Bio Data Management

Kapitel 5a

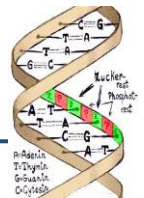
Sequenzierung und Alignments

Wintersemester 2014/15
Dr. Anika Groß
Universität Leipzig, Institut für Informatik, Abteilung Datenbanken
http://dbs.uni-leipzig.de

UNIVERSITÄT LEIPZIG

Vorläufiges Inhaltsverzeichnis

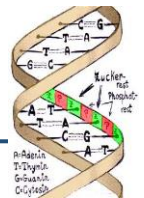
1. Motivation und Grundlagen
2. Bio-Datenbanken
3. Datenmodelle und Anfragesprachen
4. Modellierung von Bio-Datenbanken
5. a) Sequenzierung und Alignments
   b) NGS Data Management
6. Genexpressionsanalyse
7. Annotationen
8. Matching
9. Datenintegration: Ansätze und Systeme
10. Versionierung von Datenbeständen
11. Neue Ansätze



Lernziele und Gliederung

- Kenntnisse zu Sequenzierverfahren und Sequenzvergleich unter Berücksichtigung der Besonderheiten biologischer Daten/Sequenzen

- Sequenzierverfahren: Experimentelle Ansätze
- Sequenzvergleich (Alignments): Algorithmen und Methoden



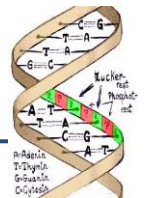
Sequenzierung

- Ziel: Aufdeckung einer unbekannten Sequenz
- DNA-Sequenzierung: Bestimmung der Abfolge der Basen in einem DNA-Molekül
  - Wegen Basenkomplementarität genügt es *einen* der beiden komplementären Stränge (Texte) zu bestimmen

  - Kettenabbruchmethode
  - "Shotgun" Sequenzierung
  - Next Generation Sequencing
  - ...

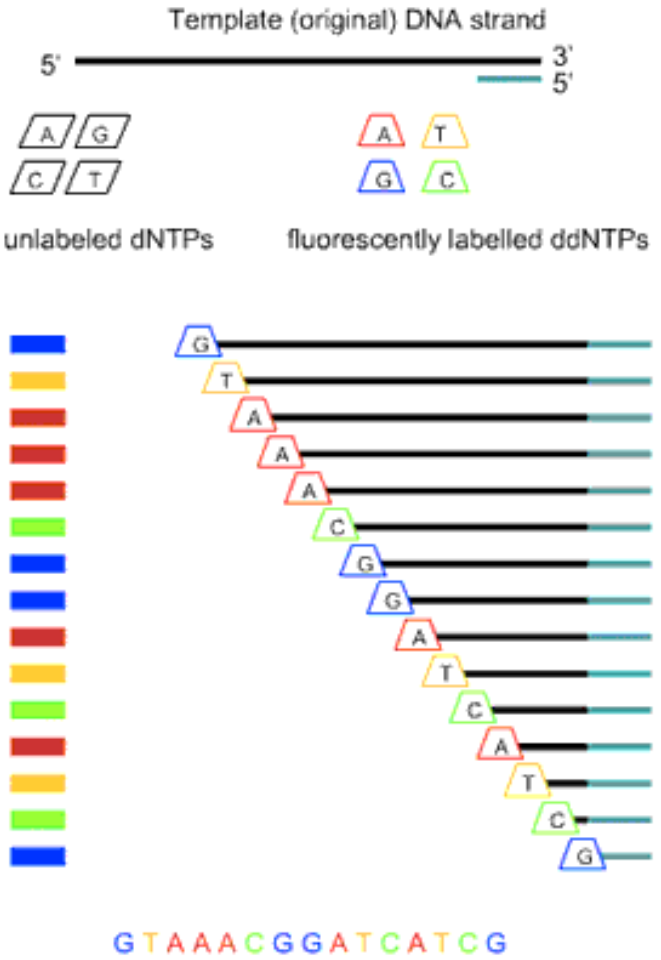
A row of DNA sequencing machines (*3730xl DNA Analyzer* machines from *Applied Biosystems*). Flickr user jurvetson: http://www.flickr.com/photos/jurvetson/57080968/

- Protein-Sequenzierung: Bestimmung der Abfolge der Aminosäuren in einem Protein (Primärstruktur) → kaum mehr verwendet

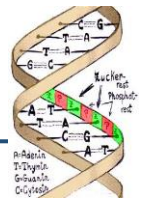


DNA-Sequenzierung

- Kettenabbruchmethode nach Sanger
  - Start: kurzes bekanntes DNA Stück (Primer)
  - Denaturierung der Doppelhelix → Einzelstränge
  - Untersch. Fluoreszenz-Markierung der 4 Basen
  - Basen hinzugeben → Strang wird neu synthetisiert
  - 4 Ansätze: je 1 Base teilweise mit Abbruchkriterium (ddNTP) markiert
  - Polymerase synthetisiert Fragmente unterschiedlicher Länge
  - Nach Größe sortieren → Sequenz (Laser, Detektoren …)
- Output einer Sequenziermaschine: trace file
- Übersetzen der „trace“ in Sequenz mittels einer Software

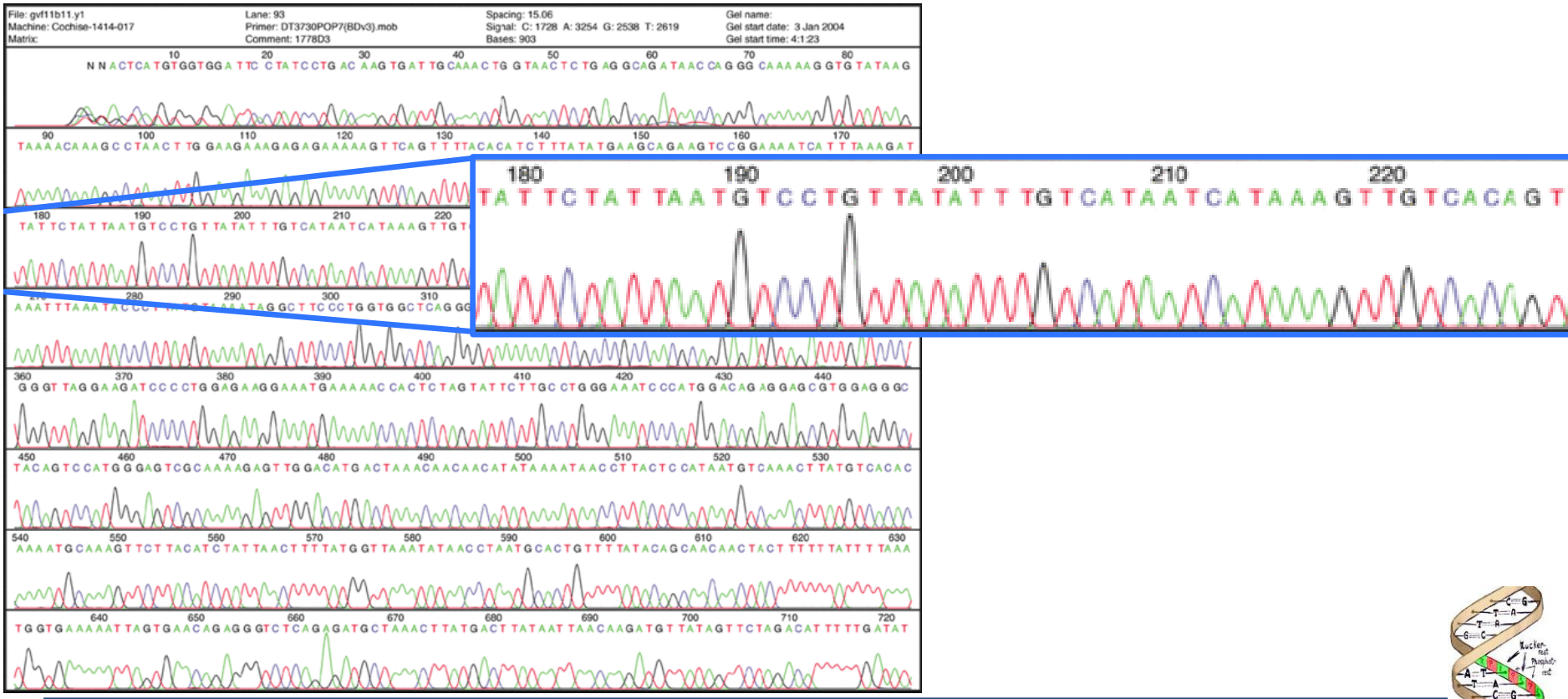


Bildquelle: http://www.genomebc.ca/files/3212/7439/2451/6.6.24%20Sequencing%20image.gif



