Universität Ulm

Institut für Medizinische Systembiologie

Prof. Dr. H. Kestler

ERWEITERTES WILD-BASEN NUKLEOTID-PROTEIN-SEQUENZALIGNMENT

Dissertation zur Erlangung des Doktorgrades der Humanbiologie der Medizinischen Fakultät der Universität Ulm

Axel Fürstberger
Memmingen
2016

Amtierender Dekan: Professor Dr. Thomas Wirth

1. Berichterstatter: Professor Dr. Hans A. Kestler

2. Berichterstatter: Juniorprofessor Dr. Jens von Einem

Tag der Promotion: 17.02.2017

Gewidmet meiner Familie

INHALTSVERZEICHNIS

Abkürzungsverzeichnis V

1 EINLEITUNG 1
1.1 Biologische Grundlagen 1
1.2 Sequenzalignment 13
1.3 Motivation 27

2 METHODEN 29
2.1 Übersicht und Taxonomie 29
2.2 Dynamisches Programmieren 32
2.3 Globales und lokales Sequenzalignment 32
2.4 Lokale Datenbank-Alignments 58
2.5 DNS-Protein-Alignment 65
2.6 Implementierungen 73
2.7 Generischer Webseiten-Analyzer 94

3 ERGEBNISSE 99
3.1 Simulationen 103

4 DISKUSSION 120

5 ZUSAMMENFASSUNG 126

6 LITERATURVERZEICHNIS 127

Lebenslauf 134

ABKÜRZUNGSVERZEICHNIS

A Adenin

Ala Alanin

api application programming interface

Arg Arginin

Asn Asparagin

Asp Asparaginsäure

Blast Basic Local Alignment Search Tool

Blastx Basic Local Alignment Search Tool X

BLAT BLAST-like alignment tool

BLOSUM BLOck SUbstitution Matrix

BWT Burrows-Wheeler-Transformation

C Cytosin

CSS Cascading Style Sheet

CSV Comma Seperated Value

Cys Cystein

DNS Desoxiribonukleinsäure

FAAST Flow-space Assisted Alignment Search Tool

Fasta Fast Nucleic Acid Data Base Search

FastLSA Fast Local Similarity Analysis

FastP Fast Protein Search

Fasty Fast Y

G Guanin

Gln Glutamin

Glu Glutaminsäure

Gly Glycin

Haxat Homopolymer Aware Cross Alignment Tool

HCMV Humanes Cytomegalievirus

HCV Hepatitis-C-Virus

His Histidin

HSP High Scoring Pair

HSV Herpes simplex-Virus

MSP Maximal Scoring Pair

Ile Isoleucin

IUPAC International Union of Pure and Applied Chemistry

js JavaScript

json JavaScript Object Notation

LAP Local Alignment Program

Leu Leucin

Lys Lysin

Met Methionin

MRA Mutations Resistance Analyzer

mRNS messenger-Ribonukleinsäure

NAP Nucleotide-Amino acid alignment Program

PAM Point Accepted Mutation oder Percent of Accepted Mutations

PDF Portable Document Format

Phe Phenylalanin

PHP PHP: Hypertext Preprocessor

PROTOMAT Protein-Alignment-Automat

Pro Prolin

PSI-BLAST Position-Specific Iterated BLAST

RAND Research ANd Development

RNS Ribonukleinsäure

Ser Serin

SSAHA Sequence Search and Alignment by Hashing Algorithm

Swat Smith Waterman extended Alignment Tool

T Thymin

Thr Threonin

Trp Tryptophan

Tyr Tyrosin

U Uracil

Val Valin

ZPM Zhang Pearson Miller Algorithmus

Teile dieser Dissertation wurden in folgender Publikation veröffentlicht:

Axel Fürstberger, Markus Maucher and Hans A. Kestler (2014): Extended pairwise local alignment of wild card DNA/RNA sequences using dynamic programming, Journal of Statistical Computation and Simulation, DOI: 10.1080/00949655.2014.928294

Mit freundlicher Genehmigung des Journal of Statistical Computation and Simulation

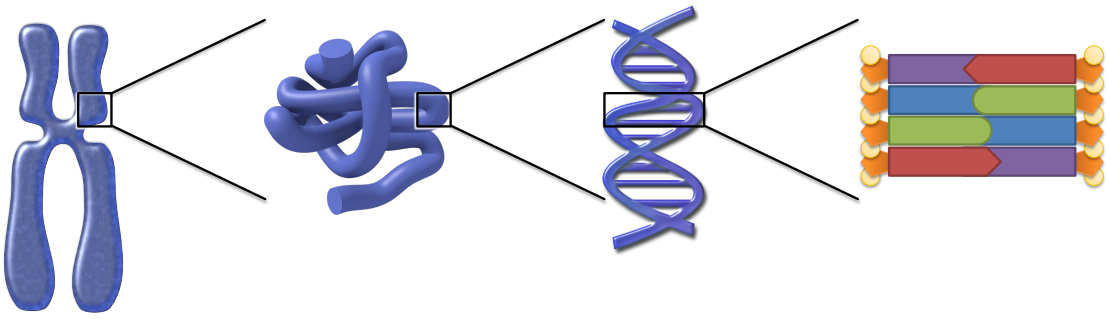
1

EINLEITUNG

Zur Ähnlichkeitsanalyse biologischer Sequenzen wird das sogenannte Sequenzalignment eingesetzt. Bei einem solchen Alignment werden die Elemente einer zu untersuchenden Sequenz so einer oder mehreren Sequenzen zugeordnet, dass die Reihenfolge der Elemente erhalten bleibt und jedes Element einem anderen Element oder einer Lücke in der entsprechenden Sequenz zugeordnet wird. Die Resultate dieser Analyse lassen sich in verschiedenen Bereichen einsetzen, beispielsweise in der Genetik und der Medizin. In dieser Arbeit wurde ein neuer Alignmentalgorithmus entwickelt, welcher die Verarbeitung von Wild-Basen im Alignmentprozess unterstützt. Der neue Algorithmus basiert auf dem Vorgehen der dynamischen Programmierung.

1.1 BIOLOGISCHE GRUNDLAGEN

Die genetischen Informationen jedes lebenden Organismus sind in der DNS, der Desoxyribonukleinsäure, gespeichert. In Eukaryoten liegt sie im Zellkern aufgewickelt und verdrillt in Form von Chromosomen vor. In Prokaryoten liegt sie frei im Zytoplasma, der ausfüllenden Grundsubstanz der Zelle. Die DNS ist eine Doppelhelix aus zwei Nukleotid-Polymer-Strängen. Diese Nukleotide bestehen aus einem Zucker, der 2-Desoxyribose, einem Phosphat und einer aus vier möglichen Nukleinbasen. So ergeben sich die vier Nukleotide der DNS: Adenin, Thymin, Cytosin und Guanin, die mit ihren Anfangsbuchstaben A, T, C und G abgekürzt werden und über das Zucker-Phosphat-Rückgrat verbunden sind. Über das sogenannte Watson-Crick-pairing [67] der Komplementärbasen werden die Einzelstränge miteinander zur Doppelhelix verknüpft (siehe Abbildung 1).



(a) Chromosom **(b) verdrillter DNS-Strang** **(c) DNS-Doppelhelix** **(d) Basenpaare mit Zucker-Phosphat-Rückgrat**

Abbildung 1: In den Chromosomen (a) befindet sich der verdrillte DNS-Strang (b), in welcher die DNS in ihrer Strickleiterform, der DNS-Doppelhelix (c) vorliegt. Diese besteht aus einem Zucker-Phosphat-Rückgrat auf beiden Seiten und den verknüpften Basenpaaren dazwischen (d).

Das Zucker-Phosphat-Rückgrat verbindet die Nukleotide über die Kohlenstoffe der 2-Desoxyribose an Stelle 3′ bzw. 5′ (siehe Abbildung 2 und 3).

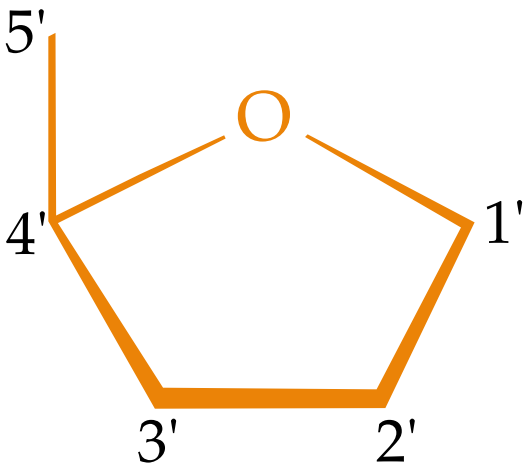


Abbildung 2: Strukturformel der 2-Desoxyribose: Die nummerierten Stellen geben die Positionen der Kohlenstoffe an.

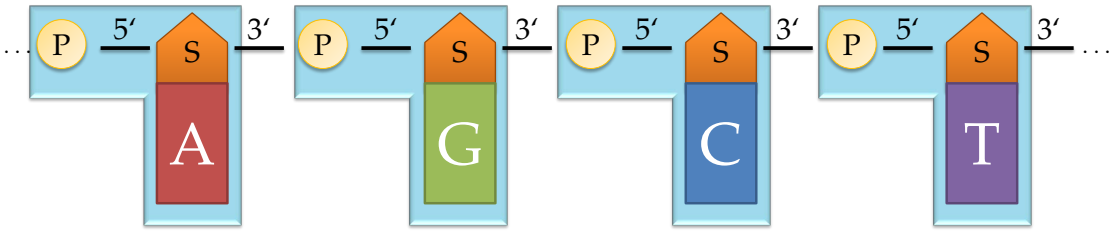


Abbildung 3: Schematische Darstellung eines Einzelstrangs der DNS: Phosphatgruppen (bezeichnet mit P) werden an 2-Desoxyribose (bezeichnet mit S für sugar) über 5′ bzw. über 3′ verbunden. An die Zucker sind unterschiedliche Nukleinbasen an Stelle 1′ gebunden. Eine solche Gruppe (in hellblau) aus Phosphat, Zucker und Base wird als Nukleotid bezeichnet.

Die DNS enthält das Erbgut, also Informationen, welche ein Organismus benötigt, um zu funktionieren und wird nach Replikation an die Tochterzellen übertragen. Das *Zentrale Dogma der Molekularbiologie*, formuliert von Crick (1958 bzw. 1970) [14, 16], besagt, dass die Übertragung von Informationen von Nukleinsäure zu Nukleinsäure bzw. von Nukleinsäure zu Protein möglich ist, jedoch die Übertragung von Protein zu Protein bzw. von Protein zu Nukleinsäure nicht (siehe Abbildung 4).

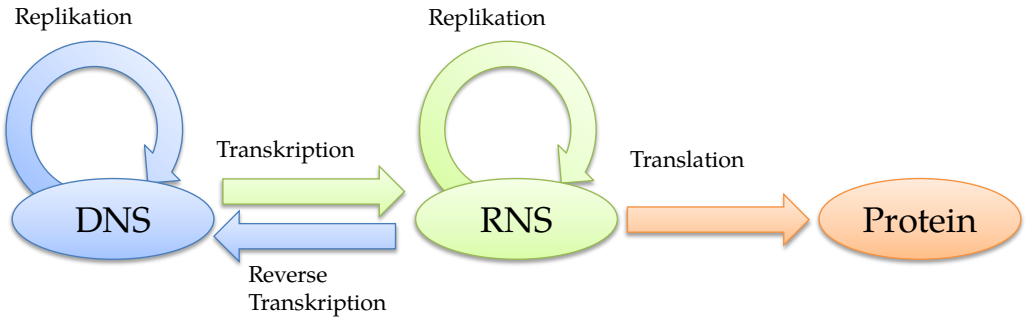


Abbildung 4: Grafische Darstellung des "Zentralen Dogmas der Molekularbiologie": DNS und RNS replizieren sich und tauschen ebenfalls Informationen aus. Doch weder Replizieren sich Proteine, noch findet ein Informationsfluss vom Protein an die RNS oder DNS statt.

Die komplementäre Struktur der DNS bildet die Basis für die semikonservative Replikation. Durch Auftrennung der Doppelhelix in Einzelstränge bilden sich zwei Vorlagen, welche für die Replikation verwendet werden. Im Fall der DNS-Replikation werden beide Einzelstränge kopiert. Im Fall der Transkription wird

nur einer der Stränge in RNS (Ribonukleinsäure) übersetzt. Die RNS transportiert die Informationen aus dem Zellkern heraus; die originale DNS verlässt den Kern nicht. Die Nukleotide, welche die RNS bilden, sind A, C, G (wie bei der DNS) und Uracil (U), welches Thymin ersetzt. Als Sekundärstrukturen bildet die RNS vor allem Hairpin-, Stem- und andere Loop-Strukturen zur Stabilisierung (siehe Abbildung 5).

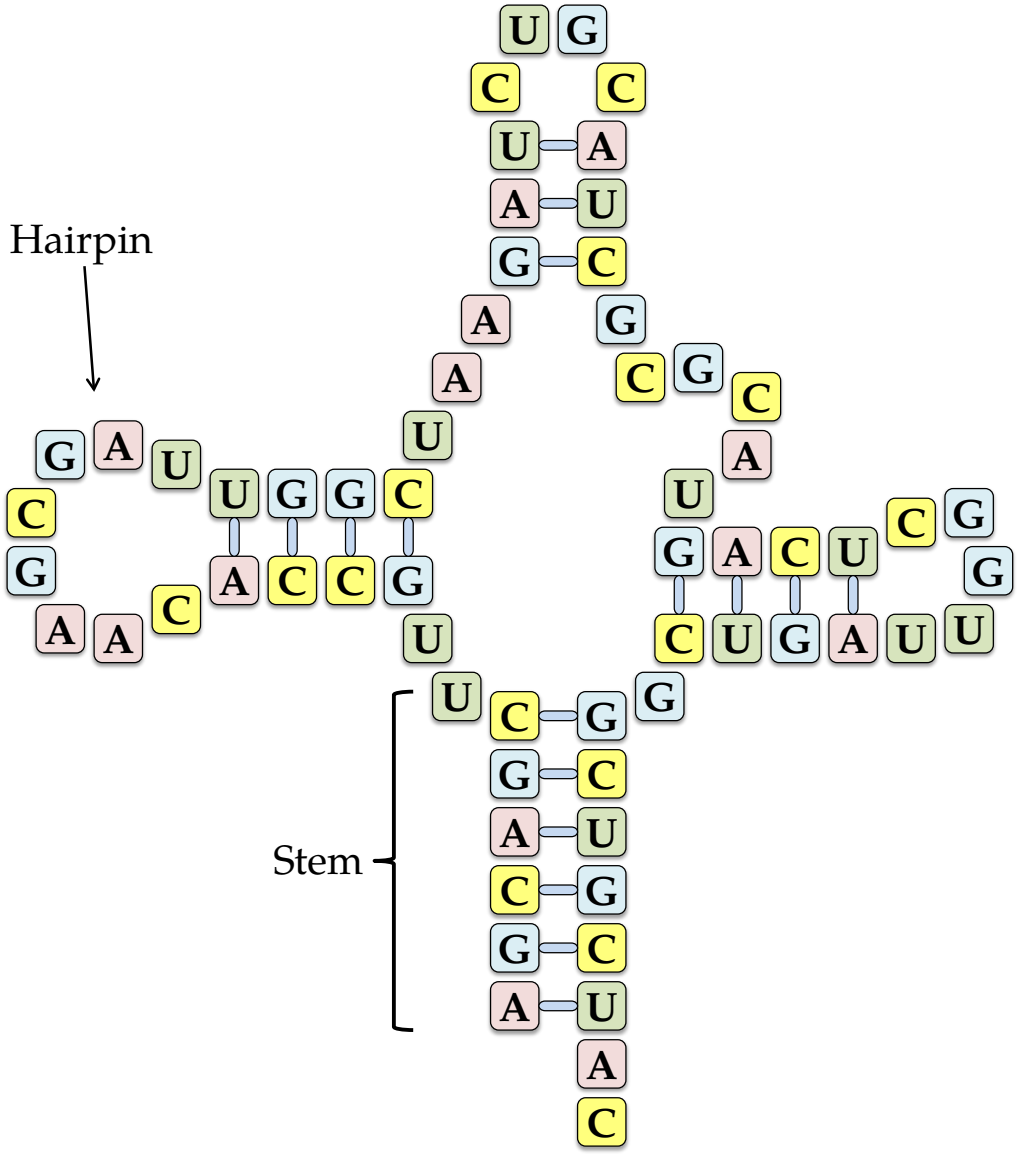


Abbildung 5: Schematische Darstellung einer RNS-Sekundärstruktur mit Stem- und Hairpin-Strukturen. Durch die unterschiedlichen Teile der Sekundärstruktur entsteht eine stabile Struktur der RNS.

Neben dem Übertragen genetischer Informationen kann die RNS auch zur Übersetzung von Nukleotidketten in Protein beitragen, sowie katalytische Funktionen, ähnlich Enzymen haben [8].
In der DNS sind Geninformationen gespeichert. Nur ca. $4\% - 6\%$ der DNS kodieren Gene. In der Proteinsynthese werden diese DNS-Sequenzen in Proteine übersetzt. Die Synthese erfolgt in zwei Schritten: der Transkription und der Translation. In der Transkriptionsphase wird eine Kopie der Gensequenz, die Boten-RNS oder messenger-RNS (kurz: mRNS) erstellt, in der Thymin durch Uracil ersetzt ist. In der Translationsphase werden Tripletts der RNS-Nukleotide in Aminosäuren übersetzt. Dabei werden je drei Nukleotide zu Tripletts zusammengefasst, welche jeweils eine Aminosäure kodieren (siehe auch Abbildung 9). Eine solche Abfolge von drei aufeinanderfolgenden Basen wird auch Codon genannt. Die Übersetzung ist nicht bijektiv, da trotz $4^3 = 64$ Triplett-Basen-Kombinationen nur 20 Aminosäuren entstehen. Crick hat 1966 eine Hypothese formuliert, die dieses Phänomen erklärt: Die Bindung kann an verschiedenen Stellen des Codons "wackeln" und trotzdem bildet das Codon die gleiche Aminosäure [15]. Sowohl Abbildung 6 als auch Abbildung 7 zeigen, wie die Zuordnung der verschiedenen Basentripletts / Codons zu den Aminosäuren ist. In diesen Abbildungen sind ebenfalls Startcodon (Methionin), ein Triplett, welches den Beginn der Transkription markiert, und die drei Stopcodons, welche das Ende der zu transkribierenden Sequenz kennzeichnet, annotiert.

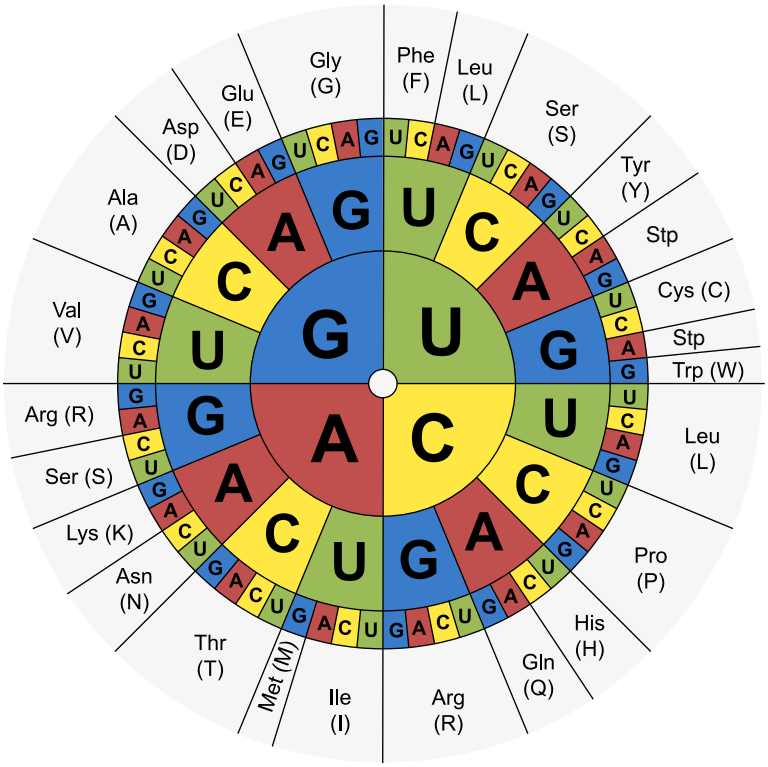


Abbildung 6: Darstellung der Triplett-Kombinationen zu Aminosäuren in Form der sogenannten Amino-Sonne: Ausgehend vom Mittelpunkt der Sonne durchläuft man nach außen hin drei Kreissegmente. Kombiniert ergeben die in den Segmenten vermerkten Buchstaben Nukleotidtripletts, welche in die im äußersten Kreis angegebenen Aminosäuren bzw. Stopcodons (abgekürzt mit Stp) übersetzt werden.

1. Base	2. Base U		2. Base C		2. Base A		2. Base G	
U	UUU	Phenylalanin	UCU	Serin	UAU	Tyrosin	UGU	Cystein
	UUC		UCC		UAC		UGC	
	UUA	Leucin	UCA		UAA	Stop	UGA	Stop
	UUG		UCG		UAG		UGG	Tryptophan
C	CUU		CCU	Prolin	CAU	Histidin	CGU	Arginin
	CUC		CCC		CAC		CGC	
	CUA		CCA		CAA	Glutamin	CGA	
	CUG		CCG		CAG		CGG	
A	AUU	Isoleucin	ACU	Threonin	AAU	Asparagin	AGU	Serin
	AUC		ACC		AAC		AGC	
	AUA		ACA		AAA	Lysin	AGA	Arginin
	AUG	Methionin	ACG		AAG		AGG	
G	GUU	Valin	GCU	Alanin	GAU	Asparaginsäure	GGU	Glycin
	GUC		GCC		GAC		GGC	
	GUA		GCA		GAA	Glutaminsäure	GGA	
	GUG		GCG		GAG		GGG	

Abbildung 7: Die Codon-Tabelle: Eine tabellarische Darstellung der Nukleotid-Tripletts und der zugehörigen Aminosäuren. Neben den Zeilen für die erste Base, die Spalten für die zweiten Basen und der Zellenbeschriftung inklusive der dritten Base, fasst die farbliche Markierung die chemischen Eigenschaften der Aminosäure zusammen:
gelb: hydrophob (unpolar)
grün: hydrophil und neutral (polar)
blau: hydrophil und positiv geladen (basisch)
rot: hydrophil und negativ geladen (sauer)

Proteine spielen in unserem Körper eine sehr wesentliche und entscheidende Rolle. Die Analyse dieser Polymere ist eine wichtige Aufgabe in der modernen Bioinformatik. Es lassen sich Vergleiche von biologischen Funktionen, gemeinsamen Vorfahren und der räumlichen Struktur von Molekülketten innerhalb einer Spezies oder über verschiedene Arten hinweg anstellen [34].

1.1.1 *Mutationen*

Das Zusammenfassen der Nukleotide zu Tripletts und die Umwandlung dieser Codons zu Aminosäuren ist der Kern der Proteinsynthese. Wenn die zu verarbeitende RNS-Sequenz fehlerfrei und unverfälscht ist, lassen sich Start und Ende

bestimmen (Startcodon in Form von Methionin beziehungsweise Stopcodon). Die Kombination je dreier Nukleotide zu Aminosäuren ist eindeutig (siehe Abbildung 6 und 7). Abbildung 8 zeigt drei möglichen Startpositionen, auch Leserahmen genannt, welche sich ergeben, wenn unklar ist, welches die erste zu kodierende Base ist. Da immer drei Basen zu einem Codon zusammengefasst werden, wiederholen sich die Leserahmen triperiodisch.

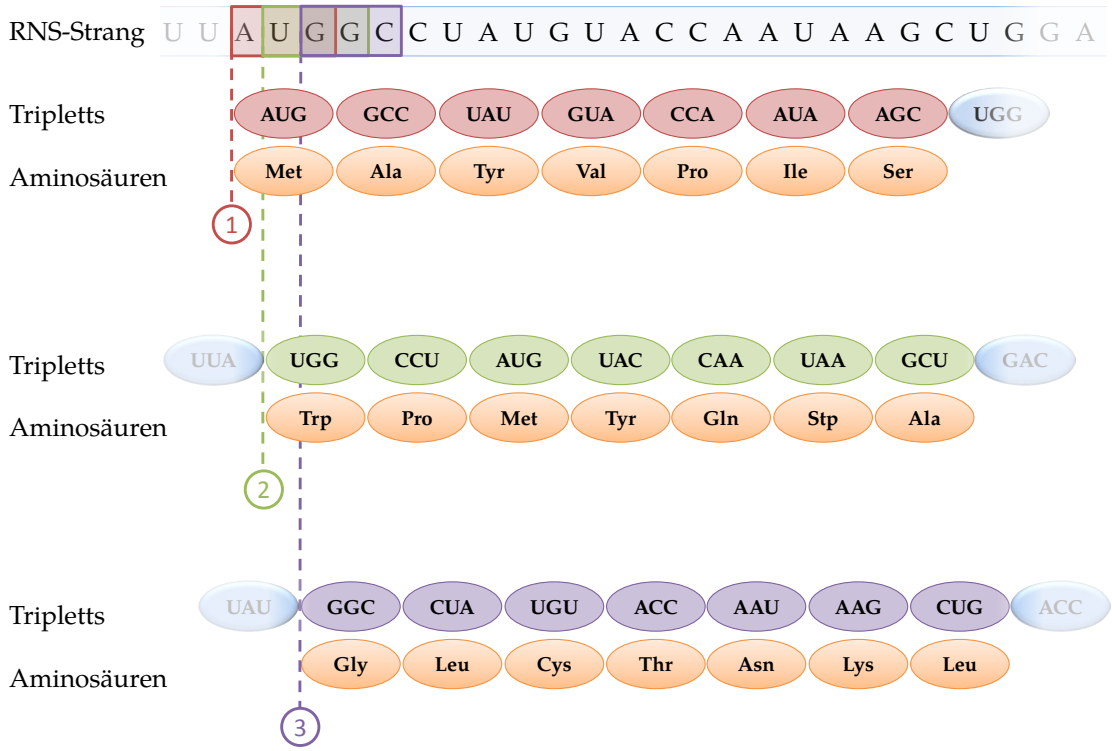


Abbildung 8: Schematische Darstellung dreier Leserahmen für eine RNS-Sequenz. Die Codon-Umwandlung ergibt je nach Startposition (markiert mit 1-3) der Tripletts unterschiedliche Aminosäuren und damit auch unterschiedliche Proteine.

Den Leserahmen exakt festzulegen, ist entscheidend für die Proteinsynthese. Die Startposition gibt den Leserahmen für die Codon-Umwandlungen vor. Punktmutationen in Form von Nukleotid-Insertionen bzw. -Deletionen können den Leserahmen verschieben und damit das betroffene und die folgenden Tripletts verändern. Diese Frameshiftmutation sind auf Aminosäureebene schwer zu erkennen [57]. Ihr

Einfluss auf die Proteinsynthese kann schwerwiegender sein, als die Insertion oder Deletion einer kompletten Aminosäure [11].
Eine weitere Form der Punktmutationen ist die Substitution einer Base. Dabei können vier Fälle eintreten:

- eine sinnverändernde Mutation (missense mutation): In diesem Fall wird eine Aminosäure innerhalb der Protein-Sequenz verändert (siehe Abbildung 10).
- eine sinnentstellende Mutation (nonsense mutation): Ein Triplett kodiert ein Stop-Codon und führt zum vorzeitigem Abbruch der Proteinsynthese (siehe Abbildung 11).
- eine stille Mutation (silent mutation): Eine Base wird ersetzt, kodiert aber im Triplett noch immer die gleiche Aminosäure (Wobble-Stelle).
- eine überlesende Mutation (readthrough mutation): Das Stopcodon wird zu einer anderen Aminosäure verändert, wodurch das Kodierungsende entfernt wird.

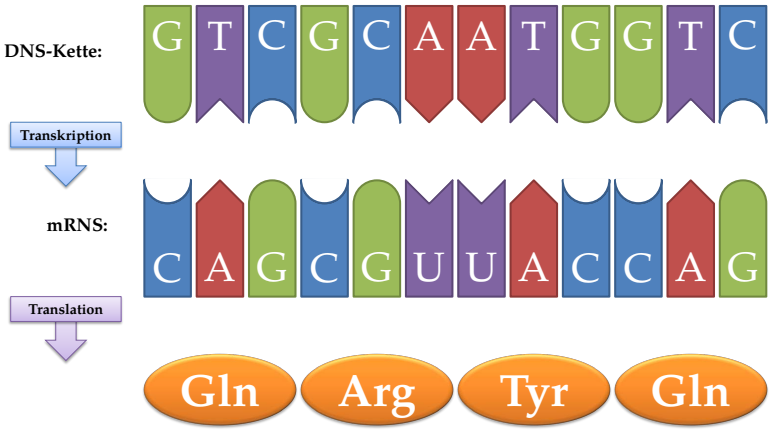


Abbildung 9: Schrittweise Umwandlung des DNS-Strangs in Aminosäuren: In der Transkriptionsphase wird die RNS über Bindung der erforderlichen Nukleotide an die Komplementärbasen der DNS synthetisiert und zur messenger-RNS umgewandelt. In der Translationsphase werden dann die Tripletts in Aminosäuren übersetzt.

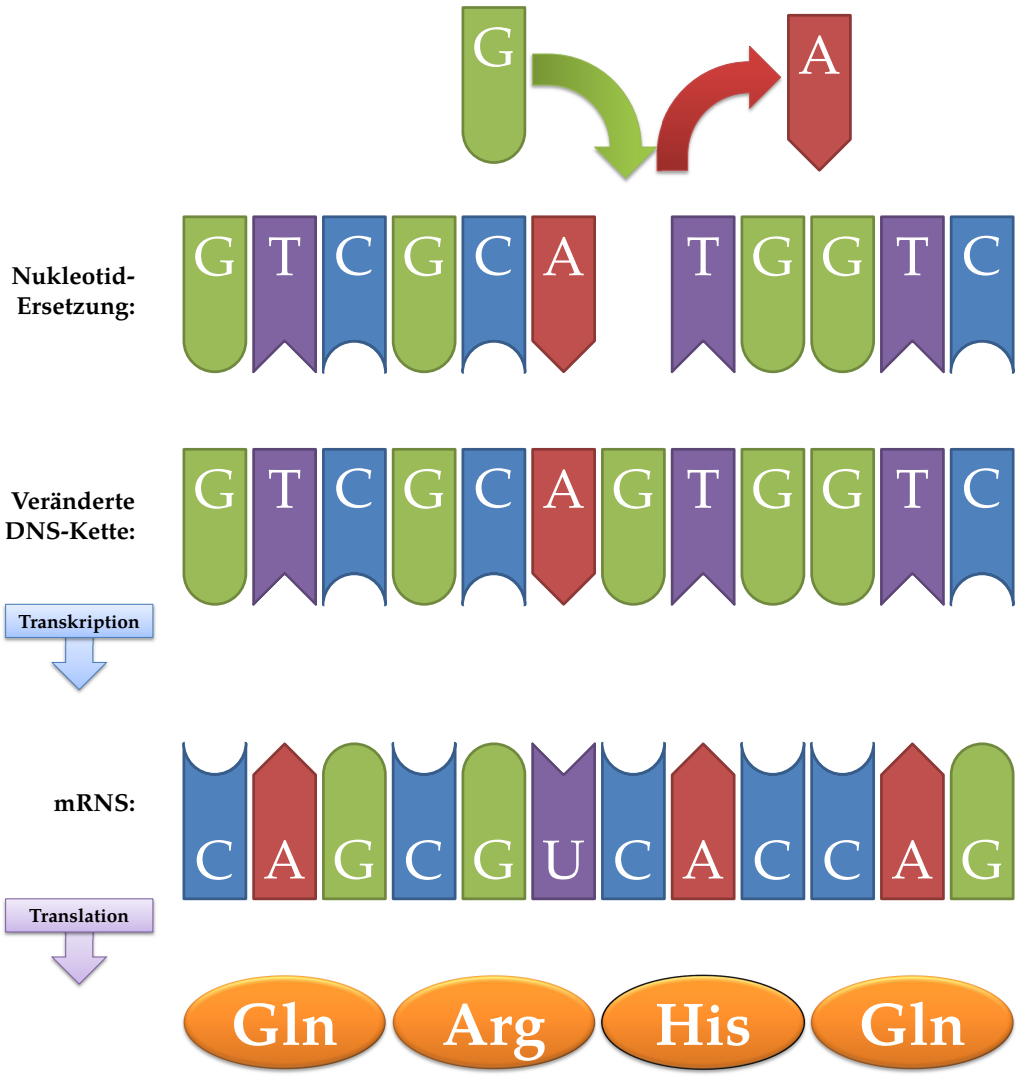


Abbildung 10: Beispiel einer sinnverändernden Mutation: Bei der Transkription wird das in der DNS durch Guanin ersetzte Adenin als Cytosin in der RNS gespeichert. Dies führt bei der Translation zum Einbau von Histidin statt Tyrosin.

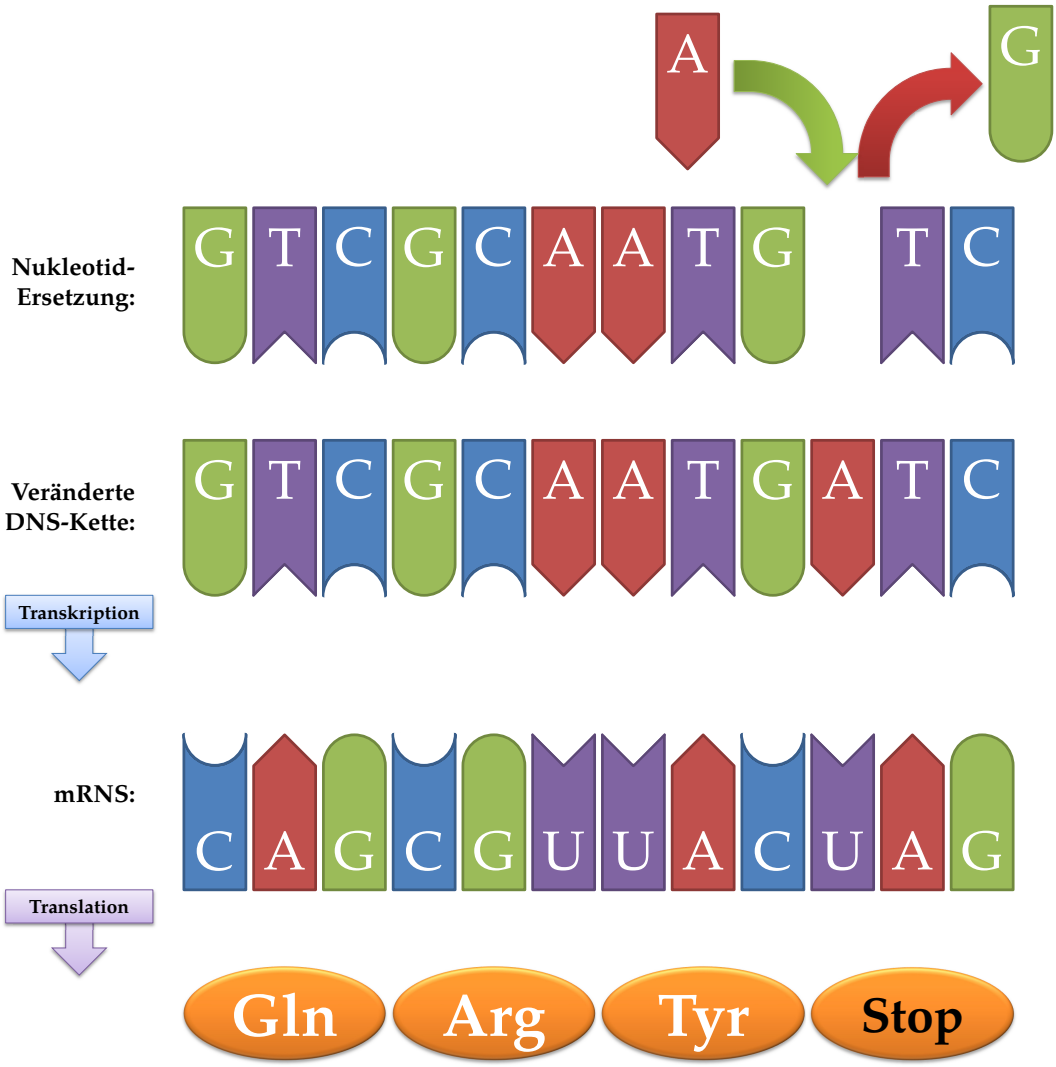


Abbildung 11: Beispiel einer sinnentstellenden Mutation: Durch die Ersetzung der Base Guanin mit Adenin entsteht bei der Übersetzung der veränderten RNS ein Stop-Codon. Dies führt zum vorzeitigen Abbruch der Proteinsynthese.

Das Einfügen bzw. Löschen eines Nukleotids führt zu einer Verschiebung des Leserahmens (siehe Abbildung 12).

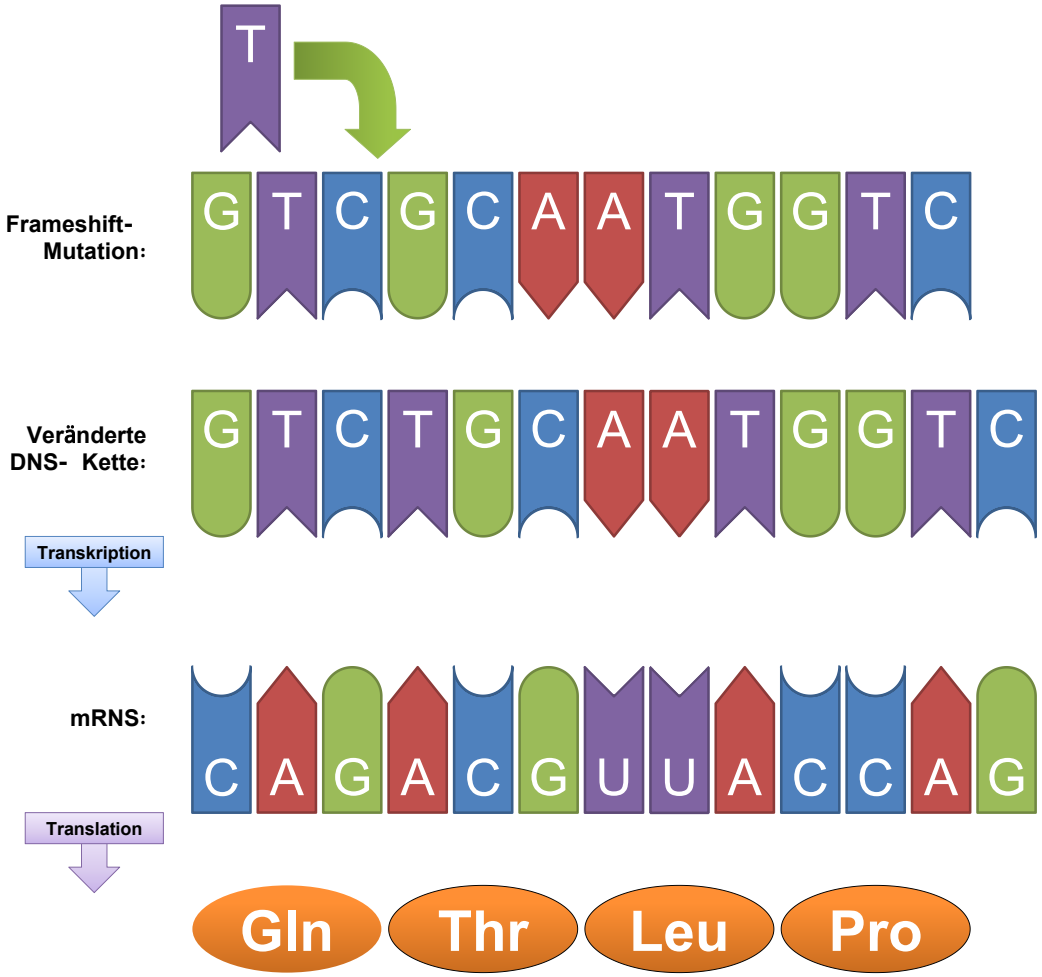


Abbildung 12: Das Einfügen von Thymin führt zur Verschiebung des Leserahmens. Die Frameshiftmutation verändert alle Folge-Aminosäuren in der Translations-Phase und verändert das Protein.

Störungen bzw. Fehler in der DNS können verheerende Folgen haben (zum Beispiel genetische Erkrankungen [22, 33, 28, 40]). Das bekannteste Beispiel hierfür ist Krebs [66]. Diese Störungen können durch unterschiedlichste Faktoren ausgelöst werden, zum Beispiel durch Strahlung, biologische oder chemische Substanzen. Doch nicht jede Störung in der DNS bedeutet ein direktes Risiko. Manche Störungen sind in Regionen, die nicht für die Kodierung der Gene bzw. Proteine zuständig sind [48], andere werden durch das mehrfache Vorliegen bestimmter Abschnitte kompensiert, ähnlich wie bei fehlerkorrigierenden Codes [20]. Wieder andere sind so extrem,

dass die DNS nicht weiter als Bauplan verwendet werden kann und so komplett verworfen wird - die Zelle stirbt [50].

1.2 SEQUENZALIGNMENT

Zur Suche von Übereinstimmungen bestimmter (Teil)Wörter oder Sätze in Texten werden Textmatching-Algorithmen eingesetzt. Eine besondere Unterklasse des (exakten) Textmatching-Problems ist das Alignmentproblem. Dieses berechnet ein Maß für die Ähnlichkeit zweier (oder mehrerer) Sequenzen [17]. Für die einzelnen Nukleotide der DNS bzw. der RNS als auch für die einzelnen Aminosäuren kann ein Abstandsmaß definiert werden. Somit induziert diese Norm auf der Menge der DNS- und RNS-, als auch Proteinsequenzen eine Metrik. Eine Metrik (oder auch Abstandsfunktion) weist zwei Sequenzen einen nicht negativen Wert zu, welcher als Abstand oder auch Ähnlichkeitsmaß betrachtet werden kann. Dieser Wert gibt an, wie stark sich zwei Sequenzen unterscheiden bzw. wie ähnlich sie sind. Eine solche Metrik muss folgende Eigenschaften erfüllen:

1. positiv semidefinit: Der Abstand zweier Sequenzen s und t ist stets größer oder gleich Null. $d(s,t) \geqslant 0$.
2. symmetrisch: Der Abstand von s zu t ist gleich dem Abstand von t zu s. $d(s,t) = s(t,s)$.
3. Dreiecksungleichung: Der Abstand von s zu u ist kleiner oder gleich dem addierten Abstand von s zu t und t zu u. $d(s,u) \leqslant d(s,t) + d(t,u)$.

