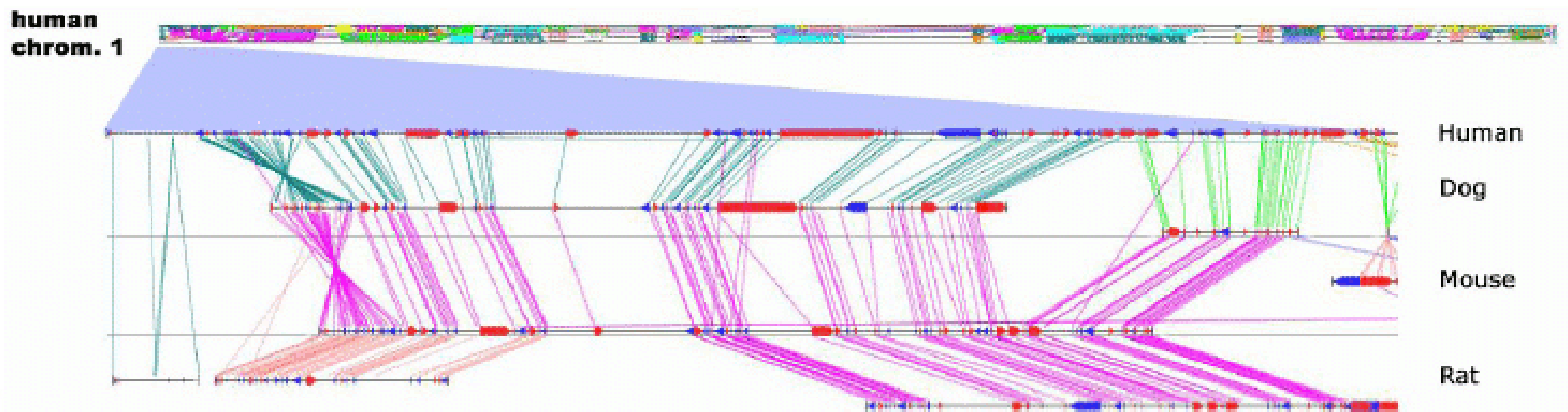
Bioinformatik

Approximative Stringvergleiche

Ulf Leser
Wissensmanagement in der
Bioinformatik

Whole Genome Aligment

- Genome unterliegen Evolution
- Neben Punktmutationen treten großräumige Veränderungen auf
  - Duplikation von Chromosomen
  - Duplikation von langen DNA Abschnitten
- Duplikate unterliegen dann unabhängiger Evolution
- In nahe verwandten Organismen (z.B. Säugetiere) sind Genome „durcheinander geschüttelte“, approximative Kopien voneinander



Finden von MUMs

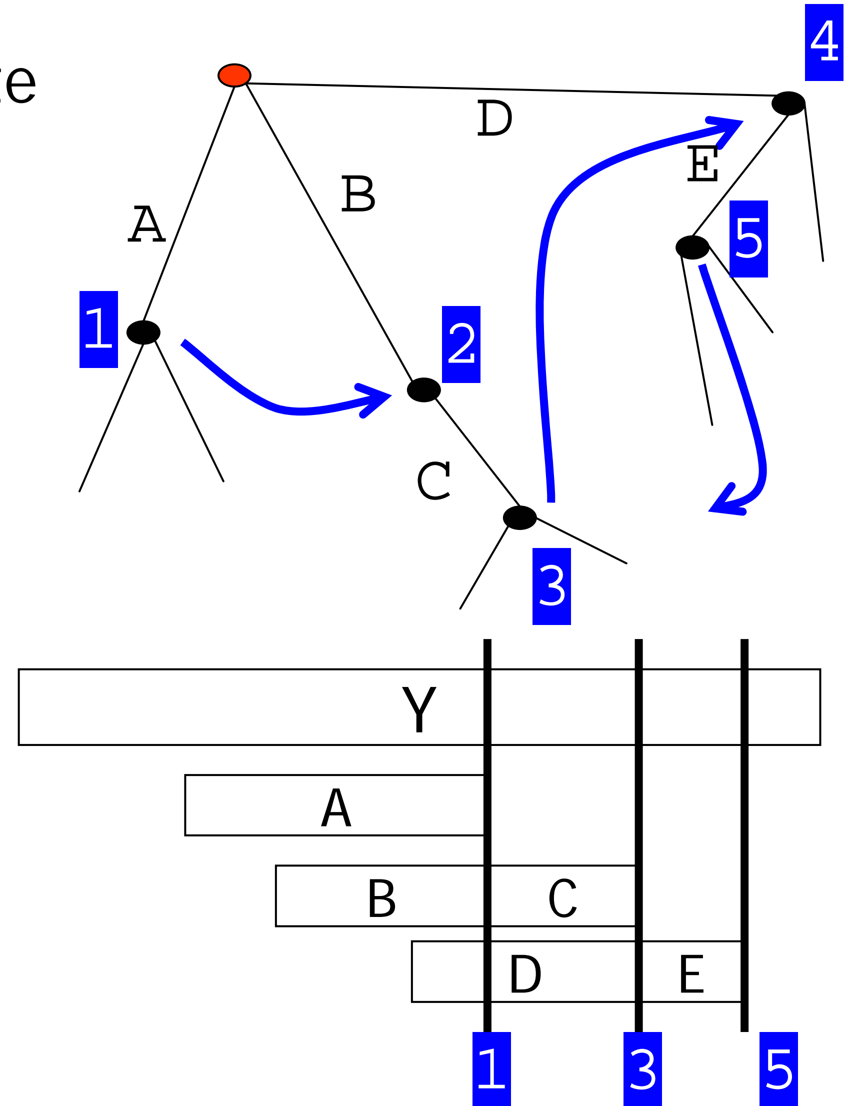
- Beobachtung
  - Evolutionärer Abstand in nahe verwandten Spezies ist klein
  - Trotz unabhängiger Evolution findet man genügend lange, identische Sequenzabschnitte
  - Also: Anwendung für exaktes Stringmatching
- Aufgabe
  - Gegeben zwei Chromosome X, Y (Mensch und Maus)
  - Finde Bereiche von X, die identisch in Y vorkommen
- Genauer: Finde längste identische Teilsequenzen über einer Mindestlänge
- Außerdem: Um klare Zuordnungen zu ermöglichen, beschränkt man sich auf eindeutige Teilsequenzen
  - Dürfen in X und Y nur jeweils einmal vorkommen
- Wir wollen alle MUMs zwischen Maus und Menschen finden
  - Vorschläge?