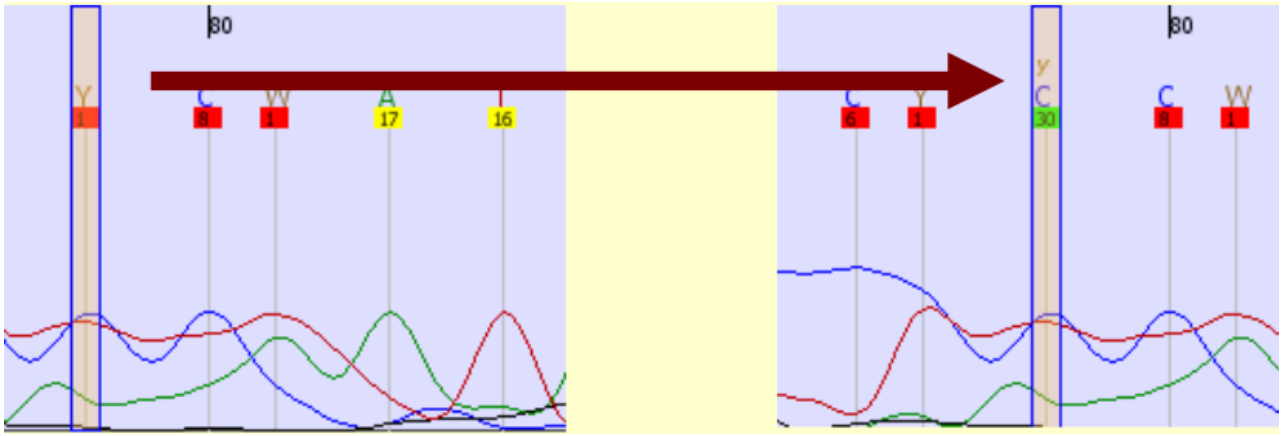
Elektropherogram

- Auch Chromatogramm
- Graphische Darstellung von Resultaten einer Elektrophorese-Analyse (wie z.B. Sequenzierung)
- Plot der Floureszenz-Einheiten über die Zeit



Elektropherogram

- Probleme: Rauschen (kein klares Signal), Verschiebung (versetzt wie Echo), Luftblase in Kapillare, …
- Visuelle Inspektion und manuelle Korrektur der Sequenz (Auflösung von Ambiguitäten)
- IUPAC Nucleotide Codes



Tool: Ridom TraceEdit

IUPAC nucl.code	Base
A	Adenine
C	Cytosine
G	Guanine
T (or U)	Thymine (or Uracil)
R	A or G
Y	C or T
S	G or C
W	A or T
K	G or T
M	A or C
B	C or G or T
D	A or G or T
H	A or C or T
V	A or C or G
N	any base
. or -	gap

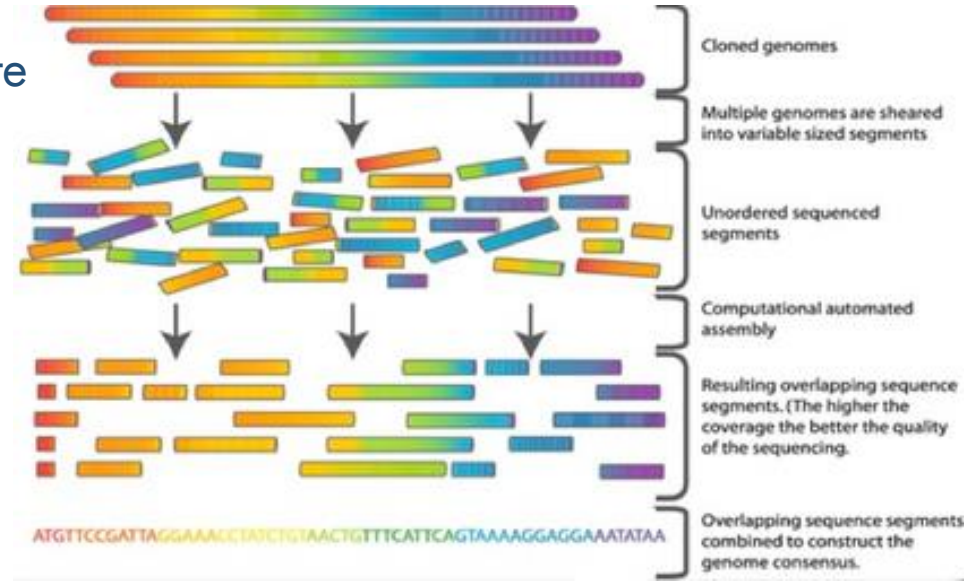
Shotgun-Sequenzierung

Auch High-Throughput Sanger Sequenzierung

- Problem: Sequenzierverfahren meist limitiert bzgl. Länge
- Lösungsansatz: Fragmentierung und Sequenzierung der Fragmente

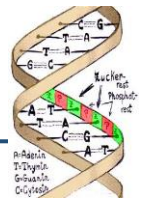
- Klonierung (=Vervielfältigung der DNA) bessere Signalstärke
- Fragmentierung (Länge 500-900 bp)
- Sequenzierung

→ Ungeordnete, sequenzierte Fragmente



Bildquelle:http://biobook.nerinxhs.org/bb/genetics/biotechnology/Whole_genome_shotgun_sequencing_versus_Hierarchical_shotgun_sequencing.png

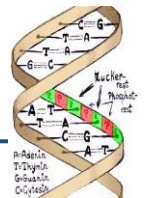
- Überlappungen der sequenzierten Fragmente (reads) bestimmen:
  - alle Fragmente miteinander vergleichen um wahrscheinlichste Reihenfolge zu bestimmen (Assembly)
  - heuristisches Verfahren zur Bestimmung eines multiplen Sequenzalignments
- Probleme bei Wiederholungen (*repeats*) in der Sequenz
  - Mehr Überlappung & mehr Redundanz → Höhere Qualität ↑



Immer noch ...

- … limitiertes Level an Parallelisierung:
  parallele Verarbeitung von 40-100 reads
- Hohe Genauigkeit von 99.999% pro Base
- Im Bereich der high-throughput shotgun Sequenzierung kostet die Sanger Sequenzierung ca. $0.50 pro Kilobase

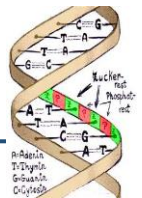
→ **Next Generation Sequencing (Kapitel 5b)**



Problematik und Nutzen der Sequenzierung

+ Verbesserung der Krankheitsdiagnostik
+ Frühere Erkennung von Prädispositionen für Krankheiten
+ Medikamenten-Design
+ Gentherapie
+ Organersatz (Eignung des Spenders, in vitro Herstellung)
– Ethische und rechtliche Problematik
  – Gentests zur Krankheitsdiagnose, z.B.: Präimplantationsdiagnostik
  – Soll/darf ein Gentest durchgeführt werden, wenn noch keine Therapie verfügbar ist? Wer hat Zugang zu den Testergebnissen? Wie verlässlich sind die Gentests?
  – Kommerzialisierung: Darf ein Gen patentiert werden?
– Kosten der Sequenzierung noch immer sehr hoch

… … …



Ähnliche Sequenzen...

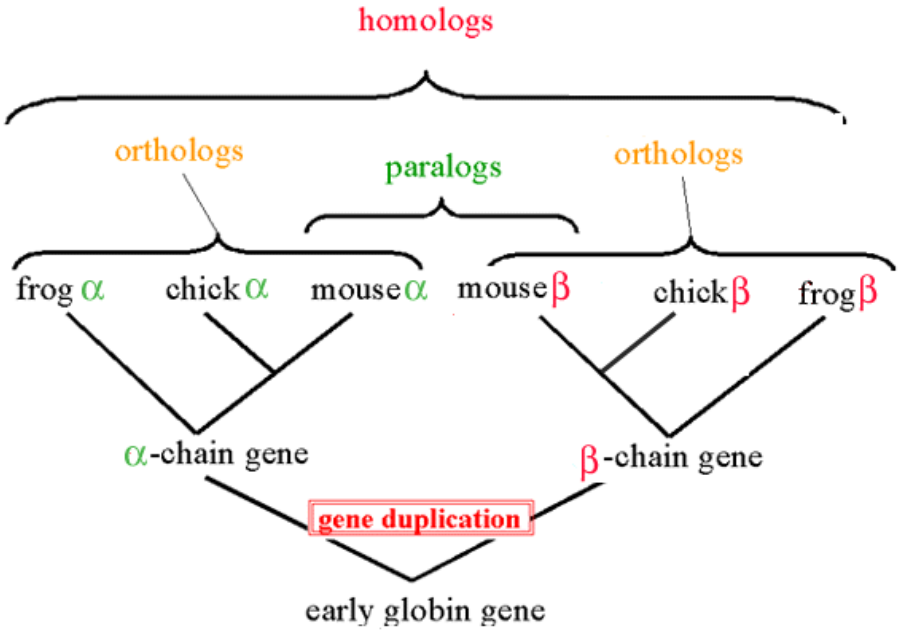
Grundannahme der Bioinformatik:

Ähnliche Sequenz → ähnliche Struktur → ähnliche Funktion

- Funktionale Annotationen (zu neuer DNA Funktion finden, Suche nach ähnlichen bekannten Proteindomänen / Gensequenzen)
- EST Clustering z.B. zur Expressionsanalyse
- Kartierungsproblematik
  - Auf welchem Chromosom befindet sich welches Gen, (welche Sequenz) an welcher Stelle?
- Sequenz-Assemblierungs-Problem (Assembly):
  - Gegeben die Überlappungsinformationen und Alignments von Fragmenten einer "unbekannten" Sequenz. Man bestimme die Reihenfolge der Buchstaben (Basen) der "unbekannten" Sequenz (Konsensus-Sequenz)
- Datenbanksuche nach ähnlichen Sequenzen (z.B. für Verwandschaftsbeziehungen, Homologie)
  - Gegeben ein Pattern p und eine Menge von Sequenzen $S = \{s_1, s_2, \ldots, s_n\}$: Suche alle Sequenzen s_i, die p ähneln
  - Gegeben ein Pattern p und eine lange Sequenz s^L: Suche alle Teilsequenzen von s^L, die dem Pattern p oder Teilsequenzen des Pattern ähneln

Sequenzhomologie

- Homologe: gemeinsamer Vorfahr, mind. 30% (10%) Sequenzähnlichkeit bei Genen (Proteinen)
- Orthologe: homologe Gene in verschiedenen Organismen, gemeinsamer Vorfahr, Artbildungsereignis
- Paraloge: Genverdopplung, 2 Kopien des Gens innerhalb eines Organismus

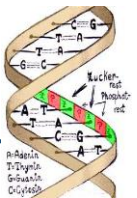


Bildquelle: http://www.stanford.edu/group/pandegroup/folding/education/orthologs3.gif

Sequenz-Alignments

Als Quelle diente teilweise die Vorlesung „Algorithmische Bioinformatik“ von Prof. Leser, Humboldt-Universität zu Berlin

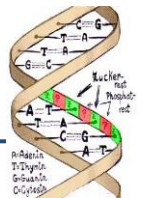
- Ziel: Vergleich zwischen bekannten Sequenzen
  - DNA: $s_1, s_2 \in S \subset (A|C|G|T)^*$
  - Proteine: $s_1, s_2 \in S \subset (A|C|D|E|F|G|H|I|K|L|M|N|P|Q|R|S|T|V|W|Y)^*$
- Ergebnis → Alignment:
  Übereinanderstellen zweier oder mehrerer Sequenzen zur Identifikation ähnlicher Bereiche
  (Konservierte Regionen, Homologien, funktionelle oder evolutionäre Verwandtschaft)
- Bioinformatik braucht exaktes und approximatives Matching
- Beispiele DB-Suche:
  - Welche Abschnitte in DB-Sequenzen enthalten eine gesuchte Exon-Sequenz?
  - Gibt es Gene in anderen Spezies, die (fast) mit dem gesuchten Gen übereinstimmen?
  - Gibt es „überlappende“ Abschnitte zwischen der Suchsequenz und den Sequenzen in der DB?



Sequenz-Alignments

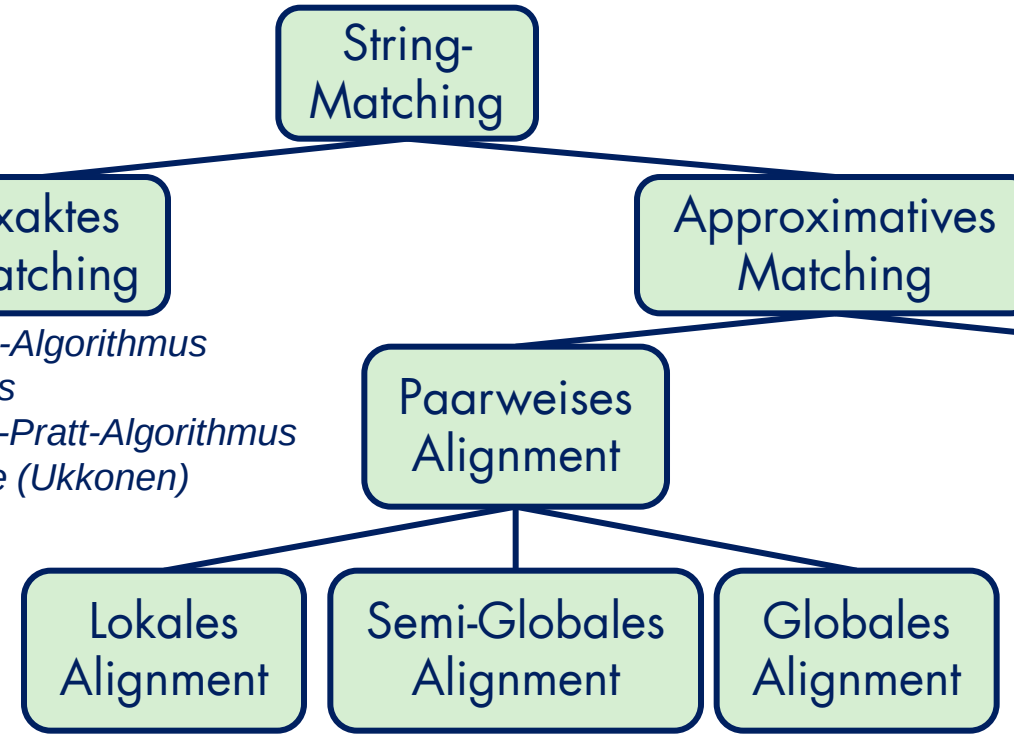
- Komplette Übereinstimmung
- Änderungsoperationen
  - Einschübe
  - Löschungen
  - Substitutionen

- Ausgewählte Methoden von Alignments
  - Visuelle Alignments
  - Lokale vs. globale Alignments
  - Paarweise vs. multiple Alignments
  - Exakte/Optimale vs. Heuristische Verfahren



String-Matching in der Bioinformatik

- Mögliche Klassifikation vorhandener Verfahren (als Überblick!)
- Teilweise optimierte Algorithmen bzgl. Laufzeit, Speicher



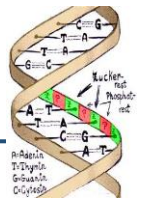
- *Boyer-Moore-Algorithmus*
- *Z-Algorithmus*
- *Knuth-Morris-Pratt-Algorithmus*
- *Suffix-Bäume (Ukkonen)*
- *…*

- *Heuristische Verfahren:*
  ClustalW
  CenterStar
  PSI-BLAST
  …

Genug Stoff für 1-2 Semester! Hier nur Überblick zu einem Teil der Verfahren …

- *Optimales Alignment: Smith-Waterman*
- *Heuristische Algorithmen: FASTA,BLAST*
- *…*

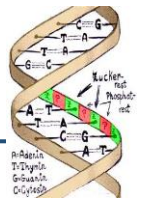
- *Optimales Alignment: Needleman-Wunsch Gotoh-Algorithmus*
- *Heuristische Algorithmen: FASTA,BLAST*
- *…*



Exaktes Matching – kurz: Boyer-Moore

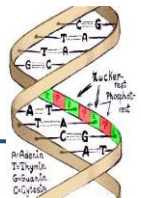
- Suche nach exakten Vorkommen von Pattern in Template
- Schiebe P l $\rightarrow$ r, vergleiche P gegen T l $\leftarrow$ r
- (Extended) Bad-Character-Rule: „mehr schieben als nur ein Zeichen“
  - Schieben des Musters bis zu dem Zeichen x, das Mismatch verursacht
    - x kommt nicht vor: springe bis an die Position nach dem x in T
    - x kommt vor: verschiebe zum rechtesten x in P, das links von der Mismatchposition liegt (dazu Positionen in Lookup-Table ablegen)
- Außerdem Good-Suffix-Rule
  - „P nicht immer wieder von Anfang an matchen“
  - Eventuell rechts schon langen Match m gefunden..kommt m nochmal n P vor?
  - Weitere Fallunterscheidungen!!

- Laufzeit
  - Gut bei großen Alphabeten!
  - Average sublinear
  - Apostolico-Giancarlo-Variante (1986): linear!