1.2.1 *Problemstellung*

Betrachtet man die Problemstellung eines Alignments als Maximierungsproblem, so gibt ein höherer Wert eine größere Ähnlichkeit an, indem beispielsweise die übereinstimmenden Zeichen gezählt werden. Als Minimierungsproblem hingegen, ist ein kleiner Wert die Zielsetzung, um einen möglichst geringen Abstand zweier Sequenzen zu finden. Um ein optimales Alignment bei DNS-, RNS- oder Proteinsequenzen zu berechnen, werden mehrere Komponenten berücksichtigt, angefangen bei der Länge einer betrachteten Sequenz bis hin zu den unterschiedlichen Formen der möglichen Mutationen. Eine entsprechende Metrik deckt diese Merkmale ab und gewichtet sie.

1.2.2 *Der Hamming-Abstand*

Eine Metrik, die auch in der Kodierungstheorie angewendet wird, ist die sogenannte Hamming-Distanz oder der Hamming-Abstand [32]. Sie misst die Ähnlichkeit zweier Sequenzen, indem sie die Übereinstimmungen bzw. Unterscheide an den einzelnen Positionen zählt:
Es seien zwei gleichlange Wörter s ($s = s_1, ..., s_n$) und t ($t = t_1, ..., t_n$) aus einem Alphabet Σ gegeben. Dann ist der Hamming-Abstand zwischen s und t definiert als die Anzahl der Positionen, an denen sich s und t unterschieden (siehe Formel 1).

$$\Delta(s,t) := \sum_{s_i \neq t_i} 1 \quad \text{für} \quad i = 1, .., n$$

Formel 1: Definition des Hamming-Abstandes.

Diese Formel erfüllt die Eigenschaften einer Metrik:

1) positiv semidefinit: $\Delta(s,t) \geqslant 0$, da $\sum_{s_i \neq t_i} 1 \geqslant 0$

2) symmetrisch: $\Delta(s,t) = \sum_{s_i \neq t_i} 1 = \sum_{t_i \neq s_i} 1 = \Delta(t,s)$

3) Dreiecksungleichung: $\Delta(s,u) \leqslant \Delta(s,t) + \Delta(t,u)$

Beweis:
Falls $s = u$, so ist die Bedingung erfüllt, da dann $\Delta(s,u) = 0$ und die positive Semidefinitheit schon gezeigt wurde.
Falls $s \neq u$, so ist mindestens ein $s_i \neq u_i$.
Ist $s = t$, so ist auch $\Delta(s,u) = \Delta(t,u)$. Damit gilt $\Delta(s,u) \leqslant \Delta(s,t) + \Delta(t,u)$, da $\Delta(s,t) \geqslant 0$ (positiv semidefinit).
Ist $t = u$, so ist $\Delta(s,u) = \Delta(s,t)$, also auch hier ist die Ungleichung erfüllt. Ist $s \neq t \neq u$, so gilt:
$k \in \{i | s_i \neq u_i\} \Rightarrow s_k \neq u_k \Rightarrow s_k \neq t_k \vee t_k \neq u_k \Rightarrow k \in \{i | s_i \neq t_i\} \vee k \in \{i | t_i \neq u_i\}$.
Jedes Zeichen, welches sich in s von u unterscheidet ist entweder in t oder in u verschieden. Somit ist für alle Fälle die Ungleichung erfüllt.
Damit ist der Hamming-Abstand eine Metrik.

■

Hier wird das Maß der Ähnlichkeit als Minimierungsproblem verstanden. Je weniger Buchstaben sich an den gleichen Positionen in den beiden Sequenzen

unterscheiden, desto ähnlicher sind s und t.
Für die Überprüfung eines Passworts ist dieses Vorgehen durchaus denkbar. Ist der Hamming-Abstand des gespeicherten Passworts und der eingegebenen Zeichenkette 0, so stimmen beide Wörter überein und das eingegebenen Passwort ist korrekt. Betrachtet man als weiteres Beispiel die zwei Geburtsdaten 01.07.1978 und 1.7.78, so beschreiben diese den gleichen Tag. Der Hamming-Abstand ist jedoch maximal und kann auch auf Grund der unterschiedlichen Länge der beiden Zeichenketten nicht verwendet werden.

1.2.3 *Die Levenshtein-Distanz*

Im Falle biologischer Daten (DNS, RNS oder Proteine) repräsentiert das Sequenzalignment die führende Homologie-Hypothese. Dabei werden zum Beispiel zwei Gene als homolog bezeichnet, wenn sie von einem gemeinsamen Vorfahren abstammen. Doch verändern sich Sequenzen im Laufe der Evolution oder durch Umwelteinflüsse. Die Veränderungen führen nicht nur zum Austausch von Basen(-Ketten) sondern auch zu Insertionen und Deletionen innerhalb der Sequenzen. Um diese Veränderungen im Alignment zu berücksichtigen, werden sogenannte "Lücken" (Gaps) in den Sequenzen eingeführt. Li beschreibt diese Lücken wie folgt [44]: Deletionen und Insertionen werden gemeinsam unter dem Begriff Gaps geführt, da der Vergleich einer Sequenz mit Insertionen oder Deletionen mit der Originalsequenz zu einer Lücke in entweder dieser oder der Originalsequenz zur Folge hat. Li unterscheidet ebenfalls zwei Arten von Gaps, abhängig von ihrer Länge: Kurze Lücken entstehen hauptsächlich durch Fehler in der DNS-Replikation, längere Lücken durch ungleiche Überkreuzungen oder DNS-Transpositionen [44].
Um diese Lücken im Alignmentprozess zu berücksichtigen, muss ein weiteres Zeichen dem Arbeitsalphabet hinzugefügt werden, welches nicht in den normalen Zeichenketten vorhanden ist. Der Standard, welches sich hierfür durchgesetzt hat ist "_".
Für zwei Zeichenketten bzw. Buchstabensequenzen s mit Länge n und t mit Länge m soll die minimale Anzahl an Änderungen berechnet werden, die nötig sind, um (ohne Beschränkung der Allgemeinheit) die Sequenz t in die Sequenz s zu überführen. Diese Änderungen umfassen lediglich die Operationen Einfügen, Löschen (dargestellt als _ in der jeweiligen Sequenz) oder Ersetzen. Für jede Änderung, die an der Sequenz vorgenommen wird, wird ein Malus vergeben und aufaddiert (siehe Formel 2).

$$\sum_{i=1}^{n} \sigma(x_i, y_i) \text{ mit } \sigma(x_i, y_i) = \begin{cases} 0 \text{ , falls } x_i = y_i \\ 1 \text{ , falls } x_i = _ \text{ oder } y_i = _ \\ 1 \text{ , falls } x_i \neq y_i \end{cases}$$

Formel 2: Beispiel einer Kostenfunktion für die Levenshtein-Distanz

Eine obere Schranke für die Anzahl der Änderungen kann bestimmt werden, indem alle Zeichen von Sequenz t durch die Zeichen von Sequenz s ersetzt werden und fehlende oder überschüssige Zeichen ergänzt, respektive gelöscht werden, also: $n + |n - m|$. Dies gibt auch eine obere Schranke für die Kosten vor. Somit kann ein Maß für die Ähnlichkeit zweier (oder mehrerer) Sequenzen errechnet werden, indem die Anzahl an Operationen berechnet wird, um eine Sequenz identisch zu einer anderen Sequenz zu machen. Diese Metrik wird auch als Levenshtein-Distanz oder Editierdistanz bezeichnet [43].

Beispiel:

Gegeben seien die beiden Sequenzen s und t mit
`s=TACCAGTG`
`t=AACCATAGG`.

Gesucht wird nun die minimale Levenshtein-Distanz dieser beiden Sequenzen.
Mit der zuvor definierten Kostenfunktion (Formel 2) beträgt dieser Abstand 4.

Ein mögliches minimales Alignment könnte so aussehen:

```
TACCA_GTG
AACCATAGG
```

Auch andere Kombinationen der Zeichen in den Sequenzen sind möglich, um diesen minimalen Abstand zu erreichen.

z.B.:
```
TACCAGTG_
AACCATAGG
```
oder:
```
TACCAGT__G
AACCA_TAGG
```

Um beide vorgestellte Metriken (Hammingabstand und Levenshteindistanz) auf das Maximierungsproblem anzuwenden, wird die Differenz von Sequenzlänge

zu Distanz gebildet. Je näher der berechnete Wert an der Sequenzlänge ist, desto ähnlicher sind die beiden Sequenzen, desto besser ist das Alignment.

1.2.4 *Dotplot*

Eine Möglichkeit, die Ähnlichkeit zweier Sequenzen graphisch darzustellen, ist die Dotplot-Methode, welche 1970 von Gibbs und Mcintyre unter dem Namen "The Diagram" veröffentlicht wurde [27]. Dabei werden die Übereinstimmungen einzelner Elemente / Zeichen in den Sequenzen in einer Matrix markiert. Zusammenhängende diagonale Abschnitte entsprechen identischen Teilsequenzen innerhalb der beiden Zeichenketten. In dieser Darstellung lassen sich neben den Übereinstimmungen auch sich wiederholende Sequenzabschnitte identifizieren, welche durch übereinstimmende Muster zu erkennen sind (siehe Abbildung 13).

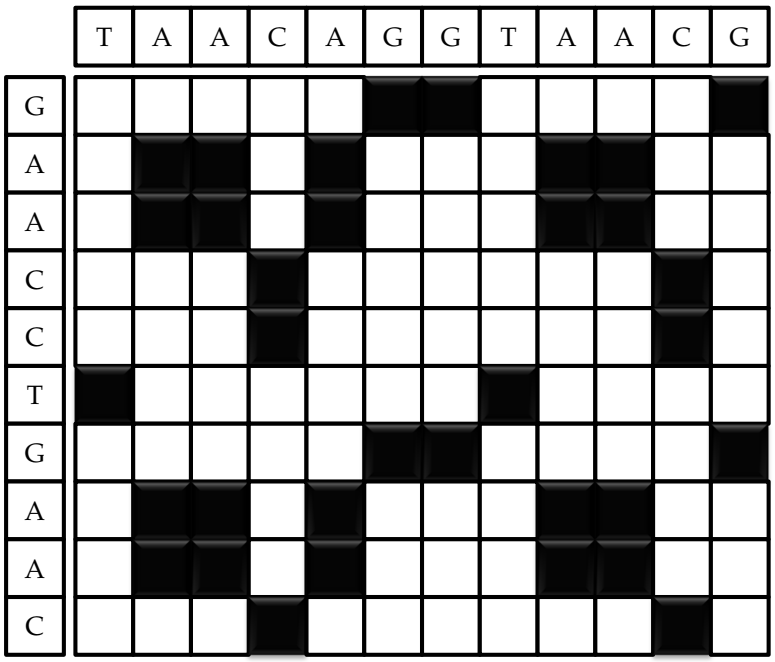


Abbildung 13: Die Dotplot-Matrix der Sequenzen TAACAGGTAACG und GAACCATGAAC. Die schwarz markierten diagonalen Abschnitte zeigen übereinstimmende Teilsequenzen an. Auch sich wiederholende Zeichenblöcke lassen sich erkennen (z.B. die Zeichenkette AAC).

1.2.5 *Scoring-Matrizen*

Im Bereich des DNS- bzw. Protein-Sequenzalignments werden neben den Zeichen, aus denen eine Sequenz besteht, auch die Größe, die Hydrophobie oder andere biochemischen Eigenschaften berücksichtigt. Um diese ebenfalls in einer Metrik zu verrechnen, werden für die Kombination zweier Elemente individuelle Bewertungen benötigt. Findet eine Ersetzung im Protein-Alignment statt, kann ebenfalls erfasst werden, ob eine ersetzte Aminosäure bestimmte biochemische Eigenschaften hat, welche unter Umständen die ursprünglichen Aminosäure ebenfalls hatte oder auch nicht. Beispielsweise sind Isoleucin und Valin beide klein und hydrophob, Serin und Threonin beide polar [13].
Ebenso spielt die Häufigkeit (Wahrscheinlichkeit), mit der eine Aminosäure auftritt, eine Rolle für die Bewertung.
Konservierung der Eigenschaften einer Aminosäure und Frequenz sind die Schlüsselgrößen für die Bewertung [63, 46].
Diese Überlegungen führen zu den Substitutionsmatrizen oder auch Scoring-Matrizen.
Um diese Matrizen im Alignmentprozess zu verwenden, wird die Aufgabe als Maximierungsproblem verstanden, also nicht die minimale Distanz berechnet, sondern die maximale Ähnlichkeit zweier Sequenzen.
Nicht nur für Aminosäuren, auch für die DNS-Basen können Bewertungsmatrizen aufgestellt werden. Eine häufige Form der Substitutionsmatrix ist die Einheitsmatrix (siehe Abbildung 14).

	A	C	G	T
A	1	0	0	0
C	0	1	0	0
G	0	0	1	0
T	0	0	0	1

Abbildung 14: Beispiel einer Einheitssubstitutionsmatrix für die Basen der DNS. Jede Übereinstimmung wird mit 1 bewertet, jeder Kombination unterschiedlicher Buchstaben (Mismatch) mit 0.

Hier werden übereinstimmende Paarungen im Alignment positiv, alle anderen Paarungen von Nukleotiden als neutral bewertet. Dabei wird davon ausgegangen, dass die Mutation bzw. die Veränderungen der einzelnen DNS-Bausteine gleich verteilt und damit gleich wahrscheinlich ist.
Die Matrix lässt sich ebenfalls über das Kronecker-Delta oder Kronecker-Symbol δ schreiben, bei dem übereinstimmende Indizes mit 1 und alle übrigen den Wert

null erhalten (siehe Formel 3).

$$\delta_{ij} = \begin{cases} 0 \text{ , falls } i \neq j \\ 1 \text{ , falls } i = j \end{cases}$$

Formel 3: Definition des Kronecker-Deltas.

Für die komplexere Struktur der Proteine ist diese Einheitsmatrix nicht ausreichend. Sowohl biochemische Prozesse als auch stochastische Betrachtungen erfordern eine detailliertere Aufschlüsselung der Bewertungen [29].

1.2.5.1 *PAM*

In den 1970er Jahren wurde der erste Versuch unternommen, die Wahrscheinlichkeit für eine Änderung einer Aminosäure in einem Alignment zu bestimmen. 1966 trug M. O. Dayhoff ihre Ergebnisse zusammen unter dem Begriff "Point Accepted Mutation" oder "Percent of Accepted Mutations"-Matrix oder kurz PAM-Matrix [18]. Die postulierte Überlegung war, dass beispielsweise die biochemischen Eigenschaften einer Aminosäure bei der Vererbung unangetastet bleiben, womit eine saure Aminosäure eher wieder durch eine saure Aminosäure substituiert wird, als durch eine basische. So wurde ein probabilistisches Modell in Form von Matrizen erstellt, welche durch Markov-Mutationsketten, also einem stochastischen Prozess, für die entsprechenden Positionen und Aminosäuren hergeleitet wurde. Dieses Modell wurde mit Experimenten belegt und die so gewonnenen Daten aufbereitet, um die Mutationswahrscheinlichkeiten der einzelnen Aminosäuren zu erfassen.
Als Basis-Größe wird die sogenannte PAM-Einheit verwendet. Zwei Proteinsequenzen s und t sind eine PAM-Einheit von einander entfernt, wenn sie sich durchschnittlich pro 100 Aminosäuren nur um eine Aminosäure unterscheiden (point mutation). Ebenfalls müssen diese Sequenzen ein funktionierendes Protein erzeugen (accepted mutation). Da eine Stelle im Laufe der Zeit verschiedene Mutationen durchlaufen kann, kann dabei auch die ursprüngliche Aminosäure wieder auftreten. Bei einer Abweichung von 60 PAM-Einheiten stimmen die Sequenzen noch zu ca. 60% überein, bei 120 PAM-Einheiten noch zu ca. 40% [41].
Die Anzahl der beobachteten Mutationen wird mit der Anzahl der zu erwarteten Mutationen ins Verhältnis gesetzt. Es wird also eine Schätzung berechnet, dass in der betrachteten Sequenz nach einer (evolutionären) Zeiteinheit aus Aminosäure A_i Aminosäure A_j wird. Wesentlich für die Werte innerhalb der PAM-Matrizen ist der Grad der Ähnlichkeit der betrachteten Aminosäuresequenzen. Die PAM1-

Matrix gibt an, wie hoch die Wahrscheinlichkeit einer Substitution wäre, wenn 1 Prozent der Aminosäuren mutiert wäre. Eine solch geringe Mutationsrate ist bei sehr nah verwandten Spezies beispielsweise möglich [18] .
Die erste Datenbasis für die Experimente bestand aus 71 Proteinen, welche nur 1572 Mutationen (nur Substitutionen, keine Betrachtung von Insertionen oder Deletionen) enthielten. Alle Sequenzen innerhalb der betrachteten Proteine stimmten zu mindestens 85% überein. Um die PAM1-Matrix zu berechnen, wurden die bereits erstellten Alignments als Grundlage verwendet, jedoch nur die alignierten Sequenzen, welche eine PAM-Einheit entfernt sind. In diesen Sequenzen werden die Frequenzen f (also das Vorkommen) zweier bestimmter Aminosäuren an einer entsprechenden Position gezählt und ins Verhältnis R zur Anzahl der Mutation, dass die eine Aminosäure durch die andere ersetzt wurde, gesetzt:

$$R(A_i, A_j) = \frac{f(A_i \to A_j)}{f(A_i) \cdot f(A_j)}$$

Dieses Verhältnis wird für die entsprechende PAM-Matrix logarithmiert:

$$m(A_i, A_j) = \log_2(R(A_i, A_j)) = \log_2 \frac{f(A_i \to A_j)}{f(A_i) \cdot f(A_j)}$$

Im Idealfall würden so auch die Matrizen berechnet werden, welche für Alignments weiter entfernter Sequenzen verwendet werden, z.B. PAM60 oder PAM250. Jedoch fehlte Dayhoff hier die notwendige Datenbasis mit alignierten Sequenzen, die den entsprechenden PAM-Abstand besitzen. Daher extrapolierte sie ihre Ergebnisse, indem sie - ausgehend von der Hypothese der molekularen Uhr [71], welche besagt, dass die Mutationsrate konstant ist - die initial berechnete PAM1-Matrix X mal mit sich selbst multiplizierte, um die PAMX-Matrix zu erhalten.

$$PAMX = PAM1^X$$

Die errechneten Werte m in den symmetrischen Matrizen können wie folgt interpretiert werden:
- Ist $m(A_i, A_j) = 0$, so entspricht der Wert der statistischen Erwartung, dass A_i zu A_j (und umgekehrt) mutiert.
- Ist $m(A_i, A_j) < 0$, so ist eine Mutation von A_i zu A_j (und umgekehrt) unwahrscheinlich.
- Ist $m(A_i, A_j) > 0$, so ist eine Mutation von A_i zu A_j (und umgekehrt) wahrscheinlich.

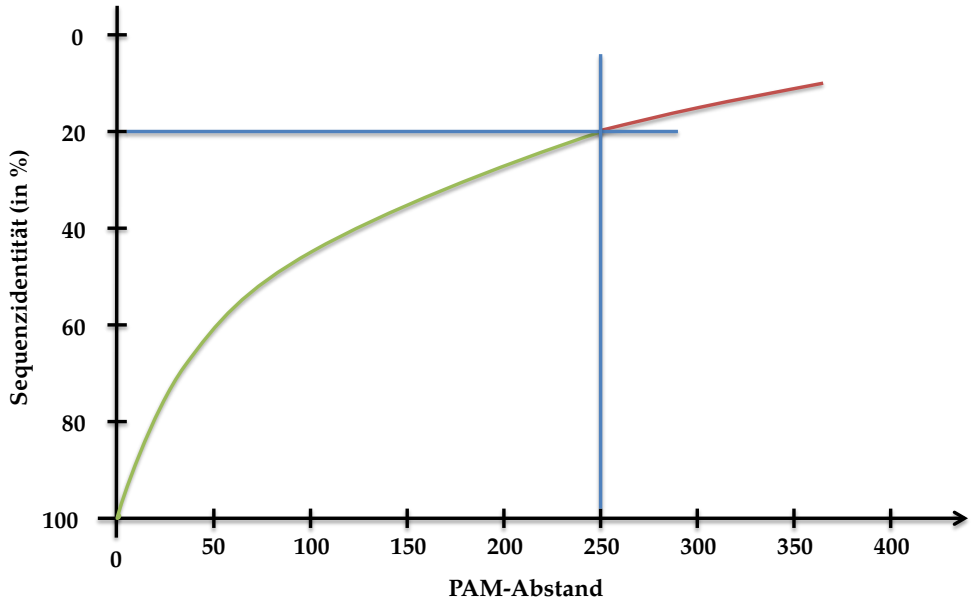


Abbildung 15: Darstellung von Sequenz-Übereinstimmung im Vergleich zum PAM-Abstand. Für Sequenzen mit einem Abstand von über 250 PAM-Einheiten kann nicht mehr von Ähnlichkeit gesprochen werden, da hier die gleiche Übereinstimmung besteht wie bei zufälligem Rauschen.

Publiziert wurden diese Ergebnisse erstmals 1979 im "Atlas of Protein Sequence and Structure" [19]. 1992 wurden die Experimente und Berechnungen mit einer größeren Datenbasis (59190 PAMs in 16130 Sequenzen) wiederholt, um eine feinere Abstimmung in den Matrizen zu ermöglichen [41].
Eine solche Substitutionsmatrix für Aminosäuren ist eine 24×24 große Matrix in der für jede Kombination von zwei Aminosäuren die Bewertung der Alignierung der zwei betroffenen Aminosäuren annotiert ist. Lediglich die letzte Zeile bzw. Spalte repräsentiert keine Aminosäure sondern ein unbekanntes Zeichen (siehe Abbildung 16).

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	J	Z	X
A	2	-2	0	0	-2	0	0	1	-1	-1	-2	-1	-1	-3	1	1	1	-6	-3	0	0	-1	0	-1
R	-2	6	0	-1	-4	1	-1	-3	2	-2	-3	3	0	-4	0	0	-1	2	-4	-2	-1	-3	0	-1
N	0	0	2	2	-4	1	1	0	2	-2	-3	1	-2	-3	0	1	0	-4	-2	-2	2	-3	1	-1
D	0	-1	2	4	-5	2	3	1	1	-2	-4	0	-3	-6	-1	0	0	-7	-4	-2	3	-3	3	-1
C	-2	-4	-4	-5	12	-5	-5	-3	-3	-2	-6	-5	-5	-4	-3	0	-2	-8	0	-2	-4	-5	-5	-1
Q	0	1	1	2	-5	4	2	-1	3	-2	-2	1	-1	-5	0	-1	-1	-5	-4	-2	1	-2	3	-1
E	0	-1	1	3	-5	2	4	0	1	-2	-3	0	-2	-5	-1	0	0	-7	-4	-2	3	-3	3	-1
G	1	-3	0	1	-3	-1	0	5	-2	-3	-4	-2	-3	-5	0	1	0	-7	-5	-1	0	-4	0	-1
H	-1	2	2	1	-3	3	1	-2	6	-2	-2	0	-2	-2	0	-1	-1	-3	0	-2	1	-2	2	-1
I	-1	-2	-2	-2	-2	-2	-2	-3	-2	5	2	-2	2	1	-2	-1	0	-5	-1	4	-2	3	-2	-1
L	-2	-3	-3	-4	-6	-2	-3	-4	-2	2	6	-3	4	2	-3	-3	-2	-2	-1	2	-3	5	-3	-1
K	-1	3	1	0	-5	1	0	-2	0	-2	-3	5	0	-5	-1	0	0	-3	-4	-2	1	-3	0	-1
M	-1	0	-2	-3	-5	-1	-2	-3	-2	2	4	0	6	0	-2	-2	-1	-4	-2	2	-2	3	-2	-1
F	-3	-4	-3	-6	-4	-5	-5	-5	-2	1	2	-5	0	9	-5	-3	-3	0	7	-1	-4	2	-5	-1
P	1	0	0	-1	-3	0	-1	0	0	-2	-3	-1	-2	-5	6	1	0	-6	-5	-1	-1	-2	0	-1
S	1	0	1	0	0	-1	0	1	-1	-1	-3	0	-2	-3	1	2	1	-2	-3	-1	0	-2	0	-1
T	1	-1	0	0	-2	-1	0	0	-1	0	-2	0	-1	-3	0	1	3	-5	-3	0	0	-1	-1	-1
W	-6	2	-4	-7	-8	-5	-7	-7	-3	-5	-2	-3	-4	0	-6	-2	-5	17	0	-6	-5	-3	-6	-1
Y	-3	-4	-2	-4	0	-4	-4	-5	0	-1	-1	-4	-2	7	-5	-3	-3	0	10	-2	-3	-1	-4	-1
V	0	-2	-2	-2	-2	-2	-2	-1	-2	4	2	-2	2	-1	-1	-1	0	-6	-2	4	-2	2	-2	-1
B	0	-1	2	3	-4	1	3	0	1	-2	-3	1	-2	-4	-1	0	0	-5	-3	-2	3	-3	2	-1
J	-1	-3	-3	-3	-5	-2	-3	-4	-2	3	5	-3	3	2	-2	-2	-1	-3	-1	2	-3	5	-2	-1
Z	0	0	1	3	-5	3	3	0	2	-2	-3	0	-2	-5	0	0	-1	-6	-4	-2	2	-2	3	-1
X	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Abbildung 16: Die PAM250-Matrix: Der Wert in den einzelnen Stellen gibt das Verhältnis der Wahrscheinlichkeit an, dass dieses Alignment plausibel ist, zur Wahrscheinlichkeit, dass es zufällig ist, wenn die Sequenzen 250 PAM-Einheiten entfernt sind. Positive Werte bedeuten, dass die Alignierung zweier Aminosäuren wahrscheinlich ist, da sich z.B. die biochemischen Eigenschaften ähneln; ein negativer Wert verdeutlicht, dass die Aminosäuren schlecht zusammen passen. Das X in der letzten Zeile bzw. Spalte steht für ein unbekanntes Zeichen, welches keiner Aminosäure entspricht.

1.2.5.2 *BLOSUM*

Im Jahr 1992 verwendeten S. und J. Henikoff eine ähnliche Methode, um Substitutionsmatrizen, genannt BLOSUM (BLOck SUbstitution Matrix), zu bestimmen. Da für die PAM-Matrizen nur sehr ähnliche Sequenzen als Grundlage verwendet wurden, spiegelt dies nur einen Teil eines möglichen evolutionären Szenarios wieder. Es ist nicht auszuschließen, dass bei relativ unähnlichen Sequenzen andere Mechanismen greifen und sich die Proteinsequenzen anders verändern, als bei nah

verwandten Sequenzen [36]. Als Datenbasis verwendeten Henikoffs ihre BLOCKS-Datenbank [55], in welcher sich ca. 3000 kurze, stark konservierte Subsequenzen, genannt Blöcke (Motifs), von ca. 800 Gruppen verwandter Proteine gespeichert waren. Diese Datenbank wurde aus Ergebnissen von Multi-Sequenzalignments (erstellt mit PROTOMAT, einem Protein-Alignment-Automaten [35]) aufgebaut. Um eine entsprechende Distribution über die Proteine zu erlangen, wurden zu ähnliche Blöcke aus der betrachteten Menge zur Berechnung einer BLOSU-Matrix gelöscht, um so einen entsprechenden (PAM-)Abstand zu erreichen (siehe Abbildungen 17 bis 19).



Abbildung 17: Phase 1: Berechnung eines Multi-Sequenzalignment um stark konservierte Subsequenzen zu finden und zu Blöcken zusammenzufassen

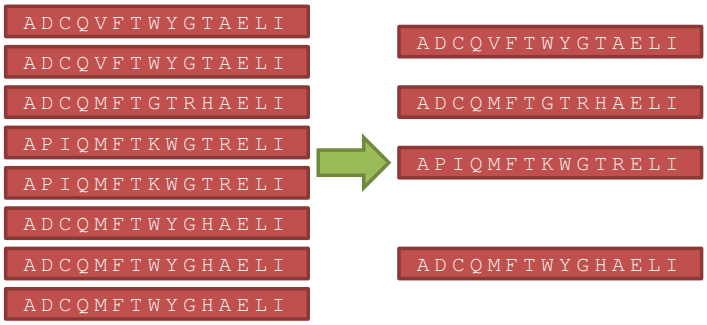


Abbildung 18: Phase 2: Die Blöcke werden analysiert und zu ähnliche Subsequenzen zusammengefasst in Abhängigkeit des festgelegten Schwellenwerts.
(z.B. 62% Übereinstimmung → BLOSUM62)

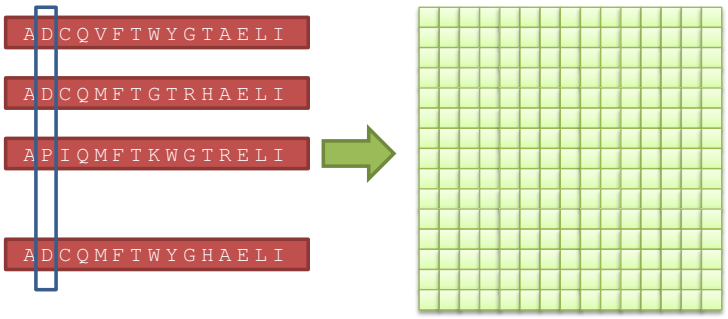


Abbildung 19: Phase 3: Frequenzen und Mutationen werden gezählt und ausgewertet, um die Matrix-Einträge zu bestimmen

Die Kennzahl, welche an die BLOSUM-Bezeichnung annotiert ist, gibt an, wie weit die verwendeten Sequenz-Blöcke maximal übereinstimmen. BLOSUMX bedeutet also höchstens X% Übereinstimmung in den betrachteten Blöcken der Sequenzen. Für die Berechnung der Matrix-Werte wird die gleiche Formel wie bei PAM verwendet:

$$m(A_i, A_j) = \log_2(R(A_i, A_j)) = \log_2 \frac{f(A_i \to A_j)}{f(A_i) \cdot f(A_j)}$$

Auch hier wurden die Ergebnisse später (1999) mit einer größeren Datenbasis (8909 Blöcke aus 1995 Proteinfamilien) verfeinert [37]. Wie bei den PAM-Matrizen haben die BLOSU-Matrizen eine Größe von 24×24 (siehe Abbildung 20)

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

Abbildung 20: BLOSUM62: Die Werte in der Matrix wurden aus Sequenz-Blöcken errechnet, welche maximal zu 62% übereinstimmen. Die Zahlen in den Zellen der Matrix geben an, wie das an der entsprechenden Zeile bzw. Spalte annotierte Aminosäurepaar bewertet wird. In der letzten Zeile bzw. Spalte ist die Bewertung mit einem unbekannten Zeichen X gespeichert.

1.2.5.3 *Vergleich der Scoring-Matrizen*

Dayhoffs PAM-Matrizen werden durch ein Markov-Mutation-Modell berechnet, welches jede Änderung einer Aminosäure unabhängig von der zuvor stattgefundenen Mutation bewertet. Durch die Prädiktion des phylogenetischen Baumes, also durch Vorhersage des Abstammungsbaumes von Nachkommen und gemeinsamen Vorfahren der betrachteten Sequenzen jeder Proteinfamilie, werden die Änderungen, welche als Abweichung in den Proteinketten eines gemeinsamen Vorfahren auftreten, erkannt und bewertet. Dabei werden Matrizen für größere PAM-Abstände extrapoliert von Matrizen mit kleinem evolutionären Abstand (PAM1).

Die Berechnung von BLOSU-Matrizen hingegen basiert auf den beobachteten Veränderungen einer alignierten Region (Block) einer Proteinfamilie, also einem lokalen Alignment und nicht einem globalen. Die unterschiedlichen Abstandsmatrizen werden aus einer Menge an Sequenzen bestimmt, welche nur zu einem gewissen Prozentsatz übereinstimmen, um einen entsprechenden evolutionären Abstand zu simulieren (maximal 62% Übereinstimmung führt zu BLOSUM62).

1.2.6 *Affine Gaps*

Empirische Untersuchungen in der Genetik haben ergeben, dass durch Mutation oder Evolution verursachte Deletionen oder Insertionen in Sequenzen wesentlich häufiger mehrere Zeichen am Stück umfassen, als nur einzelne Positionen [23, 59, 1]. Um die Berechnung für Lücken im Alignment entsprechend anzupassen, wird das Starten oder Öffnen einer Lücke mit einem größeren Malus bewertet und die eigentlichen Lücken mit einem geringeren Malus. Daher wird eine zusammenhängende Lücke über mehrere Positionen günstiger bewertet, als einzelne Lücken in diesem Bereich. Diese Kostenfunktion wird als affine Gap-Kostenfunktion bezeichnet (siehe Formel 4).

$$f(x) = g + k \cdot e$$

Formel 4: Für die Kosten einer Lücke wird die Funktion f verwendet. Das Starten einer Lücke wird mit den Kosten g verrechnet, die Länge k der Lücke wird mit dem Faktor e multipliziert. In der Regel wird $g > e$ gewählt.

1.2.7 *Optimales Alignment*

Die naive Methode, also einfach alle möglichen Alignments zu berechnen und danach das beste Ergebnis zu wählen, ist nur für sehr kurze Sequenzen (Länge ≈ 10) möglich. Alignmentsequenzen von Proteinen liegen jedoch im Bereich von ca. 1.000 bis mehrere 10.000 Aminosäuren.
Für zwei Sequenzen s und t, mit Länge n bzw. m gibt es $0 \leqslant k_1 \leqslant m$ Möglichkeiten für die Anzahl der Lücken in der ersten Sequenz und $0 \leqslant k_2 \leqslant n$ Möglichkeiten für die Anzahl der Lücken in der zweiten Sequenz. Die Länge L des Alignments liegt somit zwischen

$$\max\{n, m\} \leqslant L \leqslant n + m.$$

Um k Lücken mit $0 \leqslant k \leqslant \min\{n, m\}$ im Alignments zu verteilen, gibt es $\binom{n}{k}$ Möglichkeiten in der ersten und $\binom{m}{k}$ Möglichkeiten in der zweiten Sequenz. Somit lässt sich die Anzahl aller möglichen Alignments zweier Sequenzen berechnen mit:

$$\sum_{k=0}^{\min\{n,m\}} \binom{n}{k} \cdot \binom{m}{k} = \binom{n+m}{n}$$

Unter der Annahme, die beiden Sequenzen s und t sind gleich lang, lässt sich über Anwendung der Stirling-Formel folgende Beziehung zur Anzahl Al der Alignments aufstellen:

$$Al = \binom{2n}{n} \approx \frac{2^{2n}}{\sqrt{\pi n}}$$

Somit ergibt sich für eine Sequenzlänge von $n = 10$ bereits eine Anzahl an unterschiedlichen Alignments von $\approx 10^5$. Bei $n = 500$ liegt die Alignmentsanzahl bereits bei $\approx 10^{301}$. Das exponentielle Wachstum der Anzahl möglicher Alignments in Abhängigkeit der Sequenzlänge verhindert, den naiven Weg, alle Möglichkeiten zu berechnen.

1.3 MOTIVATION

Bedingt durch Mutationen reagieren Virenpopulationen unterschiedlich auf Medikamente [62]. Um herauszufinden, ob ein Virenstamm Resistenzen oder Sensibilitäten gegenüber Medikamenten aufweist, hat sich das Testen auf genotypische Resistenzen mittels Sequenzalignment durchgesetzt [21]. Dabei werden die entsprechenden Nukleotid-Sequenzen mit einer Referenz-Protein-Sequenz (auch Wild-Typ-Protein-Sequenz) aligniert. Nicht übereinstimmende Elemente oder Lücken im Alignment werden als Mutationen aufgefasst und mit einer Resistenzdatenbank verglichen. Diese Datenbank beschreibt bereits bekannte Mutationen, welche mit Resistenzen oder Sensibilitäten assoziiert sind. Die Einträge der Datenbank wurden aus unterschiedlichen Publikationen zusammengetragen [10].

Bisher wurde in diesem Verfahren das Alignment mit dem heuristischen Algorithmus Blastx durchgeführt. Blastx ist nicht in der Lage Frameshiftmutation zu erfassen oder Wild-Basen zu verarbeiten. Ebenso kann kein Malus für mögliche Stopcodons vergeben werden.

In der vorliegenden Arbeit wurde ein neuer Algorithmus *Swat* entwickelt, basierend auf dem Prinzip der sogenannten dynamischen Programmierung. Anders als Blastx kann *Swat* zusätzlich Frameshiftmutation verarbeiten, Wild-Basen in den Eingabesequenzen aufschlüsseln und alignieren, sowie für Stopcodons einen speziellen Malus vergeben.

Frameshiftmutation verschieben den Leserahmen und können somit die Translation der nachfolgenden Teilsequenz komplett verändern [26]. *Swat* berücksichtigt diese Mutationen beim Alignment und bewertet sowohl einfache als auch doppelte Frameshifts unabhängig.
Treten beim Sequenzierungsprozess Unsicherheiten auf [56], so dass sich die Base an einer Position nicht exakt bestimmen lässt, werden Platzhalter, sogenannte Wild-Basen, an dieser Position in die Sequenz eingesetzt. Die möglichen Nukleotide, welche durch die Wild-Base repräsentiert werden, werden beim Alignment mit *Swat* durch eine best-case- oder worst-case-Analyse der zu berechnenden Position aufgeschlüsselt.
Stopcodons führen zum vorzeitigen Abbruch der Proteinsynthese und stellen eine sinnentstellende Mutation da (siehe Abschnitt 1.1.1). Um dies in *Swat* zu verarbeiten, ist ein spezieller Malus für Stopcodons in den Algorithmus eingeflossen.

Die Arbeit ist in folgende Kapitel unterteilt:

Kapitel 2 enthält die notwendigen Methoden und vergleichbare Ansätze auf denen der neue Algorithmus basiert. Zusätzlich wird das erste Verfahren zur Analyse der Mutationen und deren Abgleich mit der Resistenzdatenbank beschrieben. Im Anschluss wird ein erweiterte Verfahren für den *Swat*-Algorithmus vorgestellt und seine Implementierung. Abschließend wird der neue Algorithmus in eine generische Webanwendung für die obige Problemstellung eingearbeitet.
Die Simulationsergebnisse zum Vergleich der Güte des *Swat*-Algorithmus gegenüber bestehender Verfahren findet sich **Kapitel 3**. Hierzu wurde eine Testumgebung implementiert, um unterschiedliche Mutationen zu generieren und diese im Anschluss auszuwerten.
In **Kapitel 4** werden die Ergebnisse der Arbeit diskutiert und anschließend in **Kapitel 5** zusammengefasst.

2

METHODEN

2.1 ÜBERSICHT UND TAXONOMIE

Distanzmetriken können in Alignmentalgorithmen umgewandelt werden, um sie als Computerprogramme zu implementiert. Sie lassen sich in zwei Unterklassen einteilen: Algorithmen, welche auf dem Prinzip der dynamischen Programmierung basieren und solche, die Heuristiken aufbauen. Der in dieser Arbeit entwickelte Algorithmus *Swat* gehört zur Unterklasse der dynamischen Programmierung (siehe Abbildung 21).

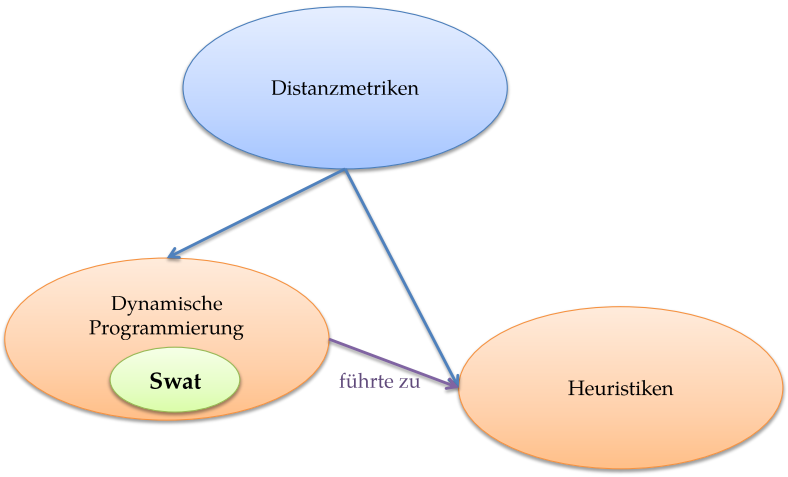


Abbildung 21: Eine grobe schematische Übersicht der Vorgehen, welche für das Sequenzalignment verwendet werden. Ausgehend von Distanzmetriken haben sich Algorithmen entwickelt, welche auf dynamischer Programmierung bzw. auf Heuristiken basieren. Der hier vorgestellte *Swat*-Algorithmus fällt in die Klasse der dynamischen Programmierung.

Alignmentalgorithmen lassen sich in weitere Unterklassen unterteilen, in Abhängigkeit davon, welche Art von Alignment sie berechnen (globales oder lokales Alignment) und auf welchen Algorithmusgrundlagen sie basieren (siehe Abbildung 22).

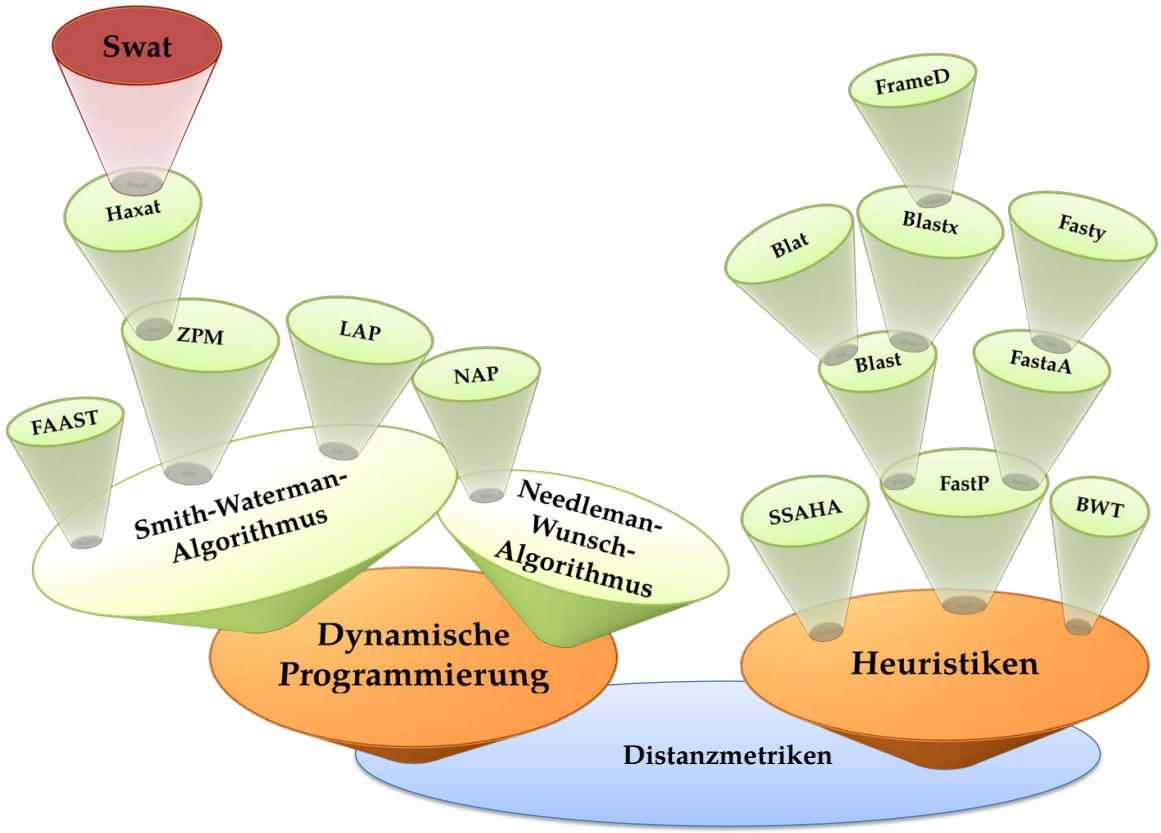


Abbildung 22: Schematische Übersicht der Zusammenhänge einiger Alignmentalgorithmen. Herausgegriffen sind lediglich einige Vertreter zu den verschiedenen Algorithmen. Basieren die Algorithmen auf einem bereits bestehenden Algorithmus, so gehen diese in der Abbildung trichterförmig aus ihnen hervor.