Möglichkeit 2

- Idee nach Chang and Lawler; verwendet in MUMmer 2
  - Wir bauen Suffixtrees nur für ein Chromosom X
  - Suffix-Links werden behalten
  - Alle Chromosomen der anderen Spezies „streamen“ wir gegen X
- Komplexität
  - Man baut 2c „kleinere“ Suffixbäume in jeweils O(m)
  - Gegen jeden streamen wir c mal in O(m)
  - Zusammen: $O(2c*m) + O(c^2*m)$
- Hauptvorteil: IO vernachlässigbar
  - Suffixbäume kann man im Hauptspeicher bauen
  - Vergleichssequenz wird sequentiell gelesen

Illustration

- Vereinfachte Annahme: Letzte Matches immer in Knoten
- Mismatch in Knoten 1
  - mit Label A
- SL zu Knoten 2
  - mit Label B
  - B ist Suffix von A
- Weiter in X matchen bis Mismatch in Knoten 3
  - mit Label BC
- SL zu 4
  - mit Label D
  - D ist Suffix BC
- Weiter in Y matchen bis …
- …



Fast richtig

- Wenn man ab k' weiter matched
  - Und einen MUM findet
  - Dann ist der nicht notwendigerweise maximal
  - Das rechte Ende ist klar
    - Der aktuelle Mismatch
  - Aber das linke Ende nicht
- Linke Enden müssen explizit geprüft werden
  - In Y und in T vor dem Suffix nach links vergleichen bis Mismatch
- Damit matched man Zeichen in Y mehrmals
  - Außerdem matched man die Zeichen zwischen k und einem Mismatch zweimal
- Aber: Wenn minimale MUM Länge sinnvoll groß, gibt es nur wenige MUMs – praktisch lineare Laufzeit (in |Y|)

Suffixarrays

- Definition
  - *Das Suffixarray A für String S ist ein Integerarray der Länge |S|, in dem A[i] die Startposition des i-ten Suffix von S, sortiert nach lexikographischer Ordnung, enthält*
- Beispiel

```
12345678901
mississippi
```

```
mississippi    i              A[1]=11
ississippi     ippi           A[2]=8
ssissippi      issippi        A[3]=5
sissippi       ississippi     A[4]=2
issippi        mississippi    A[5]=1
ssippi         pi             A[6]=10
sippi          ppi            A[7]=9
ippi           sippi          A[8]=7
ppi            sissippi       A[9]=4
pi             ssippi         A[10]=6
i              ssissippi      A[11]=3
```

Trick zur linearen Konstruktion

- Ablauf in „lexikographischer“ Depth-First Ordnung
  - Wir laufen den Suffixbaum Depth-First ab
  - An jedem Knoten wählen wir die Kinder in der lexikographischen Reihenfolge der Kantenlabel
    - „$“ gilt als kleiner als alle anderen Zeichen
  - Reihenfolge der Blätter bildet das Array
    - Erstes Blatt mit Beschriftung $i_1 \Rightarrow A[1] = i_1$
    - Zweites Blatt mit Beschriftung $i_2 \Rightarrow A[2] = i_2$
- Komplexität: O(m)
  - Depth-First ist abhängig von Anzahl Knoten
  - Wenn die Kinder als sortierte verkettete Liste oder als Array gespeichert sind ist jede Entscheidung konstant
- Aber
  - Wir haben unser Platzproblem damit natürlich nicht gelöst

Suche mit Suffixarrays

- Finde alle Vorkommen eines Pattern P im Suffixarray für String T
- Ideen?

- Suche alle Vorkommen von

  P=„ssi" in „mississippi"

  - Erinnerung: Jeder Substring ist Präfix (mindestens) eines Suffix
  - P liegt also am Anfang eines Suffix (wenn P in S)
  - Suffixe liegen alle sortiert vor
  - ➢ Binäre Suche im Suffixarray

```
a[1]=11     i
a[2]=8      ippi
a[3]=5      issippi
a[4]=2      ississippi
a[5]=1      mississippi
a[6]=10     pi
a[7]=9      ppi
a[8]=7      sippi
a[9]=4      sissippi
a[10]=6     ssippi
a[11]=3     ssissippi
```

Manber/Myers – Grundaufbau 1

- Wir erzeugen ein Array A mit den Suffixen der Länge nach absteigend sortiert
  - Genauer: Startpositionen der Suffixe
- A wird mit Bucket-Sort nach dem ersten Zeichen sortiert
- Dann sind die Buckets schon richtig sortiert – wir müssen aber noch innerhalb jedes Buckets sortieren
- Um nach den zweiten Zeichen in einem Bucket B zu sortieren
  - Jedes zweite Zeichen ist natürlich das erste Zeichen des nächstkleineren Suffix
  - Nach denen haben wir schon sortiert
  - Ausnutzen – erste Sortierung nachschlagen statt neu vergleichen
  - Das kann man für alle Buckets mit einem Durchlauf von A machen

Beispiel

```
123456789012
MISSISSIPPI$
```

$	12: $
I	2: ISSISSIPPI$ 5: ISSIPPI$ 8: IPPI$ 11: I$
M	1: MISSISSIPPI$
P	9: PPI$ 10: PI$
S	3: SSISSIPPI$ 4: SISSIPPI$ 6: SSIPPI$ 7: SIPPI$

- Sortierung der zweiten Zeichen entspricht Sortierung der ersten Zeichen
  - Im ersten Schritt wurden schon alle im String vorkommenden Zeichen sortiert
- Setze in jedem Bucket einen Pointer auf das erste Element
- Wir gehen einmal durch das Array (i)
  - Sei A[i] = j. Dann ist S[j] das zweite Zeichen von dem Suffix an Position j-1
  - Da wir die i sortiert durchlaufen, ist das Suffix ab Position j-1 das nächst kleinere in seinem Bucket
  - Also schieben wir den Wert an den Startpointer seines Buckets und verschieben den Startpointer um eins

Sortierung nach 3. und 4. Zeichen

- Bisher
  - Alle Elemente eines Buckets haben das gleiche, 2-buchstabige Präfix
  - Alle Buckets liegen sortiert nach dem Präfix im Array
- Nächste Schritte
  - Seien S[i..] = s_i … s_n\$ und S[j..] = s_j … s_n\$ wie vorher
  - Man müsste nun s_{i+2} mit s_{j+2} vergleichen
  - Das ist das gleiche wie S[i+2..] = s_{i+2} … s_n\$ mit S[j+2..] = s_{j+2} … s_n\$
  - Die S[i+2..] und S[j+2..] haben wir schon sortiert, und zwar nach den ersten beiden Zeichen
- Wir sortieren simultan nach dem 3. und 4. Zeichen
  - Wir gehen wieder durch das Array
  - Sei A[i] = j. Dann ist S[j] das dritte Zeichen von dem Suffix ab Position j-2
  - Schiebe den Index j-2 an den Startpointer seines Buckets und verschieben den Startpointer um eins
  - Das sortiert automatisch nach dem 3. und 4. Zeichen
- Dann nach 5-8. Zeichen
- Etc.