Suffixbäume

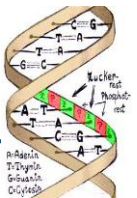
- Datenstruktur für effiziente Lösungen zahlreicher String-Verarbeitungsprobleme (Indexierung)
- Kurz: Speichern aller Suffixe (Endungen) einer Zeichenkette
- Schnelles Ausführen von z.B. Suche nach Wörtern in langen Texten
  - Besonders hilfreich wenn zu durchsuchender Text bekannt ist (z.B. Referenzgenom)
- Für exakte Suche sind Suffixbäume die schnellste Datenstruktur
  - Gut geeignet für DB-Suche
  - Konstruktion in linearer Zeit möglich, aber: speicheraufwendig
  - Vorverarbeitung von zu durchsuchenden Text → einmalige Kosten
  - Kosten der Konstruktion werden nicht in Kosten der Suchen eingerechnet
- Anwendungen in der Bioinformatik:
  - Sequenz- Suche in bekannten Sequenzdatenbanken
  - Vorstufe der approximativen Suche: Suche nach „Seeds“
  - Suche längste gemeinsame Subsequenzen (Vergleich zweier Genome)
  - Suche längste Repeats (Finden von typischen, sich im Genom wiederholenden Sequenzen)



Suffixbäume

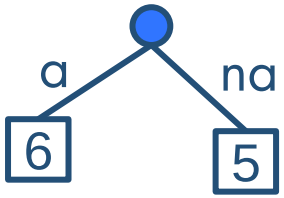
- Wir bauen einen Suffixbaum T für String S mit $|S|=m$

- Definition: Der Suffixbaum T für S ist ein Baum mit
  - T hat eine Wurzel und m Blätter, markiert mit $1,\ldots,m$
  - Jede Kante E ist mit einem Substring $label(E)\neq\emptyset$ von S beschriftet
  - Jeder innere Knoten k hat mindestens 2 Kinder
  - Alle Label der Kanten von einem Knoten k aus beginnen mit unterschiedlichen Zeichen
  - Sei $(k_1,k_2,\ldots,k_n)$ ein Pfad von der Wurzel zu einem Blatt mit Markierung i. Dann ist die Konkatenation der Label der Kanten auf dem Pfad gleich $S[i..m]$
- Intuition: Kompakte Repräsentation aller Suffixe von S in einem Baum



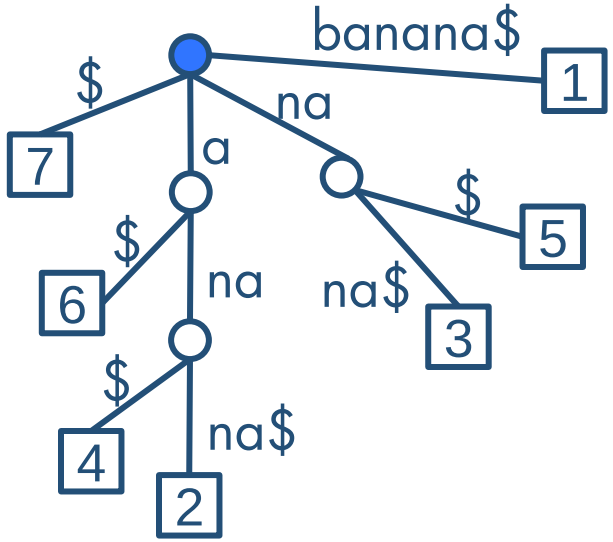
Beispiel

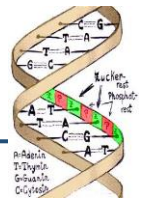
	1	2	3	4	5	6
S =	B	A	N	A	N	A



- Verlängerung von „a“ nicht erlaubt, da 6 sonst kein Blatt
- Problem: Suffix „a“ ist Präfix von „ana“
- Brauchen „Stoppzeichen“ $ ∉ Alphabet(S)

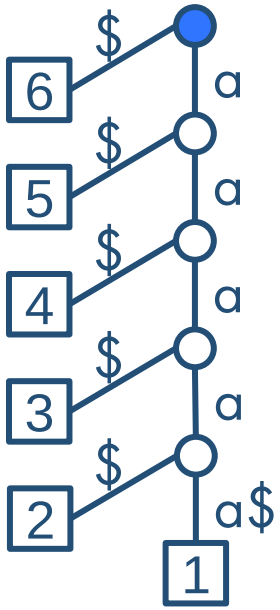
	1	2	3	4	5	6	7
S =	B	A	N	A	N	A	$



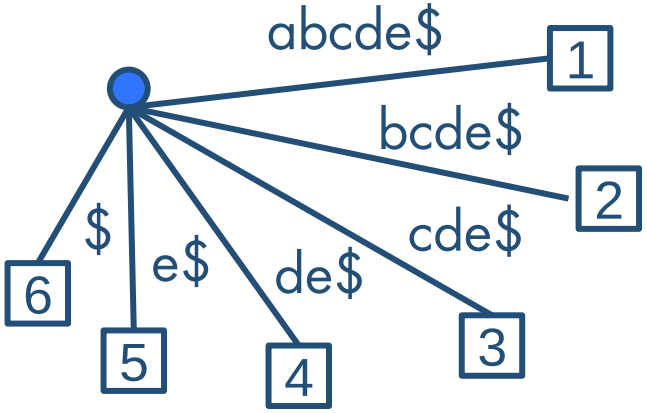


„Extreme“

	1	2	3	4	5	6
S =	A	A	A	A	A	$

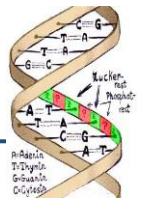


	1	2	3	4	5	6
S =	A	B	C	D	E	$



Eigenschaften von Suffixbäumen

- Zu jedem String (plus $) gibt es genau einen Suffixbaum.
- Jeder Pfad von der Wurzel zu einem Blatt ist unterschiedlich.
  (→ unterschiedlich lang)
- Jede Verzweigung an einem inneren Knoten ist eindeutig bzgl. des nächsten Zeichens auf dem Pfad.
- Gleiche Substrings können an mehreren Kanten stehen.



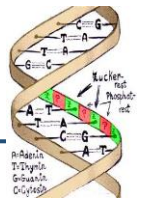
Suche mit Suffixbäumen

- Finde alle Vorkommen eines Pattern P in String S
- Intuition: Jedes Vorkommen von P muss Präfix eines Suffix von S sein
  - Sind alle als Pfad von der Wurzel aus vorhanden

1) Konstruiere den Suffixbaum T zu S → *in O(|S|) möglich*
2) Matche P auf einen Pfad in T ab der Wurzel
   - Wenn das nicht geht, kommt P in S nicht vor
   - P endet in einem Knoten k → merke k
   - P endet in einem Kantenlabel → merke Endknoten k dieser Kante
3) Die Markierungen aller unterhalb von k gelegenen Blätter sind Startpunkte von Vorkommen von P in S

Weitere Anwendungen

- Längster gemeinsamer Substring zweier Strings → Suffixbaum für $S_1\$S_2\%$
- Längstes Palindrom
  → Längster gemeinsamer Substring von S und reverse(S)



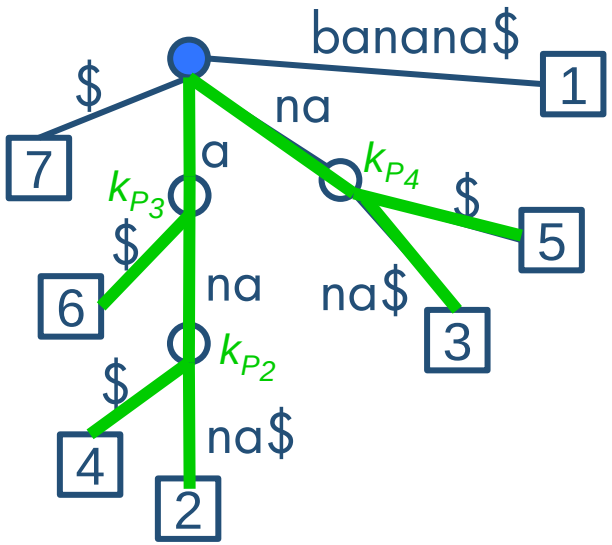
Beispiel

P_1 = G A ✗

P_2 = A N 2 4

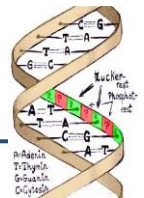
P_3 = A 2 4 6

P_4 = N A 3 5



Komplexität der Suche

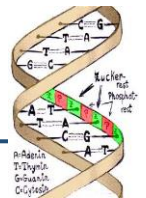
- Theorem: *Sei T der Suffixbaum für S+"$". Die Suche nach allen Vorkommen eines Pattern P, |P|=n, in S ist O(n+k), wenn k die Anzahl Vorkommen von P in S ist.*
- Beweisidee
  - P in T matchen kostet O(n)
    - Pfade sind eindeutig → Entscheidung an jedem Knoten ist klar
    - Damit maximal O(n) Zeichenvergleiche
  - Blätter aufsammeln ist O(k)
    - Baum unterhalb Knoten K hat k Blätter
    - Die kann man in O(k) finden