Im Laufe der letzten 45 Jahre sind verschiedene Algorithmen für das Sequenzalignment und Verbesserungen dieser Algorithmen entstanden, sowohl Algorithmen, welche auf dem Prinzip der dynamischen Programmierung basieren, als auch Heu-

ristiken, welche die k-Tupel-Methode als Grundlage nutzen (siehe Abbildung 23).

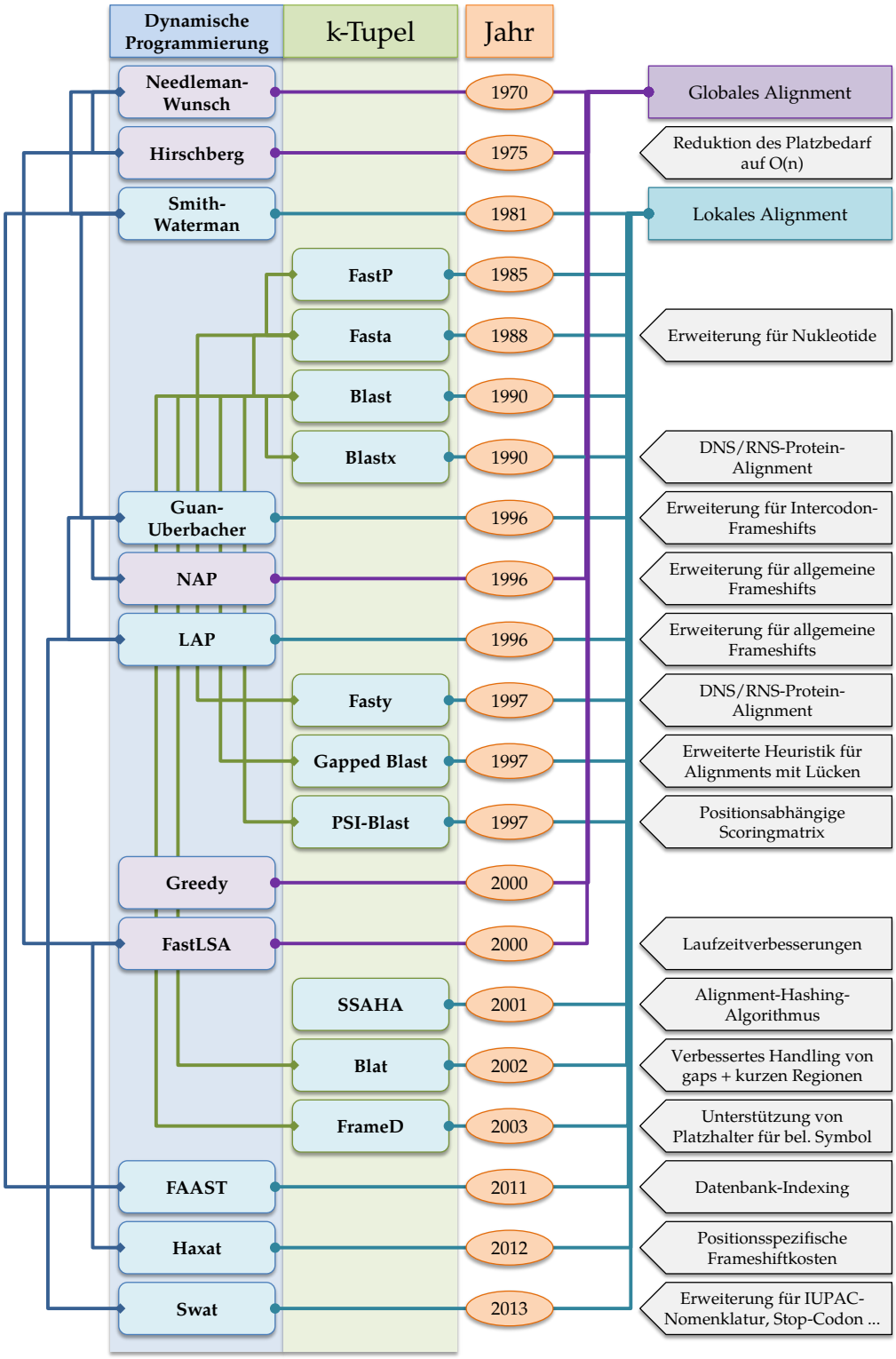


Abbildung 23: Eine Übersicht wichtiger Alignmentprogramme unterteilt in dynamische Programmieransätze und heuristische Verfahren, nach dem Jahr ihrer Veröffentlichung sortiert. Besonderheiten und Erweiterungen wurden rechts annotiert. Die Verzweigungen geben an, auf welchem Algorithmus die jeweiligen Programme basieren.

2.2 DYNAMISCHES PROGRAMMIEREN

In den 1940er Jahren arbeitete Richard Bellman bei der RAND-Corperation (Research ANd Development). Dort war es eine seiner ersten Aufgaben, einen Namen für ein Vorgehen in mehrstufigen Entscheidungsmodellen zu finden, welches nichts mit den - vom damaligen Verteidigungsminister vorverurteilten - Begriffen Forschung oder Mathematik zu tun hatte [4]. So kreierte er den Begriff *dynamisches Programmieren*. Ca. zehn Jahre später wurde *On the Theory of Dynamic Programming, Proceedings of the National Academy of Sciences* (1952) veröffentlicht [3]. Das *Optimalitätsprinzip von Bellman*, welches sich hinter dem dynamischen Programmieren verbirgt, kann so formuliert werden: "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision." (siehe Bellman, 1957, Chap. III.3.) [5]. Das bedeutet, sind die vorangegangenen Teilprobleme bereits optimal gelöst, so kann auf deren Basis eine optimale Gesamtlösung gefunden werden.
Anders wie also der Name vermuten lässt, handelt es sich hier nicht um ein Vorgehen zur Programmierung sondern um ein Prinzip der algorithmischen Problemlösung, bei der eine große Aufgabe in Teilprobleme zerlegt wird. Diese Teilprobleme sind wegen ihrer geringeren Komplexität einfacher zu lösen. Durch optimale Lösung aller Teilprobleme führt dies zu einer optimalen Lösung des Gesamtproblems. Nicht immer kann dieses Prinzip angewendet werden, da das betrachtete Problem diese Zerlegung zulassen muss.
Überlegungen basierend auf diesem Optimalitätsprinzip führten zur Anwendung auf Algorithmen in der Sequenzanalyse. Die Anwendung solcher Verfahren auf diesen Bereich ist möglich, da eine (lange) Sequenz in (kurze) Teilsequenzen zerlegt werden kann. Eine optimale Lösung eines Alignments für eine kurze Teilsequenz führt zu einer optimalen Alignmentlösung für die gesamte Sequenz, ohne, dass Teillösungen erneut berechnet werden müssen. Diese Strategie führt zu einer Verbesserung der Laufzeitkomplexität von exponentiell zu polynomiell und macht damit eine optimale Sequenzanalyse mittels Alignment in vielen Fällen in vertretbarer Zeit möglich.

2.3 GLOBALES UND LOKALES SEQUENZALIGNMENT

Grundsätzlich lassen sich zwei Klassen von Sequenzalignmentalgorithmen unterscheiden: Die Berechnung des lokalen und des globalen Sequenzalignments. Die Variante des lokalen Alignments findet hauptsächlich bei der Erkennung und Bestimmung von ähnlichen kurzen Abschnitten innerhalb von längeren Sequenzen Anwendung, welche zwar in diesen kurzen Abschnitten eine starke Ähnlichkeit

aufweisen, jedoch insgesamt stark unterschiedlich sein können. Die Algorithmen für das globale Alignment hingegen sind darauf abgestimmt, ein optimales Alignment über die komplette Sequenz zu bestimmen.
Wird beispielsweise die vollständige Sequenz eines Proteins betrachtet, so können aus dem globalen Alignment Rückschlüsse auf die evolutionäre Entwicklung einzelner Vertreter einer Proteinfamilie gezogen werden [58].
Andererseits ist zur Identifikation von Genen in langen Amonisäureketten oder auch zur Bestimmung von strukturellen oder funktionellen Abschnitten in Proteinen die Berechnung eines lokalen Alignments die zu bevorzugende Algorithmenklasse [24].

2.3.1 *Needleman-Wunsch*

Der erste Algorithmus, der auf dynamischen Programmieren basiert und für die Berechnung globaler Alignments eingesetzt wird, ist der Needleman-Wunsch Algorithmus. Dieser wurde 1970 vorgestellt [52]. Ohne Mutationswahrscheinlichkeiten oder komplexe Substitutionsmatrizen für biochemisch ähnliche Aminosäuren zu berücksichtigen, ist das Vorgehen wie folgt: Berechnet wird durch einen Maximierungsansatz ein optimales Alignment für eine komplette Sequenz, indem die zuvor berechneten optimalen Alignments für die vorangegangenen Teilsequenzen verwendet werden und um die Bewertung für das nachfolgende Zeichen ergänzt werden. Für zwei Sequenzen s und t wird eine Bewertungsmatrix erzeugt, welche für jede Position in s respektive t den (optimalen) Score S berechnet und diesen Score in die entsprechende Zelle der Matrix annotiert. Die Scoringfunktion m gibt die Bewertungen für übereinstimmende Zeichen (Matches) und sich unterscheidende Zeichen (Mismatches) vor. Für die Alignierungen von Zeichen mit Lücken (Gaps) wird der Malus g verwendet (siehe Formel 5).

$$S[i,j] = \max \begin{cases} S[(i-1),(j-1)] + m(i,j) & \text{, für Match/Mismatch an } i,j \\ S[(i-1),j] + g & \text{, für eine Lücke in } s \\ S[i,(j-1)] + g & \text{, für eine Lücke in } t \end{cases}$$

Formel 5: Rekursive Formel des Needleman-Wunsch-Algorithmus: Für eine Matrixzelle (i,j) ergibt sich der Wert aus den vorangegangen Nachbarzellen (Quellen) $S[(i-1),(j-1)]$, $S[i,(j-1)]$ und $S[(i-1),j]$, sowie die Bewertung m an der Stelle (i,j) bzw. den Kosten für eine Lücke g.

Die Bewertungsmatrix kann so sukzessive für die Teilsequenzen erzeugt werden und um je ein Zeichen ergänzt werden, bis die komplette Sequenz bewertet ist. Initialisiert wird die Matrix an der Stelle $(0,0)$ mit $S[0,0] = 0$, der Bewertung für ein leeres Alignment. Ebenso lassen sich die Werte an den Stellen $S[i,0]$ und $S[0,j]$ bestimmen, da dies einem Alignment mit einer Lücke bzw. einer Lücke der Länge i bzw. j entspricht. Somit gilt: $S[i,0] = i \cdot g$ und $S[0,j] = j \cdot g$ für g den Wert des Malus für eine Lücke. Die weiteren Werte der Matrix lassen sich über die rekursive Formel 5 bestimmen (siehe Abbildung 24).

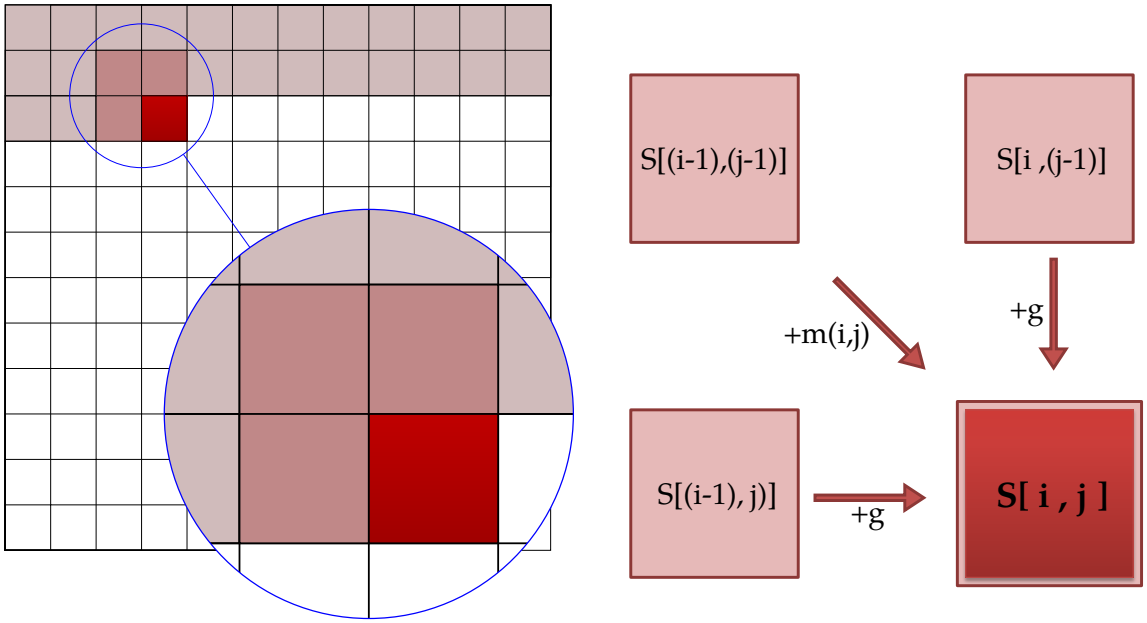


Abbildung 24: Schematische Darstellung des rekursiven Vorgehens bei Needleman-Wunsch: Für die Alignmentbewertung S der Zelle (i,j) wird das Maximum aus den Zellen $S[(i-1),(j-1)]$+ Bewertung m der Zeichen an der Stelle i mit j, $S[i-1,j]$+Malus g einer Lücke und $S[i,(j-1)]$+ Malus g einer Lücke berechnet. Die Bewertung und die Kosten für die Lücke hängen von der verwendeten Kostenfunktion ab.

Für Sequenzen s und t mit Länge n und m findet sich an $S[n,m]$ die Bewertung für ein optimales globales Alignment. Es kann dabei vorkommen, dass die Schritte, die zu diesem optimalen Alignment geführt haben, nicht eindeutig sind. Dies bedeutet, dass es zuvor mehrere Möglichkeiten für ein optimales (Sub-)Alignment gegeben hat und das Alignment mit der besten Bewertung nicht eindeutig ist.

Die folgenden Grafiken zeigen das Vorgehen und die entsprechende Schritte an einem kleinen Beispiel:
Für die Sequenzen s und t wird ein optimales globales Alignment berechnet.

```
s=TACCAGTG
t=AACCATAGG.
```

Zur Bewertung der Matches bzw. Mismatches wird folgende Scoringmatrix verwendet:

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

Abbildung 25: Scoringmatrix für DNS-Sequenzen: Übereinstimmende Basen werden mit $+5$, Mismatches mit -4 bewertet.

Eine Lücke wird mit $g = -2$ bewertet.

Zuerst wird die leere Bewertungsmatrix erzeugt und die beiden Sequenzen s und t elementweise an die entsprechende Zeile bzw. Spalte annotiert (siehe Abbildung 26).

		0	1	2	3	4	5	6	7	8
			T	**A**	**C**	**C**	**A**	**G**	**T**	**G**
0										
1	**A**									
2	**A**									
3	**C**									
4	**C**									
5	**A**									
6	**T**									
7	**A**									
8	**G**									
9	**G**									

(Match: +5 , Mismatch: -4 , Gap: -2)

Abbildung 26: Leere Bewertungsmatrix für die dynamische Programmierung: Die Sequenzen s und t werden in ihre Elemente (Buchstaben / Basen) aufgesplittet und an die entsprechenden Zeile bzw. Spalte annotiert.

Die Bewertung des leeren Alignments wird an Position $(0,0)$ mit 0 gespeichert (siehe Abbildung 27).

		0	1	2	3	4	5	6	7	8
			T	**A**	**C**	**C**	**A**	**G**	**T**	**G**
0		0								
1	**A**									
2	**A**									
3	**C**									
4	**C**									
5	**A**									
6	**T**									
7	**A**									
8	**G**									
9	**G**									

(Match: +5 , Mismatch: -4 , Gap: -2)

Abbildung 27: Initialisierung für Position $(0, 0)$: Das leere Alignment bekommt dem Wert 0.

Im nächsten Schritt werden die Bewertungen der (Teil-)Sequenzen von s und t mit Lücken berechnet und in der Matrix gespeichert. Die Anzahl der Buchstaben wird mit dem Lückenmalus multipliziert und ergibt die Bewertung in der Matrix (siehe Abbildung 28).

		0	1	2	3	4	5	6	7	8
			T	**A**	**C**	**C**	**A**	**G**	**T**	**G**
0		0	-2	-4	-6	-8	-10	-12	-14	-16
1	**A**	-2								
2	**A**	-4								
3	**C**	-6								
4	**C**	-8								
5	**A**	-10								
6	**T**	-12								
7	**A**	-14								
8	**G**	-16								
9	**G**	-18								

(Match: +5 , Mismatch: -4 , Gap: -2)

Abbildung 28: Initialisierung für die Positionen $(i, 0)$ und $(0, j)$: Für i von 1 bis 8 und j von 1 bis 9 berechnet sich die Bewertung des Alignments $S[i, 0] = i \cdot (-2)$ und $S[0, j] = j \cdot (-2)$.

Wird die Zelle (i, j) mit $1 \leqslant i \leqslant n$ und $1 \leqslant j \leqslant m$ berechnet, werden die zuvor bestimmten Ergebnisse $S[(i-1), j]$, $S[i, (j-1)]$ und $S[(i-1), (j-1)]$ benötigt. Von $S[(i-1), j]$ und $S[i, (j-1)]$ wird noch der Malus g für eine Lücke subtrahiert und zu $S[(i-1), (j-1)]$ noch der Wert für einen Match oder Mismatch $m(i, j)$ für die Position hinzu addiert. So wird die Matrix von links oben nach recht unten zeilenweise aufgefüllt. Für die Bewertung der Position $(1, 1)$ wird das Maximum aus den Bewertungen an Position $(0, 0) + m(A, T)$, $(0, 1) + (-2)$ und $(1, 0) + (-2)$ bestimmt. Für diese Position ist das Maximum -4 welches sich in allen drei Möglichkeiten ergibt (siehe Abbildung 29).

		0	1	2	3	4	5	6	7	8
			T	**A**	**C**	**C**	**A**	**G**	**T**	**G**
0		0	-2	-4	-6	-8	-10	-12	-14	-16
1	**A**	-2	-4							
2	**A**	-4								
3	**C**	-6								
4	**C**	-8								
5	**A**	-10								
6	**T**	-12								
7	**A**	-14								
8	**G**	-16								
9	**G**	-18								

(Match: +5 , Mismatch: -4 , Gap: -2)

Abbildung 29: Berechnung des ersten Wertes $S[1,1]$ aus $S[0,0]$, $S[0,1]$ und $S[1,0]$:

$$S[1,1] = \max \begin{cases} S[0,0] + m(A,T) \\ S[0,1] - 2 \\ S[1,0] - 2 \end{cases} = \max \begin{cases} 0-4 \\ -2-2 \\ -2-2 \end{cases} = -4.$$

Für die Stelle $(1,1)$ führen alle drei Möglichkeiten zu einem optimalen Alignment.

Die Berechnung der Bewertungen in der Matrix erfolgt hier zeilenweise von links nach rechts (siehe Abbildung 30).

		0	1	2	3	4	5	6	7	8
			T	**A**	**C**	**C**	**A**	**G**	**T**	**G**
0		0	-2	-4	-6	-8	-10	-12	-14	-16
1	**A**	-2	-4	3	1	-1	-3	-5	-7	-9
2	**A**	-4								
3	**C**	-6								
4	**C**	-8								
5	**A**	-10								
6	**T**	-12								
7	**A**	-14								
8	**G**	-16								
9	**G**	-18								

(Match: +5 , Mismatch: -4 , Gap: -2)

Abbildung 30: Zeilenweise Berechnung der Matrixwerte nach dem rekursiven Schema aus Formel 5: Für jede Position wird die optimale Bewertung in der Matrix notiert. Die Berechnung erfolgt von links nach rechts.

Ist die Matrix vollständig mit den Bewertungen ausgefüllt, findet sich an Position (n, m) die Bewertung des optimalen globalen Alignments. Das Alignment von s und t wird mit $S[8, 9] = 20$ bewertet (siehe Abbildung 31).

		0	1	2	3	4	5	6	7	8
			T	**A**	**C**	**C**	**A**	**G**	**T**	**G**
0		0	-2	-4	-6	-8	-10	-12	-14	-16
1	**A**	-2	**-4**	3	1	-1	-3	-5	-7	-9
2	**A**	-4	-6	**1**	-1	-3	4	2	0	-2
3	**C**	-6	-8	-1	**6**	4	2	0	-2	-4
4	**C**	-8	-10	-3	4	**11**	9	7	5	3
5	**A**	-10	-12	-5	2	9	**16**	**14**	12	10
6	**T**	-12	-5	-7	0	7	14	12	**19**	17
7	**A**	-14	-7	0	-2	5	12	10	**17**	15
8	**G**	-16	-9	-2	-4	3	10	17	**15**	22
9	**G**	-18	-11	-3	-6	1	8	15	13	**20**

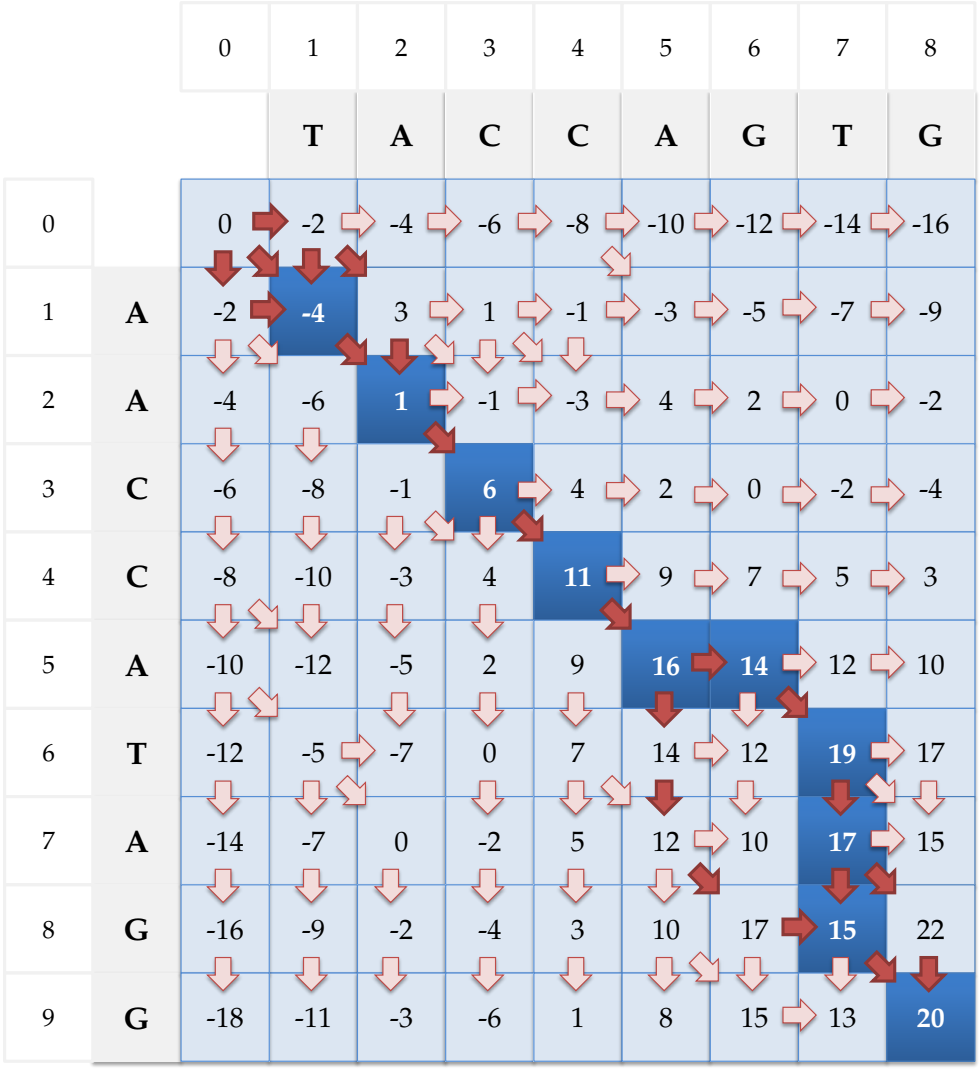
(Match: +5 , Mismatch: -4 , Gap: -2)

Abbildung 31: Vollständige Matrix mit der Bewertung des globalen Alignments an Position $(8, 9)$ mit $S[8, 9] = 20$ und der Markierung einer optimalen (Teil)Sequenz-Alignment-Folge.

Es gibt mehr als einen Weg, um dieses optimale Alignment zu erhalten. Um alle Wege zu finden, welche zu einem optimalen Alignment führen, wird Backtracking verwendet.

2.3.2 *Backtracking*

Die Rückverfolgung (Backtracking) der Schritte, welche zur optimalen Bewertung der jeweiligen (Teil-)Sequenz geführt hat, lässt eine Rekonstruktion der alignierten Sequenzen zu. Es wird von der Endposition der Matrix (n, m) ausgehend untersucht, welche der drei Fälle zum jeweiligen optimalen Alignment geführt haben können. Zuerst an Position (n, m) danach entweder zu $(n, m-1)$, $(n-1, m)$ oder $(n-1, m-1)$, je nachdem, ob eine Lücke in der zweiten Sequenz oder in der ersten Sequenz zum Optimum geführt hat oder das Alignieren von je einem Zeichen aus s und t diese Wert ergeben hat. Dieser Schritt wird für jede Optimum-Position im Alignment durchgeführt, bis zur Stelle $(0, 0)$. Es kann vorkommen, dass an Positionen auch mehr als ein optimaler Weg möglich ist. In diesem Algorithmus gibt es keine Präferenz für den zu verfolgenden Weg, womit alle möglichen optimalen Alignments gleichwertig sind. In späteren Abwandlungen werden zusätzliche Nebenbedingungen eingeführt, welche auch eine Präferenz der möglichen optimalen Backtracking-Schritte mit einschließt [38]. Sind alle Backtracking-Schritte durchgeführt, ergibt sein ein Weg von Position (n, m) der Matrix zur Position $(0, 0)$ (siehe Abbildung 32).



(Match: +5 , Mismatch: -4 , Gap: -2)

Abbildung 32: Hervorgehoben ist ein Beispiel einer möglichen Route durch die Alignmentmatrix. Die hellroten Pfeile geben optimale Wege in der Matrix an. Die dunkelroten Pfeile zeigen alle Wege an, die zu einem optimalen globalen Alignment führen. Der markierte Pfad für ein optimales globales Alignment lautet: $[1,1] \rightarrow [2,2] \rightarrow [3,3] \rightarrow [4,4] \rightarrow [5,5] \rightarrow [5,6] \rightarrow [6,7] \rightarrow [7,7] \rightarrow [8,7] \rightarrow [9,8]$.

Das in Abbildung 32 hervorgehobene Alignment der beiden Sequenzen entspricht diesem:

s_1 =TACCAGT__G
t_1 =AACCA_TAGG

Um alle optimalen Alignments aufzulisten, werden alle möglichen optimalen Pfade abgelaufen und die daraus entstehenden Sequenzen paarweise annotiert. Beispiele weiterer optimaler globaler Alignments sind folgende Sequenzpaare:

s_2 =T_ACCAGT__G
t_2 =_AACCA_TAGG

s_3 =_TACCAGT__G
t_3 =A_ACCA_TAGG

s_4 =TA_CCAGT__G
t_4 =_AACCA_TAGG

s_5 =TA_CCAGT__G
t_5 =_AACCA_TAGG

2.3.3 *Komplexität*

Die Komplexität dieser Verfahren und unter Umständen auch die Anzahl der Lösungen nimmt in Abhängigkeit der Länge der Sequenzen zu. Für zwei Sequenzen s und t der Länge n bzw. m ist die Dimension der Matrix $[n+1][m+1] = O(n \cdot m)$ und kann in der Zeit $O(m \cdot n)$ gefüllt werden, da die Berechnung jeder Zelle in konstanter Zeit ausgeführt werden kann. Die Zeit für das Backtracking beträgt $O(n + m)$. Somit ergibt sich eine Laufzeit von $O(m \cdot n) \approx O(n^2)$ für gleichlange Sequenzen [30].
Aus der zuvor hergeleiteten exponentiellen Laufzeit des naiven Ansatzes, wird ein polynomieller Aufwand.

HIRSCHBERG'S ALGORITHMUS

1975 wurde in den Algorithmus von Needleman und Wunsch eine Optimierung für den Platzbedarf eingearbeitet [38]. Hirschberg stellte ein Teile-und-Herrsche-Verfahren vor, welches den Platzbedarf für die Berechnung eines Alignments von

polynomiell auf linear reduziert. Für zwei Sequenzen s und t, unter der Annahme, dass $|s| > |t|$, wird die Sequenz t in zwei etwa gleich lange Subsequenzen t_1 (vordere Hälfte) und t_2 (hintere Hälfte) zerteilt. Danach wird der Needleman-Wunsch-Algorithmus von links oben nach rechts unten für s und t_1 ausgeführt und von rechts unten nach links oben für s und t_2 (siehe Abbildung 33).

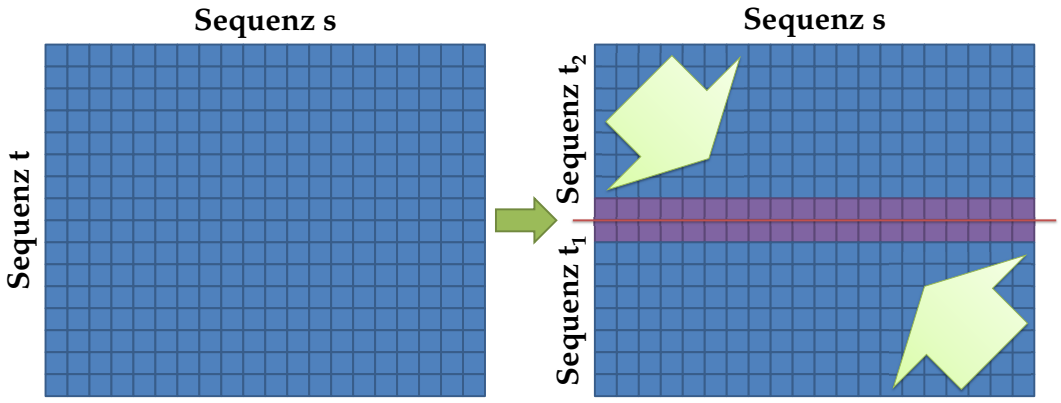


Abbildung 33: Erste Teil-Phase der Hirschberg-Verbesserung für den Needleman-Wunsch-Algorithmus: Es werden die optimalen Bewertungen für die Sequenz s mit den Teilsequenzen t_1 und t_2 berechnet. Dabei erfolgt die Berechnung des optimalen Alignments von s mit t_1 bzw. t_2 von oben nach unten bzw. von unten nach oben (in Pfeilrichtung).

Da die Berechnung der optimalen Teilalignments zeilenweise erfolgt, genügt es, auch für die beiden Teilalignments nur je eine Zeile zu speichern. Nach der ersten Teil-Phase erfolgt eine Suche nach der Stelle, in welcher optimale Bewertungen beider Teilalignments aufeinandertreffen. An dieser Stelle wird dann die Sequenz s aufgeteilt in s_1 und s_2. Danach erfolgt wieder eine Aufteilung der Sequenzen t_1 und t_2 in t_{1_1} und t_{1_2} bzw. t_{2_1} und t_{2_2} sowie eine erneute Berechnung der optimalen Teilalignments für s_1 mit t_{2_1} und t_{2_2} und s_2 mit t_{1_1} und t_{1_2} (siehe Abbildung 34).

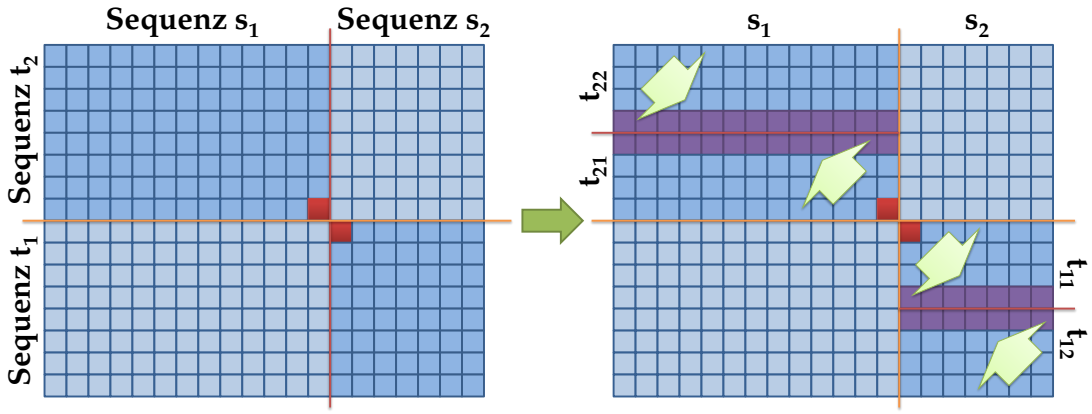


Abbildung 34: Die Position, in der optimale Bewertungen der Teilsequenzalignments aufeinandertreffen, dient als Trenner für die Aufteilung der Sequenz s in s_1 und s_2. Danach wird erneut die Teil-Phase für t_1 und t_2 ausgeführt und erneut in diesen Abschnitten das optimale Teilalignment berechnet. Für s_1 und die t_1-Teilsequenzen, sowie s_2 und die t_2-Teilsequenzen muss kein Alignment erfolgen, da diese Bereiche kein optimales Alignment liefern.

Diese Schritte werden wiederholt durchgeführt, bis alle optimalen zu alignierenden Positionen der jeweiligen Teilsequenzen gefunden sind. Danach werden diese Positionen via Gaps zusammengeführt und bilden damit ein optimales globales Alignment (siehe Abbildung 35).

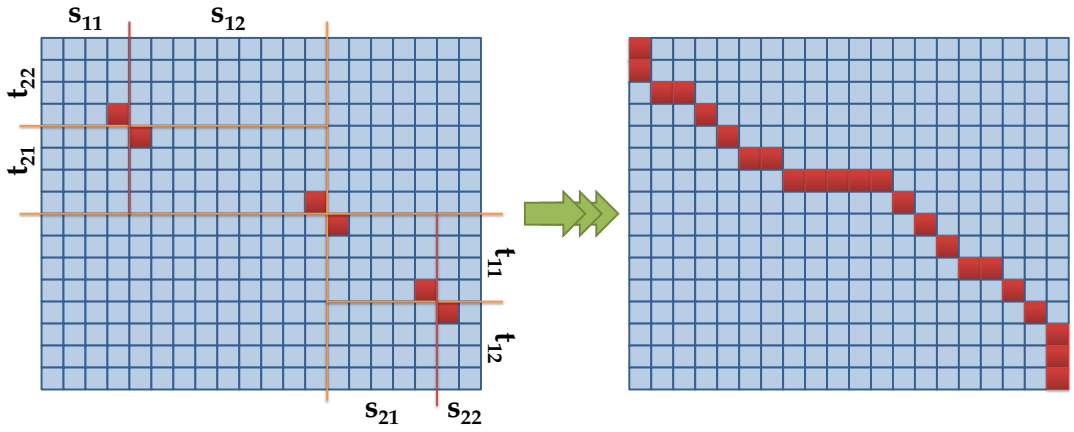


Abbildung 35: Die Teile-und-Herrsche-Schritte werden solange ausgeführt, bis keine Aufteilung der Sequenzen mehr möglich ist. Der gefundene Pfad der optimalen Positionen wird mit Gaps zu einem optimalen globalen Alignment verbunden.

Das beschriebene Vorgehen benötigt für die Alignmentberechnungen in den Teil-Phasen nur den Platz für die Vorgänger-Zeile, $O(n)$, und den Platz für das Speichern je einer Zeile, $O(n)$. Zur Speicherung der optimalen Zellen bzw. deren Positionen wird $O(n+m)$ Platz benötigt. Somit ergibt sich ein Platzbedarf von $O(n)+O(n)+O(n+m) = 2 \cdot O(n) + O(2 \cdot n) = O(n)$.
Für die Zeitkomplexität sind sowohl das rekursive Vorgehen als auch die Teile-und-Herrsche-Schritte maßgebend. In der ersten Iteration wird zweimal jeweils die halbe Matrix berechnet, also $O(2 \cdot \frac{1}{2} \cdot n \cdot m)$. In der zweiten Iteration wird nur noch je ca. ein Viertel berechnet, $O(2 \cdot \frac{1}{4} \cdot n \cdot m)$ usw. Aufsummiert ergibt dies

$$O\left(2 \cdot \left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + ...\right) \cdot n \cdot m\right) = O(2 \cdot n \cdot m) = O(n \cdot m).$$

Somit ergibt sich für zwei ähnlich lange Sequenzen der Länge n wie beim Needleman-Wunsch-Algorithmus eine Laufzeit von $O(n^2)$.

2.3.4 *Smith-Waterman*

1981, also etwa elf Jahre nach der Veröffentlichung des Needleman-Wunsch-Algorithmus, wurde von Smith und Waterman das Problem des optimalen lokalen Alignments mittels dynamischer Programmierung gelöst [61]. Dieses ist dem globalen Alignment in vielen Punkten ähnlich, jedoch wird nicht eine optimale Lösung für die kompletten Sequenzen gesucht, sondern nur noch eine optimale Lösung für

alle möglichen Subsequenzen. Gesucht werden also zwei Teilsequenzen aus s und t, welche das beste Alignment unter allen möglichen Teilsequenzen repräsentierten. Das naive Vorgehen, zu allen Subsequenzen beider Sequenzen je ein optimales globales Alignment zu berechnen und dann das Alignment mit dem höchsten Wert auszuwählen benötigt $O(n^6)$ Zeit: Die Anzahl der möglichen Subsequenzen der Sequenz s ist

$$\sum_{k=1}^{n} k \text{ , für } n \text{ Anzahl der Buchstaben .}$$

Nach der Gaußschen Summenformel ergibt dies

$$\sum_{k=1}^{n} k = \frac{n \cdot (n+1)}{2} = \frac{n^2+n}{2}.$$

Für die Sequenz t mit Länge m ergibt sich

$$\sum_{k=1}^{m} k = \frac{m \cdot (m+1)}{2} = \frac{m^2+m}{2}.$$

Für jedes (Sub-)Sequenzpaar aus s und t wird der Needleman-Wunsch-Algorithmus aufgerufen mit einer Laufzeit von $O(n \cdot m)$. Dies ergibt insgesamt eine Laufzeit von $O(n^3 \cdot m^3)$, für gleich lange Sequenzen $O(n^6)$. Dies ist für n im Bereich von 1.000 bis mehrere 10.000 bereits ein hoher Aufwand ($\approx 10^{18}$ bis 10^{24}).
Durch das Einführen einer neutralen Bewertung 0 für eine Alignmentposition, welche den Beginn eines neuen lokalen Alignments repräsentiert, lässt sich der Needleman-Wunsch-Algorithmus modifizieren, um mit einem Durchlauf das Problem des optimalen lokalen Alignment zu lösen und die Laufzeit auch für den Fall des optimalen lokalen Alignments auf $O(n \cdot m)$ zu reduzieren.
Die zusätzliche Option, ein neues optimales lokales Alignment an jeder Position zu beginnen, führt zu drei Änderungen im Needleman-Wunsch-Algorithmus. Zum einen ändert sich die Initialisierung der Bewertungsmatrix, zum anderen muss die Formel für die rekursive Berechnung der Matrixwerte erweitert werden und zum dritten kann die Bewertung für das optimale lokale Alignment nun überall in der Matrix zu finden sein.
Das leere Alignment $S[0,0]$ hat den Wert 0 und da der Algorithmus für lokale Alignments auch Teil- bzw. Subsequenzen zulässt, kann ein lokales Alignment an jeder Position in s bzw. t starten und auch enden. Daher werden Lücken am Start nicht mit einem Malus bewertet und die Matrix für die Berechnung des optimalen lokalen Alignments wird an $S[0,0]$, $S[0,i]$ mit $1 \leqslant i \leqslant n$ und $S[j,0]$ mit $1 \leqslant j \leqslant m$ mit 0 initialisiert.
Die Formel zur Berechnung des optimalen lokalen Alignments kann analog zum Needleman-Wunsch-Algorithmus rekursiv dargestellt werden (siehe Formel 6).

$$S[i,j] = \max \begin{cases} S[(i-1),(j-1)] + m(i,j) & \text{, für Match / Mismatch an } i,j \\ S[(i-1),j] + g & \text{, für eine Lücke in } s \\ S[i,(j-1)] + g & \text{, für eine Lücke in } t \\ 0 \end{cases}$$

Formel 6: Rekursive Formel des Smith-Waterman-Algorithmus: Für die Berechnung der Matrixbewertung der Zelle (i,j) werden die Werte der Quellen $S[(i-1),(j-1)]$, $S[i,(j-1)]$ und $S[(i-1),j]$ verrechnet. Die Bewertung m an der Stelle (i,j) wird zur Bewertung an $(i-1),(j-1)$ addiert. Zu den anderen beiden Quellen wird der Kostenmalus g für eine Lücke addiert. Die neutrale Bewertung 0 für den Beginn eines neuen lokalen Alignments bildet die vierte Möglichkeit für die maximale Bewertung an der Position (i,j).