Für die nächsten Wochen

Approximatives Matching

Inhalt dieser Vorlesung

- Approximative Stringvergleiche
- Dotplots
- Edit-Abstand und Alignment
- Naiver Algorithmus

Approximativer Stringvergleich

- Bisher: Suche nach exakten Vorkommen von Pattern im Template
- Aber
  - Auch nicht-exakte Vorkommen hochgradig interessant
  - Manchmal interessieren gerade die Unterschiede
  - Gleitender Übergang zwischen
    - Exakter Match
    - Guter Match
    - Schlechter Match
- Was ist ein „approximativer“, „guter“, „schlechter“ Match?

Wie ähnlich sind sich zwei Sequenzen?

- „AGGTAG“ und
  - AGTAG — G zu wenig
  - AGGTAG — Identisch = überaus ähnlich
  - AGGATAG — A zu viel
  - AGTTCAG — G durch T ersetzen und C löschen
  - ...TGAGGTAGGTT... — Sehr viel löschen
- Welche Matches sind besser?
  - „Ähnlichkeit“ muss quantifiziert werden
  - Idee: Wie sehr muss man eine Sequenz verändern, um die andere zu erzeugen
    - Man konnte auch Buchstaben zählen, Länge vergleichen, Anzahl GC nehmen, ...

Suche mit Sequenzen

- Ähnlichkeit vergleicht zwei Strings
  - Globales Alignment
- Suchprobleme
  - Gegeben Sequenz (Exon), welche Datenbanksequenzen (Clone, Genome, Chromosomen) enthalten einen ähnlichen Abschnitt?
  - Gegeben Sequenz (Gen) , welche Datenbanksequenzen (Clone, Genome, Chromosomen) enthalten einen Abschnitt, der einem Abschnitt meiner Suchsequenz ähnlich ist?
    - Lokales Alignment

Motivation

- BLAST / FASTA und Verwandte sind *die* Bioinformatik Anwendung
  - Teilweise synonym für „Bioinformatik"
- Grundlegende Gesetzmäßigkeit der Bioinformatik

Hohe Sequenzähnlichkeit bei DNA, RNA und Proteinen heißt in der Regel ähnliche Funktion bzw. Struktur

Begründung

- Biochemische Aktivität der Proteine wird bestimmt durch
  - 3D Faltung der Proteine
  - Vorkommen bestimmter Aminosäuren an bestimmten Stellen (Motiv, Domäne)
  - Interaktion und Modifikation von Proteinen
- Wesentliches Element ist die 3D Struktur von Proteinen
- Der Weg DNA – Struktur ist nicht eindeutig
  - Code zur Übersetzung DNA-Triplet – Aminosäure ist redundant
  - Abweichungen verändern die Funktion oftmals nicht
  - Nicht alle Aminosäuren sind (gleich-)wichtig für die Struktur
- Durch Evolution entstehen Variationen
  - Funktionale Änderungen schaffen Varianten (oder sind letal)
  - Evolution geht meist in kleinen Schritten: Kleine Änderungen, leicht veränderte Struktur, leicht verändertes funktionales Verhalten
  - Stammbaumentstehung (Phylogenie)

Genetischer Code

Wildtyp

C	T	T	A	G	T	G	A	C	T	A	C	G	G	T	A	A	A	DNA

Leu	Ser	Asp	Tyr	Gly	Lys	Protein

Fatale Mutationen

C	T	T	A	G	T	G	A	C	T	A	G	G	G	T	A	A	A	DNA

Leu	Ser	Asp	Stop-Codon	Protein

Leseraster Mutationen

C	T	T	A	G	T	G	A	A	C	T	A	C	G	G	T	A	A	A	DNA

Leu	Ser	His	Asp	Leu	Thr	Protein

neutrale Mutationen

C	T	T	A	G	C	G	A	C	T	A	C	G	G	T	A	A	A	DNA

Leu	Ser	Asp	Tyr	Gly	Lys	Protein

Funktionale Mutationen

C	T	T	A	G	T	G	A	A	T	A	C	G	G	T	A	A	A	DNA

Leu	Ser	Glu	Tyr	Gly	Lys	Protein

Variationen in Proteinsequenzen

- Änderungen in Proteinsequenz sind „oft“ harmlos
  - Viele Änderungen ändern die Struktur nicht
  - Nicht alle Teile einer Struktur sind gleich wichtig: Periphere Teile versus funktionale Domänen
- Proteine
  - Ab Ähnlichkeit von 20-30% geht man von verwandter Funktion aus
  - Aber auch 85% Identität ist keine Garantie
  - Eine einzelne Mutation kann schon die Funktion verändern

Fazit

- Relevant ist die biologische Funktion, nicht die Sequenz
- Sequenz und Funktion hängen eng zusammen, sind aber nicht direkt ableitbar
- Bestimmung von Funktion ist extrem aufwändig (wenn überhaupt möglich), Bestimmung von Sequenzen dagegen sehr billig
- Also: Annäherung der Funktion über Sequenzähnlichkeiten
  - Geburtsüberlegung der Bioinformatik
  - Basiert auf approximativem Stringmatching

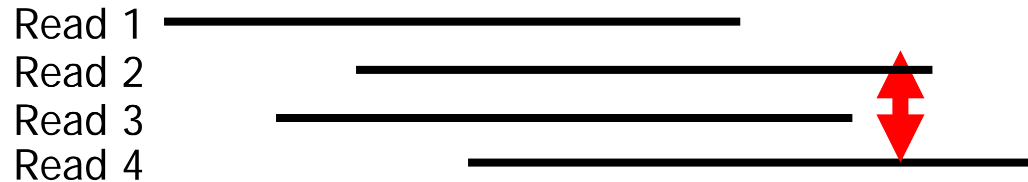
Drei konkrete Beispiele

- Sequenzieren: Assembly
- Bestimmung funktionaler Domänen
- Comparative Genomics

Assembly

- Szenario: Shotgun Sequenzierung
- Ergebnis des Base Calling: Einzelne Reads
- Gesucht: Gesamtsequenz
- … bzw.:
  - zusammenhängende Stücke (Contigs)
  - möglichst sichere Sequenz (Redundanz!)
- Assembly: Berechnung der „Konsensussequenz“