Komplexität der Konstruktion

- Naiver Algorithmus zur Konstruktion von Suffixbäumen: $O(m^2)$
- Suffixbäume nur sinnvoll, wenn Konstruktion in O(m) gelingt (sonst nimmt man lieber z.B. Boyer-Moore)
- Lösung: Ukkonen's Algorithmus → verwendet Suffix-Links in impliziten Suffixbäumen + einige Regeln&Tricks
  - Konstruktion O(m)
  - Speicher:
    - Sehr schlecht auf Sekundärspeichern, braucht den ganzen Baum im Hauptspeicher (durch Suffix-Links)

    → Eingrenzung der Verwendbarkeit (keine Genom-Genom-Vergleiche)
    - Außerdem sehr speicherintensiv (durch viele Pointer, Kantenrepräsentation)
- Andere Optimierungen: z.B. Teile des wachsenden Baumes zeitweise auf Platte auslagern
- Alternative: Suffix-Arrays

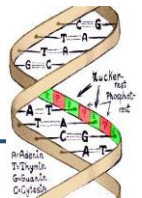


Algorithmen für den Sequenzvergleich

- Suchalgorithmen nicht ausreichend
  - Fordern Gleichheit statt Ähnlichkeit

- Ziel: hohe Ähnlichkeit, niedrige Distanz von Strings
  - Verschiedene Metriken zur Berechnung
  - Grundlage: metrischer Raum $X \times X \rightarrow R$ für beliebige Menge X und reelle Zahlen R
  - Bsp. Hamming-Distanz
    - Einfaches Distanzmaß zur Bestimmung der Unterschiedlichkeit von Zeichenketten
    - Haus ↔ Baum: Hamming-Abstand=2
  - Editier-/Levenshtein-Distanz

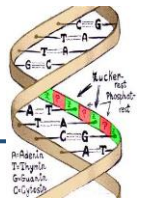
- Bioinformatik → viele approximative Matchalgorithmen, sowie heuristische Verfahren



Visuelle Sequenz-Alignments: Dotplot

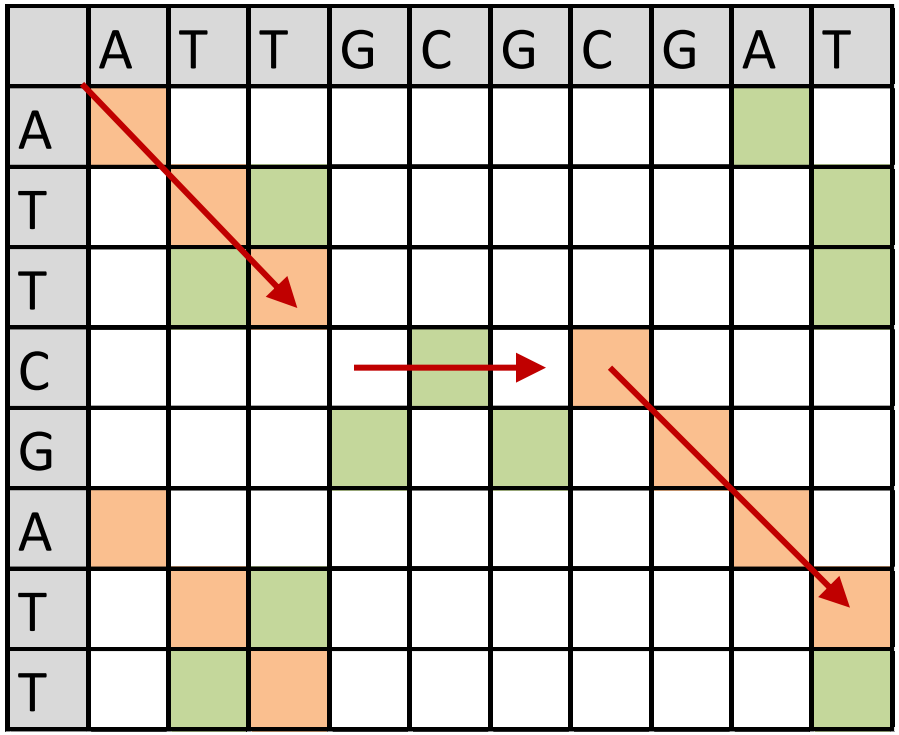
- Graphische Methode zum Vergleich zweier Sequenzen $s_1, s_2 \in (A, C, G, T)^*$
  - Matrix mit s_1 und s_2 als Titelzeile und –spalte
  - Matrix hat Dimension $M_{m,n}$ mit $m=|s_1|$, und $n=|s_2|$,
  - Markierung im Feld $m_{i,j}$ wenn $s_{1,i} = s_{2,j}$

- Ziele
  - Auffinden von ähnlichen bzw. übereinstimmenden Regionen zwischen s_1 und s_2
  - Richtung der Übereinstimmung
- Sonderfall: $s_1=s_2$ – Auffinden repetitiver Regionen innerhalb der gegebenen Sequenz
- Vorteile/Nachteile



Beispiel Dotplot

- s_1: ATTGCGCGAT
- s_2: ATTCGATT
- Längste Diagonale (von links-oben nach rechts-unten)
  → größter gemeinsamer Teilstring



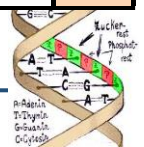
Beispiel Dotplot

Palindrome

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

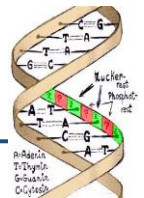
Repetitive Sequenz (ATC)

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



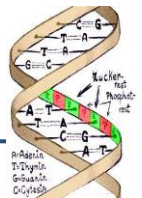
Editier-Distanz

- Bestimmung eines *Alignments* zwischen s_1 und s_2:
  - Übereinanderstellen von s_1 und s_2 und durch Einfügen von Gap-Zeichen Sequenzen auf dieselbe Länge bringen: Jedes Zeichenpaar repräsentiert zugehörige Editier-Operation
  - Kosten des Alignments:
    Summe der Kosten der Editier-Operationen
  - *optimales Alignment*: Alignment mit *minimalen Kosten*
    (= Editierdistanz)
  - Komplexität: O (n*m) mit n, m Länge der beiden Sequenzen



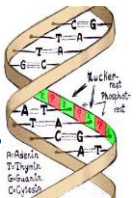
Editier-Distanz: Operationen

- Drei Arten von Editier-Operationen:
  - *Löschen* (Delete) eines Zeichens,
  - *Einfügen* (Insert) eines Zeichens
  - *Ersetzen* (Replace/Substitute) eines Zeichens x durch y (x≠y)
- Editier-Operationen korrespondieren zu je einer Mismatch-Situation zwischen s_1 und s_2, wobei „_“ für leeres Wort / Zeichen (gap) steht:
  - (-,y) Einfügung von y in s_2 gegenüber s_1
  - (x,-) Löschung von x in s_1
  - (x,y) Ersetzung von x durch y
  - (x,x) Match-Situation (keine Änderung)
- Lücke in beiden Strings ist nicht erlaubt!
- Zuweisung von Kosten *w(x,y)* je Operation;
  x, y = Zeichen an spezieller Positionen in s_1 (s_2)
- *Einheitskostenmodell*: $w(x,y) = w(-,y) = w(x,-) = 1 \quad ; w(x,x) = 0$
- *Editier-Distanz* $D(s_1,s_2)$*:* Minimale Kosten einer Folge von Editier-Operationen, um s_1 nach s_2 zu überführen
  - bei Einheitskostenmodell spricht man auch von *Levenshtein-Distanz*
  - *im Einheitskostenmodell gilt* $D(s_1,s_2)=D(s_2,s_1)$
    und für Kardinalitäten n und m von s_1 und s_2:
    $abs\ (n - m) \leq D\ (s_1,s_2) \leq max\ (m,n)$