Als weitere Option für die Bewertung S einer Zelle wird der (neutrale) Wert 0 eingeführt (siehe Abbildung 36).

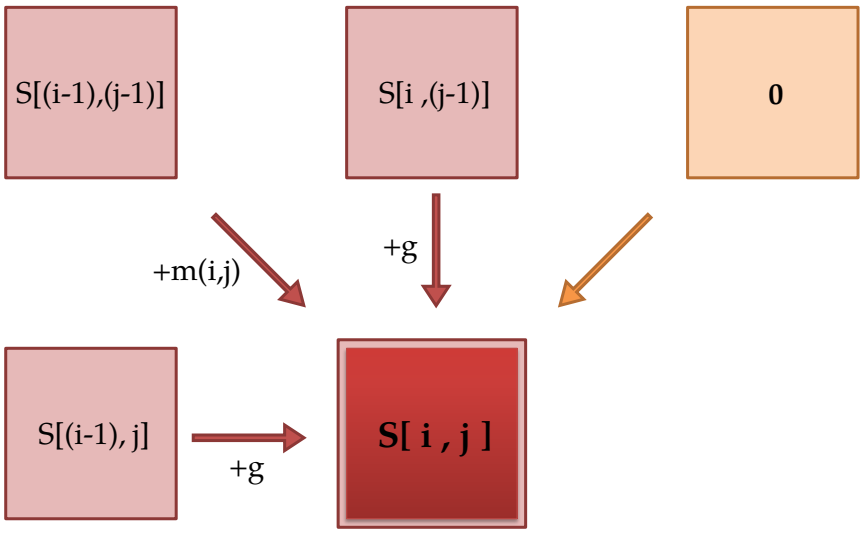


Abbildung 36: Graphische Darstellung des rekursiven Vorgehens bei Smith-Waterman: Analog zum Needleman-Wunsch-Algorithmus ist die Bewertung der gesuchten Zelle $S[i,j]$ das Maximum aus den Bewertungen der Zellen $S[(i-1),(j-1)] + m(i,j)$, der Bewertung der Zeichen an der Stelle i mit j, $S[(i-1),j]$+Lückenmalus, $S[i,(j-1)]$+Lückenmalus und zusätzlich dem neutralen Wert 0.

Für die Berechnung des Maximums einer Zelle bedeutet dies folgendes: Wird ein anfänglich positiv bewertetes Alignment einer Teilsequenz erweitert und würde es durch die Erweiterung negativ bewertet werden, so ist es besser ein neues Alignment zu beginnen, als zu versuchen, einen negativen Wert zu verbessern. Dadurch ergibt sich die Favorisierung der kürzeren lokalen Alignments mit besserer Bewertung in Relation zu den längeren (Teil)Sequenzen mit schlechterer Bewertung. Ein lokales Alignment startet bzw. endet nie mit einer Lücke oder einem Mismatch, so diese Alignierungen mit Kosten in Form eines Malus verbunden sind.
Dieses Vorgehen wird durch folgendes Beispiel illustriert:
Gegeben sind die Sequenzen s und t aus dem Beispiel des Needleman-Wunsch-Algorithmus (siehe Abschnitt 2.3.1):
`s=TACCAGTG`
`t=AACCATAGG`.
Die Scoringmatrix ist wie folgt definiert (analog zum Beispiel aus Abschnitt 2.3.1):

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

Abbildung 37: Scoringmatrix für die DNS-Sequenzen: Übereinstimmende Basen werden mit +5, Mismatches mit -4 bewertet.

Eine Lücke wird mit $g = -2$ bewertet.
Die ersten zwei Schritte sind identisch zum Needleman-Wunsch-Algorithmus. Die leere Bewertungsmatrix wird erstellt und die Elemente der Sequenzen an die entsprechende Zeile bzw. Spalte annotiert (siehe Abbildung 38).

		0	1	2	3	4	5	6	7	8
			T	**A**	**C**	**C**	**A**	**G**	**T**	**G**
0										
1	**A**									
2	**A**									
3	**C**									
4	**C**									
5	**A**									
6	**T**									
7	**A**									
8	**G**									
9	**G**									

(Match: +5 , Mismatch: -4 , Gap: -2)

Abbildung 38: Leere Matrix für die dynamische Programmierung. Die Sequenzen s und t werden in ihre Elemente (Buchstaben / Basen) aufgesplittet und an die entsprechenden Zeilen bzw. Spalten annotiert.

Im Anschluss wird die Position $(0, 0)$ mit dem Wert 0 initialisiert (siehe Abbildung 39).

		0	1	2	3	4	5	6	7	8
			T	**A**	**C**	**C**	**A**	**G**	**T**	**G**
0		0								
1	**A**									
2	**A**									
3	**C**									
4	**C**									
5	**A**									
6	**T**									
7	**A**									
8	**G**									
9	**G**									

(Match: +5 , Mismatch: -4 , Gap: -2)

Abbildung 39: Initialisierung für Position $(0, 0)$. Das leere Alignment bekommt dem Wert 0.

Danach werden die Alignments des ersten Zeichens der entsprechenden Teilsequenzen initial ebenfalls mit 0 bewertet, da so der Beginn eines neuen lokalen Alignments bewertet wird (siehe Abbildung 40).

		0	1	2	3	4	5	6	7	8
			T	**A**	**C**	**C**	**A**	**G**	**T**	**G**
0		0	0	0	0	0	0	0	0	0
1	**A**	0								
2	**A**	0								
3	**C**	0								
4	**C**	0								
5	**A**	0								
6	**T**	0								
7	**A**	0								
8	**G**	0								
9	**G**	0								

(Match: +5 , Mismatch: -4 , Gap: -2)

Abbildung 40: Initialisierung für die Positionen $(i, 0)$ und $(0, j)$ für i von 1 bis 8 und j von 1 bis 9 mit $S[i, 0] = 0$ und $S[0, j] = 0$.

Wird eine Zelle (i, j) mit $1 \leqslant i \leqslant n$ und $1 \leqslant j \leqslant m$ berechnet, werden die zuvor ermittelten Ergebnisse von $(i-1, j)$, $(i, j-1)$ und $(i-1, j-1)$ benötigt. Von $S[(i-1), j]$ und $S[i, (j-1)]$ wird noch der Malus g für eine Lücke subtrahiert und zu $S[(i-1), (j-1)]$ noch der Wert für einen Match oder Mismatch $m(i, j)$ für die Position hinzu addiert oder subtrahiert. Zusätzlich wird die Option für den Start eines neuen lokalen Alignments (Bewertung 0) berücksichtigt. Das Maximum der vier Möglichkeiten wird berechnet und in der Matrix annotiert (siehe Abbildung 41).

		0	1	2	3	4	5	6	7	8
			T	**A**	**C**	**C**	**A**	**G**	**T**	**G**
0		0	0	0	0	0	0	0	0	0
1	**A**	0	0							
2	**A**	0								
3	**C**	0								
4	**C**	0								
5	**A**	0								
6	**T**	0								
7	**A**	0								
8	**G**	0								
9	**G**	0								

(Match: +5 , Mismatch: -4 , Gap: -2)

Abbildung 41: Berechnung des ersten Wertes $S[1,1]$ aus $S[0,0]$, $S[0,1]$, $S[1,0]$ und 0:

$$S[1,1] = \max \begin{cases} S[0,0] + m(A,T) \\ S[0,1] - 2 \\ S[1,0] - 2 \\ 0 \end{cases} = \max \begin{cases} 0-4 \\ -2-2 \\ -2-2 \\ 0 \end{cases} = 0.$$

Die Stelle $(1,1)$ erhält die optimale Bewertung durch den Beginn eines neuen lokalen Alignments. Die anderen Möglichkeiten (mit einer Lücke oder einem Mismatch) zu starten werden schlechter bewertet.

Die Matrix wird zeilenweise von links nach rechts ausgefüllt (siehe Abbildung 42).

		0	1	2	3	4	5	6	7	8
			T	**A**	**C**	**C**	**A**	**G**	**T**	**G**
0		0	0	0	0	0	0	0	0	0
1	**A**	0	0	5	3	1	0	0	0	0
2	**A**	0								
3	**C**	0								
4	**C**	0								
5	**A**	0								
6	**T**	0								
7	**A**	0								
8	**G**	0								
9	**G**	0								

(Match: +5 , Mismatch: -4 , Gap: -2)

Abbildung 42: Zeilenweise Berechnung der Matrixwerte nach der rekursiven Formel 6 des Smith-Waterman-Algorithmus: Für jede Position wird die optimale Bewertung in der Matrix annotiert. Ist der Beginn einen lokales Alignments besser, als die Fortsetzung, so ergibt sich die 0 (orange markiert). Die dunkelblauen Zellen geben die Bewertungen eines optimalen lokalen Alignments für die jeweiligen Teilsequenzen vor.

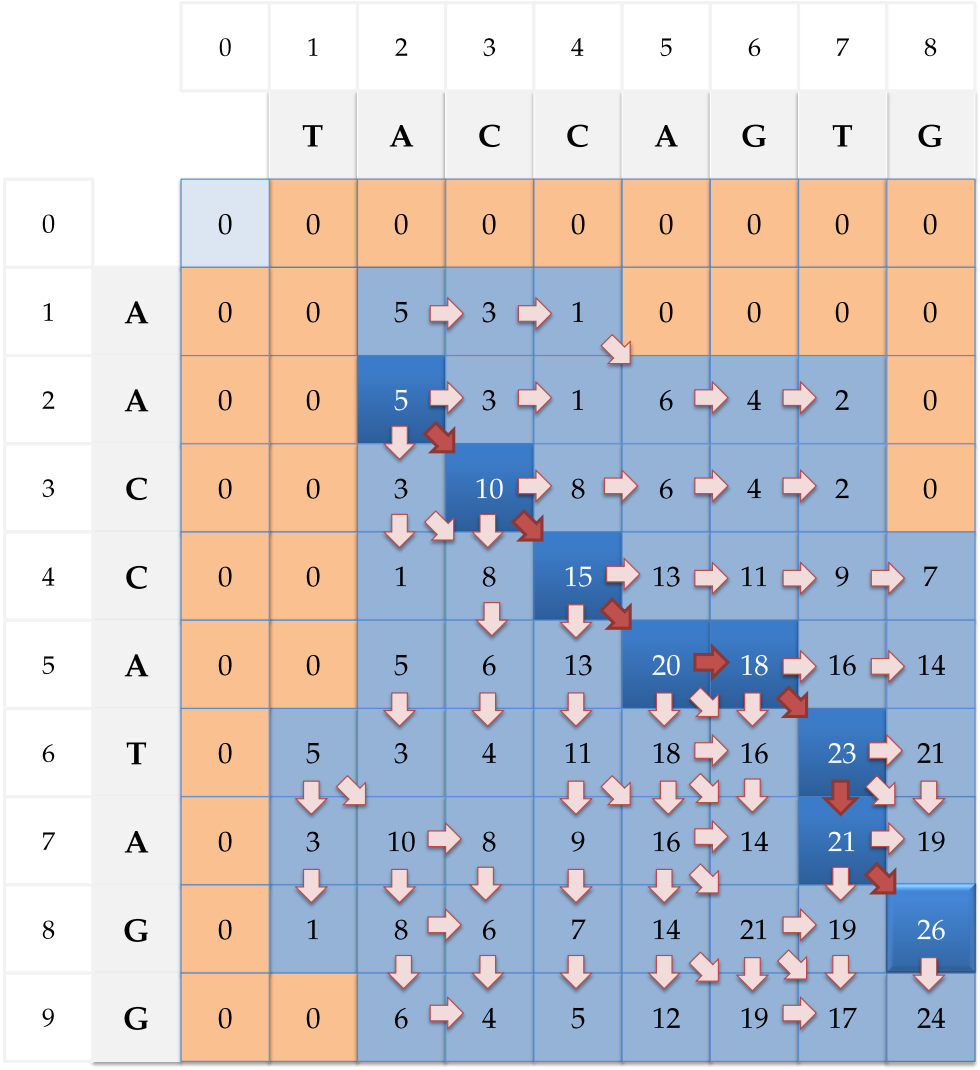
Für Sequenzen s und t mit den Längen $n = 8$ und $m = 9$ findet sich die höchste Bewertung an $(8, 8)$ mit $S[8, 8] = 26$, die Bewertung für ein optimales lokales Alignment. Diese Bewertung wird auch Smith-Waterman-Bewertung oder Smith-Waterman-Score genannt (siehe Abbildung 43).

		0	1	2	3	4	5	6	7	8
			T	A	C	C	A	G	T	G
0		0	0	0	0	0	0	0	0	0
1	A	0	0	5	3	1	0	0	0	0
2	A	0	0	5	3	1	6	4	2	0
3	C	0	0	3	10	8	6	4	2	0
4	C	0	0	1	8	15	13	11	9	7
5	A	0	0	5	6	13	20	18	16	14
6	T	0	5	3	4	11	18	16	23	21
7	A	0	3	10	8	9	16	14	21	19
8	G	0	1	8	6	7	14	21	19	26
9	G	0	0	6	4	5	12	19	17	24

(Match: +5 , Mismatch: -4 , Gap: -2)

Abbildung 43: Vollständige Matrix mit der Bewertung des optimalen lokalen Alignments an Position $S[8,8]$ mit 26 und den mit hellroten Pfeilen markierten Wegen, welche zu den jeweiligen optimalen Bewertungen der einzelnen Matrixpositionen geführt haben.

Um den Weg und damit das Alignment der Sequenzen nachvollziehen zu können wird wieder das Backtracking eingesetzt. Es ergibt sich hierbei der Weg $[2,2] \rightarrow [3,3] \rightarrow [4,4] \rightarrow [5,5] \rightarrow [5,6] \rightarrow [6,7] \rightarrow [7,7] \rightarrow [8,8]$ (siehe Abbildung 44).



(Match: +5 , Mismatch: -4 , Gap: -2)

Abbildung 44: Darstellung der Bewertungsmatrix mit den Markierungen der optimalen Wege. Mit dunkelroten Pfeilen ist der Weg zum optimalen lokalen Alignment gekennzeichnet: $[2,2] \rightarrow [3,3] \rightarrow [4,4] \rightarrow [5,5] \rightarrow [5,6] \rightarrow [6,7] \rightarrow [7,7] \rightarrow [8,8]$.

Das optimale lokale Alignment der beiden Sequenzen lautet:
s_1 =ACCAGT_G
t_1 =ACCA_TAG

Diese beiden Algorithmen, basierend auf dem Prinzip der Optimalität nach Bellman, also dem Prinzip der dynamischen Programmierung, bilden die Basis für vie-

le Abwandlungen und Variationen von Sequenzalignmentalgorithmen und deren Implementierungen, angefangen mit der ersten Variation für Sequenzdatenbankalignments und deren Umsetzung, dem Fasta-Algorithmus (fast all) [53], dem sehr populären und verbreiteten Blast-Werkzeugkasten (Basic Local Alignment Search Tool) [2], bis hin zu neuen Programmen wie Haxat (Homopolymer Aware Cross Alignment Tool) [47] und dem in dieser Arbeit entwickelten *Swat*-Algorithmus (Smith Waterman extended Alignment Tool).

2.4 LOKALE DATENBANK-ALIGNMENTS

Der Needleman-Wunsch- und auch der Smith-Waterman-Algorithmus finden das beste (globale oder lokale) paarweise Alignment auf Zeichensequenzen und lassen sich auch auf DNS- bzw. Proteinsequenzen anwenden. Hierbei werden sowohl $O(n^2)$ Zeiteinheiten und auch $O(n^2)$ Platz benötigt, um die Berechnung durchzuführen. Ist nun zu einer Sequenz eine Zweite, möglichst ähnliche, aus einer Menge von anderen Sequenzen gesucht, so ist ein Abgleich mit der kompletten Menge an Sequenzen notwendig. Um den Platzbedarf und auch die benötigten Zeiteinheiten zu reduzieren, wurden heuristische Algorithmen entwickelt, um das Alignment einer Sequenz mit einer zweiten Sequenz aus einer großen Sequenzdatenbank zu beschleunigen. Diese sind unter dem Begriff "Wort-Methode" oder auch "k-Tupel-Methode" bekannt [45]. Einsatzgebiete hierfür sind beispielsweise evolutionäre Abstammungsbäume oder auch Sequenz-Profiling [60]. Dieses heuristische Vorgehen, welches nicht garantieren kann, das beste Ergebnis zu finden, ist im Vergleich zum wiederholten Anwenden des Smith-Waterman-Algorithmus zeiteffizienter [9, 49]. Alle auf diesem Vorgehen basierenden Algorithmen verarbeiten den Sequenz-Pool vor, aus welchem die beste zur untersuchten Zeichenkette passende Sequenz gesucht wird. Alle Pool-Sequenzen werden in kleinere Teilsequenzen zerlegt, über ein komprimierendes Verfahren in einer Datenbank abgelegt und danach für den Alignmentprozess zur Verfügung gestellt. Die Verfahren nutzen Hashtabellen bzw. lookup-Tabellen, um die Sequenzen aufzubereiten [53, 2] oder nutzen die Burrows-Wheeler-Transformation, um Suffixbäume bzw. Suffixarrays zu erstellen [69]. Diese Indexstrukturen wenden mathematische Funktionen an, um die Daten so umzukodieren, dass ein schneller Zugriff auf die einzelnen Elemente möglich ist bzw. dass das Ablegen der Daten mit einer Komprimierung verbunden ist, welches die Zugriffe beschleunigt. Die Datenstruktur der Hashtabelle beispielsweise ermöglicht einen parallelen Suchzugriff und jeder Zugriff kann (im Idealfall) in konstanter Zeit $O(1)$ verarbeitet werden ([42]). Damit lassen sich für das Sequenzalignment übereinstimmende Teilsequenzen in den verschiedenen Sequenzen effizient finden. Dieses Vorgehen wird im Fasta-Algorithmus [53] als erster Schritt eingesetzt, um potenzielle gute Sequenzen zu einer vorgegebene Sequenz zu finden. Eine leichte

Variation hiervon findet sich im Blast-Algorithmus wieder [2]. Dort ist das Auffinden der Teilsequenzen etwas relaxierter, da hier auch stark ähnliche Teilsequenzen erlaubt sind und nicht exaktes Übereinstimmen Voraussetzung ist.

2.4.1 *Fasta*

Lipman und Pearson entwickelten auf der Basis des beschriebenen heuristischen Vorgehens einen Alignment-Algorithmus [45]. Der 1985 veröffentliche Algorithmus trug den Namen FastP, welche eine schnelle (fast) Suche auf einer Protein-Datenbank (P) ermöglichte. Dieser Algorithmus wurde von beiden zum Fasta-Algorithmus weiterentwickelt, um ebenfalls Nukleotidsequenzen zu verarbeiten [53]. In diesem Zusammenhang wurde das erste Fasta-Werkzeugpaket geschnürt, welches bis heute erweitert und verbessert wird.
Der Fasta-Algorithmus kann in die vier Schritte Lookup, Rescoring, Expansion und Alignment zerlegt werden.
Zuerst werden exakt übereinstimmende Teilsequenzen gesucht und diese werden dann, wenn sie auf der selben Diagonalen nah beieinander liegen, zu Regionen zusammengefasst. Im Anschluss werden die gefundenen Regionen mit einer Scoringmatrix bewertet und die am höchsten bewertete Region, auch mit $init_1$ bezeichnet, wird als Ausgangspunkt verwendet, um weitere Regionen in anderen Diagonalen mit dieser zusammenzufassen (bezeichnet mit $init_n$). Dies entspricht dem Einfügen von Lücken in ein Alignment. Im letzten Schritt wird dann im Bereich der zusammengefassten Regionen ein Alignment mit dem Smith-Waterman-Algorithmus vorgenommen, um das beste Alignment über den Bereich zu finden.

2.4.1.1 *Fasta-Algorithmus*

Formal lässt sich der Fasta-Algorithmus wie folgt beschreiben:

1. Lookup: Bestimme die Menge "hot spots" H aller Teilsequenzen der Länge k mit $s_1[i, i+k] = s_x[j, j+k]$. Ist $|f(h_i), f(h_j)| < t$ mit $h_i, h_j \in H$, $f(x)$ eine Bewertungsfunktion für eine Teilsequenz und Schwellwert t, fasse h_i und h_j zusammen zur Region h_{ij} und ersetze h_i und h_j in H durch h_{ij}. Wiederhole diesen Schritt, bis H sich nicht mehr ändert.
2. Rescoring: Bewerte jede Region $h_i \in H$ mit einer Substitutionsmatrix M: $S(h_i) = \sum_{z=1}^{|h_i|} m(s_z, t_z)$ mit $m(x, y)$ dem Eintrag in M für x und y. Die Region mit der höchsten Bewertung wird $init_1$ genannt.

3. Expansion: Ist $h_a + h_b - l(g) < k$ mit h_a und h_b aus der Menge H, $l(x)$ eine Lückenbewertungsfunktion für den Abstand g zwischen h_a und h_b und Schwellwert k, so fasse diese Regionen zusammen. Diese Menge wird als $init_n$ bezeichnet.

4. Alignment: Berechne optimales Alignment im Abstand d um $init_1$, also im Bereich von $init_1 - d$ bis $init_1 + d$

Anschaulicher bedeutet dies:

1. Lookup: Es werden identische Teilsequenzen gesucht, welche mindestens eine Länge von k Zeichen haben, also $s_1[i, i+k] = s_x[j, j+k]$ für eine Eingabesequenz s_1 und einer Sequenz s_x aus der Datenbank. Diese Teilsequenzen werden als "hot spots" bezeichnet. Hot spots, welche auf einer Diagonalen nah beieinander liegen, können sukzessive zu Regionen zusammengefasst werden, so ihre gemeinsame Bewertung einen Schwellenwert nicht unterschreitet (siehe Abbildung 45).

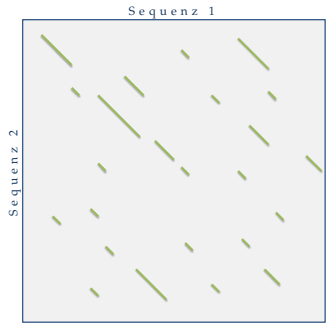


Abbildung 45: Erster Schritt im Fasta-Algorithmus: Lookup. Identische Teilsequenzen werden ermittelt und als hot spots markiert. Nahe beieinander liegende hot spots auf der gleichen Diagonalen in der Matrix werden zu Regionen, also Abschnitten mit kurzen Bereichen von Mismatches, zusammengefasst, so sie einen Schwellwert nicht unterschreiten.

2. Rescoring: Wenn diese Verarbeitung abgeschlossen ist, werden die Regionen über eine Scoringmatrix (inklusive Mismatches) bewertet. Die Region mit der höchsten Bewertung wird als $init_1$ bezeichnet (siehe Abbildung 46).

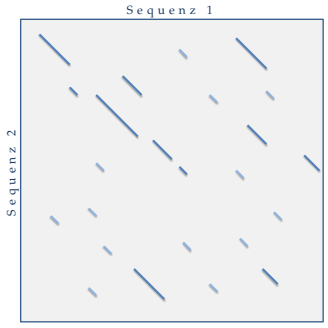


Abbildung 46: Zweiter Schritt im Fasta-Algorithmus: Rescoring. Mit Hilfe einer Substitutionsmatrix werden die Regionen aus dem ersten Schritt bewertet. Dabei wird die Region mit der höchsten Bewertung mit $init_1$ bezeichnet.

3. Expansion: Diese $init_1$-Region wird nun mit Hilfe einer Lückenbewertungsfunktion um andere Regionen erweitert, so auch hier ein Schwellwert nicht unterschritten wird. Die Bewertung der besten Menge von Regionen wird als $init_n$ bezeichnet (siehe Abbildung 47).

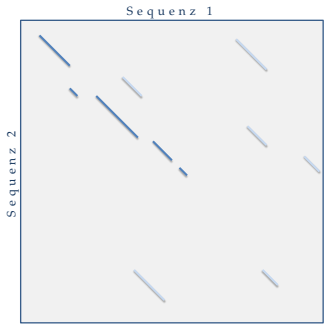


Abbildung 47: Dritter Schritt im Fasta-Algorithmus: Expansion. Regionen, welche zwar auf unterschiedlichen Diagonalen, jedoch nicht zu weit entfernt von einander liegen, werden ebenfalls zusammengefasst, um einen initialen Bereich für das Alignment zu bilden.

4. Alignment: Die $init_n$-Bewertungen werden verglichen und das beste Paar ausgewählt, um eine Abwandlung des Smith-Waterman-Algorithmus zu durchlaufen. Um den Bereich von $init_1$ wird ein Alignment berechnet. Dieses wird in einem Band der Breite d Zeichen um $init_1$, also $2 \cdot d$ Diagonalen, ermittelt. Dieses finale Alignment wird dann als Ergebnis des Algorithmus ausgegeben (siehe Abbildung

48).

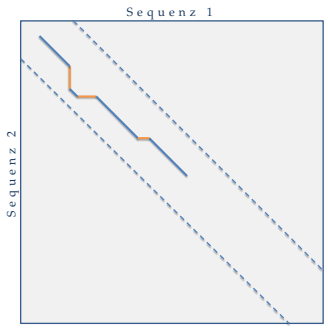


Abbildung 48: Vierter Schritt im Fasta-Algorithmus: Alignment. Der gefundene Bereich wird in einem Band einer Auswertung durch eine Variante des Smith-Waterman-Algorithmus unterzogen, welche ein optimales Alignment für diesen Teil der Matrix berechnet.

2.4.2 *Blast*

Altschul et al. haben 1990 einen weiteren Algorithmus veröffentlicht, welcher ebenfalls einen heuristischen Prozess modelliert. Der Blast-Algorithmus (Basic Local Alignment Search Tool) wird zur Suche in Datenbanken nach ähnlichen, homologen Sequenzen verwendet [2]. Auch dieser Algorithmus besteht aus vier Schritten: Bestimmung und Bewertung von sehr ähnlichen oder identischen Teilsequenzen, Erweiterung und Zusammenfassung der gefundenen Teilsequenzen mit der besten Bewertung und schließlich eine Alignierung der Sequenz mit Sequenzen aus der Datenbank welche die Abschnitte mit der höchsten Bewertung enthalten.

2.4.2.1 *Blast-Algorithmus*

Die vier Schritte des Blast-Algorithmus lassen sich formal beschreiben:

1. Seeding: Bestimme die Menge "seeds" H aller stark ähnlicher oder identischer Teilsequenzen der Länge k mit $s_1[i, i+k] \approx s_x[j, j+k]$.
2. Scoring: Bewerte die gefundenen seeds h_i der Menge H mit einer Scoring-matrix M: $S(h_i) = \sum_{z=1}^{|h_i|} m(s_z, t_z)$ mit $m(x, y)$ dem Eintrag in M für x und y.

Solange die Bewertung einen Schwellwert übersteigt, werden diese seeds im weiteren Prozess berücksichtigt.

3. Extension: Modifiziere $h_i \in H$ zu HSPs (high scoring pairs) $\forall g(h_i) > l$ mit $1 < i < |H|$ und $g(x)$ eine Bewertungsfunktion um $\sigma_1 \pm h_i \pm \sigma_2$, sodass h_i keine bessere Bewertung erreicht, durch Hinzufügen oder Entfernen von Zeichen. Die Regionen mit der höchsten Bewertung werden als maximal segment pairs (MSP) bezeichnet.

4. Alignment: Berechne ein optimales Alignment um die maximum scoring pairs.

Anschaulicher bedeutet dies:

1. Seeding: Ähnlich wie Fasta betrachtet Blast zuerst kurze Zeichenketten innerhalb der Eingabesequenz und vorverarbeiteten Datenbanksequenzen und erstellt sogenannte Seeds. Diese Seeds unterscheiden sich jedoch von den hot spots des Fasta-Vorgehens, da zwar die Länge dieser Seeds wie bei Fasta gewählt wird, jedoch werden hier auch ähnliche und nicht nur identische Zeichenketten betrachtet (siehe Abbildung 49).

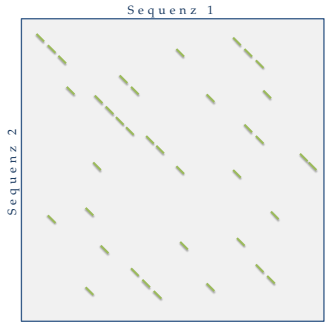


Abbildung 49: Erster Schritt im Blast-Algorithmus: Seeding. Finde alle (kurzen) Teilsequenzen, welche stark ähnlich oder identisch sind.

2. Scoring: Die gefundenen seeds werden mit einer Substitutionsmatrix bewertet (ala BLOSUM62 oder PAM250 etc.). Solange bei der Bewertung der seeds eine Schwelle nicht unterschritten wird, wird dieser Bereich in den folgenden Schritten berücksichtigt (siehe Abbildung 50).

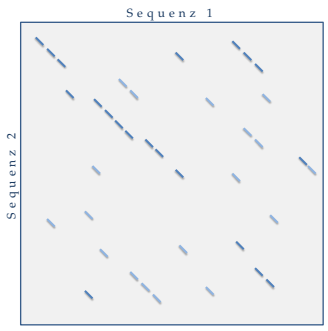


Abbildung 50: Zweiter Schritt im Blast-Algorithmus: Scoring. Bewerte die gefundenen Seeds mit einer Substitutionsmatrix und filtere die Positionen heraus, welche einen zu geringen Wert aufweisen.

3. Extension: Der nächste Schritt erinnert an die Expansion im Fasta-Algorithmus: Hoch bewertete Abschnitte (high scoring pairs) werden nach vorne und hinten um Zeichen erweitert oder verkürzt, bis die Bewertung dieser Teilsequenz nicht mehr verbessert wird. Es entstehen unterschiedlich lange Teilsequenzen, welche zwar Mismatches zulassen, aber keine Lücken (analog zu $\mathtt{init}_1$). Als weiterer Teil dieser Expansion werden erneut die Strategien von Fasta benutzt. Sind zwei HSPs weniger als einen bestimmten Wert auseinander, so werden sie zu MSPs (maximal scoring pairs) zusammengefasst (analog zu $\mathtt{init}_n$) (siehe Abbildung 51).

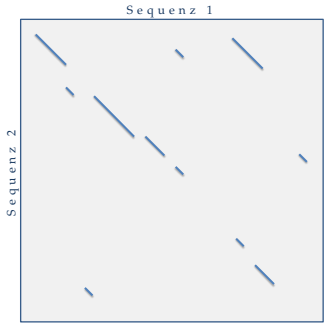


Abbildung 51: Dritter Schritt im Blast-Algorithmus: Extension. Modifiziere die gefundenen high scoring pairs, um deren Bewertung zu optimieren und filtere die besten Abschnitte heraus

4. Alignment: Schließlich werden alle MSPs mit den entsprechenden alignierten Sequenzen der Datenbank verglichen und das beste Ergebnis wird als optimales lokales Alignment verwendet (siehe Abbildung 52).

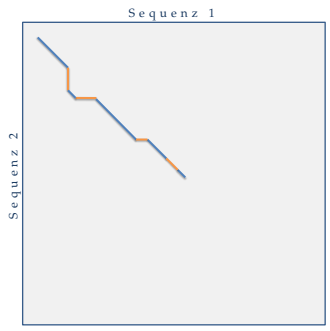


Abbildung 52: Vierter Schritt im Blast-Algorithmus: Alignment. Erstelle mit einer modifizierten Version des Smith-Waterman-Algorithmus ein Alignment um die gefundenen maximum scoring pairs und berechne das optimale lokale Alignment.

2.4.3 *Verbesserungen*

In beiden Heuristiken ist ein Schwellwert nötig, um die Ergebnisse einzuschränken. Hierbei kann es vorkommen, dass Regionen, die nur knapp unter dem Schwellwert liegen nicht mehr weiter verarbeitet werden, obwohl diese ein sehr gutes Ergebnis liefern würden, wenn sie im späteren Alignmentprozess berücksichtigt werden würden. Um diesen Umstand zu verbessern, kann neben einem Schwellwert auch eine minimale Anzahl an zu untersuchenden Sequenzen vorgegeben werden.

2.5 DNS-PROTEIN-ALIGNMENT

Zu Beginn der Arbeit wurde die Umwandlung von DNS-Sequenzen in Proteinsequenzen vorgestellt. Somit kann auch ein Alignment von DNS- und Proteinsequenzen vorgenommen werden kann. Der Leserahmen gibt dabei die Basentripletts vor, welche die Codons für Aminosäuren kodieren. Nach der Übersetzung in Aminosäuren kann ein Alignment mit einer Proteinsequenz erfolgen.
Die in Abschnitt 1.1.1 aufgeführten Mutationsarten (Substitutionen, Insertionen und Deletionen) können für ein optimales Alignment berücksichtigt werden. Es werden sowohl Änderungen in der Nukleotidkette und als auch in der Aminosäuresequenz unterschieden.

2.5.1 *Aminosäure-Mutationen*

Die Mutationen (Substitutionen, Insertionen und Deletionen) auf Aminosäure-Ebene wird wie in den vorangegangenen Algorithmen (z.B. Needleman-Wunsch oder Smith-Waterman) über eine Gapkostenfunktion und über eine Kostenmatrix für Substitutionen behandelt.

2.5.2 *Nukleotid-Substitutionen*

Punktmutationen, welche einer Substitution in der DNS-Sequenz entsprechen, können zu Änderung einer Aminosäure in der Proteinsequenz führen (siehe Abschnitt 1.1.1). Nicht jede Nukleotidsubstitution führt jedoch zu einer Änderung in der Aminosäuresequenz. Wie in Abbildung 7 zu sehen ist, werden unterschiedliche Basentripletts in die gleiche Aminosäure umgewandelt (siehe Abschnitt 1.1.1). Zum Beispiel werden CUA, CUC, CUG und CUU in Leucin übersetzt. Bilden die ersten zwei Basen die Kombination CU, so kodiert das Codon unabhängig von der dritten Base für die Aminosäure Leucin. DNS-Sequenzen können sich somit voneinander unterscheiden und trotzdem für die gleiche Aminosäuresequenz kodieren, wenn sie nur in Wobble-Positionen variieren [68].
Die anderen Formen der Punktmutation (sinnverändernde, sinnentstellende und überlesende Mutationen) lassen sich wie bei Aminosäuremutationen über eine Bewertungsmatrix erfassen.

2.5.3 *Nukleotid-Insertionen bzw. -Deletionen: Frameshifts*

Wie in Abbildung 12 an einem Beispiel illustriert, ist ein konsistenter Leserahmen maßgebend für die Übersetzung in Aminosäuren. Treten Insertionen oder Deletionen innerhalb der DNS-Sequenz auf, verschieben diese den Leserahmen und die Codons kodieren für andere Aminosäuren und damit ein anderes Protein. Wie auch in Abbildung 8 zu sehen ist, werden je nach Startposition andere Aminosäuren kombiniert.
1991 wurde von States und Botstein eine erste Untersuchung zu Frameshifts in Alignments veröffentlicht [6]. Sie verwendeten einen Bayesian Alignment Algorithmus, um ihre Ergebnisse auszuwerten. Dabei stellten sie fest, dass schon ein 2%iger Anteil an Frameshift-Fehlern, welche zu einer Verschiebung im Leserahmen führen, ein mögliches Alignment zerstört und das Ergebnis nicht mehr von einem zufälligen Alignment zu unterscheiden ist.
Diese Frameshift-Mutationen sind tri-periodisch, das heißt, wenn drei Einfüge- oder Lösch-Änderungen in der Nukleotidsequenz aufeinander folgen, so ist der

Leserahmen wieder intakt. In so einem Fall wird eine Insertion oder Deletion einer Aminosäure angenommen.

2.5.3.1 *verschiedene Ansätze zur Lösung des Frameshift-Problems*

Es ist möglich Frameshift-Mutationen bei Alignments von DNS/RNS-Sequenzen und Proteinsequenzen zu berücksichtigen, indem alle drei möglichen Leserahmen der DNS/RNS erstellt werden (siehe Abbildung 8) und die drei so entstehenden Aminosäuresequenzen mit der Proteinsequenz aligniert werden. Das beste Ergebnis wird dann verwendet und als optimales Alignment der DNS-Sequenz und der Proteinsequenz angenommen.
Guan und Uberbacher haben 1996 einen Algorithmus vorgestellt, welcher auf dem Smith-Waterman-Algorithmus basiert [31] und in jeder Alignment-Position drei aufeinanderfolgende Tripletts aus der DNS-Sequenz übersetzt und mit der Aminosäure der Proteinsequenz aligniert. Das Maximum der Berechnungen wird verwendet, um ein optimales lokales Alignment mit dieser Stelle in den Sequenzen zu bilden. Dieser Algorithmus ist in der Lage auch Insertionen und Deletionen von Nukleotiden zwischen den Codons zu ermitteln.
Statt nur einer Bewertungsmatrix für das Alignment werden drei Matrizen benötigt, um die möglichen Bewertungen die sich durch die drei Leserahmen ergeben zu speichern. Da dies sowohl für die aktuelle Position gilt, welche bewertet wird, als auch für die entsprechenden Positionen davor, werden insgesamt neun Matrizen benötigt (bezeichnet mit $S_1, S_2, S_3, P_1, P_2, P_3, Q_1, Q_2$ und Q_3). Analog zum Smith-Waterman-Algorithmus werden die vier Fälle unterschieden, welche zur Berechnung des Maximums an der Position herangezogen werden: Das Alignieren des nächsten Zeichens aus der übersetzten Nukleotidsequenz N_i und der Proteinsequenz A_j, dem Öffnen bzw. Fortsetzen einer Lücke in der übersetzten Nukleotidsequenz, dem Öffnen oder Fortsetzen einer Lücke in der Proteinsequenz und das Starten eines neuen lokalen Alignments. Als Kostenfunktion für die Lücken wurde die affine Lückenkostenfunktion verwendet. Das Öffnen einer Lücke bekommt den Malus $g + e$ und das Fortsetzen einer Lücke den Malus e. Zusätzlich werden abhängig vom aktuellen Leserahmen die sogenannten Shift-Kosten (δ) von der Bewertung subtrahiert, so in einen benachbarten Leserahmen gewechselt wird. Dies verhindert, dass ein Alignment konstruiert wird, welches vorwiegend aus zufälligen Matches besteht [31]. Somit sind neun Maximumsberechnungen notwendig pro Leserahmen (siehe Formel 7).

$$
\begin{aligned}
P_1(i,j) &= \max\{S_1(i-1,j)+g+e\ ,\ P_1(i-1,j)+e\} \\
Q_1(i,j) &= \max\{S_1(i,j-1)+g+e\ ,\ Q_1(i,j-1)+e\} \\
S_1(i,j) &= \max\begin{cases} S_1(i-1,j-1)+m(N_i,A_j) \\ P_1(i,j) \\ Q_1(i,j) \\ 0 \end{cases} \\
t_1 &= \max\{S_2(i-1,j)+g+e\ ,\ P_2(i-1,j)+e\} \\
t_2 &= \max\{S_2(i,j-1)+g+e\ ,\ Q_2(i,j-1)+e\} \\
t_3 &= \max\begin{cases} S_2(i-1,j-1)+m(N_i,A_j) \\ t_1 \\ t_2 \\ 0 \end{cases} \\
\text{Falls} \quad & t_3-\delta > S_1(i,j) : \{P_1(i,j)=t_1-\delta\ ,\ Q_1(i,j)=t_2-\delta\ ,\ S_1(i,j)=t_3-\delta\} \\
t_1 &= \max\{S_3(i-1,j)+g+e\ ,\ P_3(i-1,j)+e\} \\
t_2 &= \max\{S_3(i,j-1)+g+e\ ,\ Q_3(i,j-1)+e\} \\
t_3 &= \max\begin{cases} S_3(i-1,j-1)+m(N_i,A_j) \\ t_1 \\ t_2 \\ 0 \end{cases} \\
\text{Falls} \quad & t_3-\delta > S_1(i,j) : \{P_1(i,j)=t_1-\delta\ ,\ Q_1(i,j)=t_2-\delta\ ,\ S_1(i,j)=t_3-\delta\}
\end{aligned}
$$

Formel 7: Formel zur Berechnung des Maximums für einen Leserahmen an der Position i,j im Alignment: Matrizen P_1 bis P_3 und Q_1 bis Q_3 enthalten die Berechnungen für die besten Bewertungen für das Öffnen bzw. Fortsetzen einer Lücke des bisherigen Alignments im entsprechenden Leserahmen von 1 bis 3. Für den Leserahmen 1 berechnet S_1 die maximale Bewertung, analog zum Smith-Waterman-Algorithmus, aus den drei Vorgängerframes der aktuellen Position, den jeweiligen Lücken-Bewertungen aus diesen Frames und dem Starten eines neuen lokalen Alignments (Bewertung 0). Werden dabei eine oder zwei Nukleotid-Positionen übersprungen, werden die Shift-Kosten (δ) von der Bewertung abgezogen.

Die Komplexität des Alignmentalgorithmus erhöht sich dabei um einen konstanten Faktor. Insgesamt bleibt die Laufzeit von $O(n^2)$ erhalten. Der Platzbedarf wächst auf $O(n^2)$.

Im gleichen Jahr publizierten Huang und Zhang zwei weitere Algorithmen, einen für das globale und einen für das lokale Alignment von DNS- und Proteinsequenzen mit möglichen Frameshift-Mutationen [39]. NAP (Nucleotide-Amino acid alignment Program) und LAP (Local Alignment Program) sind die Bezeichnungen für die beiden Programme. Neben Frameshift-Mutationen zwischen den Codons werden auch mögliche Frameshift-Mutationen innerhalb der Basentripletts berechnet. Die Formel dazu begründet sich wieder auf dem Needleman-Wunsch- bzw. dem Smith-Waterman-Algorithmus. Anstatt immer ein Triplett zu alignieren, werden auch einzelne Nukleotide oder Nukleotidpaare verwendet, welche um zwei bzw. ein Nukleotid erweitert werden. Hierbei wird für die Bewertung das arithmetische Mittel σ aller möglichen Codons angenommen, welche nicht zu einem Stop-Codon evaluieren (siehe Formel 8).

$$\sigma(N_1N_2N^*, A) = \frac{1}{|\{N_1N_2N^*\}|} m(N_1N_2N^*, A) \text{ mit } N_1N_2N^* \in AA$$

Formel 8: Beispiel einer Frameshift-Berechnung: Falls nur zwei Nukleotide aus der Sequenz (N_1N_2) aligniert werden, wird aus der Menge an möglichen Nukleotiden jedes Element gewählt (N^*), um ein Triplett zu bilden und dann mit dem arithmetischen Mittel verrechnet. Die Aminosäure A stammt dabei aus der Menge der Aminosäuren AA.