Assembly

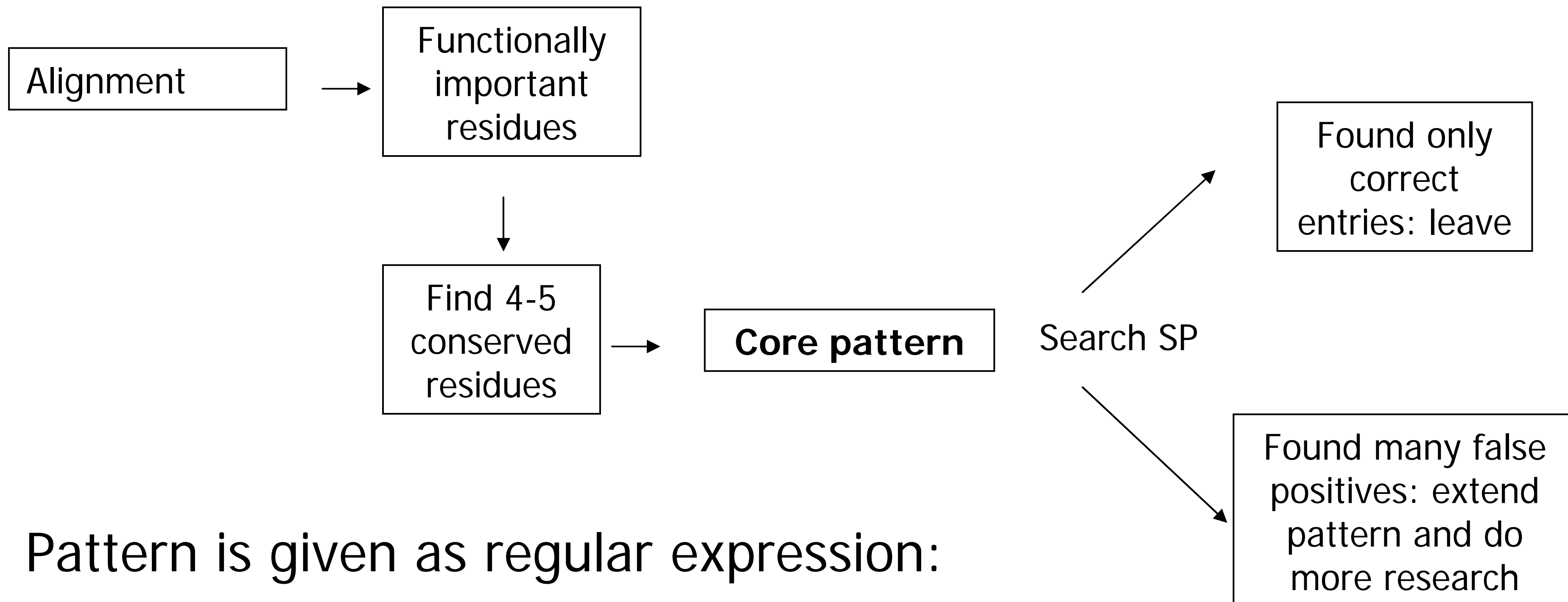


- Gesucht
  - Überlappungen der einzelnen Reads
  - Fehler sind immer vorhanden (insb. fehlende/eingeschobene Blöcke durch Fehler in bakterieller Replikation)
  - Dadurch keine perfekten Überlappungen
  - Oftmals mehrere „unterschiedlich gute“ Möglichkeiten
- Grundverfahren: Schnelle Bestimmung möglichst guter Überlappungen
  - Approximatives Stringmatching

Proteindomänen

- Pattern, Domäne, Motiv, Site: Teile einer Proteinsequenz mit funktionaler Bedeutung
  - Bindungsstellen, enzymatische Aktivität, Signal, etc.
  - Beschrieben z.B. durch reguläre Ausdrücke, Fingerprints, Profile, …
- Datenbanken von Domänen
  - PROSITE, Pfam, InterPro, BLOCKS, PRINTS, …
- Beispiel PROSITE
  - Beginn: Finden einer „interessanten“ Teilsequenz in der Literatur
  - Identifikation ähnlicher Sequenzen in anderen Proteinen
    - Approximatives Stringmatching
  - Identifikation der konservierten Aminosäuren
    - Multiple Sequence Alignment

Entstehung von Prosite Pattern



Pattern is given as regular expression:

[AC]-x-V-x(4)-{ED}

ala/cys-any-val-any-any-any-any-(any except glu or asp)

Quelle: „PROTEIN PATTERN DATABASES", EBI

Comparative Genomics

- Bestimmung von Protein/Genfunktion in anderen Spezies wesentlich leichter als beim Menschen
  - Bakterien, Knock-out Mäuse, etc.
- Viele Gene sind hochgradig konserviert
  - Maus – Mensch: 97% Sequenzidentität
  - „Housekeeping Genes“ in allen Organismen ähnlich vorhanden
  - Die 4% „aktivsten“ (am besten verbundenen) Proteine sind in allen (bisher sequenzierten) bekannten Organismen vorhanden
- Vorwärts
  - Finden und sequenzieren eines neues Genes beim Menschen
  - Suchen nach ähnlichen Sequenzen in anderen Organismen
  - BLAST gegen Genbank / EMBL
- Rückwärts
  - Bestimmung der Funktion eines Genes einer anderen Spezies
  - Suche nach ähnlichen Sequenzen beim Menschen
  - BLAST gegen Genbank / EMBL

Approximatives Matchen außerhalb der Bioinformatik

- Anwendungen außerhalb der Bioinformatik
  - Unscharfe Volltextsuche
    - Suche mit „Xylofon“ und finde auch „Xhylophon“
  - Personenabgleich
    - Ist „Herr Müller, 27, Stargarder Str 54“ identisch zu „Hr. Mueller, 27, Stagarder Str. 54“ ?
  - Phonetische Suche
    - Finde alle Meyer, Meier, Maier, Mair, …