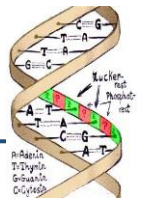
Editier-Distanz: Anwendungszenarien

- Problem 1: Berechnung der Editier-Distanz
  - Berechne für zwei Sequenzen s_1 und s_2 möglichst effizient die Editier-Distanz $D(s_1,s_2)$ und eine kostenminimale Folge von Editier-Operationen, die s_1 in s_2 überführt
  - entspricht Bestimmung eines optimalen Alignments
- Problem 2: Approximative Suche
  - Suche zu einem (kurzen) Muster p alle Vorkommen von Strings *p'* in einem Text, so dass die Edit-Distanz $D(p,p') \leq k$ ist (für ein vorgegebenes *k*) z.B. Motif-Suche
  - Spezialfall 1: exakte Stringsuche (*k=0*)
  - Spezialfall 2: *k*-Mismatch-Problem, falls nur Ersetzungen und keine Einfüge- oder Lösch-Operationen zugelassen werden
  - Variationen
    - Suche zu Muster/Sequenz das ähnlichste Vorkommen (lokales Alignment) z.B. Exon-Suche
    - bestimme zwischen 2 Sequenzen s_1 und s_2 die ähnlichsten Teilsequenzen s_1' und s_2'



Berechnung der Editier-Distanz

- Berechnung der Editier-Matrix
  Sei $s_1 = (a_1, \ldots a_n)$, $s_2=(b_1, \ldots b_m)$.
  D_{ij} sei Editierdistanz für Präfixe $(a_1, \ldots a_i)$ und $(b_1, \ldots b_j)$;
  $0 \leq i \leq n$; $0 \leq j \leq m$
  - D_{ij} kann ausschließlich aus $D_{i-1,\,j}$, $D_{i,j-1}$ und $D_{i-1,j-1}$ bestimmt werden
  - Triviale Teillösungen für $D_{0,0}$, $D_{0,j}$,$D_{i,0}$
  - Bestimmung der $D_{i,j}$ aus „Vorgängerzellen“
  - Editierdistanz zwischen s_1 und s_2 ergibt sich für i=n, j=m
- Backtracking
  - Durchlaufen der Matrix rückwärts
  - Diagonal: Match / Mismatch
  - Vertikal: Einfügung in s_2
  - Horizontal: Löschung in s_1



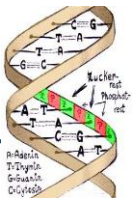
Matrix-Rekurrenzen

$$D(0,0) = 0,\ D(i,0) = i,\ D(0,j) = j$$

$$m=|s_1|,\ m=|s_2|,\ 1 \leq i \leq m,\ 1 \leq j \leq n$$

$$D(i,j) = min \begin{cases} D(i,j-1)+1 \\ D(i-1,j)+1 \\ D(i-1,j-1)+t(i,j) \end{cases}$$

$$t(i,j) = \begin{cases} 1\text{: wenn } s_{1,i} \neq s_{2,j} \\ 0\text{: sonst} \end{cases}$$

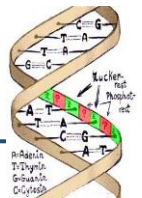


Berechnung der Editier-Distanz

- Rekursiver Ansatz
  - Lösen eines Problems durch Lösen mehrerer kleinerer Teilprobleme, aus denen sich die Lösung für das Ausgangsproblem zusammensetzt

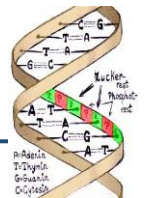
```
function D(i,j) {
        if (i = 0)              return j;
        else if (j = 0)         return i;
        else
                return min (    D(i-1,j) + 1,
                                D(i,j-1) + 1,
                                D(i-1,j-1) + t(s1[i],s2[j]));
}
function t(c1, c2) {
        if (c1 = c2) return 0;
        else return 1;
}
```

→ Mehrfachberechnungen der Teillösungen!



Berechnung der Editier-Distanz

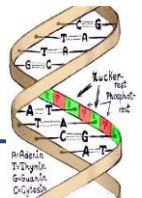
- Nutzung folgender Eigenschaften zur Begrenzung zu prüfender Editier-Operationen
  - optimale Folge von Editier-Operationen ändert jedes Zeichen höchstens einmal
  - jede Zerlegung einer optimalen Anordnung führt zur optimalen Anordnung der entsprechenden Teilsequenzen



Dynamische Programmierung

- Lösung des Optimierungsproblems durch Ansatz der dynamischen Programmierung (DP)
  - Konstruktion der optimalen Gesamtlösung durch Kombination von Lösungen für Teilprobleme
  - Speichern einmal berechnete Lösungen in einer Tabelle für spätere Zugriffe (Wiederverwendung)
  - Von einfachen zu komplexen Fällen, wobei jedes Unterproblem gelöst wird, bevor es durch ein anderes benötigt wird
  - Möglich: merken der Zeiger, aus welcher die jeweilige Zelle berechnet wurde (spart ein paar Berechnungen beim Traceback)

- Berechnung Editier-Distanz in zwei Phasen
  - Berechnung der Editier-Matrix mit DP
  - Finden des optimalen Alignments (Backtracking)
  - Merken der Zeiger hilfreich (Gesamtkomplexität unverändert)

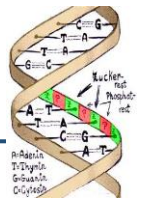


Needleman-Wunsch-Algorithmus

- Suchen optimales, globales Alignment
- Bisher: Abstand minimiert
- Jetzt: Score maximieren
- Ähnlichkeit des Alignments von s_1 und s_2 bzgl. Score-Funktion

$$sim(s_1, s_2) = \sum_{i=1}^{n} score(s_1[i], s_2[i])$$

- DP-Algorithmus zur Berechnung der Levenshtein-Distanz, jedoch mit beliebiger *score*-Funktion für Einfüge- und Löschoperationen (auch mehr als ein Zeichen → Gap-Länge unbekannt) $O(n^3)$
- Bei beschränkter Kostenfunktion (z.B. einheitliche Gapkosten) $O(n^2)$
- Mehrere optimale, globale Alignments möglich
- Hirschberg-Algorithmus
  - Reduziert Speicherbedarf von quadratisch auf linear durch Divide-and-Conquer-Methode



Matrix-Rekurrenzen

$$d(i,0) = \sum_{k=1}^{i} score(s_1[k], \_) \qquad d(0,j) = \sum_{k=1}^{j} score(\_, s_2[k])$$

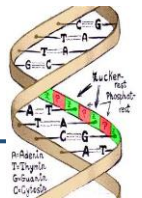
$$d(i,j) = max \begin{cases} d(i,j-1) + score(\_, s_2[j]) \\ d(i-1,j) + score(s_1[i], \_) \\ d(i-1,j-1) + score(s_1[i], s_2[j]) \end{cases}$$

Alternative zu Einheitskostenmodell (eine von vielen):

- $score(s_1[i],s_2[j]) = \mathbf{1}$, falls $s_1[i] = s_2[j]$ // pos. score für match
- $score(s_1[i],s_2[j]) = \mathbf{-1}$, falls $s_1[i] \neq s_2[j]$ //neg. score für mismatch/replace, oder score aus Substitutionsmatrix
- $score(s_1[i],\_) = score(\_,s_2[j]) = \mathbf{-2}$ //höhere Bestrafung (neg. score) für Löschung/Einfügung (Rasterverschiebung)

Smith-Waterman

- Wie Needleman-Wunsch-Algorithmus, aber zur Berechnung des optimalen lokalen Alignments
- Scoring: Positive Werte für Matches, negative für Mismatches
- Unterschiede
  - 1. Zeile und 1.Spalte mit 0 initialisieren
  - Zusätzlicher Fall: Maximierung über 0
    (kein Wert kann kleiner als 0 werden)
  - Lokales Alignment steht irgendwo in der Matrix:
    Bei der höchsten Zahl in der Matrix mit Backtracking starten, bei 0 stoppen → optimales lokales Alignment
  - Mehrere optimale lokale Alignments
- $O(n^2)$ (da keine beliebige score-Funktion)



Matrix-Rekurrenzen

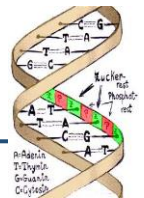
$$v(i,j) = \max_{\forall s_1' = s_1'[x..i],\, s_2' = s_2[y..j]} (sim(s_1', s_2'))$$

// Suche Suffixe s_1' von s_1 und s_2' von s_2, so dass die Ähnlichkeit zwischen s_1 und s_2 maximal ist

$$v(i,0) = 0$$
$$v(0,j) = 0$$

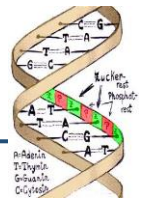
// Initialisierung der Ränder mit 0

$$v(i,j) = max \begin{cases} 0 \\ v(i,j-1) + score(\_, s_2[j]) \\ v(i-1,j) + score(s_1[i], \_) \\ v(i-1,j-1) + score(s_1[i], s_2[j]) \end{cases}$$



Optimierung: k-Band

- Gute Alignments müssen eng an der Hauptdiagonale bleiben
- Berechnung des optimalen globalen Alignments innerhalb eines Bandes der Breite 2*k
- Erhöhen k iterativ bis opt. Alignment gefunden wurde (es lässt sich beweisen, dass man das optimale A. bereits gefunden hat)
- Alignment in (meistens) weniger als O(n*m)



