-Matrix dar. Matches (z.B. GG) erhalten die höchsten Werte, Mismatches (z.B. SD) werden je nach evolutionärer Distanz bewertet.

Algorithmen für MSA

1. Exakte Verfahren – DP
2. Heuristiken
   - Progressive Verfahren
   - Iterative Verfahren
3. Konsistenzbasierte Verfahren

- Cédric Notredame: Recent progresses in multiple sequence alignment: a survey, Pharmacogenomics, Vol. 3, Nr. 1, 2002.

31

Algorithmen für MSA

Exakte Verfahren basieren meist auf einer Verallgemeinerung des DP-Ansatzes für paarweises Alignment. Sie können nur auf eine relative geringe Anzahl von Sequenzen angewendet werden (< 20) und sind meist auf die Sum-of-Pairs-Zielfunktion beschränkt. Daher werden in der Praxis für die Berechnung von multiplen Alignments vieler Sequenzen häufig Heuristiken angewendet, die allerdings das Auffinden der optimale Lösung hinsichtlich der gewählten Zielfunktion nicht garantieren können.

Man kann folgende Heuristiken unterscheiden:

- Progressive Verfahren, die ein multiples Alignment durch wiederholte Berechnung paarweiser Alignments bilden. Dabei wird in jedem Durchgang eine weitere Sequenz oder ein Alignment hinzugefügt.
- Iterative Verfahren, die ein bereits vorhandenes multiples Alignment wiederholt durch kleine Änderungen verbessern, bis keine Verbesserungen mehr möglich sind.

Bei konsistenzbasierten Verfahren ist das Ziel, jenes multiple Alignment zu finden, das am besten mit allen möglichen optimalen paarweisen Alignments übereinstimmt. Sie können exakt oder heuristisch gelöst werden.

Einen guten Überblick über die gebräuchlichsten progressiven und iterativen, aber auch exakten und konsistenzbasierten Verfahren gibt Cédric Notredame in seinem oben genannten Artikel (http://www.isrec.isb-sib.ch/DEA/module5/Course_Cedric/pharmacogenomics.pdf).

Exakte Verfahren für MSA

Variante des DP-Algorithmus
(Vereinfachung: $s_1, \ldots, s_k$ mit Länge n)

- k-dimensionales Feld a mit n+1 Einträgen pro Dim.
- $a[i_1, \ldots, i_k]$ enthält Wert des besten Alignments von $s_1[i_1], \ldots, s_k[i_k]$
- Initialisierung: $a[0, \ldots, 0]$ mit 0
- Feldeinträge: $a[i_1, \ldots, i_k]$ auf Basis der 2^k-1 Vorgänger und dem aktuellen Spaltenwert ermitteln
- Optimales Alignment durch Rückverfolgung

32

Exaktes Verfahren für MSA

Der DP-Algorithmus, den wir für das paarweise Alignment kennen gelernt haben, kann leicht für k>2 Sequenzen verallgemeinert werden. Der Einfachheit halber gehen wir nun davon aus, dass alle Sequenzen dieselbe Länge n haben. Wir benötigen nun ein k-dimensionales Feld a mit n+1 Einträgen pro Dimension, um die optimalen Werte für die multiplen Alignments der Präfixe der Sequenzen zu speichern. Der Eintrag $a[i_1, ..., i_k]$ enthält nun also Wert des besten Alignments von $s_1[i_1], ..., s_k[i_k]$. Das Feld $a[0, ..., 0]$ wird wiederum mit 0 initialisiert. Die übrigen Feldeinträge werden auf Basis ihrer 2^k-1 Vorgänger und dem aktuellen Spaltenwert ermittelt. Davon wird wiederum der maximale Wert genommen.

Das optimale globale multiple Alignment kann wiederum ausgehend vom Feld $a[n,...,n]$ durch Rückverfolgen des Weges gefunden werden. Dieses Prinzip lässt sich leicht auf Sequenzen unterschiedlicher Längen $n_1, ..., n_k$ anpassen.

Aufwand für dynamische Programmierung

- Berechnung des kompletten Feldes: n^k Einträge

$\Rightarrow$ Speicherplatz $O(n^k)$, Laufzeit: $\Omega(n^k)$

- Berechnung eines Eintrags: 2^k-1 Vorgänger (Symbol oder Leerzeichen jeder Sequenz) $\Rightarrow O(2^k)$
- Berechnung der Spaltenwerte:
  - einfaches Schema (Symbole zählen) $\Rightarrow O(k)$
  - SP-Score: $k(k-1)/2$ Paare $\Rightarrow O(k^2)$

$\Rightarrow$ Gesamtlaufzeit: $O(k2^k n^k)$ bzw. $O(k^2 2^k n^k)$

MSA mit SP-Maß ist *NP*-vollständig!

33

Aufwand für dynamische Programmierung

Um den Aufwand für den Ansatz mit Dynamischer Programmierung abzuschätzen, müssen wir die verschiedenen Komponenten des Algorithmus betrachten.

Es werden n^k Feldeinträge benötigt, daher ist der benötigte Speicherplatz in $O(n^k)$. Dieser Wert ist auch die untere Schranke für die Laufzeit des Algorithmus, da das Ausfüllen des gesamten Feldes einen zeitlichen Aufwand von $O(n^k)$ verursacht.

Zur Berechnung eines einzelnen Eintrages müssen ferner 2^k-1 Vorgängereinträge herangezogen werden, da es jeweils 2^k-1 Möglichkeiten für den Aufbau der aktuellen Spalte des Alignments gibt: für jede Sequenz kann entweder das Symbol oder ein Leerzeichen eingesetzt werden. Die Variante mit Leerzeichen in allen Sequenzen scheidet natürlich aus.

Der Aufwand für die Berechnung der Spaltenwerte hängt vom jeweiligen Bewertungssschema ab. Für ein einfaches Schema, bei dem beispielsweise nur die Anzahl der Symbole in einer Spalte gezählt wird, ist das in $O(k)$ möglich. Verwendet man das Sum-of-Pairs-Maß, dann erhöht sich die Laufzeit dieses Teils auf $O(k^2)$. Hier müssen nämlich $k(k-1)/2$ paarweise Werte aufsummiert werden.

Die Gesamtlaufzeit des Algorithmus ist in jedem Fall exponentiell in der Anzahl der Sequenzen. Es wurde gezeigt, dass das *Multiple Alignment*-Problem mit dem SP-Maß *NP*-vollständig ist.