- Erste Annäherung: Dotplots

Dotplot

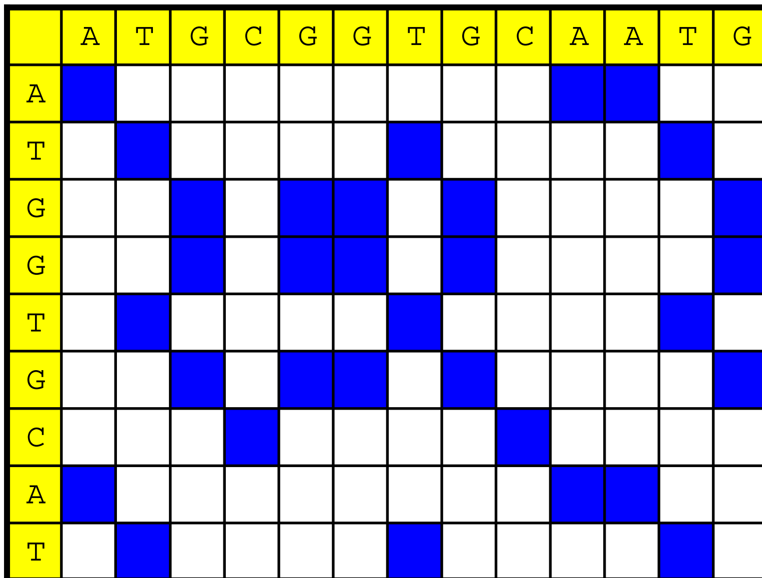
- Definition

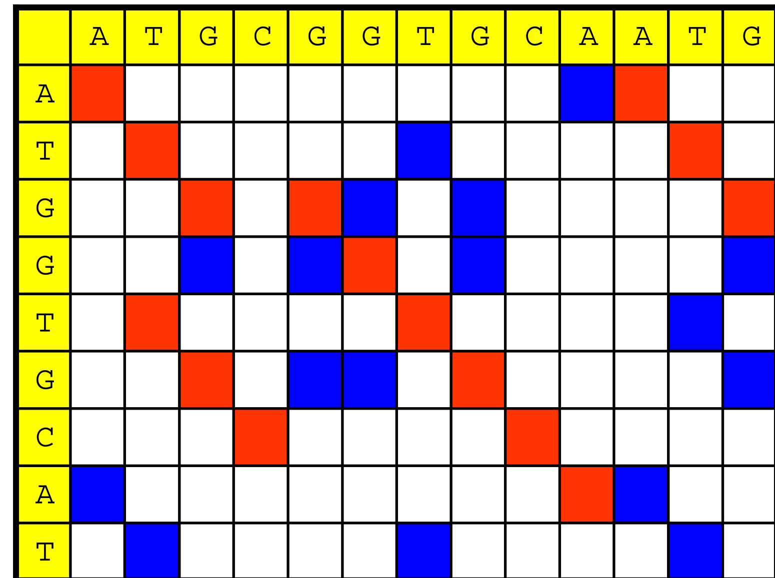
  Ein Dotplot zweier Strings A, B ist eine Matrix M mit
  - *Die Spalten entsprechen den Zeichen von A*
  - *Die Zeilen entsprechen den Zeichen von B*
  - *M[a,b]=1 gdw. A[a] = B[b]; sonst 0*

	A	T	G	C	G	G	T	G	C	A	A	T	G
A	■									■	■		
T		■					■					■	
G			■		■	■		■					■
G			■		■	■		■					■
T		■					■					■	
G			■		■	■		■					■
C				■					■				
A	■									■	■		
T		■					■					■	

Dotplot und gleiche Teilstrings

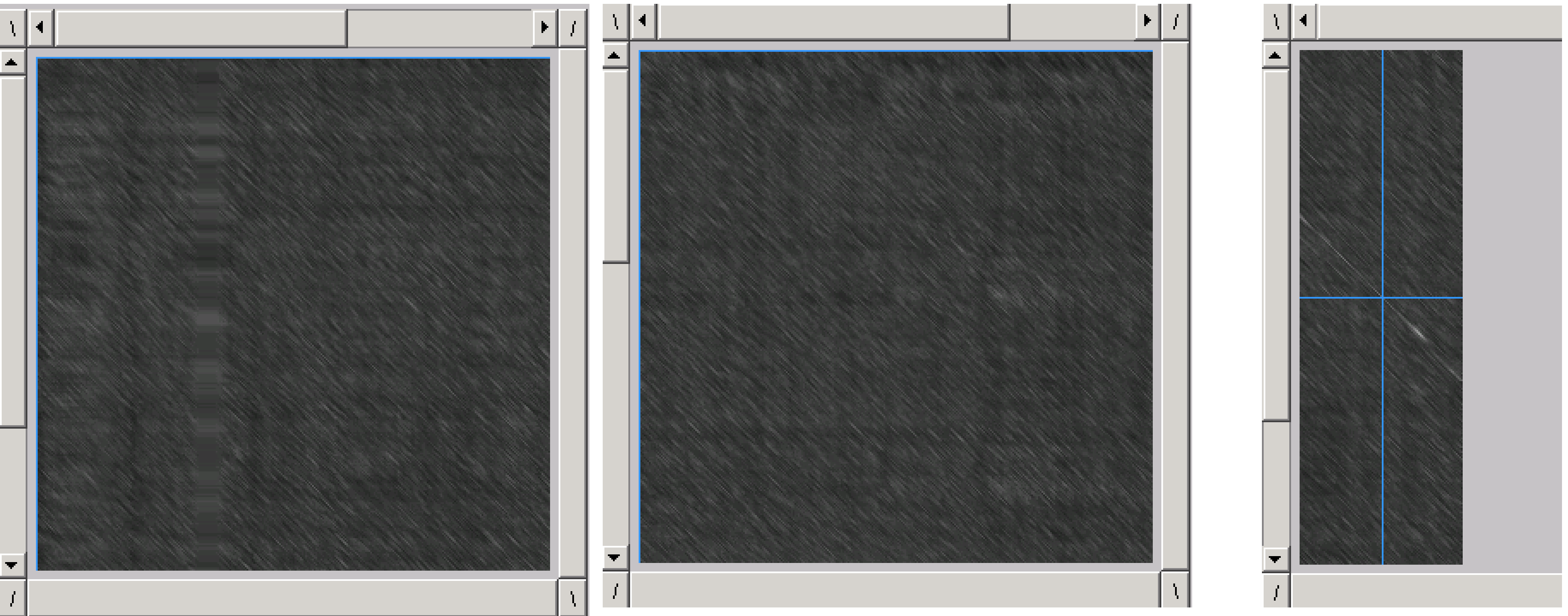
- Wie erkennt man gleiche Teilstrings im Dotplot?





- Diagonalen von links-oben nach rechts-unten
  - Größter gemeinsamer Teilstring – längste Diagonale
  - Visuell bei kurzen Strings möglich