Die rekursive Formel für das lokale Alignment berechnet für eine Position (i, j) das Maximum über alle möglichen Single-, Double- und Triplett-Kombinationen, so wie kombinierbare Tripletts aus vier bzw. fünf Nukleotiden aus der Nukleotide-Kette und verknüpft diese mit den vorangegangenen (optimalen) Bewertungen aus den entsprechenden Positionen davor. Für das Hinzufügen bzw. Eliminieren von Nukleotiden wird analog zu den Lücken auf der Ebene der Aminosäuren eine Kostenfunktion eingeführt. Folgendes Beispiel erläutert die Vorgehensweise:

```
DNS: GATGA  GCTTC  ATCCTACTATGGTGAAAAGGGGT   GTCT
     -|||.--|||..--..|||.-. |||--------|||---|||-
Prot: MetArgAlaCysTyrProM etTrp        GlyLeuVal
```

$$
\begin{aligned}
S(\mathrm{DNS}, \mathrm{Prot}) = \quad & \sigma(\mathrm{ATG}, \mathrm{Met}) + \sigma(\mathrm{A**}, \mathrm{Arg}) - (g + 2 \cdot e) + \sigma(\mathrm{GCT}, \mathrm{Ala}) + \\
& \sigma(\mathrm{TG*}, \mathrm{Cys}) + \sigma(\mathrm{*AT}, \mathrm{Tyr}) - (g + 2 \cdot e) + \sigma(\mathrm{CCT}, \mathrm{Pro}) + \\
& \sigma(\mathrm{ATA}, \mathrm{Met}) - (g + e) + \sigma(\mathrm{TGG}, \mathrm{Trp}) - (g + 8 \cdot e) + \\
& \sigma(\mathrm{GGT}, \mathrm{Gly}) - (g + 3 \cdot e) + \sigma(\mathrm{GTC}, \mathrm{Val})
\end{aligned}
$$

Abbildung 53: Beispiel eines möglichen Alignments einer DNS- und einer Proteinsequenz und dessen Bewertung mit dem LAP-Algorithmus. Triplett-Übereinstimmungen sind mit "|||" gekennzeichnet, Lücken (sowohl in der Nukleotid-Sequenz als auch in der Protein-Sequenz sind mit "-" markiert und partielle Übereinstimmungen sind mit "." annotiert. Die Bewertung $S(\mathrm{DNS}, \mathrm{Prot})$ setzt sich aus den σ-Berechnungen und den Lückenkosten zusammen. Hier wird die gleiche Kostenfunktion für Nukleotid- und Aminosäure-Lücken angenommen. Für Lückenpositionen entstehen die Kosten zum Öffnen einer Lücke (g) und die Kosten pro Längeneinheit k der Lücke ($k \cdot e$). Das "*"-Symbol in der Berechnung bezeichnet eine Nukleotid-Insertion.

Ein weiterer Algorithmus wurde von Zhang, Pearson und Miller 1997 formuliert, welcher unter Modifikationen auch Einzug in die Fasta-Werkzeuge erhielt [70]. Dieser Ansatz vereinfacht die rekursive Formel für die Berechnung des optimalen Alignments einer Nukleotid- und einer Proteinsequenz (siehe Formel 9).

$$
fs \geqslant \max \left\{ \frac{g + e + m_{max} - m_{min}}{3}, \frac{m_{max} - m_{min}}{2} \right\}
$$

Formel 9: Bestimmung des Frameshiftmalus fs: Die Kosten für einen Frameshift werden so hoch gewählt, dass kein entartetes Alignment entsteht. g entspricht den Gapkosten, e der Erweiterung einer Lücke und m_{max} bzw. m_{min} sind maximaler bzw. minimaler Eintrag in der Bewertungsmatrix.

Unter der Annahme, dass der Frameshift-Malus größer oder gleich den Kosten für ein Drittel einer Aminosäure-Lücke der Länge eins und der Differenz aus dem Maximum und dem Minimum der Einträge aus der Bewertungsmatrix für die

Aminosäuren ist oder der Hälfte der Differenz des maximalen und minimalen Eintrags in der Aminosäurebewertungsmatrix entspricht, wird eine Nebenbedingung im Alignmentalgorithmus formuliert, welche verhindert, dass ein zufälliges oder entartetes Alignment entsteht.
Unter der Voraussetzung, dass für den Frameshiftmalus fs Formel 9 gilt, ergibt sich die rekursive Formel des 1,5-Alignments für den Nukleotide-Protein-Alignment-Algorithmus von Zhang et al. [70]. Es wird für jede Position (i, j) in der Bewertungsmatrix das Maximum aus dem Starten bzw. Fortsetzen einer Lücke in der Nukleotid- bzw. der Proteinsequenz und den Alignierungsbewertungen der ein- bis fünfelementigen Teilsequenzen aus der Nukleotidsequenz mit dem Element der Proteinsequenz bestimmt. Dabei Berechnet eine Funktion R alle möglichen Kombinationen und Permutationen, welche in den ein- bis fünfelementigen Teilsequenzen möglich sind, um ein Triplett zu erzeugen. Ebenso verrechnet die Funktion R den Frameshiftmalus, welche durch das Einfügen bzw. Löschen von einzelnen Nukleotiden entsteht (siehe Formel 10).

$$
\begin{aligned}
D(i,j) &= \max\{D(i-3,j), S(i-3,j) - g\} - e \\
I(i,j) &= \max\{I(i,j-1), S(i,j-1) - g\} - e \\
S(i,j) &= \max \begin{cases}
D(i,j) \\
I(i,j) \\
S(i-1,j-1) + R[N_i, A_j] \\
S(i-2,j-1) + R[N_{i-1}N_i, A_j] \\
S(i-3,j-1) + R[N_{i-2}N_{i-1}N_i, A_j] \\
S(i-4,j-1) + R[N_{i-3}N_{i-2}N_{i-1}N_i, A_j] \\
S(i-5,j-1) + R[N_{i-4}N_{i-3}N_{i-2}N_{i-1}N_i, A_j]
\end{cases}
\end{aligned}
$$

Formel 10: Für die Bewertung des optimalen Alignments an der Stelle (i, j) wird das Maximum der Bewertungen aus dem Starten oder Fortsetzen von Lücken aus den vorangegangenen Positionen im Alignment berechnet (D bzw. I), ebenso wie die Alignierungen eines Nukleotids bis zu fünf Nukleotiden mit der entsprechenden Aminosäure (Funktion R). Dabei berechnet diese Funktion R alle möglichen Kombinationen und Permutationen, welche mit einem Nukleotid bzw. mit bis zu fünf Nukleotiden möglich sind und ergänzt bzw. löscht Nukleotide, um ein Triplett zu erzeugen. Das Ergänzen bzw. Löschen zieht den Frameshiftmalus nach sich, welcher ebenfalls von R verrechnet wird.

Ein Beweis für die Optimalität des 1,5-Alignments findet sich in [70].
Der Ansatz für das 1,5-Alignment findet sich auch im Programm Haxat wieder [47]. Diese 2012 entstandene Implementierung zur DNS-Protein-Alignierung besitzt als Ergänzung der Optionen getrennte Mali für einfache Frameshifts (also dem Hinzufügen bzw. Löschen einer Base) und doppelte Frameshifts (also dem Hinzufügen bzw. Löschen zweier Basen). Wie die Berechnung des Maximums einer Zelle in der Bewertungsmatrix kommen acht Fälle in Frage. Für den Beginn eines neuen lokalen Alignments steht die 0. Es ist möglich das bisherige Alignment durch eine Lücke in der DNS- oder in der Proteinsequenz fortzusetzen. Werden für diese Position Zeichen in beiden Sequenzen aligniert, kann ein Zeichen aus der Proteinsequenz, jedoch bis zu fünf Zeichen aus der Nukleotidsequenz, verwendet werden, um ein Triplett zu bilden. Würden mehr als 5 Zeichen verwendet, müssen drei oder mehr Zeichen für die Alignierung in der Position gelöscht werden, was zu mindestens einer Deletion auf Ebene der Aminosäuren führen würde. Durch die triperiodische Struktur der Nukleotidsequenz genügt es, bis zu fünf Basen pro Position zu beachten. Bei fünf Nukleotiden können durch eine doppelte Frameshift-Deletion zwei Nukleotide entfernt werden und es bleibt ein Triplett übrig, welches zu einer Aminosäure codiert wird. Bei vier Nukleotiden kann eine Nukleotid-Deletion zur entsprechenden Aminosäure führen. Werden exakt drei Nukleotide betrachtet, so werden diese direkt in eine Aminosäure übersetzt. Bei zwei und einem Nukleotid aus der Nukleotid-Kette wird die Aminosäure durch einfache bzw. doppelte Nukleotid-Insertion ergänzt, was wiederum einem einfachen bzw. doppeltem Frameshift entspricht.
Für fünf Nukleotide gibt es $\binom{5}{3} = 10$ Möglichkeiten, um ein Triplett zu bilden, welches mit dem Zeichen aus der Aminosäure aligniert wird. Für vier Zeichen gibt es $\binom{4}{3} = 4$ Fälle. Einen weiteren Fall für genau drei Nukleotide und für das Ergänzen von zwei bzw. einem Nukleotid gibt es für jedes Nukleotid und jede Position einmal 12 und einmal 48 Möglichkeiten. Zieht man sich überlappende bzw. identische Fälle ab, so bleiben noch zehn bzw. 37 Möglichkeiten für die Nukleotid-Insertionen übrig. Zusammengefasst müssen $4(\mathtt{Ins}, \mathtt{Del}, \mathtt{Match}, 0) + 10 + 4 + 1 + 10 + 37 = 66$ Berechnungen pro Zelle ausgeführt werden, um das Maximum zu bestimmen.

Daraus lässt sich das optimale lokale Alignment von Nukleotidsequenzen zu Aminosäuresequenzen mit Unterstützung von Frameshiftmutationen sukzessiv berechnen.

2.6 IMPLEMENTIERUNGEN

Mit Hilfe der Datenbasis einer Mutations-Resistenzdatenbank [10] lassen sich über das Ergebnis eines paarweisen Sequenzalignments einer Wildtyp-Proteinsequenz und der Nukleotidsequenz eines Virus bekannte Resistenzen des mutierten Virus gegenüber der Wirkung bestimmter Medikamente abgleichen. Für ein solches Verfahren wurde ein neuer Algorithmus entwickelt und um zusätzliche Methoden ergänzt, um sowohl für den Einsatz als Kommandozeilenwerkzeug als auch als Zwischenschritt in einer Webanwendung verwendet werden zu können. Hierbei wurden sowohl Useranforderungen als auch Optimierungsschritte implementiert.

2.6.1 *erstes Verfahren*

Ein erstes Verfahren zur Bestimmung von Virusresistenzen mit Hilfe von paarweisem Sequenzalignment verwendet eine Wildtyp-Proteinsequenz, also eine auf natürliche Weise durch Evolution entstandene Sequenz, um eine Nukleotidsequenz eines mutierten Virus zu alignieren und die mutierten Stellen zu analysieren (siehe Abbildung 54). Als Alignmentalgorithmus wird der Blastx-Algorithmus aus der Blast-Suite verwendet (siehe Abschnitt 2.4.2). Der Abgleich des Alignmentergebnisses mit der hinterlegten Resistenzdatenbank gibt Aufschluss auf mögliche Resistenzen bzw. Sensibilitäten des Virus gegenüber bestimmten Medikamenten.

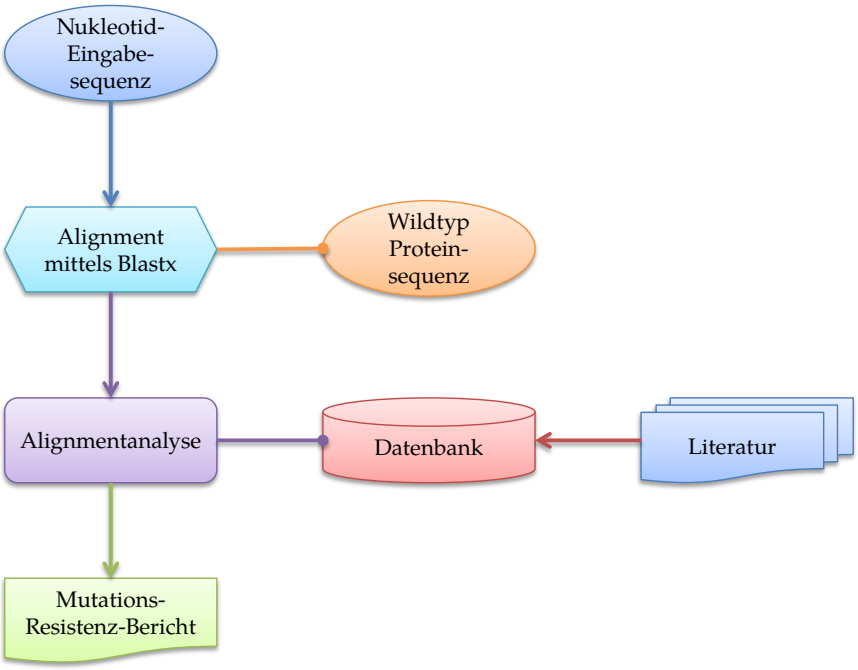


Abbildung 54: Schematische Darstellung des ersten Verfahrens: Zuerst wird ein paarweises Sequenzalignment einer Nukleotidsequenz und einer Wildtyp-Proteinsequenz durch den Blastx-Algorithmus berechnet. Das Alignment mit der höchsten Bewertung wird danach analysiert und mutierte Stellen mit der Mutations-Resistenzdatenbank abgeglichen. Die Ergebnisse werden anschließend in einem Bericht präsentiert.

Das obige Verfahren gliedert sich in drei Phasen. Zuerst wird ein paarweises lokales Sequenzalignment durchgeführt. Im Anschluss an diese Berechnung wird das am höchsten bewertete Alignment analysiert und mit Einträgen aus der Mutations-Resistenzdatenbank abgeglichen. Diese Datenbank beinhaltet Einträge, welche auf Literaturrecherche basieren. Liegen in dieser Übereinstimmungen bei Mutationen der Virussequenz vor, so lässt dies auf Resistenzen des Virus gegenüber speziellen Medikamenten schließen. Abschließend werden die Ergebnisse in Form eines Berichts aufbereitet. Die drei Phasen des Verfahrens lassen sich in weitere Abschnitte untergliedern (siehe Abbildung 55).

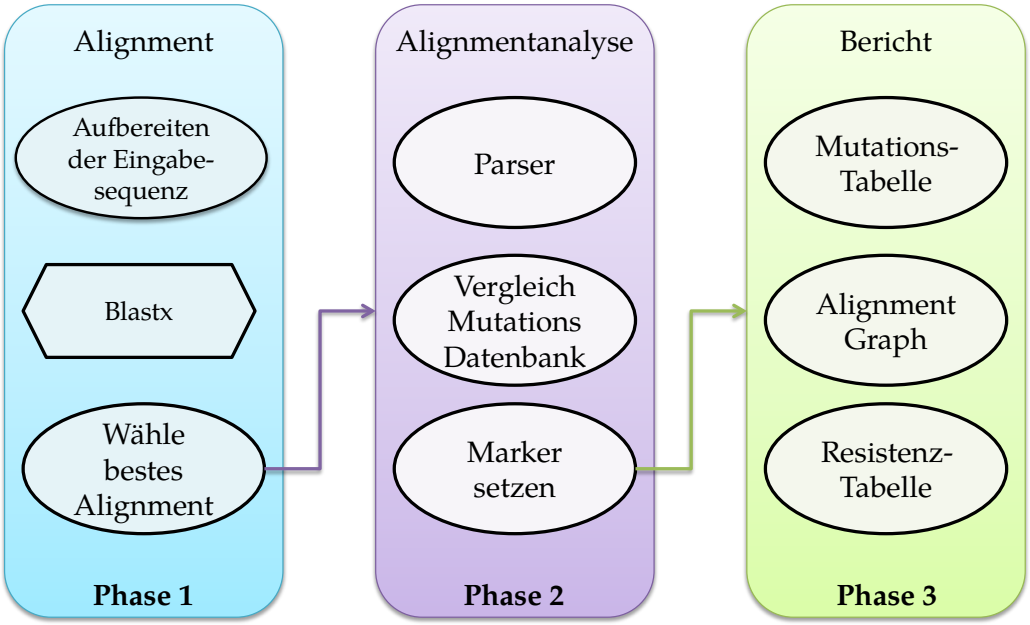


Abbildung 55: Vorgehensweise des Verfahrens aus [10] im Detail: In Phase 1 wird die Eingabesequenz aufbereitet, so dass diese im Blastx-Algorithmus verarbeitet und mit der Proteinsequenz aligniert werden kann. Danach wird das beste Alignment (hier, das Alignment mit der höchsten Smith-Waterman-Bewertung) gewählt und in Phase 2 analysiert. Ein Parser durchläuft das Alignment und erfasst alle Stellen, welche nicht übereinstimmen. Diese gesammelten Stellen werden dann mit den Einträgen aus der Mutations-Datenbank verglichen und abhängig von den gefundenen Einträgen mit entsprechenden Resistenzmarkern versehen. Phase 3 bereitet die Ergebnisse in unterschiedlichen Darstellungsformen auf (tabellarisch und als Übersichtsgrafik) und fasst diese zu einem Bericht zusammen.

Die Eingabesequenz wird im FASTA-Format übergeben. Dieses Textformat besteht aus zwei Teilen pro Sequnz: einer optionalen Überschrift bzw. Anmerkung zur Sequenz oder dessen Name, eingeleitet mit einem ”>”, und der Buchstabenabfolge der DNS- oder RNS-Sequenz, zusammengesetzt aus den Bezeichnern für die einzelnen Nukleotide (A, C, G und T bzw. U). Für den Blastx-Algorithmus wird dieses Format vorverarbeitet, sodass lediglich die zusammenhängende Nukleotidsequenz an den Algorithmus übergeben wird.
Der Blastx-Algorithmus arbeitet analog zu dem in Kapitel 2 vorgestellten Blast-Algorithmus (siehe Abschnitt 2.4.2). Statt jedoch zwei Nukleotid- bzw. Proteinsequenzen zu alignieren, werden eine Nukleotid- und eine Proteinsequenz als Eingabe verwendet. Hierbei wird zuerst die Nukleotidsequenz in eine Aminosäurekette umgewandelt, bevor diese mit der Proteinsequenz (ebenfalls in

Form einer Aminosäuresequenz) aligniert wird. Bei der Umwandlung werden die drei möglichen Leserahmen berücksichtigt (siehe Abbildung 8 und Abschnitt 1.1.1). Es wird je ein Alignment der Proteinsequenz mit den drei Nukleotid-zu-Aminosäuresequenzen berechnet. Die Ausgabe des Blastx-Algorithmus beinhaltet die verwendeten Parameter, sowie das Ergebnis des Alignments (siehe Abbildung 56. Das Alignment mit der höchsten Smith-Waterman-Bewertung wird danach ausgewählt und dessen Ausgabe in Phase 2 für die Analyse verwendet.

```
...
Database: UL97Protein.fa
           1 sequences; 707 total letters

Query= Test-Sequence

Length=2118
                                                                      Score     E
Sequences producing significant alignments:                          (Bits)  Value

  UL97 Protein Sequence                                               891    0.0

> UL97 Protein Sequence
Length=707

 Score =  891 bits (2330),  Expect = 0.0, Method: Compositional matrix adjust.
 Identities = 584/735 (79%), Positives = 600/735 (82%), Gaps = 60/735 (8%)
 Frame = +1

Query  4     SARRSRARRAS--GTTQGW--APPRHPSRARRRQWMREAAAAEQAKAVQAGAQAAAVQVA  171
             SA RSRAR AS   TTQGW   P R PSRARRRQWMREAA A    AVQA AQAAA QVA
Sbjct  3     SALRSRARSASLGTTTQGWDPPPLRRPSRARRRQWMREAAQAAAQAAVQA-AQAAAAQVA  61

Query  172   QAHVDE-EVWDLMTDEAGGGVTSSLTTLSSVSTTVQGHATFSG-VRSVMHSDGEKEDAAS  345
             QAHVDE EV DLMTDEAGGGVT+  T  S  +TTV GHATFS  VRS +  DGEKEDAAS
Sbjct  62    QAHVDEDEVVDLMTDEAGGGVTTLTTLSSVSTTTVLGHATFSACVRSDVMRDGEKEDAAS  121
...
Query  1897  FVEADMESCRVRAFRRSLFYHECSW-MLHEYVRKNVERLLATSDGLCLYNAFRRCTTSII  2073
             FVEA M SCRVRAFRR  FYHECS  MLHEYVRKNVERLLATSDGL LYNAFRR TTSII
Sbjct  636   FVEAKMSSCRVRAFRR--FYHECSQTMLHEYVRKNVERLLATSDGLYLYNAFRR-TTSII  692

Query  2074  CEDLIDGDCRQLFPE  2118
             CE+ +DGDCRQLFPE
Sbjct  693   CEEDLDGDCRQLFPE  707

...
Matrix: BLOSUM62
Gap Penalties: Existence: 8, Extension: 2
Neighboring words threshold: 12
Window for multiple hits: 40
```

Abbildung 56: Teilausgabe des Blastx-Algorithmus: Die Ausgabe beinhaltet die Namen der untersuchten Sequenzen (Database und Query), deren Längen (Length), die Bewertung (Score), sowie die Anzahl der Übereinstimmungen (Identities), der positiv bewerteten Positionen (Positives), der Lücken (Gaps) und den Leserahmen, welcher verwendet wurde (Frame). Es folgen die aufgeschlüsselten Alignments im Detail und am Ende werden die verwendeten Parameter aufgelistet (Bewertungsmatrix, Lückenkosten, Schwellwert etc.)

Ein Parser durchläuft in Phase 2 des Verfahrens das am höchsten bewertete Alignment und schlüsselt alle Positionen auf, welche nicht exakt übereinstimmen. Dabei werden nicht übereinstimmende Positionen mit positiver Bewertung (positiv Matches), mit negativer Bewertung (Mismatches) und Lücken (Gaps) in den Sequenzen unterschieden und in einer Datei abgespeichert. Diese Datei beinhaltet zusätzlich noch die Position der mutierten Stellen, die Länge des betrachteten Alignments, sowie Start- und Endposition der alignierten Sequenzen und eine Kategorie, welche festlegt, ob des als Alignments erfolgreich angesehen wird, also die Länge einen bestimmten Schwellwert übersteigt ("found": true oder false). Der in diesem Verfahren verwendete Parser erstellt eine sogenannte json-Datei (Abbildung 57).


```
{"mutations":[{"position":75,"subject":"T","query":"A"},{"position":102,"subject":"F","query":"Q"},
              {"position":103,"subject":"S","query":"F"},{"position":104,"subject":"A","query":"S"},
              {"position":105,"subject":"C","query":"G"},{"position":402,"subject":"C","query":"-"},
              {"position":520,"subject":"H","query":"Q"},{"position":581,"subject":"-","query":"C"},
              {"position":582,"subject":"-","query":"H"},{"position":583,"subject":"-","query":"A"}],
 "insertions":[{"begin":581,"end":583}],
 "deletions":[{"begin":402,"end":402}],
 "length":"707",
 "start":"1","end":"707",
 "found":true}
```


Abbildung 57: Inhalt einer durch den Parser erstellen Datei: Gruppiert werden die Mutationen (positive Matches oder Mismatches), Insertionen, Deletionen, sowie Länge, Start- und Endposition und ob das Alignment erfolgreich war (Die Länge des Alignments muss einen festgelegten Wert überschreiten).

Das Akronym json steht für JavaScript Object Notation und der Inhalt einer solchen Datei ist in Kategorien gegliedert. Daher ist dieses Dateiformat gut für die computergestützte Weiterverarbeitung geeignet.
Der gegliederte Aufbau des Inhalts der json-Datei ermöglicht einen Abgleich der gefundenen Mutationen mit Resistenzen, welche in der Mutations-Resistenzdatenbank vermerkt sind. Diese Datenbank enthält neben der Position (locus), der Art der Mutation und der möglichen assoziierten Resistenzen des untersuchten Virus auch Angaben zur Sensibilität, einen Vermerk zur Literaturrecherche sowie einen Eintrag zu verwendeten Testmethode und zu zusätzlichen Informationen (siehe Tabelle 1).

Tabelle 1: Ausschnitt aus der Mutations-Resistenz-Datenbank: Neben Position (locus), Art der Substitution (mutation), Angabe von Resistenz (resistant) und Sensibilität (sensitive) sowie Anfälligkeit (drug susceptibility und viral fitness) sind auch Angaben zur Literatur (literature) , verwendeter Testmethode (method) und ergänzende Informationen (information) gespeichert.

locus	mutation	resistant	sensitive	drug susceptibility (ratio of EC_{50})	viral fitness	literature	method	information
411	H411L		GCV	0.7x		Chou et al., 2008	marker transfer (not clonal); SEAP assay	
411	H411N		GCV	1x		Chou et al., 2008	marker transfer (not clonal); SEAP assay	
411	H411Y		GCV	0.5x		Chou et al., 2008	marker transfer (not clonal); SEAP assay	
427	A427V		GCV	1.0x		Martin et al., 2006	marker transfer (clonal); plaque reduction assay	
440	A440V		GCV	1.2x		Chou et al, 2010	marker transfer (clonal); SEAP-Assay	
442	A442G		GCV	1.2x		Chou et al, 2010	marker transfer (clonal); SEAP-Assay	
466	V466G	GCV		3.5x	impaired	Martin et al., 2010	marker transfer (clonal); plaque reduction assay	small plaque phenotype

Der Eintrag `{"position":75,"subject":"T","query":"A"}` der json-Datei wir beispielsweise zum Mutationseintrag `A75T` der Datenbank assoziiert. Falls für diese Stelle eine Resistenz oder Sensibilität vermerkt ist, wird für die entsprechende Stelle im Alignment ein Marker hinterlegt.
Sind alle Positionen abgearbeitet, liegen für alle gefundenen Mutationen Marker vor, welche die Art der Mutation und eine etwaige Resistenz bzw. Sensibilitäten anzeigen.
In Phase 3 werden nun die Mutationsinformationen in tabellarischer und graphischer Form aufbereitet. Für Mismatches und positive Matches wird eine Mutationstabelle erstellt, welche die Position und Art der Substitution angibt (siehe Tabelle 2). Für Insertionen und Deletionen wird ebenfalls eine entsprechende Tabelle generiert (siehe Tabelle 3). Neben der tabellarischen Form wird auch ein Alignmentgraph über die gefundenen Mutationen erstellt, in der die Mutationsstellen farblich kodiert markiert sind (siehe Abbildung 58).

Tabelle 2: Tabelle der positiven Matches und Mismatches: Neben dem Laufindex (S/N) sind in der Tabelle Position der Mutation, die erwartete Aminosäure (Wild Type AA) und die aus der Nukleotidsequenz übersetzte Aminosäure (Submitted AA) gespeichert.

S/N	Position	Wild Type AA	Submitted AA
1	413	D	A
2	427	H	N
3	898	D	A
4	1122	T	Q

Tabelle 3: Tabelle der Insertionen und Deletionen: Mit einem Index (S/N) versehen sind in jeder Zeile der Tabelle die Art der Mutation (Insertions / Deletions), die Startposition (Position Begin) und die Endposition (Position End) der Mutation angegeben.

S/N	Insertions / Deletions	Position Begin	Position End
1	Insertion	366	368
2	Insertion	523	524
3	Deletion	134	136

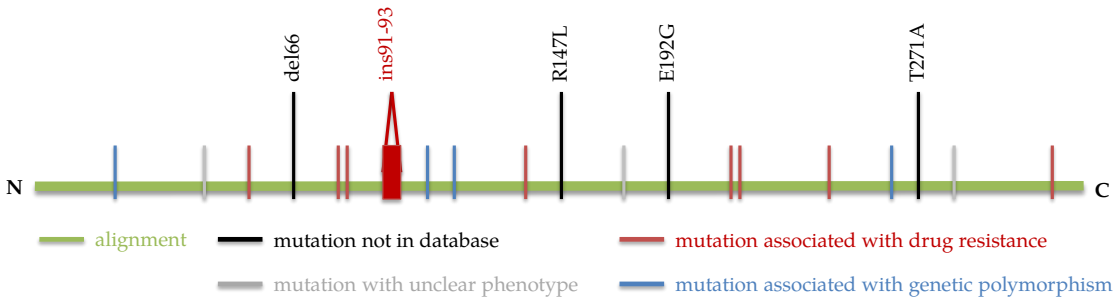


Abbildung 58: Graph eines Alignments mit den Markierungen der gefundenen Mutationen. Unterschieden werden diese farblich in ”unbekannt” (not in database), ”steht in Zusammenhang mit einer Resistenz” (associated with drug resistance), ”unklarer Phänotyp” (unclear phenotype) und ”steht im Zusammenhang mit einem Polymorphismus” (associated with genetic polymorphism).

2.6.2 *Neuerungen*

Das im vorangegangen Abschnitt (siehe Abschnitt 2.6.1) vorgestellte Verfahren kann weiterentwickelt werden, um erweiterte Eingaben zuzulassen und beim Alignment Frameshift- und Stopcodonmutationen zu berücksichtigen. Dies hat Änderungen im Ablauf des Verfahrens zur Folge (siehe Abbildung 59).

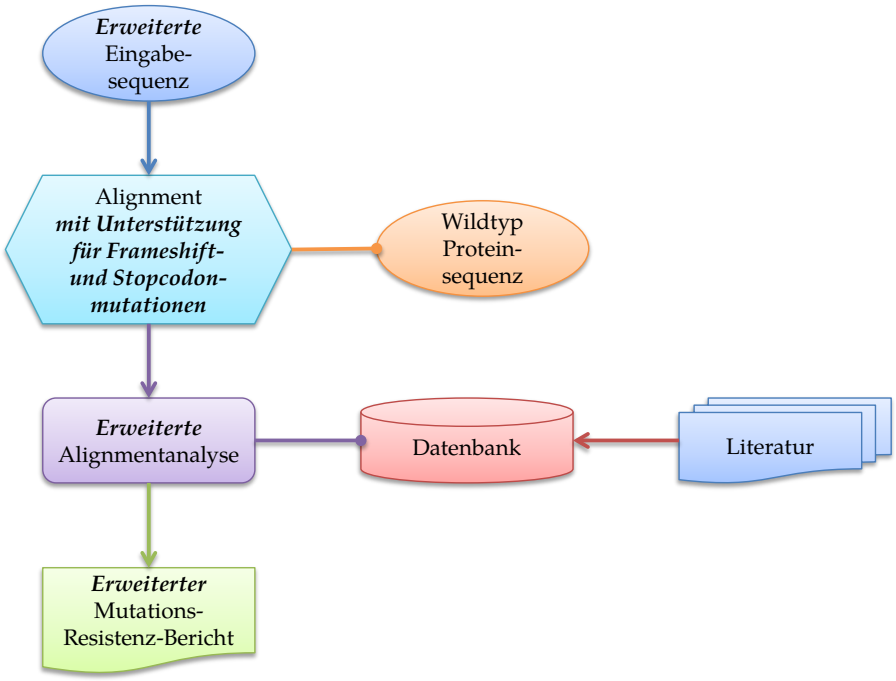


Abbildung 59: Schematische Darstellung der weiterentwickelten Verfahrens: Für die erweiterte Eingabe und die Unterstützung von Frameshift- und Stopcodonmutationen wird ein neuer Alignmentalgorithmus verwendet. Für die zusätzlich unterstützten Mutationsarten wird die Datenbank angepasst und die Ausgaben des abschließenden Berichts werden ergänzt.

ERWEITERTE EINGABESEQUENZEN IN IUPAC-NOMENKLATUR

Die IUPAC-Nomenklatur [12] erlaubt es, neben den vier Nukleotid-Symbolen für DNS- bzw. RNS-Sequenzen auch Platzhalter in Sequenzen zu verwenden, welche für zwei, drei oder auch vier mögliche Nukleotide stehen. Diese Platzhalter, auch Wild-Basen genannt, werden verwendet, wenn sich die Base des Nukleotids an einer Position nicht exakt bestimmen lässt, allerdings eine Einschränkung auf bestimmte Teilmengen möglich ist. In Tabelle 4 sind die Wild-Basen und ihre Platzhalter aufgelistet.

Tabelle 4: IUPAC-Tabelle der Nukleotid-Ersetzungen: Die Platzhalter auf der linken Seite, auch Wild-Basen genannt, repräsentieren zwei oder drei Nukleotide, welche an dieser Position der Sequenz möglich sind. Das N dient als Universal-Symbol, an dessen Stelle jedes der vier Nukleotide stehen kann.

Wild-Base:	mögliche Nukleotide:
M	A C
R	A G
W	A T
S	C G
Y	C T
K	G T
V	A C G
H	A C T
D	A G T
B	C G T
N	A G C T

2.6.3 *Algorithmus für das erweiterte Verfahren*

Die Analyse von Frameshift- oder Stopcodonmutationen wird vom Blastx-Algorithmus nicht unterstützt. Daher wird in diesem erweiterten Verfahren ein neuer Algorithmus verwendet, um das Alignment zu berechnen. Grundlage hierbei ist der Smith-Waterman-Algorithmus mit der 1,5-Alignment-Erweiterung (siehe Abschnitt 2.5). Analog zur Formel 10 lässt sich die Rekursion für den Algorithmus aufstellen (siehe Formel 11).

$$
S[A_i, N_j] = \max \begin{cases}
S[A_{i-1}, N_{j-1}] + m^*(A_i, N_j) + d & (1) \\
S[A_{i-1}, N_{j-2}] + m^*(A_i, N_{j-1..j}) + s & (2) \\
S[A_{i-1}, N_{j-3}] + m^*(A_i, N_{j-2..j}) & (3) \\
S[A_{i-1}, N_{j-4}] + m^*(A_i, N_{j-3..j}) + s & (4) \\
S[A_{i-1}, N_{j-5}] + m^*(A_i, N_{j-4..j}) + d & (5) \\
\max \begin{cases} S_1[A_{i-1}, N_j] + e \\ S[A_{i-1}, N_j] + g + e \end{cases} & (6) \\
\max \begin{cases} S_2[A_i, N_{j-3}] + e \\ S[A_i, N_{j-3}] + g + e \end{cases} & (7) \\
0 & (8)
\end{cases}
$$

Formel 11: Alignmentalgorithmus für das erweiterte Verfahren: Für die Bewertung einer Zelle S der Matrix eines Nukleotid-Protein-Alignments an der Position (i, j) wird das Maximum über acht Berechnungen gebildet. Im Fall (1) wird lediglich ein Nukleotid aus der DNS- oder RNS-Sequenz mit einer Aminosäure aus der Proteinsequenz aligniert, über die Funktion m^* bewertet und die Kosten für einen doppelten Frameshift d einberechnet. Im Fall (2) werden zwei Nukleotide mit einer Aminosäure aligniert, bewertet und die Kosten für einen einfacher Frameshift s addiert. Fall (3) aligniert ein Nukleotid-Triplett, also ein komplettes Codon mit einer Aminosäure. Fall (4) und (5) sind analog zu Fall (1) und (2), jedoch werden hier vier bzw. fünf Nukleotide aus der DNS- bzw. RNS-Sequenz mit einer Aminosäure aligniert, was wiederum einen einfachen bzw. doppelten Frameshift zur Folge hat, welcher wieder verrechnet wird. Das Einfügen bzw. Vergrößern einer Lücke in der Protein-Sequenz (gespeichert in S_1) (Fall (6)) und das Einfügen bzw. Vergrößern einer Lücke in der Nukleotid-Sequenz (gespeichert in S_2) (Fall (7)) werden mit affinen Gapkosten verrechnet. Dabei sind g die Kosten für die Initialisierung einer Lücke und e die Kosten für das Vergrößern einer Lücke. Das Starten eines neuen lokalen Alignments ist Fall (8). Die Initialisierung von S, S_1 und S_2 erfolgt identisch zur Initialisierung im Smith-Waterman-Algorithmus (siehe Abschnitt 2.3.4).

Die Korrektheit der Formel 11 kann über vollständige Induktion gezeigt werden: Die Initialisierung der Bewertungsmatrix hier ist identisch zu der beim Smith-Waterman-Algorithmus. Somit gilt die Bewertung zweier Sequenzen s und t mit $|s| = n$ und $|t| = m$:

$S[0,0] = 0$, $S[i,0] = 0$, $\forall\, i \in \{1,n\}$ und $S[0,j] = 0$, $\forall\, j \in \{1,m\}$

Für die Induktionshypothese von $S[A_i, N_j]$ gilt die dargestellte Formel 11.

Induktionsschritt:

Für $S[A_{i+1}, N_{j+1}]$ gelte:

$$S[A_{i+1}, N_{j+1}] = \max \begin{cases} S[A_i, N_j] + m^*(A_{i+1}, N_{j+1}) + d \\ S[A_i, N_{j-1}] + m^*(A_{i+1}, N_{j..j+1}) + s \\ S[A_i, N_{j-2}] + m^*(A_{i+1}, N_{j-1..j+1}) \\ S[A_i, N_{j-3}] + m^*(A_{i+1}, N_{j-2..j+1}) + s \\ S[A_i, N_{j-4}] + m^*(A_{i+1}, N_{j-3..j+1}) + d \\ \max \begin{cases} S_1[A_i, N_{j+1}] + e \\ S[A_i, N_{j+1}] + g + e \end{cases} \\ \max \begin{cases} S_2[A_{i+1}, N_{j-2}] + e \\ S[A_{i+1}, N_{j-2}] + g + e \end{cases} \\ 0 \end{cases}$$

Für die Fälle des 1,5-Alignments ist die Induktionshypothese bereits zutreffend, da in den fünf Fällen bereits ein optimales Alignment in einer Vorstufe der Induktionshypothese berechnet wurde. Noch unbewiesen sind die Berechnungen für die Lücken in den Sequenzen. Diese lassen sich jedoch auf die Initialisierung der drei Bewertungsmatrizen zurückführen. Sowohl für die $(0,0)$-Position, als auch für die 0te-Zeile und Spalte ist die Bewertung 0. Wird auf die Lückenberechnungen nun rekursiv immer wieder die Formel angewendet, so errechnen sich die Vorstufen der Induktionshypothese für die 1,5-Alignments der Lücken. Sowohl in S als auch in S_1 und S_2 lediglich der i-malige bzw. j-malige Aufruf von $S_1[0, N_{j+1}] + e$ bzw. $S[0, N_{j+1}] + g + e$ sowie $S_2[A_{i+1}, 0] + e$ und $S[A_{i+1}, 0] + g + e$ bleiben übrig. Diese sind jedoch schon durch die Initialisierung mit $0 + e$ bzw. $0 + g + e$ berechnet. Somit ist die vollständige Induktion gezeigt.

■

FRAMESHIFTMUTATIONEN

Die Bewertungen des 1,5-Alignments lassen sich weiter aufschlüsseln. Sowohl beim Hinzufügen als auch beim Löschen von Nukleotiden kann ein Maximum bestimmt

werden, indem die höchste Bewertung für die alignierten Positionen in Abhängigkeit der hinzugefügten bzw. gelöschten Nukleotiden berechnet wird. Formel 12 zeigt das Vorgehen.

$$m^*(A_i, N_j) = \max\left\{m[A_i, p] \mid p \in \left\{\pi(N_j, N_x^I, N_y^I)\right\}\right\} \quad \text{, mit } N_x^I, N_y^I \in \{A, C, G, T/U\} \tag{1}$$

$$m^*(A_i, N_{j-1..j}) = \max\left\{m[A_i, p] \mid p \in \left\{\pi(N_{j-1}, N_j, N_x^I)\right\}\right\} \quad \text{, mit } N_x^I \in \{A, C, G, T/U\} \text{ und } pos_\pi(N_{j-1}) < pos_\pi(N_j) \tag{2}$$

$$m^*(A_i, N_{j-2..j}) = m[A_i, (N_{j-2}, N_{j-1}, N_j)] \tag{3}$$

$$m^*(A_i, N_{j-3..j}) = \max\left\{m[A_i, p] \mid p \in \left\{\upsilon(N_{j-3}, N_{j-2}, N_{j-1}, N_j)\right\}\right\} \tag{4}$$

$$m^*(A_i, N_{j-4..j}) = \max\left\{m[A_i, p] \mid p \in \left\{\upsilon(N_{j-4}, N_{j-3}, N_{j-2}, N_{j-1}, N_j)\right\}\right\} \tag{5}$$

Formel 12: Berechnung der höchsten Bewertungen m^*, in Abhängigkeit der Frameshiftmutationen: Wird nur ein Nukleotid aus der betrachteten Sequenz mit einer Aminosäure aligniert (Fall (1)), so werden zwei Nukleotide aus der Menge der möglichen Nukleotide ergänzt ($N_x^I, N_y^I \in \{A, C, G, T/U\}$) und mit Hilfe der Funktion π durchpermutiert. Für Fall (2) werden zwei Nukleotide aus der Sequenz verwendet und mit einer Aminosäure aligniert, so wird nur ein Nukleotid aus der Menge der möglichen Nukleotide ($N_x^I \in \{A, C, G, T/U\}$) ergänzt und wiederum mit Hilfe von π permutiert. Nebenbedingung hierbei ist jedoch, dass die Position der Sequenz-Nukleotide nicht invertiert werden. Im Fall (3) tritt keine Frameshift-Mutation auf und es wird keine Maximumbewertung berechnet. Werden vier Nukleotide aus der DNS- bzw. RNS-Sequenz verwendet (Fall (4)), so wird ein Nukleotid gelöscht und das übrig bleibende Nukleotid-Triplett mit der Aminosäure aligniert. Die Funktion υ generiert diese Tripletts, indem drei Elemente aus der Menge der Nukleotide ausgewählt werden. Analog wird im Fall (5) vorgegangen, jedoch werden hier zwei Nukleotide aus den fünf Sequenznukleotiden entfernt, um die entsprechenden Tripletts zu bilden.

STOPCODONMUTATIONEN

Stopcodonmutationen werden gesondert behandelt. Ihre Aufgabe ist es die Translation der DNS-Sequenz zu beenden, dabei wird keine Aminosäure erzeugt. Wird die Proteinproduktion vorzeitig beendet, entsteht kein vollständiges Protein [7, 51]. Diese Form der Nukleotid-Mutation kann durch eine gesonderte Bewertung für einen Mismatch im Alignment verrechnet werden (siehe Formel 13).

$$m[A_i, (N_{j_1..j_3})] = \begin{cases} m_1[A_i, N_{j_1..j_3}] & \text{, für } r(N_{j_1..j_3}) \in AA \\ s_{Stp} & \text{, für } r(N_{j_1..j_3}) \in Stp \end{cases}$$

Formel 13: Für die Bewertung m einer Aminosäure A_i aus der Menge der Aminosäuren AA mit einer Nukleotidtriplett $N_{j_1..j_3}$ werden zwei Fälle unterschieden: Lässt sich das Nukleotid-Triplett in eine Aminosäure umwandeln ($r(N_{j_1..j_3}) \in AA$), so wird der Eintrag in der Bewertungsmatrix m_1 verwendet. Falls jedoch das Triplett zu einem Stopcodon evaluiert ($r(N_{j_1..j_3}) \in Stp$), so wird die gesonderte Bewertung s_{Stp} verwendet.