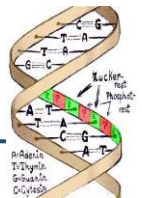
Substitutionsmatrizen

- Ersetzung einer Base/AS durch eine andere hat unterschiedliche biologische Bedeutung, keine Gleichberechtigung!
- Häufig werden anstatt einheitlichen Kosten für Substitutionen, Werte aus einer Matrix entnommen
- Werte in einer Matrixzelle beinhalten den Score für den Austausch eines Nukleotids durch ein bestimmtes anderes Nukleotid an der selben Position
- Aufstellen einer Scoring-Matrix für Substitutionen anhand der relativen Häufigkeit von Veränderungen durch Austausch
  - Transitionsmutationen (Purin↔Purin, Pyrimidin ↔ Pyrimidin) häufiger als Transversionen (Purin↔Pyrimidin)

A↔G
T↔C
(A,G)↔(T,C)

  - Beispiel

	A	T	G	C
A	20	5	10	5
T	5	20	5	10
G	10	5	20	5
C	5	10	5	20

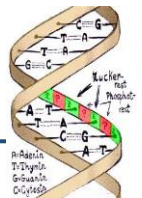


Substitutionsmatrizen – PAM, BLOSUM

- Grundlage: statistisch erfasste Werte über Sequenzunterschiede
- Nicht alle Aminosäuren sind gleich wichtig für die Struktur
  - Substitutionen, die wahrscheinlicher sind: höherer score
  - Substitutionen, die weniger wahrscheinlich sind: geringerer score
- Beobachtung/Schätzung tatsächlich entstandener evolutionärer Unterschiede
- 1PAM = 1 percent accepted mutations, d.h 2 Sequenzen, die einen Abstand von 1PAM haben sind zu 99% identisch
- Sammeln statistischer Werte über eng verwandte Sequenzen aus Alignments

PAM	0	…	110	…	250
Übereinstimmung	100%	…	60%	…	20%

- $Score\ der\ Mutation\ i \leftrightarrow j = \frac{beobachtete\ Mutationsrate\ i \leftrightarrow j}{aufgrund\ der\ AS-Frequenz\ erwartete\ Mutationsrate}$
- Messen der
  - Absoluten Häufigkeit der AS in allen Sequenzen
  - Übergangshäufigkeiten für alle möglichen AS-Paare

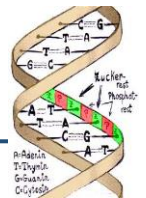


BLOSUM

- BLOSUM=BLOCKS substitution matrix
- BLOCKS = DB für Proteinsequenzen
- BLOSUM verwendet einzelne Blöcke (ohne Lücken) innerhalb der Sequenzen von homologen Proteinen (kurze multiple Alignments)

C	F	B	Q	D	Q	F	V	Y	Q	P
C	F	M	C	E	Y	Y	_	_	V	P
Y	F	B	_	U	U	V	F	D	V	P

- Basiert auf mehr Sequenzen als PAM und verwendet gezielt „entfernte Sequenzen“
- Identische Berechnung wie PAM aber andere Bedeutung der Zahl x
  - Zur Berechnung der Blosum-x Matrix werden in jedem Block alle Sequenzen mit >x% Identität zu einer Sequenz zusammengefasst
  - BLOSUM80 für evolutionär nah verwandte Proteine
  - BLOSUM45 für stark divergierende Proteine geeignet

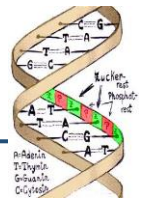


BLOSUM62

	Ala	Arg	Asn	Asp	Cys	Gln	Glu	Gly	His	Ile	Leu	Lys	Met	Phe	Pro	Ser	Thr	Trp	Tyr	Val
Ala	4																			
Arg	-1	5																		
Asn	-2	0	6																	
Asp	-2	-2	1	6																
Cys	0	-3	-3	-3	9															
Gln	-1	1	0	0	-3	5														
Glu	-1	0	0	2	-4	2	5													
Gly	0	-2	0	-1	-3	-2	-2	6												
His	-2	0	1	-1	-3	0	0	-2	8											
Ile	-1	-3	-3	-3	-1	-3	-3	-4	-3	4										
Leu	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4									
Lys	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5								
Met	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5							
Phe	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6						
Pro	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7					
Ser	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4				
Thr	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5			
Trp	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11		
Tyr	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	
Val	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4

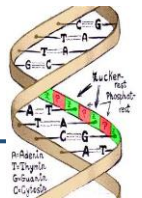
Beispiel: Substitution von Tryptophan mit Tyrosin ist 2 (nicht 0), d.h. Tryptophan zu Tyrosin (und umgekehrt) mutiert häufiger als nur durch Zufall zu erwarten wäre

häufige Ersetzung
→"nicht schlimm" +
sehr seltene Ersetzung
→"Bestrafung" -



Affine Gap-Kosten

- Konstante vs. Affine Gap-Kosten
- Biologische Sequenzen haben eher eine größere Lücke (z.B. Länge 10) anstatt 10 einzeln auftretende Lücken
- z.B. beim Alignieren von cDNA gegen Genom-DNA (Introns sind Lücken)
- Alignment mit affinen Gap-Kosten → Gotoh-Algorithmus
- Affine Gap-Kosten
  - Kosten für den Start einer Lücke (**g**ap **o**pen **p**enalty): *gop* z.B. -12
  - Kosten für die Verlängerung einer Lücke um eine Stelle (**g**ap **e**xtension **p**enalty): *gep* z.B. -1
  - affine Gap-Kosten Funktion (Länge l)
    $g(l) = gop + l \cdot gep$
- Wenige, große Lücken sind besser als viele kleine Lücken
- "Ermutigen" zur Gap extension statt Gap Opening

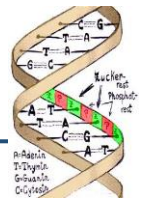


Heuristische Verfahren

- Häufige Aufgabe: Vergleich eines zu suchenden Musters gegen Datenbanksequenz(en)
- Verfahren zur Bestimmung optimaler Alignments zu komplex
- Zu langsam für Suche in sehr großen DNA-/Proteindatenbanken

→ Verwendung heuristischer Verfahren zur schnellen DB Suche

- Verzichten auf optimale Lösung (Begrenzung des Lösungsraumes)
- Finden kurzer exakter Übereinstimmungen, welche dann verlängert und zusammengefasst werden (*„seed-and-extend"*)
- Wichtige Systeme:
  - FASTA: Fast All (all= für Proteine und DNA)
  - BLAST: Basic Local Alignment Search Tool
  - Für beide existieren mehrere Varianten und verschiedene Implementierungen



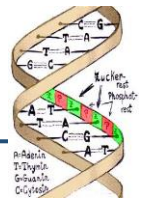
Indexierung

- Extrem große Datenmengen

→ keine direkte Anwendung von DP-Ansätzen möglich

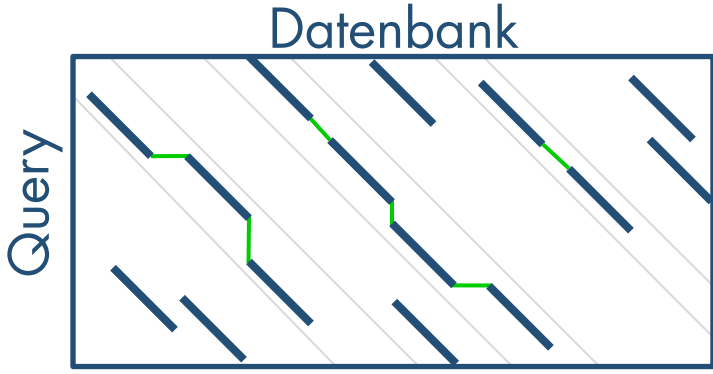
→ Indexierung für schnelle Suche, z.B. Suffixbäume, Suffixarrays, Lookup-Tabellen/Hashing, …

- Zugeständnisse:
  - Brauchen viel Hauptspeicher
  - Verwenden von Sekundärspeichern
  - In jedem Lauf neuen Index erstellen, Subindex, mehrere Durchläufe



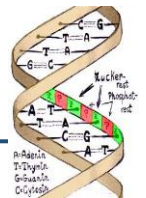
FASTA

- Suche alle exakten Matches (Hot-Spots) der Länge k zwischen Query- und DB-String *
  - Tupel: ($StartPos_{Query}$, $StartPos_{DB\text{-}Sequenz}$)
  - Effiziente Suche nach kurzen Matches durch Lookup-Tabelle, für Startpositionen der Hot-Spots in Query und DB-String