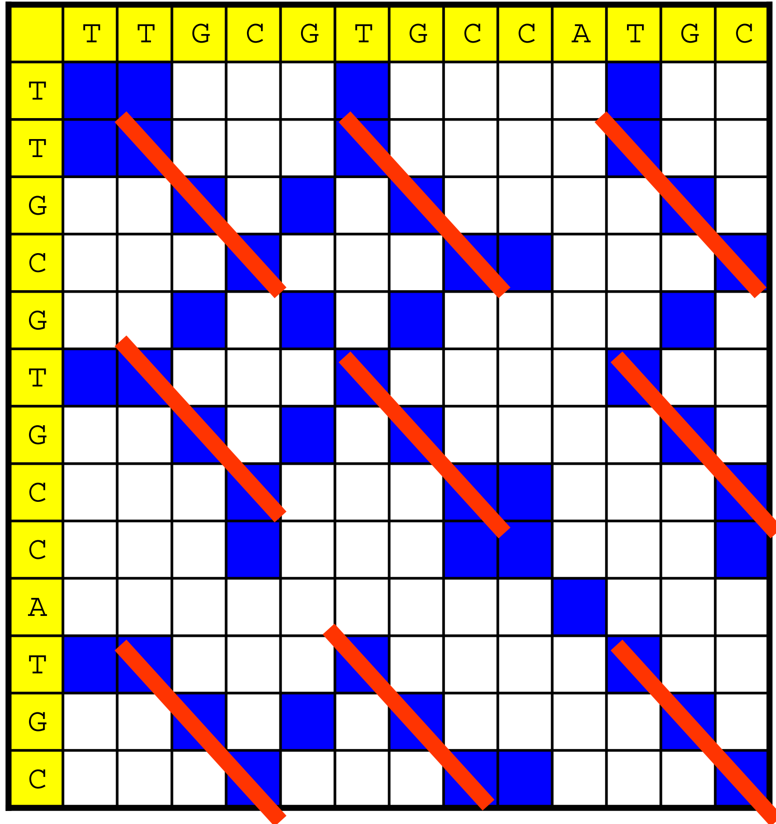
Visuelle Untersuchung

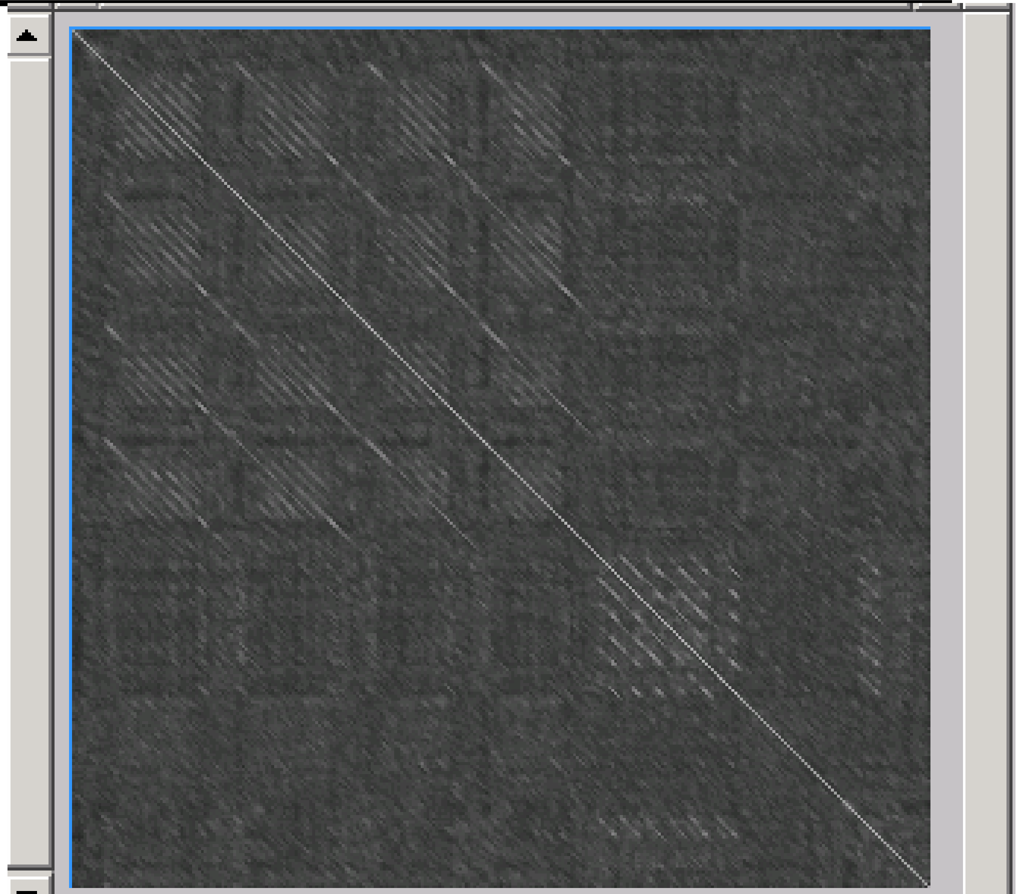


- Helligkeit: Ähnlichkeit im Umfeld eines Pixels

Quelle: Dotlet, http://www.isrec.isb-sib.ch/java/dotlet/Dotlet.html

Repetitive Sequenzen





- Dotplot mit A=B
  - Zitat (Genbank, P24014):

```
[SIMILARITY] CONTAINS 7 EGF-LIKE DOMAINS.
[SIMILARITY] Contains 24 leucine-rich (LRR) repeats.
```

Erweiterungen von Dotplots

- Trennen der signifikanten Übereinstimmungen vom Rauschen ist schwierig
- Sliding Windows
  - Statt Zeichen – Zeichen vergleiche Substring – Substring
  - Schieben eines Fensters der Länge l über beide Sequenzen
  - Dot nur dann, wenn mindestens z Zeichen in l matchen
  - Beispiel: DNA plotten mit l=15 und z=10
    - Da DNA (4 Zeichen) sehr viel Rauschen erzeugt
  - ➢ Kurze Matches verschwinden
  - ➢ Wichtige Matchregionen treten klarer zutage

Finden längster Teilstrings mit Dotplots

- Gegeben Dotplot M zweier Strings A, B
- Gesucht: Längster gemeinsamer Teilstring
  - Annahme: $|A|=|B|=m$
- Naives Verfahren
  - Prüfe jede der 2*m Diagonalen
  - Suche jeweils zusammenhänge 1'er (linear in m)
  - Merke das längste zusammenhängende Stück
  - Komplexität: $O(m^2)$
  - (Zusätzlich: Konstruktion von M - wie schwierig?)
- Wir kennen schon lineare Algorithmen
  - Welche?
- Außerdem wollen wir approximativ matchen …

Abstandsmaße und Algorithmen

- Quantifizierung von „Ähnlichkeit“
- Editabstand

Abstandsmaße

- Approximatives Stringmatching sucht Ähnlichkeiten
  - Welcher Substring von T ist am ähnlichsten zu P?
  - Welcher String $T_1,\ldots,T_n$ ist am ähnlichsten zu T?
- Voraussetzung dafür
  - Was heißt ähnlich?
  - Was heißt „am ähnlichsten"?
- Quantifizierung des Abstandes zweier Strings
  - Definition von Ähnlichkeit ist oft eine sehr schwierige Aufgabe
  - Ähnlichkeit ist abhängig vom Gegenstand und Aufgabe
    - Wann sind sich Gesichter ähnlich - Haarfarbe zählt weniger als Augenfarbe?
    - Wann sind sich Texte ähnlich – gleiche Wörter oder gleicher Inhalt?