IUPAC-BEWERTUNG

Wild-Basen in einer Nukleotid-Sequenz können auf verschiedene Arten im Alignment verarbeitet werden. Eine naive Möglichkeit ist es, jeden Platzhalter der IUPAC-Nomenklatur in der DNS- bzw. RNS-Sequenz durch ein repräsentiertes Nukleotid (siehe Tabelle 4) zu ersetzten und dann jede Sequenz, welche sich aus den möglichen Ersetzungen ergibt, zu alignieren. Angenommen, es soll eine Sequenz mit 5000 Nukleotiden untersucht werden, welche zu 1% aus Wild-Basen besteht, welche im Schnitt alle zu 3 Basen evaluieren, dann ergibt sich als Anzahl möglicher Nukleotidsequenzen $3^{50} \approx 7.18 \cdot 10^{23}$. Dies ist ein hoher Aufwand.

Ein alternatives Vorgehen ersetzt die Wild-Basen während des Alignment-Prozesses und spielt alle Möglichkeiten für die Ersetzung dann durch, wenn die Sequenzposition der Wild-Base im Alignment erreicht wird. Der Induktionsbeweis und das Optimalitätsprinzip von Bellman (siehe Abschnitt 2.6.3 und 2.2) erlauben dieses Vorgehen. Dabei kann je nach Fokus des berechneten Alignments eine best- und worst-case-Analyse vorgenommen werden. Wird die Mutation gesucht, welche für eine Wild-Base-Stelle die niedrigste Bewertung ergibt, so wird die worst-case-Ersetzung verwendet. Wird andererseits die best-case-Lösung gesucht, so wird die Ersetzung vorgenommen, welche die höchste Bewertung berechnet (siehe Formel 14).

$$
\begin{aligned}
m^*(A_i, N_j) &= \max\left\{\mathrm{opt}\left\{m[A_i, p] \mid p \in \left\{\pi(N_j, N_x^I, N_y^I)\right\}\right\}\right\} && \text{, mit } N_x^I, N_y^I \in \{A, C, G, T/U\} \\
m^*(A_i, N_{j-1..j}) &= \max\left\{\mathrm{opt}\left\{m[A_i, p] \mid p \in \left\{\pi(N_{j-1}, N_j, N_x^I)\right\}\right\}\right\} && \text{, mit } N_x^I \in \{A, C, G, T/U\} \\
& && \text{und } \mathrm{pos}_\pi(N_{j-1}) < \mathrm{pos}_\pi(N_j) \\
m^*(A_i, N_{j-2..j}) &= \mathrm{opt}\left\{m[A_i, (N_{j-2}, N_{j-1}, N_j)]\right\} \\
m^*(A_i, N_{j-3..j}) &= \max\left\{\mathrm{opt}\left\{m[A_i, p] \mid p \in \left\{v(N_{j-3}, N_{j-2}, N_{j-1}, N_j)\right\}\right\}\right\} \\
m^*(A_i, N_{j-4..j}) &= \max\left\{\mathrm{opt}\left\{m[A_i, p] \mid p \in \left\{v(N_{j-4}, N_{j-3}, N_{j-2}, N_{j-1}, N_j)\right\}\right\}\right\}
\end{aligned}
$$

Formel 14: Erweiterung der Formel 12 für die IUPAC-Nomenklatur: Zusätzlich zur maximalen Bewertung des 1,5-Alignments kann für die einzelnen Fälle noch eine Optimums-Analyse opt vorgenommen werden. Dabei kann eine best- oder worst-case-Strategie für die einzelnen Bewertungen vorgenommen werden, abhängig von vorkommenden Wild-Basen an der Position der erweiterten Nukleotid-Sequenz. Im Unterschied zu Formel 12 können hier die Symbole der Sequenz aus der Menge der Nukleotide $\{A, C, G, T/U\}$ und der IUPAC-Platzhalter $\{M, R, W, S, Y, K, V, H, D, B, N\}$ stammen.

ERWEITERTE ALIGNMENTALGORITHMUS-AUSGABE

Die Ausgabe des Alignmentalgorithmus aus Abschnitt 2.6.3 wurde ähnlich zur Ausgabe des im ursprünglichen Verfahren verwendeten Blastx-Algorithmus gehalten. Die Ausgabe beinhaltet eine detaillierte Liste aller Parameter, die Alignment-Bewertung, eine aufgeschlüsselte Darstellung des Alignments und die Zusammenfassung der unterschiedlichen Mutationen (siehe Abbildung 60).

```
SWAT-Alignment-Protocol:

Protein sequence: UL97 Protein Sequence
Nucleotide sequence: UL97-test-sequence

Length of the protein sequence:707
Length of the nucleotide sequence:2122

Parameters: gop: -11, gep: -1, sfsp: -20, dfsp: -25, matrix: BLOSUM62, special: |wild base mutations|

Alignment score: 3630
Alignment: from 1[p]|1[n]  to 707[p]|2118[n]
Length of the alignment:706[710]

     1 MetSerSerAlaLeuArgSerArgAlaArgSerAlaSerLeuGlyThrThrThrGlnGlyTrpAspProProProLeu
       ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
     1 AUGUCCUCCGCACUUCGGUCUCGGGCUCGCUCGGCCUCGCUCGGAACGACGACUCAGGGCUGGGAUCCGCCGCCAUUG
...
   365 LeuThrValTrpMetSerGlyLeuIleArgThrArgAlaAlaGlyGluGlnGlnGlnProProSerLeuValGlyThr
       |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||-12
  1093 CUCACGGUCUGGAUGUCGGGCCUGAUCCGCACGCGCGCCGCUGGCGAGCAACAGCAGCCGCCGUCGCUGGUGGGCACG
...
   573 LeuAspGluValArgMetGlyThrGluAlaLeuLeuPheLysHisAlaGlyAlaAlaCysArgAlaLeuGluAsnGly
       |||||||||||||||||||||||||+++|||||||||||||||||||||||||||||||||||||||||||||||||
  1717 CUGGACGAGGUGCGCAUGGGUACGCAGGCGUUGCUCUUUAAGCACGCCGGCGCGGCCUGCCGCGCGUUGGAGAACGGC

   599 LysLeuThrHisCysSer------CysLeuLeuIleLeuAlaAlaGlnMetSerTyrGlyAlaCysLeuLeuGlyGlu
       ||||||||||||||||||IIIIII|||||||||||||||||||||||||||||||||||||||||||||||||||||
  1795 AAGCUCACGCACUGCUCCGACGCCUGUCUGCUCAUUCUGGCGGCGCAAAUGAGCUACGGCGCCUGUCUCCUGGGCGAG

...

# of identical matches: 699(99.01%)
# of positive matches: 3(00.42%)
# of gaps: 5[# of INS:2 and # of DELS: 3](00.71%)

Mutations:
# of INS: 2 [positions of INS: 631(X:-)  631(X:-)  ]
# of DEL: 3 [positions of DEL: 130(-:V)  131(-:V)  132(-:P)  ]
# of REP: 0 [positions of REP: ]
# of FSM: 2 ( sfs: 1  and dfs: 1)  [positions of SFS: 535(SFS:X)  ] and [positions of DFS: 390(DFS:T)  ]

Warning!
detected STOP-codon in the alignment at 631(Stop:*)

# of wild bases: 4 ( K=1, S=1, Y=1, M=0, W=0, R=0, B=0, D=0, H=0, V=0, N=1 )
```

Abbildung 60: Gekürzte Ausgabe des Alignment-Algorithmus des erweiterten Verfahrens: Namen, Längen, Parameter, Bewertung, sowie die aufgeschlüsselten Alignmentpositionen für die Protein und RNS-Sequenz werden ausgegeben. Danach erfolgt die Quantifizierung der exakten und positiven Matches, der Lücken, der verschiedenen Mutationen und auch der Wild-Basen. Im aufgeschlüsselten Alignment markieren "|" exakte Matches, "+" positive Matches, I Insertionen und D Deletionen. $-n$ oder $-n_1n_2$ für $n, n_1, n_2 \in \{1..5\}$ stehen für Frameshiftdeletionen, wobei n bzw. n_1 und n_2 für die Basenposition stehen, welche aus dem Alignment gelöscht worden sind (siehe Position 390 am Ende der zweiten Zeile des aufgeschlüsselten Alignments). i steht für eine Frameshiftinsertion. Ein $*$ markiert einen Mismatch.

ERWEITERTE ALIGNMENTANALYSE

Der Algorithmus für das erweiterte Verfahren aus Abschnitt 2.6.3 ermittelt ein optimales Alignment. Die Analyse dieses Alignments erfasst neben den positiven Matches, Mismatches, Insertionen und Deletionen nun ebenfalls Frameshiftmutationen, Stopcodonmutationen und vorkommende Wild-Basen (siehe Ausgabe in Abbildung 60). Wie auch schon im ursprünglichen Verfahren erfolgt eine Analyse des gefundenen Alignments (entspricht Phase 2 des ursprünglichen Verfahrens). Ein Parser ermittelt erneut die positiven Matches, Mismatches, Insertions- und Deletionsmutationen, sowie die Frameshift- und Stopcodonmutationen. Zusätzlich zu den Angaben zur Länge, Start- sowie Endposition des gefundenen Alignments wird auch die Anzahl der unterschiedlichen Wild-Basen ermittelt. Der Parser erstellt ebenfalls eine json-Datei (vgl. Abbildung 57 und siehe Abbildung 61).


```
{"mutations":[{"position":75,"subject":"T","query":"A"},{"position":520,"subject":"H","query":"Q"},
 "insertions":[{"begin":631,"end":631},{"begin":631,"end":631}],
 "deletions":[{"begin":130,"end":130},{"begin":131,"end":131},{"begin":132,"end":132}],
 "single frame-shifts":[{"begin":535,"end":535}],
 "double frame-shifts":[{"begin":390,"end":390}],
 "stop-codons":[{"begin":631,"end":631}],
 "WB":[{"K":1},{"S":1},{"Y":1},{"N":1}],
 "length":"710",
 "start":"1","end":"707",
 "found":true}
```


Abbildung 61: Inhalt einer json-Datei für das erweiterte Verfahren: Neben den Kategorien für Mutationen (positive Matches oder Mismatches, Insertionen, Deletionen, Frameshift- und Stopcodon-Mutationen) enthält die Datei die Anzahl und Art der Wild-Basen, so diese in der untersuchten Sequenz vorkommen.

ERWEITERTER MUTATIONS-RESISTENZ-BERICHT

Für die zusätzlich erfassten Mutationsarten lassen sich weitere Tabellen erstellen und der Alignmentgraph kann ebenfalls erweitert werden, um die zusätzlichen Mutationsarten anzuzeigen (siehe Tabelle 5 und 6 und Abbildung 62).

Tabelle 5: Tabelle der Frameshiftmutationen: Neben dem Laufindex werden die Art der Frameshiftmutation, also die Verschiebung um ein (=single) oder zwei (= double) Nukleotide und die Region innerhalb der alignierten Proteinsequenz angegeben.

S/N	single / double frameshift	Protein Region
1	single frameshift	535
2	single frameshift	718
3	double frameshift	199

Tabelle 6: Tabelle der Stopcodonmutationen: Jede gefundene Stopcodonmutation innerhalb des Alignments wird mit Laufindex und Position in der Protein-Kette versehen.

S/N	stopcodon	Protein Region
1	stopcodon	1142
2	stopcodon	1598

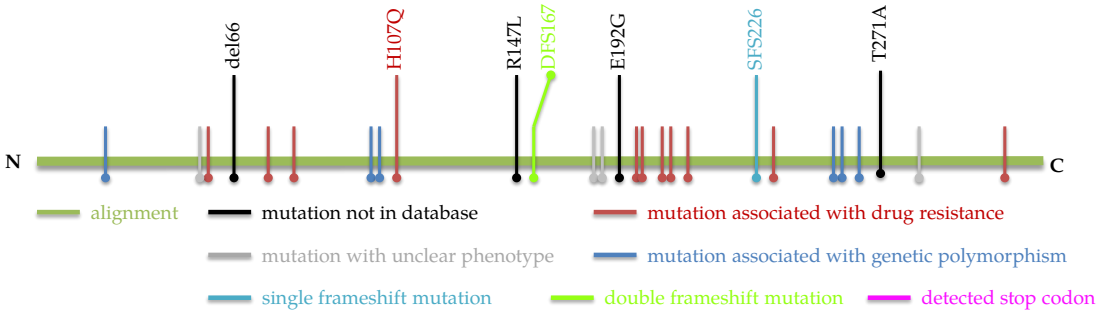


Abbildung 62: Graph eines erweiterten Alignments: Die gefundenen Mutationsstellen werden je nach Art der gefundenen Mutation farblich markiert.

2.6.4 *Swat*

Als "Proof-of-Concept" und für den Einsatz in einer entsprechenden Webapplikation wurde der im erweiterten Verfahren vorgestellte Alignment-Algorithmus *Swat* (Smith-Waterman extended Alignment Tool) implementiert [25]. Die Implementierung des Algorithmus umfasst neben anpassbaren Parametern auch bereits die Analysephase der Alignmentausgabe (siehe Abschnitt 2.6.3).

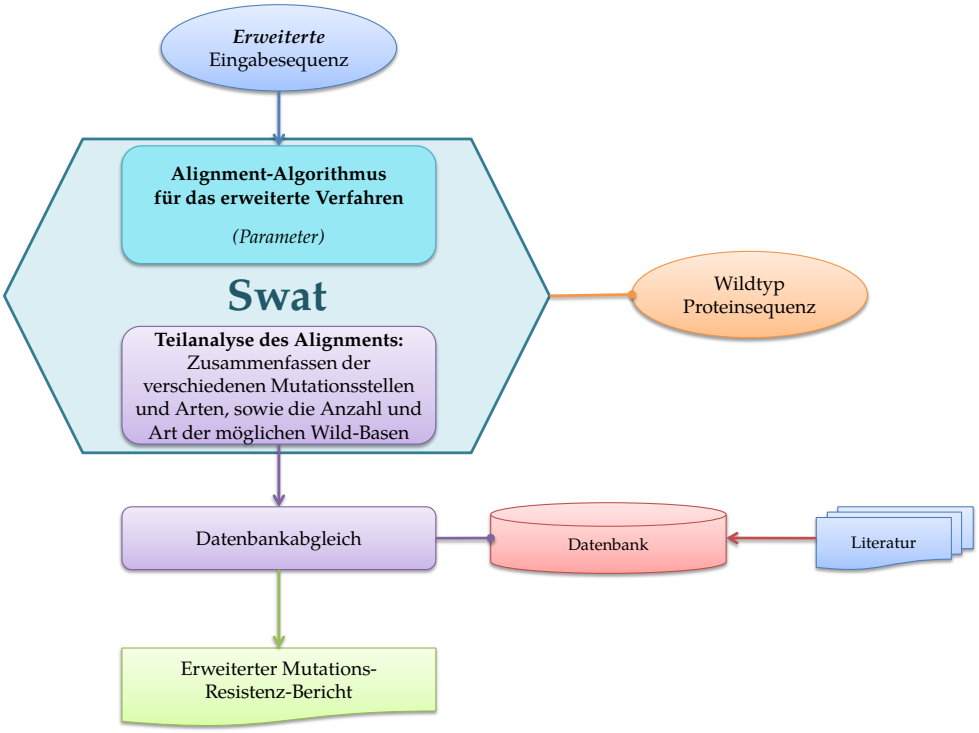


Abbildung 63: Schematische Darstellung der Integration des *Swat*-Algorithmus in das erweiterte Verfahren: Der Algorithmus erstellt beim Backtracking des Alignments die Sequenzen und erfasst dabei die mutierten Stellen ebenso wie die Wild-Basen. Ein erneutes Parsen des Ergebnisses ist daher nicht mehr nötig. Im Anschluss erfolgt der Abgleich mit der Datenbank und die Aufarbeitung der Ergebnisse im Mutations-Resistenz-Bericht.

2.6.4.1 *Parameter*

Swat besitzt eine Reihe von Parametern, welche für die Konfiguration des Alignments angepasst werden können (siehe Tabelle 7).

Tabelle 7: Liste der *Swat*-Parameter: Dem Parameter folgt eine kurze Beschreibung, der Gültigkeitsbereich und der Standartwert, welche verwendet wird, wenn der Parameter nicht explizit gesetzt wird.

Parameter	Bedeutung	Gültigkeitsbereich	Standard
g =	Malus für das Starten einer Lücke im Alignment	-10000 bis -1	-11
e =	Malus für das Erweitern einer Lücke im Alignment	-10000 bis -1	-1
s =	Malus für einen einfachen Frameshift	-10000 bis -1	-20
d =	Malus für einen doppelten Frameshift	-10000 bis -1	-40
p =	Malus für Mismatch mit Stop-Codon	-10000 bis -1	-4
m =	zu verwendende Scoringmatrix	-	BLOSUM62
ng	Gaps sind im Alignment nicht erlaubt	Ja / Nein	Nein
na	Es werden keine affinen Gaps verwendet	Ja / Nein	Nein
nf	Es werden keine Frameshiftmutationen erlaubt	Ja / Nein	Nein
w	Finde ein optimales Alignment, mit den schlechtesten Bewertungen für wild base-Stellen	Ja / Nein	Nein
o =	Speichere gefundene Mutationen in einer JSON-Datei	Ja / Nein (+Dateiname)	Nein

Die voreingestellten Standard-Parameter entsprechen den Einstellungen der verbreiteten DNS-Protein-Alignmentprogrammen (Blastx, Fasty etc.). Eine Evaluation geeigneter Parameter wurde für die Scoringmatrix und Gap-Penalties in [64] und für Frameshifts in [54] vorgestellt.

BESCHREIBUNG DER EINZELNEN PARAMETER

`g` Der Kosten-Parameter für den Start bzw. das Öffnen einer Lücke. Die Höhe des Malus sollte auf die Scoringmatrix abgestimmt werden. Ein zu kleiner Malus lässt das Alignment entarten, indem keine Mismatches mehr berechnet werden, sondern an dieser Stelle Lücken entstehen. Ein zu großer Malus lässt unter Umständen keine Lücken zu und es werden Mismatches bevorzugt.

`e` Dieser Parameter gibt die Kosten für eine Aminosäure-Lücke an. Im Normalfall ist $e < g$, um die Alignmentberechnung mit affinen Lücken durchzuführen.

`s` Für die Konfiguration der Single-Frameshift-Kosten kann der Parameter `s` verwendet werdn. Hierfür muss die Formel 9 eingehalten werden, für ein optimales Alignment.

`d` Analog zu den Kosten für den einfachen Frameshift verhält es sich beim doppelten Frameshift-Malus-Parameter `d`.

`p` Mit diesem Parameter werden die Kosten für eine Stopcodon-Mutation innerhalb des Alignments festgelegt.

`m` Die Scoring- bzw. Kostenmatrix ist ebenfalls als Parameter wählbar. Neben den bereits integrierten PAM und BLOSU-Matrizen können beliebige weitere Scoring-Matrizen (siehe auch [65]) verwendet werden.

`ng` Ist dieser Parameter aktiv, sind Lücken im Alignment nicht erlaubt. So entsteht ein lückenloses lokales Alignment, welches nur aus Matches und Mismatches besteht. Frameshiftmutationen sind weiterhin im Alignment erlaubt.

`na` Soll jede Lücke unabhängig vom zuvor stattgefundenen Alignment bewertet werden, so kann der `na`-Parameter verwendet werden. Es werden also keine affinen Gapkosten verwendet.

`nf` Ist für ein Alignment bekannt, dass die zu untersuchenden Sequenzen keine Frameshifts enthalten, so kann mit dem Parameter `nf` die Berücksichtigung und Bewertung von Frameshiftmutationen ausgesetzt werden.

`w` Da als Option wählbar ist, ob das beste oder schlechteste optimale Alignment in DNS-Sequenzen mit Wild-Basen berechnet werden soll, kann über den `w`-Parameter festgelegt werden, ob das worst case oder best case Alignment durchgeführt werden soll.

o Der Parameter o erstellt eine json-Ausgabe der mutierten Stellen. Abbildung 61 zeigt eine solche Ausgabe. Diese kategorisierte Ausgabe kann in eine Datei gespeichert werden, so ein Dateiname angegeben wird und zur späteren Verarbeitung verwendet werden.

2.7 GENERISCHER WEBSEITEN-ANALYZER

Der in dieser Arbeit entwickelte Algorithmus *Swat* wurde in eine generische Webanwendung zur Virusresistenz-Analyse integriert. In Kapitel 2.6 wurde ein Verfahren vorgestellt, um Viren auf Resistenzen gegenüber bestimmten Medikamenten zu untersuchen. Dieses Verfahren wurde in eine Webapplikation integriert, um mögliche Resistenzen des Humanen Cytomegalievirus (HCMV) auf den zwei Proteinsequenzen UL97 und UL54 zu untersuchen [10]. Zwei weitere Webseiten mit dem Verfahren wurden für das Hepatitis-C-Virus (HCV) und das Herpes-simplex-Virus 1 (HSV) erstellt. Der für das Alignment zu Beginn eingesetzte der Blastx-Algorithmus wurde durch den auf den vorgestellten Erweiterungen basierenden *Swat*-Algorithmus ersetzt. Im nächsten Schritt wurde ein Framework implementiert, welches generisch für beliebige Gruppen von Sequenzen eingesetzt werden kann. Dazu wurden die spezifischen (von der jeweiligen Sequenz abhängigen) Informationen und Menüpunkte verallgemeinert und statt als seperate Unterseite als Auswahloption innerhalb der Webanwendung aufbereitet. Dadurch ist das Web-Framework auf beliebige Sequenzen und Sequenzgruppen erweiterbar und kann ebenfalls entsprechende Resistenzdatenbanken ansprechen (siehe Abbildung 64).

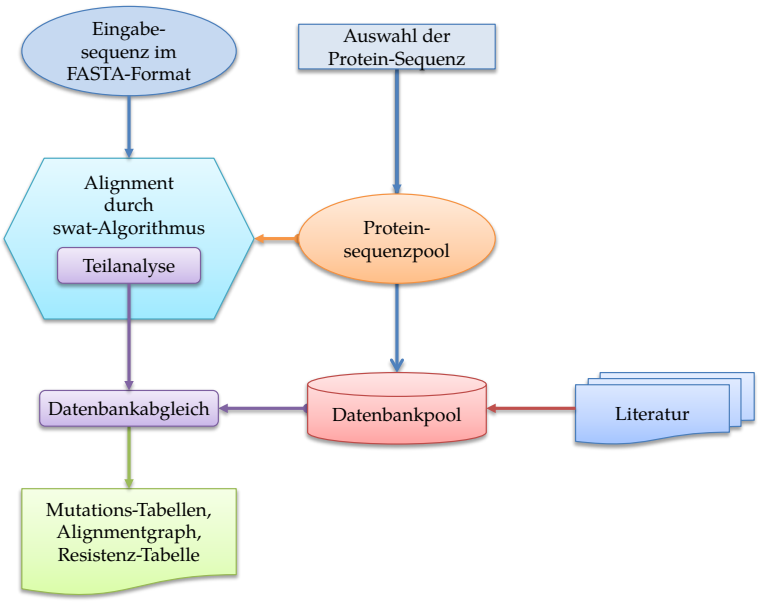


Abbildung 64: Schematische Darstellung der generischen Webanwendung: Eine Eingabesequenz wird mit einer Proteinsequenz aus dem Sequenzpool mit Hilfe des *Swat*-Algorithmus aligniert. Das Ergebnis wird mit den Datenbankeinträgen der zuvor ausgewählten Proteinsequenz abgeglichen. Gefundene Mutationen und Resistenzen werden in tabellarischer Form und als Alignmentgraph als Ergebnis dargestellt.

Zentraler Punkt der Webanwendung ist die Programmierschnittstelle (*api*). In dieser finden sich alle Basisfunktionen und -Methoden, welche für die Webseiten programmiert wurden. Die Basis-Klasse importiert externe Methoden sowie Layoutvorgaben und Menüstruktur. Neben den Webseiten für Kontaktaufnahme (*contact*), Fehler (*error*), Zusatzinformationen (*supple*), Nutzungsbedingungen (*terms*)und Updateinformationen (*update*), findet sich die Eingabemaske für Sequenzen auf der Formularwebseite (*form*). Der Alignmentalgorithmus wird nach der Übermittlung der Daten aufgerufen und das aufgearbeitete Ergebnis auf der Ergebniswebseite (*result*) dargestellt. Hierfür werden Komponenten aus dem Bild-Generator für den Alignmentgraphen (*ResultImage*) und aus dem Tabellen-Generator (*table*) benötigt. Um die Ergebnisse als PDF (Portable Document Format) zu speichern steht ebenfalls ein PDF-Export zur Verfügung. Die Kombination dieser Komponenten ergibt die Struktur der Webanwendung (siehe Abbildung 65).

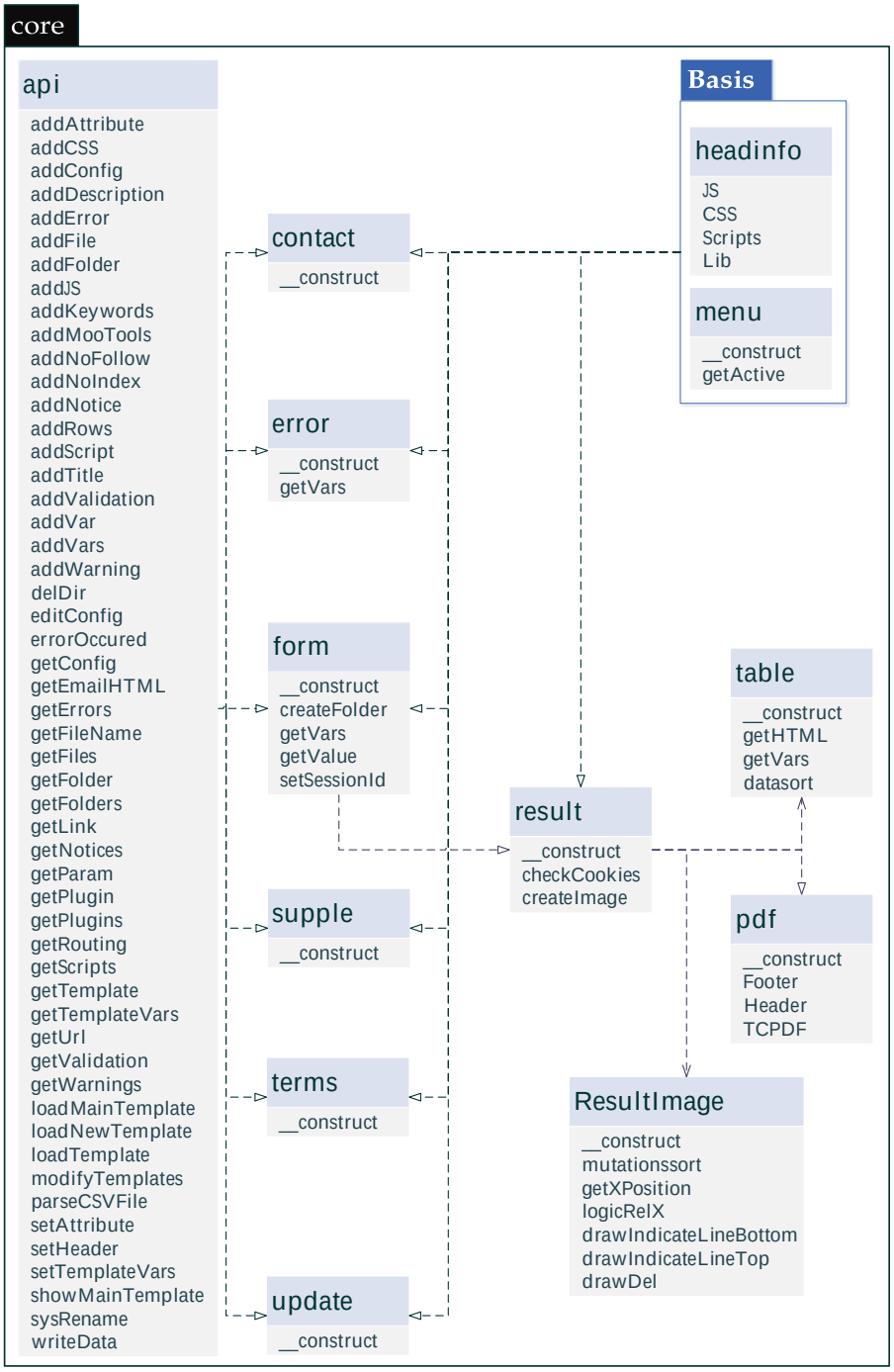


Abbildung 65: Schematische Sitemap und Abhängigkeiten der einzelnen Seiten und Klassen: Zentraler Punkt ist die api, welche in jeder Seite eingesetzt wird. Die drei Basis-Klassen bilden das grundsätzliche Layout und die Menü-Struktur, sowie die Importe aus CSS, Javascript und php. In der tieferen Ebene der *form* werden Alignment-Algorithmus und *result* aufgerufen. *result* wiederum benötigt *ResultImage, table und pdf* zur kompletten Darstellung des Ergebnisses und dessen Variationen (Alignmentgraph, Tabellen und PDF-Export).

Die Webanwendung lässt sich in eine hierarchische Ordnerstruktur ablegen (siehe Abbildung 66).

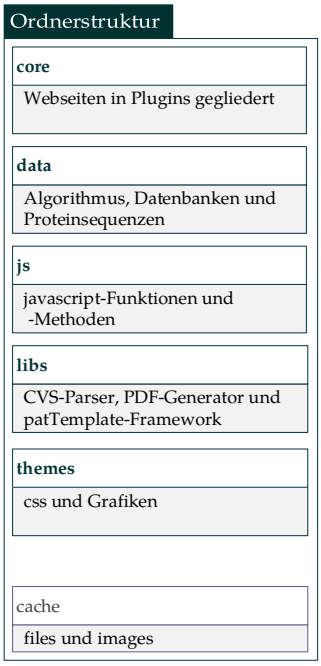


Abbildung 66: Ordner-Struktur der Webanwendung. Zur Organisation wurden die unterschiedlichen Bereiche der Webanwendung in Ordner eingeteilt. Neben dem in Abbildung 65 im Detail beschriebenen Kern der Anwendung *core*, gibt es für Algorithmus, Sequenzen und Datenbanken den Ordner *data*, in *js* und *libs* Javascript-Methoden und Funktionen, sowie php-Hilfsframeworks für CSV-Verarbeitung und PDF-Erstellung, sowie ein Template-Framework für die Seiten, um diese Sprachabhängig zu gestalten. Der *themes*-Ordner kann genutzt werden, um das Look-and-Feel der Webseiten zu verändern. Der *cache*-Ordner dient dem Speichern der flüchtigen Daten (Alignmentauswertungen und Grafiken)

Neben dem eben im Detail beschriebenen Kern der Anwendung (*core*) gibt es einen Ordner für den Algorithmus, die Proteinsequenzen und die Resistenzdatenbanken (*data*). Auf der selben Ebene befinden sich die Ordner für JavaScript-Methoden und -Funktionen (*js*), php-Hilfsframeworks (*libs*) und Gestaltungs-Frameworks (*themes*).

Die generische Webanwendung kann mit verschiedenen Sequenz- bzw. Proteingruppen umgehen. Es wurden die spezifischen Sequenzseiten aus der ursprünglichen Webanwendung durch eine dynamische Liste ersetzt. Diese Liste wird beim

Aufruf des Analyzer-Formulars generiert, indem ein Verzeichnis auf die Sequenzen untersucht wird. Dabei werden der Sequenzname und die zugehörigen Informationen aus dem FASTA-Format ausgelesen und in einer Sequenzliste zur Auswahl auf dem Formular bereitgestellt. In diesem Zusammenhang wurde auch die Ordnerstruktur für die Daten erweitert.
Die unterschiedlichen Elemente (Algorithmus, Sequenzen und Resistenzdatenbank), welche zuvor in einem Verzeichnis abgelegt waren, sind nun unterteilt in die entsprechenden Kategorien, um bei einer großen Anzahl an Sequenzen und Datenbanken die Übersicht zu erhalten. Zusätzlich können weitere Scoring-Matrizen und auch verschiedene Versionen des Alignmentalgorithmus hinterlegt werden. Weiterhin wurden alle Sequenz-abhängigen Parameter von der Übergabe über die Adresszeile des Browsers auf interne Variablen umgestellt, um die Fehleranfälligkeit zu reduzieren. Da die Zeit für die Berechnung des Alignments durch den dynamischen Programmieransatz des *Swat*-Algorithmus erhöht wurde, wurde ein Prozess-Screen erstellt, welcher angezeigt wird, solange der Algorithmus im Hintergrund arbeitet. So wird ein kontinuierliches Feedback an den Anwender gewährleistet.

3

ERGEBNISSE

Um die *Swat*-Implementierung zu testen und mit Algorithmen wie Fasty und Blastx zu vergleichen, wurden unterschiedliche Simulationen durchgeführt. Dazu wurde eine Testumgebung implementiert, welches es ermöglicht, Sequenzen mit unterschiedlichen Mutationen zu generieren und Werte für Alignmentparameter auf unterschiedlichen Intervallen anzugeben. So generierte Sequenzreihen können an unterschiedliche Alignmentprogramme mit den gewählten Parametern oder Parameterintervallen übergeben werden. Ebenso bietet das Framework eine Klasse, welche das Sammeln und Vergleichen der Ergebnisse unterstützt. Als weitere Simulation wurden Wild-Basen in Sequenzreihen erzeugt und deren Einfluss auf die Bewertung des Alignments gemessen.

MUTATIONS-GENERATOR

Für eine möglichst große Anzahl an unterschiedlichen Sequenzen, welche nur bestimmte Mutationen oder einen Mix aus allen Mutationen enthalten können, wurde ein Generator implementiert, welcher in eine Eingabesequenz verschiedene Mutationen auf Sequenzreihen einfügen kann (siehe Abbildung 67).

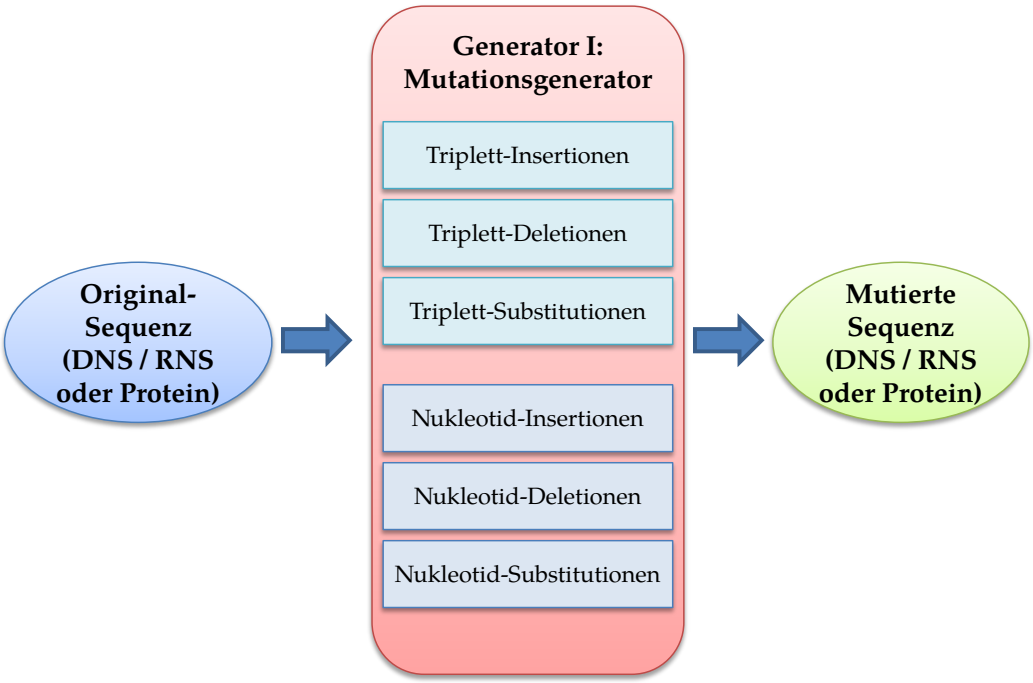


Abbildung 67: Ablaufdiagramm des Mutationsgenerators: Die originale Sequenz (DNS / RNS oder Protein) wird mit Hilfe von Mutationsmethoden verändert und erhält – in Abhängigkeit des gewünschten Rauschanteils – Aminosäure- und/oder Nukleotid-Mutationen in Form von Insertionen, Deletion und / oder Substitutionen. Die Ausgabe ist die mutierte Sequenz oder komplette mutierte Sequenzreihen.

Der Mutations-Generator ist so konzipiert, dass sich für beliebige Stellen innerhalb einer Nukleotidsequenz und einem vorgegebenen Prozentsatz (separat) Mutationen in Form von Insertionen, Deletionen oder Substitutionen in der DNS- bzw. RNS-Sequenz oder einer Proteinsequenz generieren lassen. Dabei kann zusätzlich unterschieden werden, ob es sich um Triplett-Mutationen oder um Nukleotid-Mutationen handeln soll.

WILD-BASEN-GENERATOR

Um Wild-Basen-Rauschen in Nukleotidsequenzen zu erzeugen, wurde der Wild-Basen-Generator implementiert (siehe Abbildung 68).

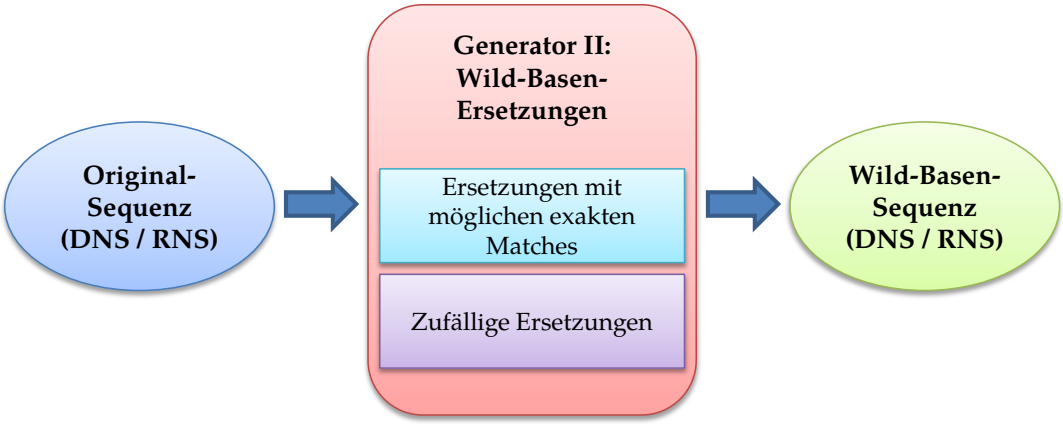


Abbildung 68: Ablaufdiagramm des Wild-Basen-Generators: In einer Nukleotidsequenz werden einzelne Basen durch Wild-Basen ersetzt. Diese Ersetzungen können Wild-Basen sein, welche das originale Nukleotid ebenfalls repräsentieren, oder willkürlich gewählt sein.

Mit ihm lassen sich Sequenzreihen erzeugen, welche zu einem vorgegebenen Prozentsatz aus Wild-Basen bestehen. Dabei kann unterschieden werden, ob die in die Sequenz eingesetzten Wild-Basen das substituierte Nukleotid ebenfalls repräsentieren oder ob eine willkürliche Substitution stattfinden soll.

PROZESSOR

Zur Automatisierung der Alignmentprozesse für Sequenzreihen wurde ein Framework implementiert, welches nach Festlegung der Parameter und der Eingabesequenzen die Berechnungen durchführt und die Ergebnisse entsprechend der Eingaben ablegt (siehe Abbildung 69).

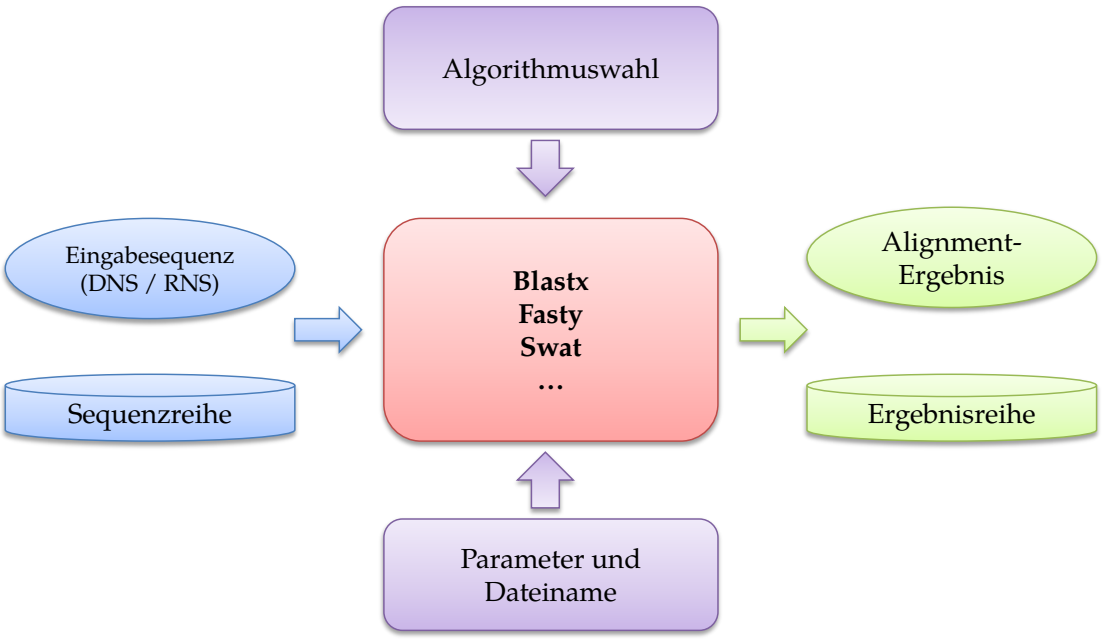


Abbildung 69: Ablaufdiagramm des Prozessors: Für eine Eingabesequenz bzw. komplette Sequenzreihen werden Algorithmus und Parameter, sowie Name der Ausgabedatei bzw. das Namensschema der Ausgabedateien gewählt. Danach wird über den automatisierten Kommandozeilenaufruf das Alignmentergebnis generiert. Weiter Algorithmen können ebenfalls in das System integriert werden, so diese über die Kommandozeile ausführbar sind.