- Re-score: Bewertung der diagonalen Läufe zur Bestimmung der 10 besten Diagonalfolgen (Nutzung Substitutionsmatrix)
- Zusammenfassen mehrerer Regionen zu längerem besser bewerteten Alignment (auch Zulassen von Lücken)
  + Eliminieren von Segmenten, die vermutlich kein Teil des „highest scoring segment“ sind
- Berechnung der optimalen Teilalignments durch Anwendung der Bandvariante des SW, um die Diagonalen

* Üblich: $k = 6$ für DNA-Sequenzen, $k = 2$ für Protein-Sequenzen

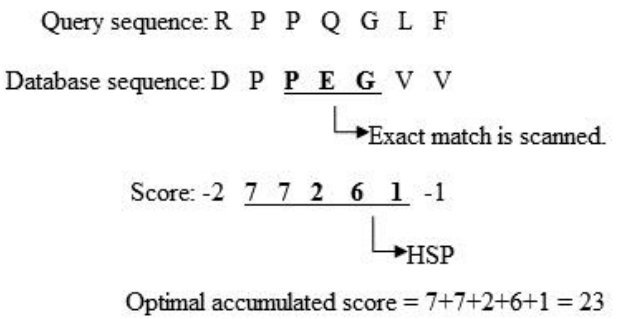


BLAST

- Preprocessing: bilde alle Teilworte einer geg. Länge w und speichere dazu alle möglichen w-mere mit score > threshold ab (unter Nutzung einer Substitutionsmatrix)*
- Suchen von *Hits*: ähnliche Teilstrings der Länge w zwischen Query- und DB-Sequenz (effizient wegen Preprocessing)
  - Unterschied FASTA: suchen nicht nur exakte Übereinstimmungen sondern alle lokalen Alignments (ohne Lücken) mit score > threshold
- Suchen aller Paare von Hits, die max. Abstand A voneinander haben
- Ausdehnen dieser Hit-Paare mit DP-Alg. bis sich die Bewertung nicht mehr erhöht (Hit-Paar über Schwellwert = *High Scoring Pair* (HSP))
- Ausgabe Blast: absteigend geordnete Liste der HSPs

Beispiel: Liste aller w-mere der Länge 2 mit Score T > 8 für die Sequenz RQCSAGW **

Teilwort	*w-mere*
RQ	RQ
QC	QC, RC, EC, NC, DC, HC, KC, MC, SC
CS	CS, CA, CN, CD, CQ, CE, CG, CK, CT
SA	
AG	AG
GW	GW, AW, RW, NW, DW, QW, EW, HW, KW, PW, SW, TW, WW



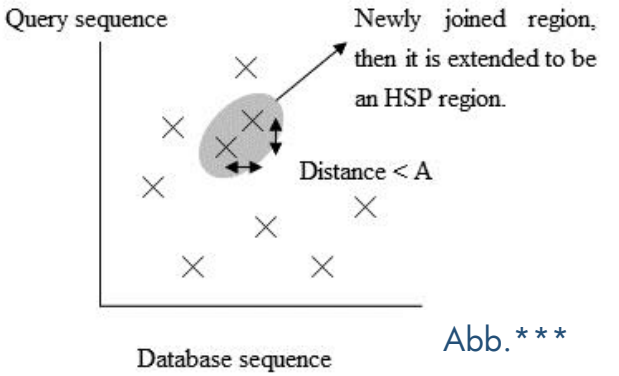
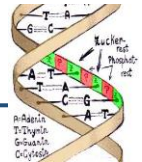


Abb.***

* Üblich: $w = 11$ für DNA-Sequenzen, $w = 3$ für Protein-Sequenzen

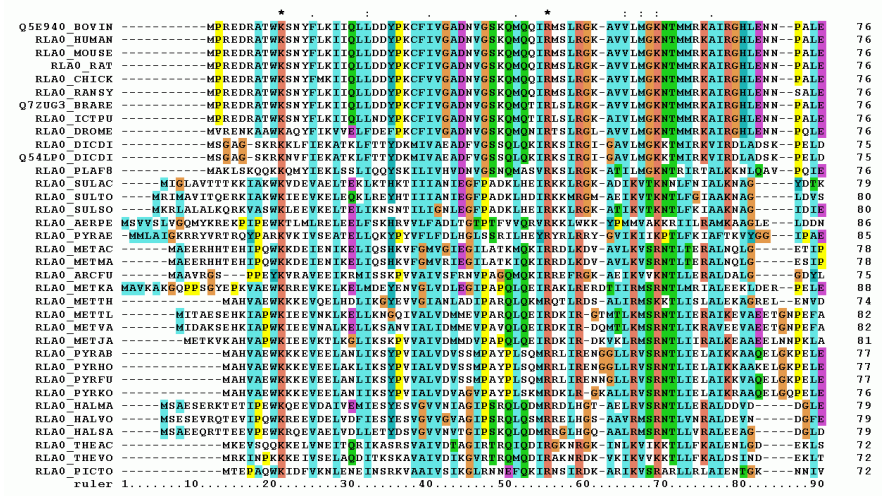
** Beispiel: http://www-lehre.img.bio.uni-goettingen.de/Bio_Inf/fastblast/fastblas.htm

*** Abbildung: http://en.wikipedia.org/wiki/File:Neighbor_HSP.jpg



Paarweise vs. multiple Alignments

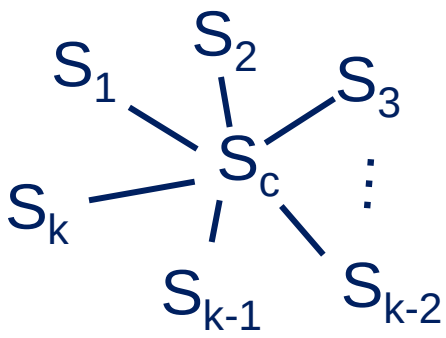
- Paarweise Alignments: Paarweiser Vergleich zweier Sequenzen
- Multiple Alignments: Vergleich über mehrere Sequenzen hinweg
  - Nutzung zur Rekonstruktion von Stammbäumen
  - Entdeckung konservierter* Regionen
- Speicherung der Alignments in RDBMS
  - Trefferregion: Start, Stopp, Strang
  - Ähnlichkeitswerte, Mismatch-Positionen

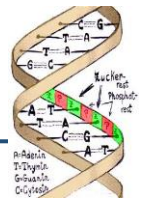


* konservieren = bewahren; konservierter Bereich der DNA blieb im Verlauf der Evolution erhalten (=existiert in vielen verschiedenen Spezies)

Multiple Alignments

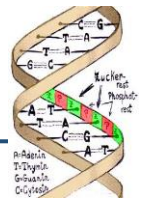
- Viele Dimensionen
- Exakte Berechnung des optimalen „Sum-of-Pairs"-Alignments:
  Laufzeit wächst exponentiell $O(2^k \cdot n^k)$
  (k - #Sequenzen, n-Länge der längsten Sequenz)
  NP-vollständiges Optimierungsproblem
- Heuristiken (optimale Lösung nicht garantiert!), z.B.
  - Multiple Alignments für kleinere Teilmengen bestimmen …
  - Zunächst alle optimalen paarweisen Alignments der zu untersuchenden Sequenzen, dann „Guide Tree" durch hierarchisches Clustern; berechnen und mergen der Teil-Alignments
  - Center-Star: Minimierung der Summe der Alignments aller Sequenzen $S_1 \ldots S_k$ gegen eine zentrale Consensussequenz S_c
  - …





Zusammenfassung

- Naive Textsuche
  - Einfache Realisierung ohne vorberechnete Hilfsinformationen
  - Worst Case O (n*m), aber oft linearer Aufwand O(n+m)
- Schnellere Ansätze zur dynamischen Textsuche
  - Vorverarbeitung des Musters, jedoch nicht des Textes
  - Boyer-Moore: Worst-Case O(n*m) bzw. O(n+m), aber im Mittel oft sehr schnell O (n/m)
- Indexierung erlaubt wesentlich schnellere Suchergebnisse
  - Vorverarbeitung des Textes bzw. der Dokumentkollektionen
  - Hohe Flexibilität von Suffixbäumen (Probleme: Größe; Externspeicher-Zuordnung)
- Approximative Suche, Alignments
  - Finden des optimalen Alignments, z.B. optimales lokales Alignment: Smith-Waterman
  - Bestimmung der optimalen Folge von Editier-Operationen sowie Editierdistanz über dynamische Programmierung (DP) in O(n*m)
  - Gap-Kosten, Substitutionsmatrizen
  - Oftmals Einsatz heuristischer Verfahren wie z.B. BLAST



Fragen ?