Mögliche Maße

- Hammingabstand
  - Voraussetzung: |A|=|B|
  - Vergleiche A und B Zeichen für Zeichen
  - Hammingabstand = Anzahl der Mismatches
  - Verwendung: Korrektur bei digitaler Signalübertragung
  - Beispiel: ha(CGTGCTCGC, ACGTGCTCG)= 9
  - Das kann nicht in unserem Sinne sein
- Jaccard-Abstand (auf q-Grammen)
  - Wir haben zwei Strings (Texte) A und B
  - Berechne alle n-Gramme in beiden Texten
  - Abstand der Texte

$$d(A,B) = \frac{qgram(A) \cap qgram(B)}{qgram(A) \cup qgram(B)}$$

Abstandsmaß für unsere Aufgabe

- Biologischen Hintergrund nicht vergessen
  - Situation: Wir haben humane Gensequenz A und suchen ähnliche Sequenzen (B) in anderen Organismen
  - Annahme: A und B haben gemeinsamen Vorfahren und sind durch evolutionäre Prozesse entstanden
  - Einfaches Modell: Basenaustausch, Baseneinfügung, Basenlöschen

Begriffe

- Sequenzen heißen
  - Homolog, wenn sie einen gemeinsamen Ursprung haben und von diesem durch Evolution divergiert sind
  - Ortholog, wenn sie in verschiedenen Spezies vorkommen, aber vom gleichen „Vorfahren“ abstammen
  - Paralog, wenn sie durch Duplikation innerhalb einer Spezies entstanden sind
- Ob zwei ähnliche Sequenzen homolog sind, kann man eigentlich nicht bestimmen
  - Nur Indizien sammeln
  - Ähnlichkeit der Sequenz ist ein sehr starkes Indiz
    - Andere: Lage im Genom, Regulationsmechanismen, Beteiligung in den gleichen Stoffwechselwegen an gleicher Stelle, …

Editskripte

- Definition
  Ein Editskript e für zwei Strings A, B aus $\Sigma^{*}=\Sigma \cup$ "_" ist eine Sequenz von Editieroperationen
  - *I (Einfügen eines Zeichen $c \in \Sigma$ in A)*
    - *Dargestellt als Lücke in A; das neue Zeichen erscheint in B*
  - *D (Löschen eines Zeichen c in A)*
    - *Dargestellt als Lücke in B; das alte Zeichen erscheint in A*
  - *R (Ersetzen eines Zeichen in A mit einem anderen Zeichen in B)*
  - *M (Match, d.h., gleiche Zeichen in A und B an dieser Stelle)*

  so, dass e(A)=B
- Beispiel: A=„ATGTA", B=„AGTGTC"
  -
```
MIMMMR          IRMMMDI
A_TGTA          _ATGTA_
AGTGTC          AGTGT_C
```

Editabstand

- Offensichtlich gibt es für A,B ziemlich viele Editskripte
- Definition
  - *Die Länge eines Editskript ist die Anzahl von Operationen o im Skript mit* $o \in \{I, R, D\}$
  - *Der Editabstand zweier Strings A, B ist die Länge des kürzesten Editskript für A, B*
- Bemerkung
  - Matchen zählt nicht – interessant sind nur die Änderungen
  - Anderer Name: Levenshtein-Abstand
  - Es gibt oft verschiedene kürzeste Editskripte
  -

```
IMMMMMD      DMMMMMI
_AGAGAG      AGAGAG_
GAGAGA_      _GAGAGA
```

Alignment

- Andere Darstellung: Alignments
- Definition
  - *Ein (globales) Alignment zweier Strings A,B ist eine Untereinanderanordnung von A und B, jeweils mit beliebigen zusätzlichen Leerzeichen (_), ohne das zwei Leerzeichen untereinander stehen*
    - Achtung: Untereinanderstehende Zeichen müssen nicht matchen
  - *Der Alignmentscore eines Alignment ist die Anzahl von Leerzeichen und Mismatches*
  - *Der Alignmentabstand zweier Strings A, B ist der minimale Alignmentscore aller Alignments der beiden Strings*
- Beispiele

```
– A_TGT_A        A_T_GTA        _AGAGAG        AGAGAG_
  AGTGTC_        _AGTGTC        GAGAGA_        _GAGAGA
```

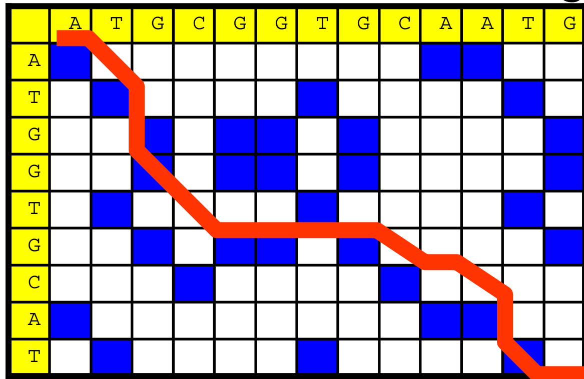
Score:	**3**	**5**	**2**	**2**

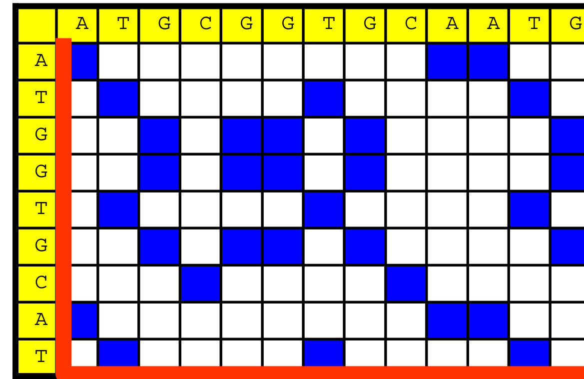
Alignment versus Editskript

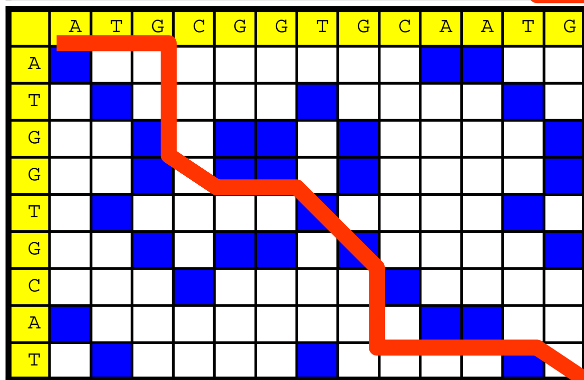
- Sind logisch äquivalent. Übersetzung ist einfach
  - Alignment – Editskript: Definiere Operationen im Skript wie folgt
    - Space in A ergibt ein I — Space in B ergibt ein D
    - Replacement ergibt ein R — Match ergibt ein M
  - Editskript – Alignment: Umgekehrter Weg
- Alignments
  - Besser zur Visualisierung
  - Charakterisieren einen Stringvergleich
- Editskripte
  - Bezeichnen einen Transformationsprozess
  - Entspringen einem Evolutionsmodell
- Wir werden diverse Varianten betrachten
  - Unterschiedliche Kosten für Operationen I, D, R
  - Unterschiedliche Kosten für R je nach Zeichenpaar
  - Gesonderte Behandlung von Gaps
  - …