Die betrachteten Algorithmen (hier Blastx, Fasty, *Swat*) lassen sich über die Unix-Kommandozeile aufrufen. Die zu untersuchenden Sequenzen, die Parameter und auch den Namen der Ausgabedateien lassen sich übergeben und erlauben eine Automatisierung der Alignment-Prozesse. Der Prozessor kann für andere Programme erweitert werden, so diese den Aufruf via Kommandozeile zulassen.

COLLECTOR

Das Collector-Programm übernimmt die Aufgabe, die Alignmentergebnisse der Sequenzen bzw. Sequenzreihen einzusammeln und in einer Datei abzulegen (siehe Abbildung 70).

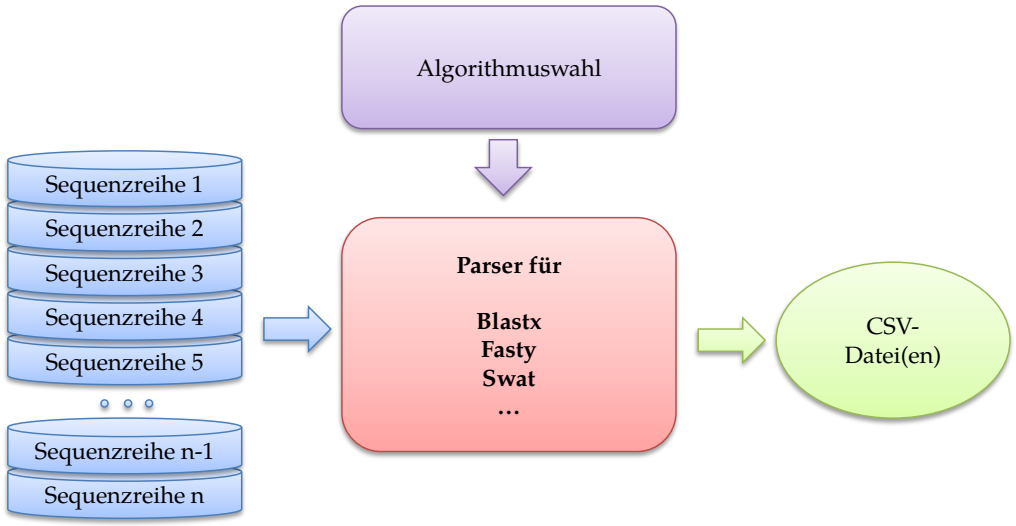


Abbildung 70: Ablaufdiagramm des Collectors: Die Ergebnisse der Sequenzreihen verschiedener Algorithmen werden geparst und im Anschluss in einer CSV-Datei gespeichert.

Für die einzelnen Alignmentalgorithmen durchlaufen individuelle Parser die Alignmentergebnisse und erfassen die gewünschten Informationen. Dabei kann es sich um den Alignment-Score handeln, die verwendeten Parameter oder auch die Anzahl der gefundenen Lücken im Alignment. Als Ergebnis erzeugt dieses Programm eine CSV (comma seperated value) -Datei.

3.1 SIMULATIONEN

Zur Einordnung der Güte des *Swat*-Algorithmus und zur Untersuchung, wie Wild-Basen die Bewertung eines Alignments beeinflussen, wurden unterschiedliche Simulationen durchgeführt.
Für die ersten zwei Simulationen werden Sequenzreihen erzeugt, welche entweder Triplett-Mutationen oder Nukleotid-Mutationen beinhalten. Die so erzeugten Sequenzreihen werden zur Alignierung an die drei Algorithmen Blastx, Fasty und *Swat* übergeben und die gesammelten Ergebnisse ausgewertet.
Die dritte Simulation vergleicht die worst- und best-case Alignmentergebnisse für Wild-Basen-Sequenzen. Hierfür werden Wild-Basen-Sequenzreihen mit Hilfe des Generators II erzeugt und im Anschluss analysiert.
In der letzten Simulation werden reale Daten zum Vergleich herangezogen, um zu

evaluieren, ob der neue Algorithmus auch damit bessere Ergebnisse erzielt.
In allen Simulationen wird der Smith-Waterman-Score als Vergleichsgröße herangezogen. Erzeugen die Algorithmen ein äquivalentes Alignment (nicht zwangsläufig identisch), so ist der Smith-Waterman-Score gleich, falls identische Parameter zur Berechnung des Alignments angenommen wurden.
Für die maschinell generierten Testreihen wurde die Referenz (Wild-Typ)-DNS-Sequenz der UL97-Proteinsequenz verwendet. Die Länge der Proteinsequenz ist 707 Aminosäuren. Dieser gegenüber steht die Nukleotid-Sequenz mit einer Länge von 2121 Basen. Unverändert stimmt diese Nukleotidsequenz perfekt mit der Wild-Typ-Proteinsequenz überein. Der Smith-Waterman-Score der hierbei mit den Standardparametern erreicht wird ist 3731.

TRIPLETT-MUTATIONEN

In der ersten Simulation werden Sequenzreihen aligniert, welche 1%, 5%, 10%, 15% oder 20% zufällige Triplett-Mutationen (Rauschen) aufweisen. Diese werden in Form von Insertionen, Deletionen, Substitutionen und schließlich ein Mix aus all diesen Veränderungen auf je 1000 Sequenzen generiert. Hierbei zeigt sich, dass die Algorithmen im Falle der Insertionen ein äquivalentes Alignment berechnen (siehe Abbildung 71).

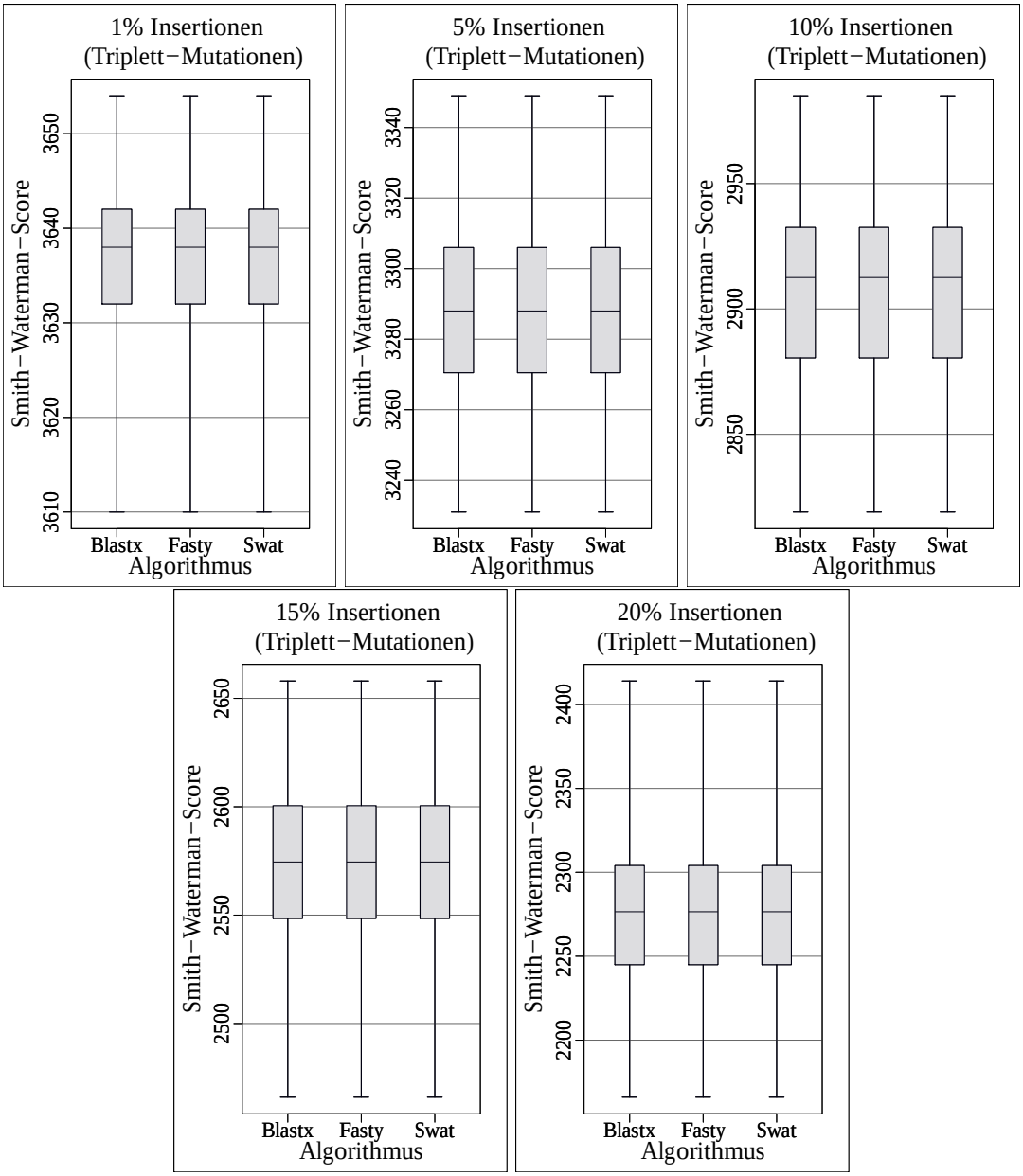


Abbildung 71: Ergebnisse der Insertions-Simulation: Für Sequenzreihen mit $n = 1000$ Sequenzen mit je 1%, 5%, 10%, 15% oder 20% Triplett-Mutations-Rauschen wurden die lokalen optimalen Alignments mit den Algorithmen Blastx, Fasty und *Swat* berechnet. Alle drei Algorithmen schneiden gleich ab, finden also äquivalente Sequenzalignments.

Im Falle der Deletionen bestimmen Blastx und *Swat* auch bei hohem Rauschanteil noch zu einem sehr hohen Prozentsatz äquivalente Alignments, lediglich Fasty errechnet ab einem Rauschanteil von 10% Alignments mit einer schlechteren Bewertung (siehe Abbildung 72).

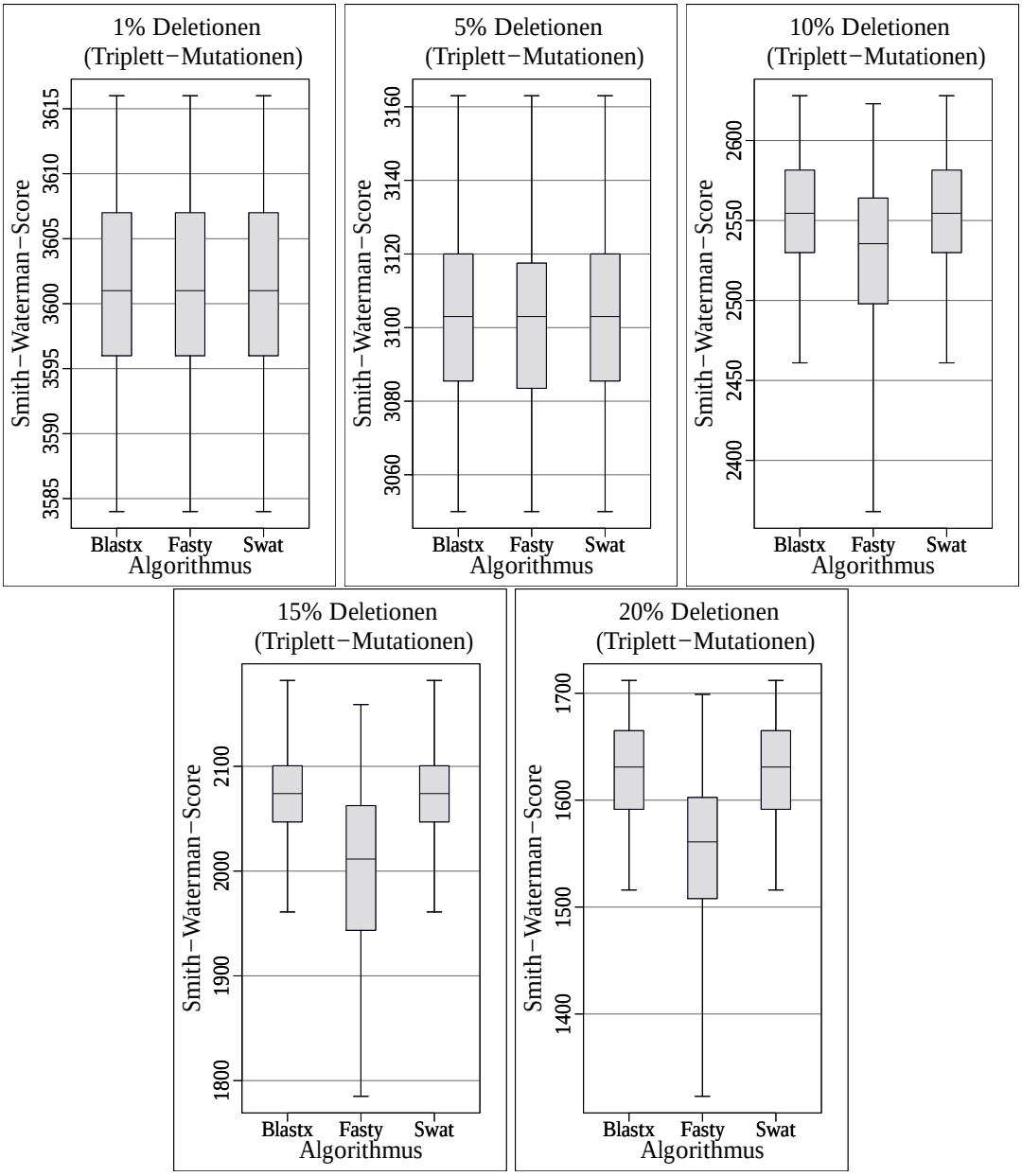


Abbildung 72: Ergebnisse der Deletions-Simulation: Es wurden je $n = 1000$ Sequenzen mit den Rauschanteilen von 1%, 5%, 10%, 15% und 20% an die drei Algorithmen Blastx, Fasty und *Swat* übergeben und die Alignment-Bewertungen verglichen. Fasty errechnet ab einem Rauschanteil von 10% in einigen Fällen Alignments mit schlechterer Bewertung als Blastx bzw. *Swat*. Blastx schneidet nur minimal schlechter im Vergleich zu *Swat* ab (1-2 Punkte) bei 15% bzw. 20% Rauschanteil; dies jedoch nur bei 0.4% bzw. 0.9% der Sequenzen. In wenigen Fällen berücksichtigt also Blastx einen einzelnen seed nicht und erreicht damit nicht die höchst mögliche Bewertung für diesen Fall.

Bei der Substitutions-Simulation berechnen bei geringem Rauschen alle drei getesteten Algorithmen ebenfalls äquivalente Alignments, schlechtere Bewertungen treten erst ab einem Rauschanteil von 15% bei Fasty auf (siehe Abbildung 73).

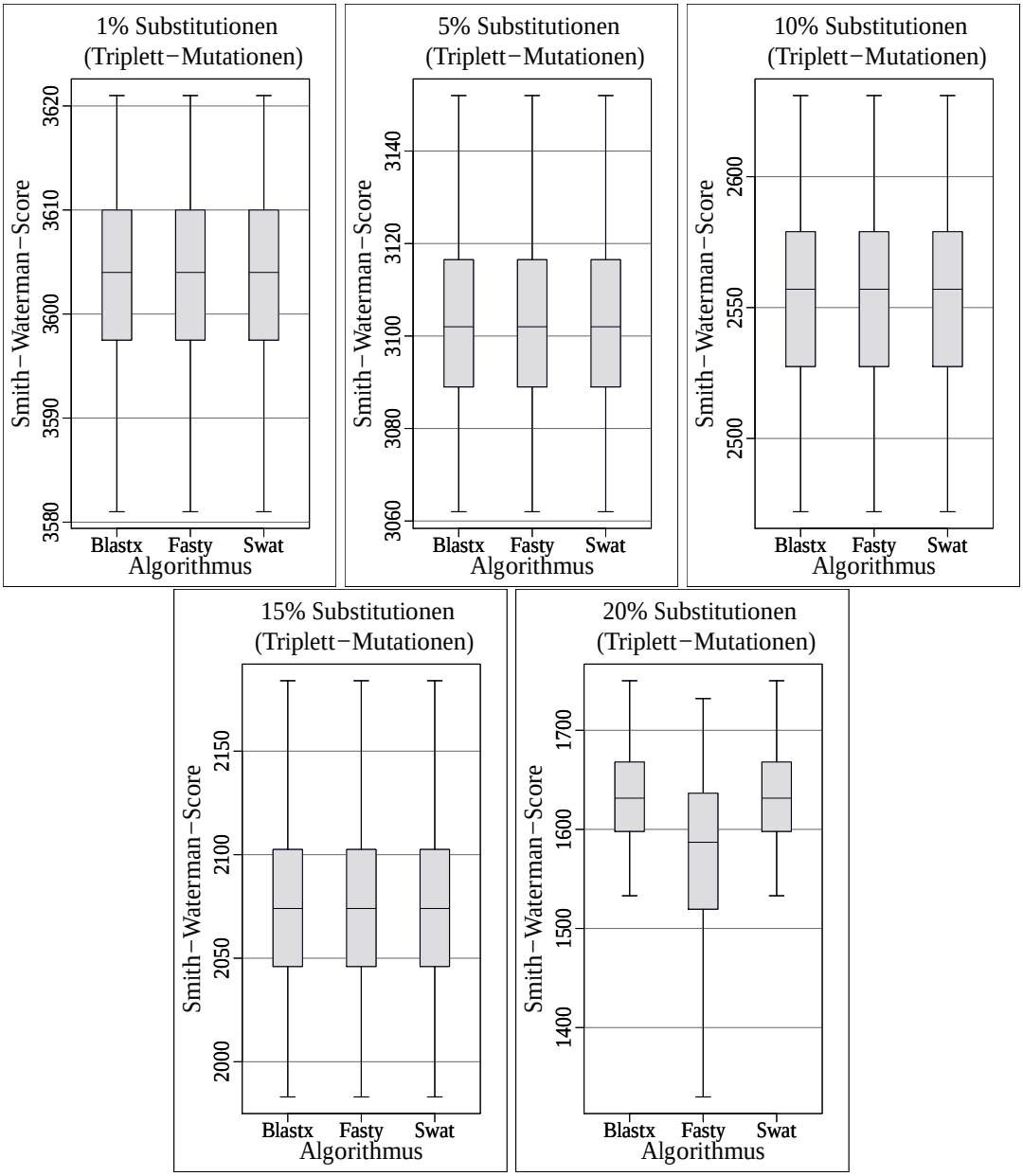


Abbildung 73: Ergebnisse der Substitutions-Simulation: Die drei Algorithmen Blastx, Fasty und *Swat* berechnen bis zu einem Rauschanteil von 15% bei je $n = 1000$ Sequenzen identische Ergebnisse und somit äquivalente Alignments. Lediglich bei 20% Rauschanteil errechnet Fasty in einigen Fällen niedrigere Alignmentergebnisse als Blastx und *Swat*.

Werden alle drei Mutationsarten vermischt, so berechnen Blastx und *Swat* bis zu einem Rauschanteil von 15% äquivalente Alignments. Ab 20% Rauschen findet *Swat* bei ca. 1% der Sequenzen ein höher bewertetes Alignment als Blastx. Fasty hingegen findet im Vergleich zu den beiden anderen Algorithmen ab einem Rauschanteil von 10% schlechter bewertete Alignments (siehe Abbildung 74).

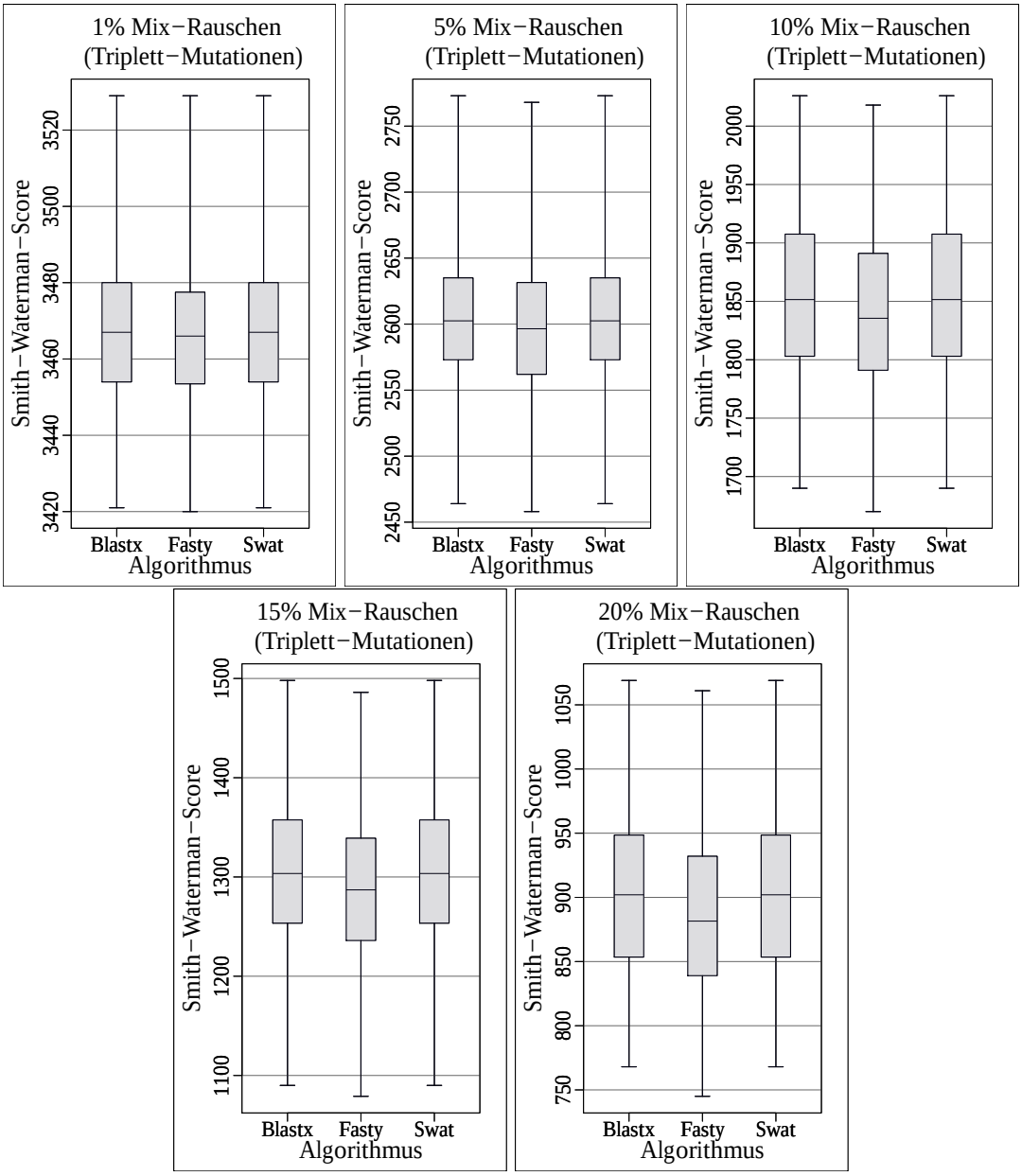


Abbildung 74: Ergebnisse der Mix-Simulation: Wie durch die vorangegangenen Simulationen zu erwarten war, schneidet Fasty schlechter ab, als Blastx oder *Swat*, wenn die verschiedenen Aminosäure-Mutationsarten vermischt werden. Erneut wurden $n = 1000$ Sequenzen mit den verschiedenen Alignmentalgorithmen aligniert.

Die erste Simulation zeigt, dass die Algorithmen bei Insertionen ein äquivalentes Alignment finden. Bei Substitutionen schneidet der fasty-Algorithmus bei starkem Rauschen ($\geqslant 10\%$) schlechter ab. Ähnlich verhält es sich bei den Deletionen und bei dem Mix aller Mutationen. Blastx und *Swat* finden bis zu einem Rauschan-

teil von 20% bei Insertionen und Substitutionen äquivalent optimale Alignments. Lediglich bei einem Rauschanteil von 20% bei Deletionen und bei der Mischung der Triplett-Mutationen findet *Swat* bei ca. 1% der Sequenzen ein Alignment mit höherer Bewertung.

NUKLEOTID-MUTATIONEN

Die zweite Simulation testet Nukleotid-Mutationen statt Triplett-Mutationen. Es werden zufällig ein oder zwei Nukleotide verändert, wodurch Frameshiftmutationen entstehen. Die Rauschanteile von 1%, 5%, 10%, 15% und 20% werden analog zur ersten Simulation gewählt. Der Blastx-Algorithmus unterstützt keine Frameshiftmutationen, dadurch wird ein optimales Alignment durch diesen Algorithmus ohne Berücksichtigung dieser Mutationsart berechnet. Die Alignmentbewertungen der Nukleotid-Insertions-Simulation zeigen, dass Fasty und *Swat* äquivalente oder nahezu äquivalente Alignments finden, selbst bei einem Rauschanteil von 20%. Blastx hingegen berechnet bereits bei 1% Rauschanteil ein wesentlich niedriger bewertetes Alignment (siehe Abbildung 75).

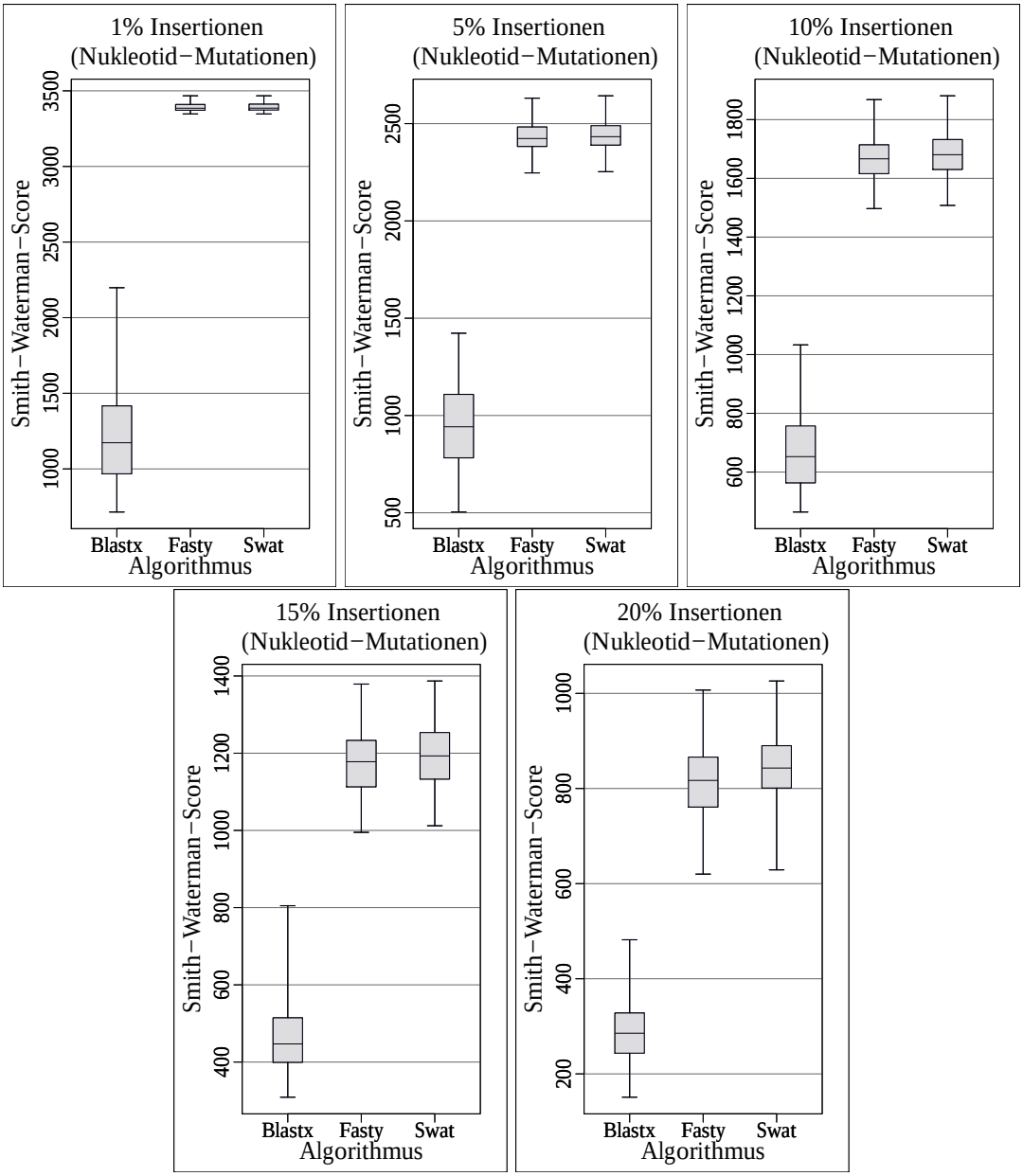


Abbildung 75: Ergebnisse der Frameshift-Insertions-Simulation: Die Ergebnisse der unterschiedlichen Rauschanteile für je $n = 1000$ Sequenzen zeigen, dass Fasty und *Swat* ähnlich gut arbeiten. Der *Swat*-Algorithmus findet in einigen wenigen Sequenzen ein Alignment mit höherem Smith-Waterman-Score. Ebenfalls zu sehen ist, dass schon ab 1% Frameshiftmutationen der Smith-Waterman-Score eines Alignments ohne Unterstützung von Frameshiftmutationen im Algorithmus um 60% niedriger berechnet wird.

Für die Nukleotid-Deletions-Simulation zeigt sich ein ähnliches Bild wie bei der Insertions-Simulation. Die höchsten Bewertungen erreicht der *Swat*-Algorithmus.

Der Fasty-Algorithmus berechnet Alignments mit etwas niedrigeren Bewertungen. Blastx berechnet deutlich niedrigere Alignment-Bewertungen (siehe Abbildung 76).

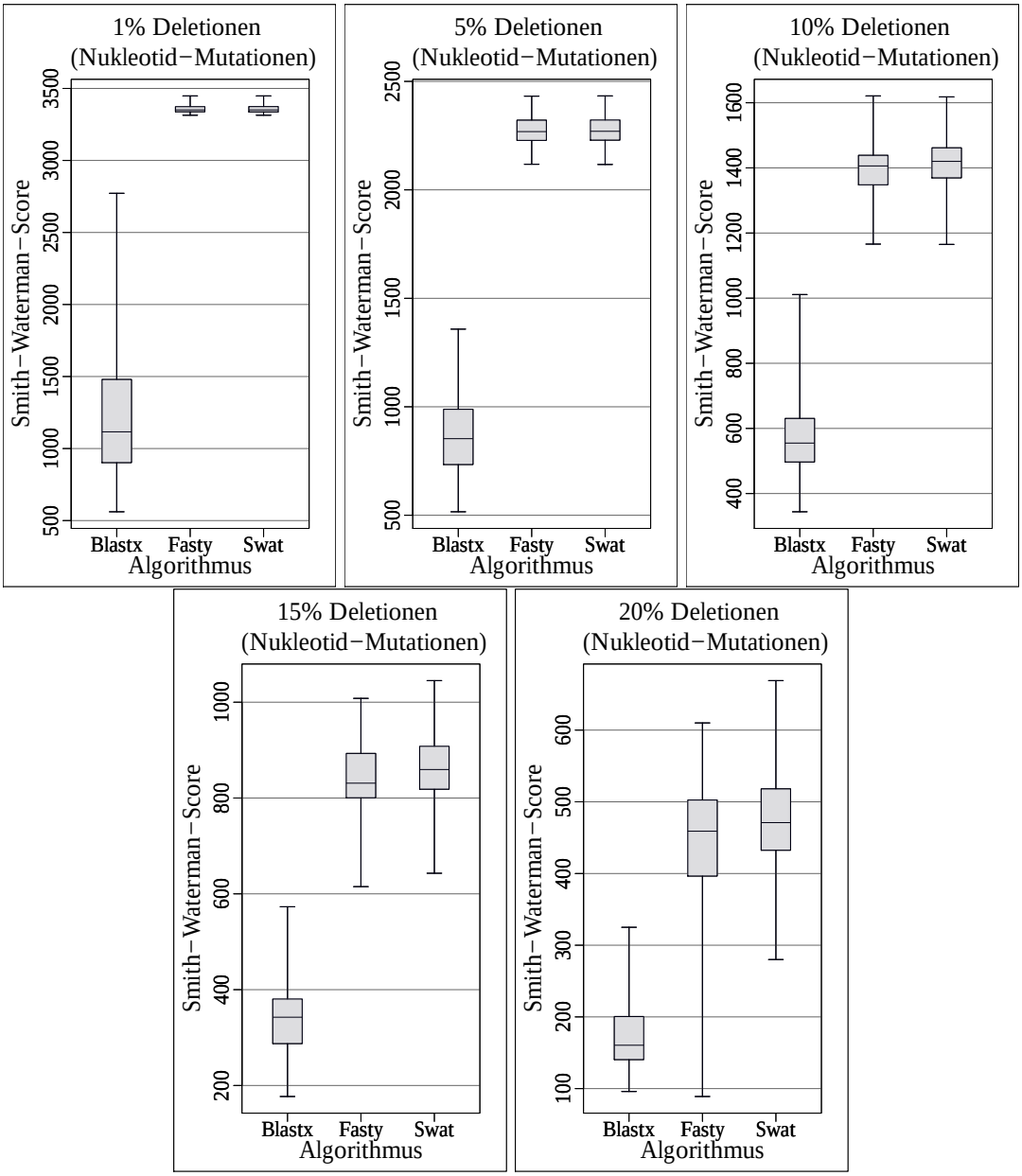


Abbildung 76: Ergebnisse der Frameshift-Deletions-Simulation: Ebenso wie bei der Frameshift-Insertions-Simulation gestalten sich die Ergebnisse hier, erneut mit $n = 1000$ Sequenzen. Betrachtet man die Alignment-Ergebnisse von Fasty im Detail, sieht man, dass in einigen Sequenzen positiv gewichtete Teilsequenzen am Anfang des Alignments übersehen werden, was zu den teilweise niedrigeren Bewertungen im Vergleich zu *Swat* führt. Dies ist durch die hot-spot-Analyse des Fasty-Algorithmus begründet.

In der Frameshift-Substitutions-Simulation finden Fasty und *Swat* Alignments mit ähnlich hohen Bewertungen. Erst bei einem Rauschanteil von 15% sind die Bewertungen der Alignments von Fasty etwas niedriger als die von *Swat*. Blastx berechnet ohne die Unterstützung von Frameshiftmutationen wieder ein signifikant niedrigeres Alignment (siehe Abbildung 77).

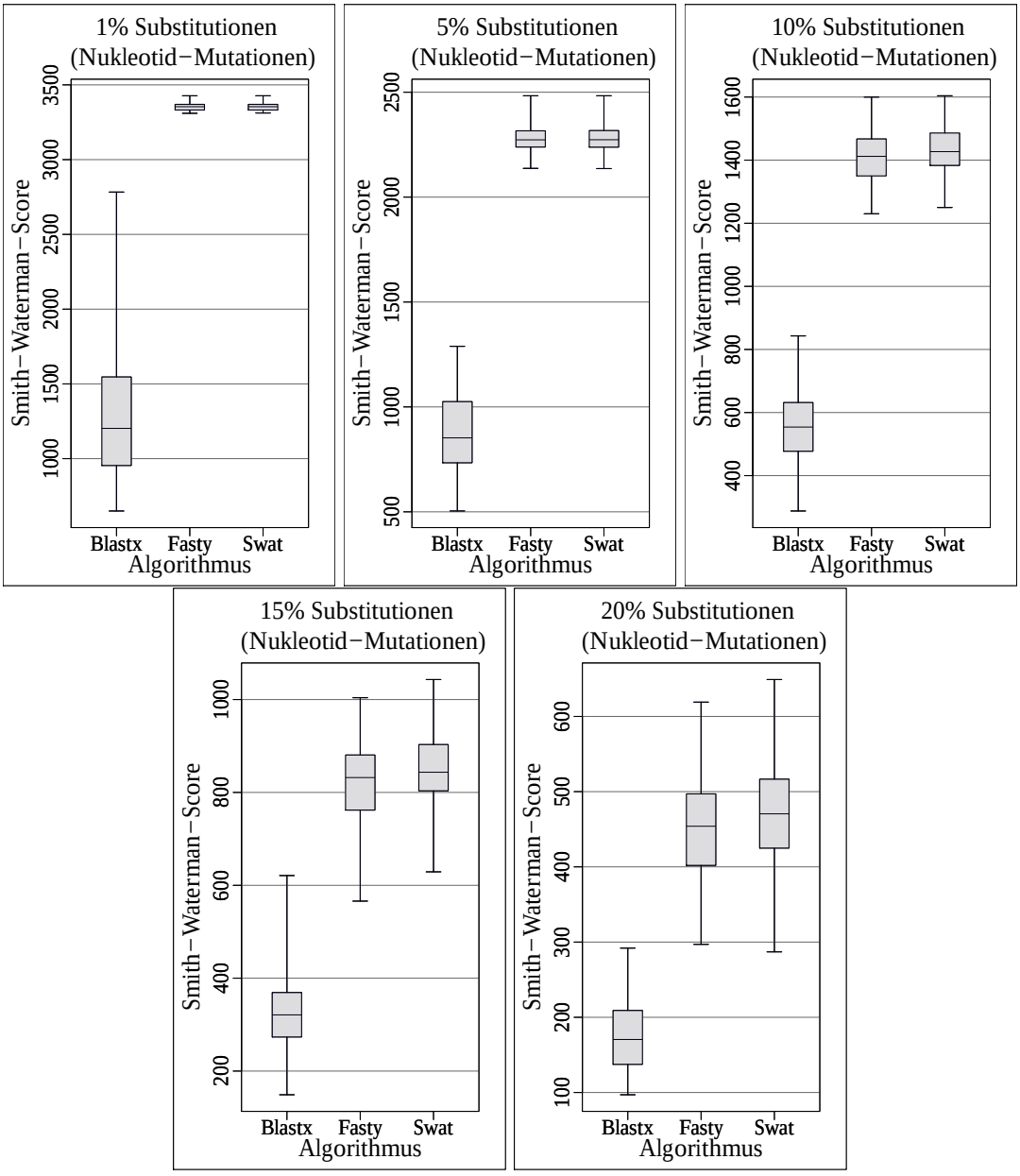


Abbildung 77: Ergebnisse der Frameshift-Substitutions-Simulation: Für $n = 1000$ Sequenzen zeigt sich ein identisches Bild zur Frameshift-Deletions-Simulation, die Unterschiede treten jedoch erst bei 15% Rauschen hervor. Blastx kommt zu deutlich niedrigeren Alignment-Ergebnissen.

Im letzten Teil der Nukleotid-Mutations-Simulation wurden Insertionen, Deletionen und Substitutionen miteinander kombiniert und die so entstandenen Sequenzen durch die drei Algorithmen Blastx, Fasty und Swat aligniert. Die Analyse der Ergebnisse zeigt die gleichen Effekte, wie in den vorangegangenen Nukleotid-Mutations-Simulationen. Blastx berechnet Alignments mit deutlich niedrigerer Bewertung als Fasty und *Swat*. Die beiden Algorithmen mit Frameshiftunterstützung berechnen erneut gleiche oder stark ähnliche Bewertungen. Erst bei einem Rauschanteil von 20% werden die Unterschiede sichtbar (siehe Abbildung 78).

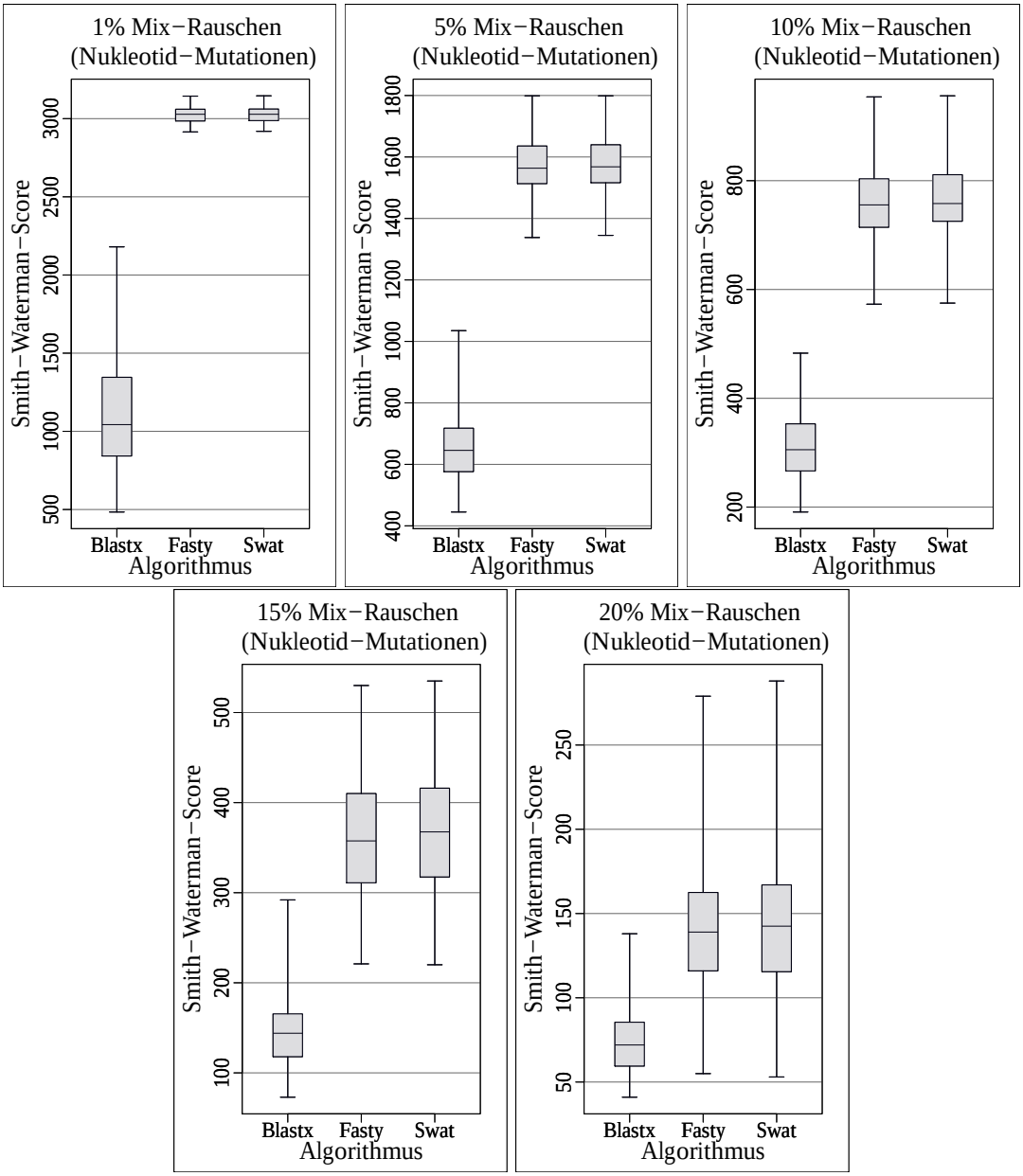


Abbildung 78: Ergebnisse der Frameshift-Mix-Simulation mit $n = 1000$ Sequenzen je Rauschstufe: Auch hier treten die zu erwartenden Ergebnisse auf, welche sich bereits in den vorangegangenen Simulationen gezeigt haben. Blastx berechnet Alignmentscores mit deutlich niedrigerer Bewertung als Fasty und *Swat*. *Swat* erzielt leicht höhere Ergebnisse als Fasty bei einem Rauschanteil von 20%.