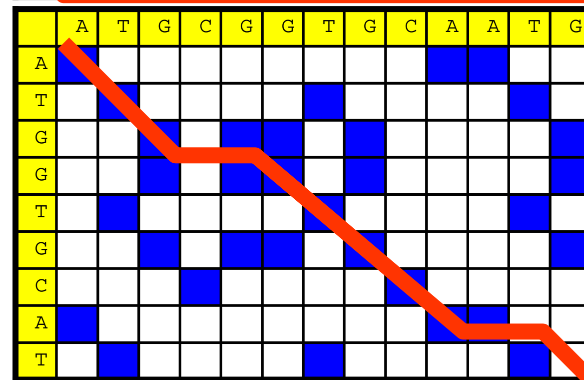
Alignments und Dotplots

- Sei |A|=m, |B|=n
- Betrachte Pfade im Dotplot von (1,1) nach (m,n)
  - Pfad startet in linker oberer Ecke
  - Erlaubte Schritte: nach rechts, nach unten und nach rechts-unten (diagonal)
  - Pfad: Zusammenhänge Menge von Schritten bis (m,n)







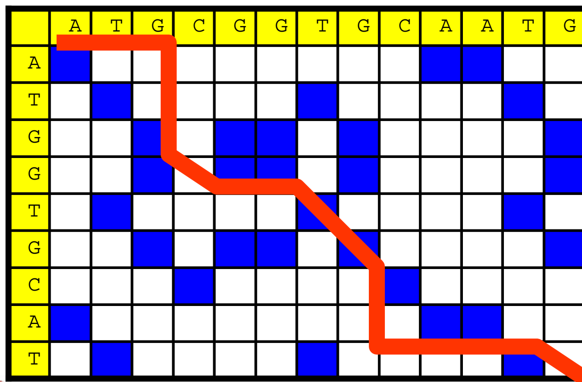


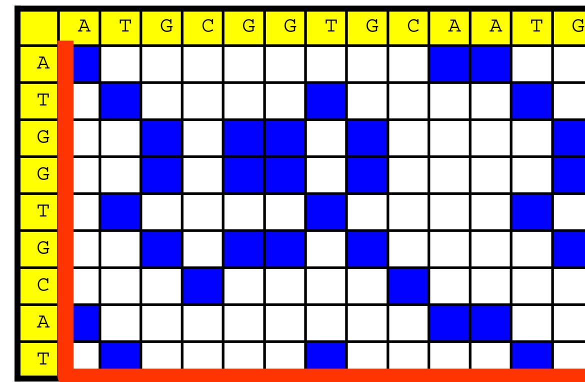
Alignments und Dotplots 2

- Übersetzung von Pfaden im Dotplot in Alignments
  - Dotplot: Sei A horizontal und B vertikal aufgetragen
  - Alignment: Sei A über B angeordnet
  - Schritt nach rechts: Nächstes Zeichen von A; „_“ in B
  - Schritt nach unten: Nächstes Zeichen von B; „_“ in A
  - Schritt nach rechts-unten: Nächstes Zeichen von A und B

```
ATG___CGGTG__CAATG
___ATGG__TGCA____T
```

```
__________ATGCGGTGCAATG
ATGGTGCCAT_____________
```





Pfadgüte

- „Gute Pfade“ haben viele Matches
  - Matches im Dotplot sind die 1‘er Felder
- Definition

  Die Güte eines Pfades P durch einen Dotplot M ist die Anzahl an diagonal durchquerten 1‘er Feldern
  - Schritte nach rechts oder unten zählen nicht
- Bemerkung
  - Der beste Pfad kann also höchstens Güte min(m,n) haben

Beispiele

	A	T	G	C	G	G	T	G	C	A	A	T	G
A													
T													
G													
G													
T													
G													
C													
A													
T													

Pfadgüte: 0

	A	T	G	C	G	G	T	G	C	A	A	T	G
A													
T													
G													
G													
T													
G													
C													
A													
T													

Pfadgüte: 2

	A	T	G	C	G	G	T	G	C	A	A	T	G
A													
T													
G													
G													
T													
G													
C													
A													
T													

Pfadgüte: 8

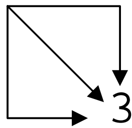
Maximale Güte?

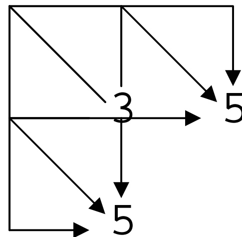
Alles das Gleiche

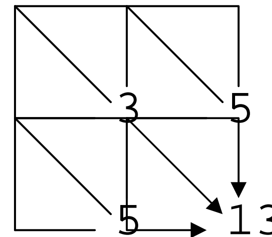
- Gegeben zwei Strings A,B und Dotplot M
- Die folgenden Probleme sind äquivalent
  - Finde das optimale Alignment von A und B (= Alignmentabstand)
  - Finde die minimale Menge an Editoroperationen von A nach B (= Editabstand)
  - Finde in M den Pfad mit maximaler Güte
- Beweis
  - Pfade lassen sich in Alignments übersetzen (und beinahe auch umgekehrt)
  - Alignments lassen sich in Editskripte übersetzen und umgekehrt

Algorithmus

- Naives Verfahren um den besten Pfad zu finden
  - Alle Pfade aufzählen
  - Das sind exponentiell viele







- Nur Pfade „um“ die Hauptdiagonale: $> 3^{\min(m,n)}$
- Inakzeptable Laufzeit
- Tatsächliche Komplexität des Problems: $O(m^2)$
- Trick: Dynamische Programmierung

Zusammenfassung

- Approximatives Matching ist das Thema der Bioinformatik schlechthin
  - Grundlage vieler Anwendungen
  - Prinzip des „Sequenzvergleich ersetzt Experiment“
- Wir werden sehen: Exaktes Matching ist oft Hilfsmittel zum approximativen Matching
- Visualisierung durch Dotplots
- Bewertung der Approximation setzt Abstandsmaß voraus
  - Editabstand
- Alignment gleichwertig zu Editabstand zu Dotplotpfaden
- Naive Algorithmen mit katastrophaler Komplexität
- Nächste Stunde: Dynamische Programmierung