Es wurden je 1000 Sequenzen mit Insertionen, Deletionen, Substitutionen und einem Mix aus allen drei Arten generiert und mit Hilfe des Frameworks ausgewertet. Der Blastx-Algorithmus unterstützt keine Frameshiftmutationen, sodass Ergebnis-

se zwischen Alignmentbewertungen mit und ohne Frameshift-Unterstützung verglichen werden können.

3.1.1 *Wild-Basen*

Die dritte Simulation ermöglicht einen quantitativen Vergleich zwischen den best- und worst-case Alignment-Berechnungen in Abhängigkeit vom Anteil der Wild-Basen in den Sequenzen.
Wild-Basen werden auf zwei Arten in eine Sequenz eingebracht: Eine Nukleotidbase wird durch eine Wild-Base ersetzt, welche jedoch auch das originale Nukleotide repräsentiert oder eine Base wird durch eine zufällig ausgewählte Wild-Base ersetzt. Dieses Vorgehen erlaubt eine Unterscheidung zwischen best-case und worst-case-Analyse, wobei für das erste Ersetzungsvorgehen im best-case ein perfektes Matching an der Position entsteht, so die ursprüngliche Sequenz ein perfektes Matching darstellt und im worst-case eine Alignment zurück liefert, welches an diesen Stellen Mutationen aufweist.

GEZIELTE WILD-BASEN-SUBSTITUTIONEN

Im ersten Teil der dritten Simulation wurden in einer Sequenz Anteile von 1%, 5%, 10%, 15% und 20% der Nukleotide durch Wild-Basen ersetzt, welche neben anderen Nukleotiden auch die ursprüngliche Base repräsentieren. Dabei berechnet der Algorithmus im best-case die Referenz-Bewertung des Alignments, da die Wild-Basen durch Nukleotide ersetzt werden können, die in der Sequenz zu Tripletts führen, so dass keine Mutation an dieser Stelle auftritt. Die worst-case Berechnungen führen im Allgemeinen zu Alignments mit niedrigeren Bewertungen, welche abhängig vom Anteil der Wild-Basen ist. Die geringere Bewertung des Alignments resultiert aus Mutationsstellen, welche an der mutierten Position zu einer niedrigeren Bewertung bei der Berechnung führen (siehe Abbildung 79).

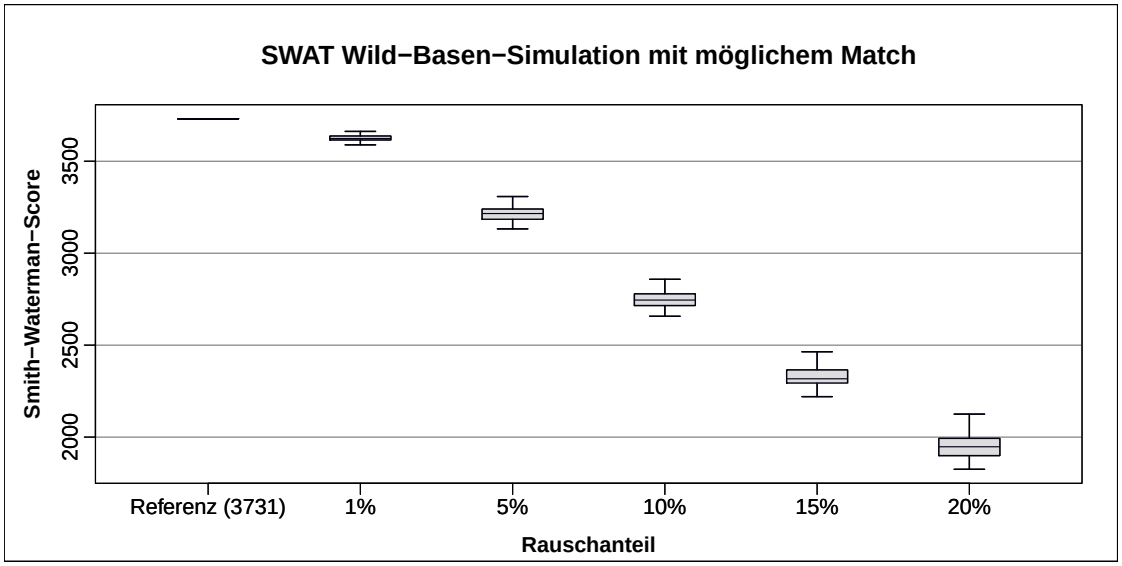


Abbildung 79: Auswertung der ersten Wild-Basen-Substitutions-Simulation: Je 1000 Variationen einer Sequenz wurden mit Wild-Basen-Rauschanteilen von 1% bis 20% generiert. Dabei wurden die Wild-Basen so gewählt, dass sie die Substituierte Base ebenso repräsentieren. Im Vergleich zum Referenz-Smith-Waterman-Score eines perfekten Alignments (3731 = exakter Match) zeigt sich, dass die worst-case-Berechnungen in den Fällen von 1% bis zu 20% Wild-Basen-Rauschen um 1.6% bis zu 51.1% niedriger ist.

WILLKÜRLICHE WILD-BASEN-SUBSTITUTIONEN

Im zweiten Teil der Wild-Basen-Simulation wurden Nukleotide einer Sequenz zu 1% bis hin zu 20% durch zufällige Wild-Basen ersetzt. So kann analysiert werden, wie sich die best-case- und worst-case-Alignmentbewertungen in Abhängigkeit vom Rauschanteil unterscheiden. Bei geringem Rauschen unterscheiden sich die Bewertungen der Alignments nur wenig, ab einem Rauschanteil von 15% ist die Bewertung der best-case Analyse mindestens doppelt so hoch, wie im worst-case (siehe Abbildung 80).

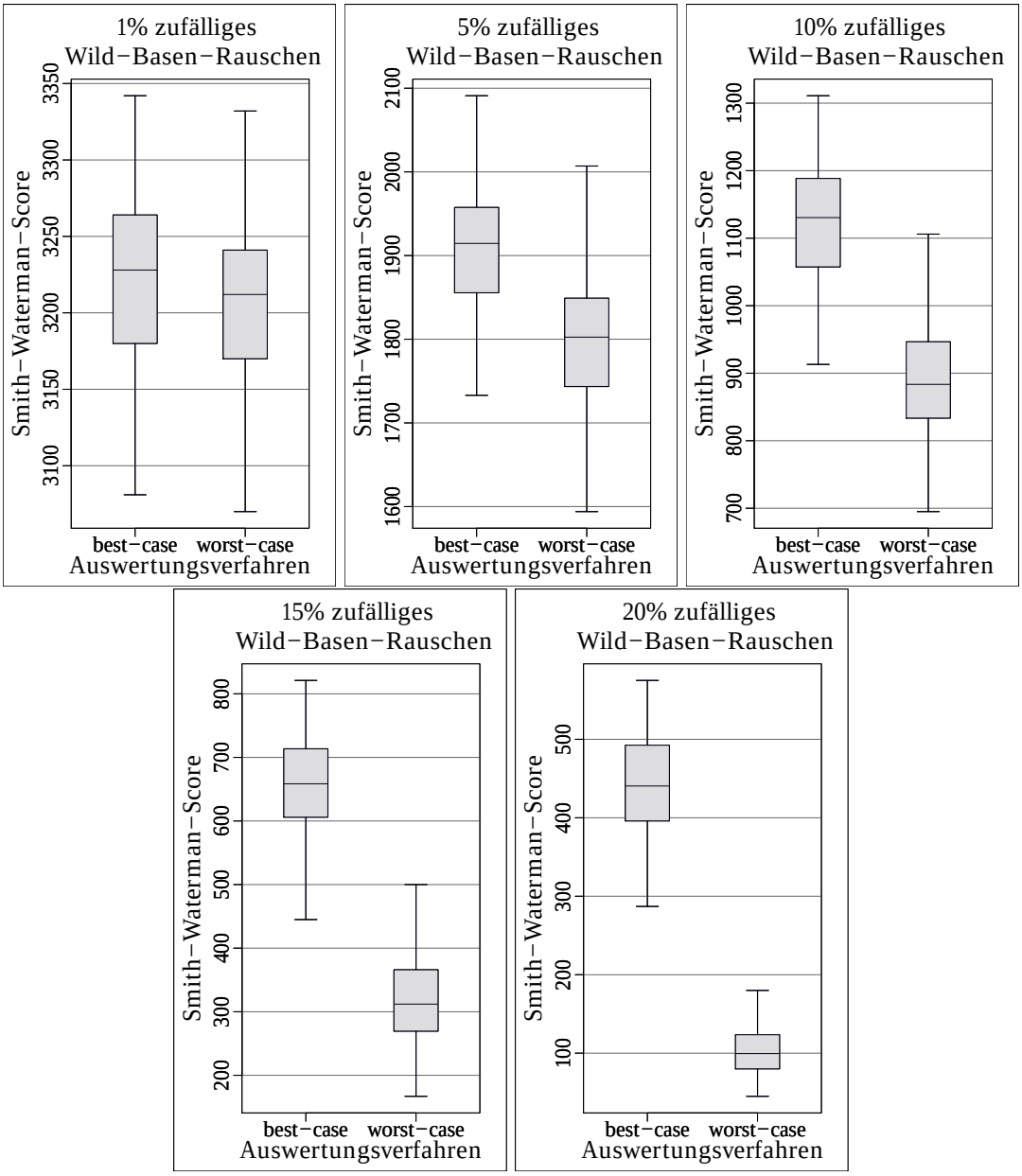


Abbildung 80: Die Auswertung der zweiten Wild-Basen-Substitutions-Simulation mit $n = 1000$ Sequenzen zeigt, dass bereits bei 15% Wild-Basen-Anteil die Alignments der best-case-Berechnung mindestens einen doppelt so hohen Smith-Waterman-Score aufweisen, wie die Alignment-Scores in der worst-case-Berechnung.

3.1.2 *Reale Daten*

Als weitere Simulation wurden Virussequenzen, gesammelt aus der HCMV-Webseite [10], für die UL54- und die UL97-Proteinsequenz untersucht. Je ca. 4000 Sequenzen wurden als Eingaben für den Blastx-Algorithmus, welcher bisher in der Webanwendung verwendet wird, und dem *Swat*-Algorithmus verwendet.
Für die UL54-Daten hat sich im Vergleich gezeigt, dass in ca. 630 Fällen der *Swat*-Algorithmus einen höheren Smith-Waterman-Score erzielt. In ca. 500 dieser Fälle ist der Unterschied kleiner als 3,5%. In den übrigen Fällen (ca. 130) ergibt sich ein Unterschied von bis zu 44%. Dies sind Fälle, in denen Frameshifts auftreten, welche vom Blastx-Algorithmus nicht berücksichtigt werden. Für die übrigen Sequenzen erzielen die beiden Algorithmen eine identische Bewertung und damit ein äquivalentes Alignment.
In der UL97-Simulation wurden durch *Swat* 280 höher bewertete Alignments in den ca. 4000 Sequenzen gefunden. Dabei wurden bis zu 50% höher bewertete Alignments in 10 Sequenzen gefunden und 20% höher bewertete Alignments in ca. 60 Sequenzen. Auch diese Sequenzen weisen Frameshift-Mutationen auf. Die übrigen Sequenzen wurden durch beide Algorithmen äquivalent analysiert.

4

DISKUSSION

In der Arbeit wurde ein erweiterter Alignmentalgorithmus *Swat* zum Testen auf genotypische Resistenzen entwickelt und implementiert. Seine Validität konnte durch Korrektheitsbeweis und Simulationen gezeigt werden.

ALGORITHMUS

Für *Swat* wurden als Algorithmus-Grundlage das Prinzip der dynamischen Programmierung und ein heuristischer Ansatz diskutiert. Heuristische Alignmentalgorithmen verwenden k-Tupel, um den Berechnungsaufwand zu reduzieren. Dabei kann es jedoch zu Ungenauigkeiten kommen, wie in der Simulation (siehe Abschnitt 3.1) zu sehen ist. Ein dynamischer Programmieransatz durchläuft für jedes Element jede mögliche Position im Alignment einzeln und kann dadurch ein optimales (Teil-)Alignment exakt bestimmen. Dem Geschwindigkeitsvorteil der Heuristik steht somit die höhere Genauigkeit der dynamischen Programmierung gegenüber. Für das Testen auf Resistenzen ist es wichtig, ein Alignment genau zu berechnen. Daher wurde der dynamische Programmieransatz gewählt.

Swat unterstützt die Analyse von Triplett-Mutationen (Insertionen, Deletionen und Substitutionen), Frameshiftmutationen, Wild-Basen und eine gesonderte Behandlung von Stopcodonmutationen. Der Algorithmus besitzt dafür entsprechende Parameter zur Anpassung der Bewertungen dieser Fehler bzw. Mutationen. Ein früheres Verfahren für das hier vorgestellte Verfahren zum Testen auf Resistenzen basiert mit Blastx auf einem heuristischen Algorithmus [2]. Dieser unterstützt Triplett-Mutationen, jedoch keine Frameshiftmutationen. Ebenso wenig werden Wild-Basen in der Eingabesequenz zugelassen, da diese von Blastx nicht korrekt verarbeitet werden können. Stopcodonmutationen werden bei Blastx im Alignment ebenfalls nicht gesondert behandelt. Dies ist jedoch wichtig, da solche Mutationen schwerwiegendere Auswirkungen auf die übersetzte Proteinsequenz haben können als einfache Substitutionen.
Swat berücksichtigt alle diese Mutationen. Mögliche Frameshiftmutationen wer-

den durch individuelle Bewertungen sowohl von einfachen als auch doppelten Frameshifts verarbeitet. Dadurch lassen sich in den Simulationen Alignments mit höherer Bewertung erzielen (siehe Abschnitt 3.1). Dies lässt auf ein besseres Alignment mit weniger Fehlern schließen.
Durch die Erweiterung zur Verarbeitung und Auflösung von Wild-Basen erlaubt *Swat* ebenfalls das Alignieren von Eingabesequenzen, die Ungenauigkeiten und Unsicherheiten enthalten. *Swat* nutzt dafür eine IUPAC-Erweiterung zum 1,5-Alignment (siehe Abschnitt 2.6.3). Unter den gegebenen Bedingungen hat sich dieser Ansatz für Nukleotid-Protein-Alignments, wie *Swat* sie berechnet, durchgesetzt [70, 47]. Aktuell kann für die Wild-Basen-Analyse entweder das Maximum oder das Minimum der Bewertung verwendet werden. Sollen andere Analysekriterien verwendet werden, können diese zusätzlich in *Swat* implementiert werden.
Um Stopcodonmutationen im Alignment gesondert zu berücksichtigen, wurde ein neuer Malus in *Swat* eingeführt. Auch dieser kann angepasst werden, um entsprechende Mutationen im Alignment mehr oder weniger stark zu bestrafen.

IMPLEMENTIERUNG

Der Algorithmus wurde in der Programmiersprache *Java* implementiert. Da diese Sprache plattformunabhängig ist, kann das Programm auf jedem aktuellen Betriebssystem eingesetzt werden. Darunter fallen unter anderem Linux, Windows und MacOSx. Durch entsprechende Parameter bei der Ausführung des *Java*-Programms können vorgegebene Hardwarespezifikationen berücksichtigt werden. So können zum Beispiel der zur Verfügung stehende Arbeitsspeicher oder Prozessoren optimal genutzt werden. Das Java Archive, in das *Swat* gepackt ist, benötigt eine Java-Laufzeitumgebung (Java Runtime Environment) der Version 1.7 oder höher. Diese kann in der Regel jedoch auf allen modernen Betriebssystemen einfach installiert werden.

WEBSEITEN-ANALYZER

Um den in der Arbeit entwickelten und implementierten Algorithmus zum Testen auf genotypische Resistenzen anzuwenden (siehe Abschnitt 2.6), wurde dieser in einen generischen Webseiten-Analyzer integriert. Neben dem Austausch von Blastx durch *Swat* wurden zusätzliche Funktionen in die Webapplikation eingepflegt. Diese ermöglichen unter anderem eine detailliertere Ausgabe, eine Erweiterung um zusätzliche Sequenzen und entsprechende Datenbanken. Die ursprünglichen Optionen zur Ein- und Ausgabe wurden beibehalten bzw. zusammengeführt. Beispielsweise wurden die bisher auf unterschiedlichen Instanzen laufenden Analyzer für

virale Gene von HCMV, HCV und HSV zu einer Webanwendung zusammenfasst. Diese werden jetzt auf der neuen Webseite durch eine Auswahlliste repräsentiert. Dies vereinfacht einerseits die Wartung der Anwendung und andererseits vereinheitlicht es die Nutzeroberfläche.

SIMULATIONSERGEBNISSE

Verschiedene künstliche Datensätze wurden simuliert, um Alignmentgüte und -qualität, sowie die Auswirkungen von Wild-Basen auf das Alignment zu untersuchen. Zusätzlich wurden reale Daten, welche über die Webanwendung gesammelt wurden, mit *Swat* aligniert. Die Ergebnisse wurden mit den Alignments von Blastx verglichen.
Um die entsprechenden Simulationsdaten zu erzeugen, wurde eine weitere Testumgebung in *Java* implementiert. Diese erzeugt in Abhängigkeit verschiedener Parametern Sequenzen bzw. Sequenzreihen, welche einen vorgegebenen Anteil bestimmter Mutationen bzw. Wild-Basen aufweisen. Die so generierten Sequenzen wurden mit drei verschiedenen Algorithmen aligniert und die Ergebnisse anschließend ausgewertet.
Auf Grund der Verbreitung heuristischer Algorithmen wie Blastx wurde mit Fasty ein zweiter heuristischer Algorithmus zum Vergleich herangezogen. Fasty kann zusätzlich Frameshiftmutationen verarbeiten, jedoch keine Wild-Basen ersetzen. *Swat* mit einen anderen Algorithmus basierend auf dem Prinzip der dynamischen Programmierung zu vergleichen, würde keine anderen Ergebnisse erzeugen, so die selben Mutationsarten unterstützt werden.
Um die k-Tupel-Berechnungen der heuristischen Ansätze der Präzision von *Swat* gegenüber zustellen, wurden verschiedene Simulationen entworfen. Zuerst wurden Sequenzreihen mit Triplettinsertionen, -deletionen und -substitutionen erzeugt. Die Ergebnisse der Triplettsimulation zeigen, dass die drei betrachteten Algorithmen Blastx, Fasty und *Swat* bei geringen Mutations-Rauschanteilen äquivalente Alignments finden. Mit steigendem Rauschanteil schneiden die heuristischen Algorithmen Blastx und Fasty schlechter ab. Analysiert man die Alignments im Detail, zeigt sich, dass Teilsequenzen zu Beginn des lokalen Alignments im Fasty-Algorithmus verworfen werden, was in der hot-spot-Analyse des Algorithmus begründet liegt. Fallen dabei die Bewertungen einer Teilsequenz unter einen bestimmten Schwellwert, werden diese Bereiche nicht für das Alignment berücksichtigt. Ähnlich, jedoch nicht so ausgeprägt, zeigt sich dies auch bei Blastx, jedoch ist hier der Schwellwert der seeds verantwortlich [49]. So die exakte Startposition eines Alignments nicht relevant ist, ist dieses Vorgehen ausreichend. Um Sequenzen auf genotypische Resistenzen zu untersuchen, ist ein möglichst vollständiges Alignment wichtig. Daher erweist sich *Swat* hier als überlegen.

In der zweiten Simulation wurden Sequenzreihen mit Nukleotidinsertionen, -deletionen und -substitutionen generiert. Da Blastx diese Art der Mutation nicht unterstützt, versucht der Algorithmus diese Frameshiftmutationen mit Triplettinsertionen, -deletionen und -substitutionen auszugleichen. Dies verfälscht das Alignment stark und führt zu wesentlich schlechteren Bewertungen. Obwohl Fasty Frameshiftmutationen unterstützt, treten bei hohen Rauschanteilen ab 10% die gleichen Schwächen wie bei den Triplettmutationen auf. Auch hier kann *Swat* somit die besten Alignmentbewertungen erzielen.

In der Wild-Basen-Simulation wurden zwei unterschiedliche Szenarien untersucht. Zuerst wurden Nukleotide durch Wild-Basen ersetzt, die auch das ursprüngliche Nukleotid repräsentierten. Danach wurden Nukleotide durch willkürliche Wild-Basen substituiert.
So eine Wild-Base das ursprüngliche Nukleotid repräsentiert, liefert die best-case Bewertung der Alignments wie erwartet stets den Smith-Waterman-Score eines perfekten Alignments. Hier wird die Wild-Base wieder durch das ursprüngliche Nukleotid ersetzt, da dies zu einer Übereinstimmung im Alignment führt. So die Wild-Base das ursprüngliche Nukleotid nicht repräsentiert, werden im Allgemeinen niedrigere Bewertungen erzielt. Im best- und worst-case werden Wild-Basen dabei jeweils so ersetzt, dass das Alignment an dieser Stelle die höchstmögliche bzw. geringstmögliche Bewertung erhält. Handelt es sich bei einer solchen Stelle um eine sogenannte Wobbel-Stelle, kann in beiden Fällen die gleiche Bewertungen erreicht werden. In allen anderen Fällen führt die worst-case Analyse zu einer schlechteren Bewertung im Alignment, als die der best-case Analyse.
Soll für eine Wild-Basen-Nukleotid-Sequenz lediglich die Proteinsequenzen mit der größte Ähnlichkeit gefunden werden, ist die best-case Bewertung des Alignments vorzuziehen. Dabei wird angenommen, dass die Wild-Basen zu Großteil nicht Mutationen entsprechen sondern zum Beispiel Unsicherheiten in der Sequenz. Wird jedoch angenommen, dass die Mehrheit der Wild-Basen tatsächlichen Mutationen entsprechen, sollen diese durch Fehler im Alignment erkannt werden. Die worst-case Berechnung maximiert die Anzahl dieser Fehler und somit auch die Anzahl der detektierten Mutationen. Damit ist sie für diese Anwendungen besser geeignet. Der Nutzer kann sich je nach Anwendungsfall selbst entscheiden, welche der beiden Methoden von primärem Interesse ist.

Die Simulationen generieren unterschiedliche Mutationen bzw. Sequenzierungsunsicherheiten, die bei Sequenzen auftreten können. Die generierten Sequenzen umfassen Abstufungen von nur wenigen Änderungen zur Referenzsequenz bis hin zu einer Sequenz mit bis zu 20% Rauschanteil. *Swat* kann all diese Sequenzen

verarbeiten und berechnet Alignments mit der höchsten Bewertung.

Um diese Ergebnisse zu validieren, wurden in einer weiteren Analyse reale UL54- und UL97-Sequenzen aus der ursprünglichen Webanwendung gesammelt [10]. Diese Sequenzen wurden mit *Swat* aligniert und mit den zuvor berechneten Ergebnissen von Blastx verglichen. 15% bzw. 7% der Sequenzen erreichen eine höhere Alignmentbewertung mit *Swat*; in allen anderen Fällen erreichen die beiden Algorithmen äquivalente Alignments. Bei 70 der 4000 Sequenzen wurden Frameshiftmutationen gefunden, welche von Blastx nicht berücksichtigt werden. Dies zeigt, dass es relevant ist diese Mutationen zu verarbeiten. Auch bei den anderen Sequenzen, die mit *Swat* eine höhere Alignmentbewertung erzielen, ist das lokale Alignment länger und umfasst somit mehr Positionen, die mit der Proteinsequenz verglichen werden können. Hier wird also deutlich, dass die Berücksichtigung von Frameshiftmutationen eine durchaus relevante Erweiterung von Alignmentalgorithmen darstellt, welche auch auf realen Daten ein besseres Ergebnis ermöglicht. Dies bestätigt zum einen die Ergebnisse der Simulationen, zeigt andererseits aber auch, dass *Swat* in der speziellen Anwendung des Resistenztestens besser abschneidet als Blastx.

LAUFZEIT

Swat besitzt eine Laufzeit der Ordnung $O(n^2)$. Wild-Basen in der Eingabesequenz führen potenziell zu exponentiell vielen Sequenzen, wenn die Wild-Basen durch alle Kombinationen der repräsentierten Nukleotide ersetzt werden. Würde *Swat* all diese Sequenzen gegen eine Proteinsequenz alignieren, würde die Laufzeit stark anwachsen. Um diesen exponentiellen Anstieg zu verhindern, berechnet *Swat* Alignments mit der höchsten bzw. niedrigsten Bewertung der Wild-Basen. Diese Vereinfachung reduziert die möglichen Alignments und erhält die quadratische Laufzeit bis auf einen konstanten Faktor.
Das Nukleotid, das bei der Optimumsanalyse einer Wild-Base in das Alignment eingesetzt wird, muss nicht dem sequenzierten Nukleotid der Sequenz entsprechen. Die best-case Analyse maximiert die Sequenzähnlichkeit und die worst-case Analyse maximiert die Fehler- und damit Mutationswahrscheinlichkeit im Alignment. So werden die wichtigsten Fälle für das Alignment abgedeckt, was dieses Vorgehen vertretbar macht und die quadratische Laufzeit von *Swat* bleibt auch bei Wild-Basen-Sequenzen erhalten.

ANWENDUNGEN UND ERWEITERUNGEN

Swat ist in der Lage eine best- und worst-case Bewertung der Alignments bei Wild-Basen-Sequenzen vorzunehmen. Die textuellen Ausgaben der Ergebnisse zu vergleichen ist aufwendig, wenn die einzelnen Positionen und deren Nukleotide bzw. Aminosäuren manuell verglichen werden müssen. Eine einfache Visualisierung der Unterschiede in den Alignments kann durch eine Gegenüberstellung der zugehörigen Alignmentgraphen erreicht werden.
Um diese Darstellung übersichtlich zu gestalten, könnten zum Beispiel nur die Abweichungen der beiden Alignments hervorgehoben werden. Für eine genauere Betrachtung der Mutationen könnten Details wahlweise wieder eingeblendet werden. Dies könnte über unterschiedliche Vergrößerungsstufen der Graphenansicht erreicht werden. So kann einerseits eine schnelle Übersicht der mutierten Stellen angeboten werden, andererseits aber auch eine detailreiche Abbildung der einzelnen Nukleotide, wenn auf einen bestimmten Teil des Alignments fokussiert werden soll. *Swat* bietet bereits die strukturierte Ausgabe der Alignments, wie sie für eine solche Anwendung benötigt wird.

Swat arbeitet auf Resistenzdatenbanken bestimmter viraler Gene. Es können ebenfalls andere Annotationdatenbanken für weitere Organismen hinterlegt werden. Diese könnten als Grundlage für Resistenzvorhersagemodelle verwendet werden, indem die Charakteristiken der Gene und die entsprechenden Datenbanken verwendet werden. So könnten mit *Swat* auch Hypothesen über weniger bekannte oder untersuchte Gene aufgestellt werden.

Der in dieser Arbeit entwickelte Alignmentalgorithmus *Swat* unterstützt die Analyse von Insertionen, Deletionen und Substitutionen, Frameshifts, Wild-Basen, sowie eine gesonderte Behandlung von Stopcodonmutationen. Vergleiche mit anderen Algorithmen haben gezeigt, dass *Swat* sowohl auf simulierten als auch auf realen Datensätzen durchwegs die besseren Ergebnisse erzielt. Ein Einsatz in anderen Gebieten als der hier speziell betrachteten Resistenzanalyse ist ebenfalls denkbar und mögliche Anwendungen wurden diskutiert. *Swat* stellt so eine Bereicherung für das Gebiet der Alignmentalgorithmen und angrenzende Arbeitsfelder dar.

5

ZUSAMMENFASSUNG

Zur Alignmentanalyse von Wild-Basen-Nukleotidsequenzen mit Proteinsequenzen wurde ein neuer Algorithmus *Swat* entwickelt und implementiert. Aufbauend auf dem algorithmischen Prinzip der dynamischen Programmierung wurde die Detektion von Insertionen, Deletionen und Substitutionen um eine Optimalitätsanalyse von Wild-Basen ergänzt. Damit unterstützt *Swat* die Eingabe von Sequenzen mit erweiterter IUPAC-Nomenklatur für paarweises Sequenzalignment von Nukleotid- und Proteinsequenzen. Treten Wild-Basen in einer Nukleotidsequenz auf, so können diese durch den erweiterten Algorithmus aufgeschlüsselt und durch ein best- und worst-case-Verfahren aligniert werden. Dabei bleibt die Ordnung der Laufzeit des dynamischen Programmieransatzes von $O(n^2)$ erhalten. Neben der Berücksichtigung der unterschiedlichen Mutationsarten ermöglicht der Algorithmus ausserdem die gesonderte Bewertung von Stopcodonmutationen.

Unterschiedliche Simulationen haben gezeigt, dass *Swat* verglichen mit den heuristischen Alignmentalgorithmen Fasty und Blastx äquivalent gute und bessere Alignments berechnet. Dies lässt sich sowohl auf künstlichen als auch auf realen Daten zeigen.

Das neue Verfahren wurde in die generische Webanwendung MRA (Mutations Resistance Analyzer) zur Bestimmung von Resistenzen und Sensibilitäten mutierter Viren gegenüber Medikamenten integriert. Diese unterstützt die Analyse unterschiedlicher Proteinsequenzen und kann um weitere Proteinsequenzen erweitert werden.

Ein weiteres Ziel ist die Vorhersage von resistenz- oder sensibilitätsverursachenden Mutationen neuer Gene auf Basis vergleichbarer Analysen bereits untersuchter Gene.

6

LITERATURVERZEICHNIS

[1] Altschul, S. F. Generalized affine gap costs for protein sequence alignment. *Proteins Structure Function and Genetics 32* (1998), 88–96.

[2] Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. Basic local alignment search tool. *Journal of Molecular Biology 215* (1990), 403–410.

[3] Bellman, R. E. On the Theory of Dynamic Programming. *Proceedings of the National Academy of Sciences of the United States of America 38* (1952), 716–719.

[4] Bellman, R. E. *Eye of the Hurricane: An Autobiography*. World Scientific Publishing Company, Singapore, 1984.

[5] Bellman, R. E. *Dynamic Programming*. Dover Publications, New York, 2003.

[6] Botstein, D., and States, D. J. Molecular sequence accuracy and the analysis of protein coding regions. *Proceedings of the National Academy of Sciences of the United States of America 88* (1991), 5518–5522.

[7] Brenner, S., Barnett, L., Katz, E. R., and Crick, F. H. C. UGA: A Third Nonsense Triplet in the Genetic Code. *Nature 213* (1967), 449–450.

[8] Brosius, J., and Tiedge, H. RNomenclature. *RNA Biology 1* (2004), 81–83.

[9] Buhler, J. Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics 17* (2001), 419–428.

[10] Chevillotte, M., von Einem, J., Meier, B. M., Lin, F.-M., Kestler, H. A., and Mertens, T. A new tool linking human cytomegalovirus drug resistance mutations to resistance phenotypes. *Antiviral Research 85* (2010), 318–327.

[11] Claverie, J.-M. Detecting Frame Shifts by Amino Acid Sequence Comparison. *Journal of Molecular Biology 234* (1993), 1140–1157.

[12] Connelly, N. G. *Nomenclature of inorganic chemistry: IUPAC recommendations 2005*. Royal Society of Chemistry, London, 2005.

[13] Cooper, G. M., and Hausman, R. E. *The Cell: A Molecular Approach*. ASM Press, New York, 2007.

[14] Crick, F. H. On protein synthesis. *Symposia of the Society for Experimental Biology 12* (1958), 138–163.

[15] Crick, F. H. Codon-anticodon pairing: The wobble hypothesis. *Journal of Molecular Biology 19* (1966), 548–555.

[16] Crick, F. H. Central Dogma of Molecular Biology. *Nature 227* (1970), 561–563.

[17] Damerau, F. J. A Technique for Computer Detection and Correction of Spelling Errors. *Communications of the ACM 7* (1964), 171–176.

[18] Dayhoff, M. O., and Eck, R. V. *Atlas of Protein Sequence and Structure: 1966*. National Biomedical Research Foundation, Washington, D.C., 1966.

[19] Dayhoff, M. O., and Foundation, N. B. R. *Atlas of Protein Sequence and Structure*. Atlas of Protein Sequence & Structure, Suppl. No. 3. National Biomedical Research Foundation, Washington, D.C., 1979.

[20] Donlin, M. J., Patel, S. S., and Johnson, K. A. Kinetic Partitioning between the Exonuclease and Polymerase Sites in DNA Error Correction. *Biochemistry 30* (1991), 538–546.

[21] Dunn, D. T., Coughlin, K., and Cane, P. A. Genotypic resistance testing in routine clinical care. *Current Opinion in HIV and AIDS 6* (2011), 251–257.

[22] Farrell, P. M., Rosenstein, B. J., White, T. B., Accurso, F. J., Castellani, C., Cutting, G. R., Durie, P. R., LeGrys, V. A., Massie, J., Parad, R. B., Rock, M. J., and Campbell III, P. W. Guidelines for Diagnosis of Cystic Fibrosis in Newborns through Older Adults: Cystic Fibrosis Foundation Consensus Report. *The Journal of Pediatrics 153* (2008), S4–S14.

[23] Fitch, W. M. Locating Gaps in Amino Acid Sequences to Optimize the Homology Between Two Proteins. *Biochemical Genetics 3* (1969), 99–108.

[24] Frith, M. C., Saunders, N. F. W., Kobe, B., and Bailey, T. L. Discovering Sequence Motifs with Arbitrary Insertions and Deletions. *PLoS Computational Biology 4* (2008), e1000071.

[25] Fürstberger, A., Maucher, M., and Kestler, H. A. Extended pairwise local alignment of wild card DNA/RNA sequences using dynamic programming. *Journal of Statistical Computation and Simulation 85* (2015), 3–13.

[26] Garcia-Diaz, M., and Kunkel, T. A. Mechanism of a genetic glissando*: structural biology of indel mutations. *Trends in Biochemical Sciences 31* (2006), 206–214.

[27] Gibbs, A. J., and Mcintyre, G. A. The Diagram, a Method for Comparing Sequences. *European Journal of Biochemistry 16* (1970), 1–11.

[28] Gîrdea, M., Noé, L., and Kucherov, G. Back-translation for discovering distant protein homologies in the presence of frameshift mutations. *Algorithms for Molecular Biology 5* (2010), 1–15.

[29] Goonesekere, N. C. W., and Lee, B. Context-specific amino acid substitution matrices and their use in the detection of protein homologs. *Proteins: Structure, Function, and Bioinformatics 71* (2008), 910–919.

[30] Gotoh, O. An Improved Algorithm for Matching Biological Sequences. *Journal of Molecular Biology 162* (1982), 705–708.

[31] Guan, X., and Uberbacher, E. C. Alignments of DNA and protein sequences containing frameshift errors. *Computer Applications in the Biosciences: CABIOS 12* (1996), 31–40.

[32] Hamming, R. W. Error Detecting and Error Correcting Codes. *Bell System Technical Journal 29* (1950), 147–160.

[33] Harfe, B. D., and Jinks-Robertson, S. Removal of Frameshift Intermediates by Mismatch Repair Proteins in Saccharomyces cerevisiae. *Molecular and Cellular Biology 19* (1999), 4766–4773.

[34] Hauben, L., Vauterin, L., Swings, J., and Moore, E. Comparison of 16S Ribosomal DNA Sequences of All Xanthomonas Species. *International Journal of Systematic Bacteriology 47* (1997), 328–335.

[35] Henikoff, S., and Henikoff, J. G. Automated assembly of protein blocks for database searching. *Nucleic Acids Research 19* (1991), 6565–6572.

[36] Henikoff, S., and Henikoff, J. G. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences of the United States of America 89* (1992), 10915–10919.

[37] Henikoff, S., Henikoff, J. G., and Pietrokovski, S. Blocks+: a non-redundant database of protein alignment blocks derived from multiple compilations. *Bioinformatics 15* (1999), 471–479.

[38] Hirschberg, D. S. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM 18* (1975), 341–343.

[39] Huang, X., and Zhang, J. Methods for comparing a DNA sequence with a protein sequence. *Computer Applications in the Biosciences : CABIOS 12* (1996), 497–506.

[40] Ikebe, S.-i., Tanaka, M., and Ozawa, T. Point mutations of mitochondrial genome in Parkinson's disease. *Molecular Brain Research 28* (1995), 281–295.

[41] Jones, D. T., Taylor, W. R., and Thornton, J. M. The rapid generation of mutation data matrices from protein sequences. *Computer Applications in the Biosciences : CABIOS 8* (1992), 275–282.

[42] Knuth, D. E. *The Art of Computer Programming, Volume 3: Sorting and Searching (2nd ed.)*. Addison Wesley, New York, 1999.

[43] Levenshtein, V. I. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. In *Soviet Physics Doklady* (1966), vol. 10, pp. 707–710.

[44] Li, W. H. *Molecular evolution.* Sinauer Associates, Massachusetts, 1997.

[45] Lipman, D. J., and Pearson, W. R. Rapid and sensitive protein similarity searches. *Science 227* (1985), 1435–1441.

[46] Luscombe, N. M., and Thornton, J. M. Protein-DNA Interactions: Amino Acid Conservation and the Effects of Mutations on Binding Specificity. *Journal of Molecular Biology 320* (2002), 991–1009.

[47] Lysholm, F. Highly improved homopolymer aware nucleotide-protein alignments with 454 data. *BMC Bioinformatics 13* (2012), 1–13.

[48] Marchington, D. R., Poulton, J., Sellar, A., and Holt, I. J. Do Sequence Variants in the Major Non-Coding Region of the Mitochondrial Genome Influence Mitochondrial Mutations Associated with Disease? *Human Molecular Genetics 5* (1996), 473–479.

[49] McGinnis, S., and Madden, T. L. Blast: at the core of a powerful and diverse set of sequence analysis tools. *Nucleic Acids Research 32* (2004), W20–W25.

[50] Müllauer, L., Gruber, P., Sebinger, D., Buch, J., Wohlfart, S., and Chott, A. Mutations in apoptosis genes: a pathogenetic factor for human disease. *Mutation Research/Reviews in Mutation Research 488* (2001), 211–231.

[51] Nakamura, Y., and Ito, K. How protein reads the stop codon and terminates translation. *Genes to Cells 3* (1998), 265–278.

[52] Needleman, S. B., and Wunsch, C. D. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. *Journal of Molecular Biology 48* (1970), 443–453.

[53] Pearson, W. R., and Lipman, D. J. Improved Tools for Biological Sequence Comparison. *Proceedings of the National Academy of Sciences of the United States of America 85* (1988), 2444–2448.

[54] Pearson, W. R., Wood, T., Zhang, Z., and Miller, W. Comparison of DNA Sequences with Protein Sequences. *Genomics 46* (1997), 24–36.

[55] Pietrokovski, S., Henikoff, J. G., and Henikoff, S. The Blocks database - a system for protein classification. *Nucleic Acids Research 24* (1996), 197–200.

[56] Quail, M. A., Smith, M., Coupland, P., Otto, T. D., Harris, S. R., Connor, T. R., Bertoni, A., Swerdlow, H. P., and Gu, Y. A tale of three next generation sequencing platforms: comparison of Ion Torrent, Pacific Biosciences and Illumina MiSeq sequencers. *BMC Genomics 13* (2012), 1–13.

[57] Ripley, L. S. Frameshift Mutation: Determinants of Specificity. *Annual Review of Genetics 24* (1990), 189–211.

[58] Russell, R. B., and Barton, G. J. Multiple Protein Sequence Alignment From Tertiary Structure Comparison: Assignment of Global and Residue Confidence Levels. *Proteins: Structure, Function, and Bioinformatics 14* (1992), 309–323.

[59] Sankoff, D., Cedergren, R. J., and Lapalme, G. Frequency of Insertion-Deletion, Transversion, and Transition in the Evolution of 5S Ribosomal RNA. *Journal of Molecular Evolution 7* (1976), 133–149.

[60] Shyu, C., Sheneman, L., and Foster, J. A. Multiple Sequence Alignment with Evolutionary Computation. *Genetic Programming and Evolvable Machines 5* (2004), 121–144.

[61] Smith, T. F., and Waterman, M. S. Identification of Common Molecular Subsequences. *Journal of Molecular Biology 147* (1981), 195–197.

[62] Strasfeld, L., and Chou, S. Antiviral Drug Resistance: Mechanisms and Clinical Implications. *Infectious Disease Clinics of North America 24* (2010), 413–437.

[63] Taylor, W. R. The Classification of Amino Acid Conservation. *Journal of Theoretical Biology 119* (1986), 205–218.

[64] Vingron, M., and Waterman, M. S. Sequence Alignment and Penalty Choice: Review of Concepts, Case Studies and Implications. *Journal of Molecular Biology 235* (1994), 1–12.

[65] Vogt, G., Etzold, T., and Argos, P. An Assessment of Amino Acid Exchange Matrices in Aligning Protein Sequences: The Twilight Zone Revisited. *Journal of Molecular Biology 249* (1995), 816–831.

[66] Walsh, T., Lee, M. K., Casadei, S., Thornton, A. M., Stray, S. M., Pennil, C., Nord, A. S., Mandell, J. B., Swisher, E. M., and King, M.-C. Detection of inherited mutations for breast and ovarian cancer using genomic capture and massively parallel sequencing. *Proceedings of the National Academy of Sciences of the United States of America 107* (2010), 12629–12633.

[67] Watson, J. D., and Crick, F. H. C. Molecular Structure of Nucleic Acids. *Nature 171* (1953), 737–738.

[68] Xia, X. Wobble Hypothesis. In *Brenner's Encyclopedia of Genetics (2nd Edition)*, S. Maloy and K. Hughes, Eds. Academic Press, San Diego, 2013, pp. 347–349.

[69] Yang, L., Zhang, X., and Wang, T. The Burrows-Wheeler similarity distribution between biological sequences based on Burrows-Wheeler transform. *Journal of Theoretical Biology 262* (2010), 742–749.

[70] Zhang, Z., Pearson, W. R., and Miller, W. Aligning a DNA Sequence with a Protein Sequence. *Journal of Computational Biology 4* (1997), 339–349.

[71] Zuckerkandl, E., and Pauling, L. Molecular Disease, Evolution, and Genetic Heterogeneity. In *Horizons in Biochemistry*, M. Kasha and B. Pullman, Eds. Academic Press, New York, 1962, pp. 189–225.

DANKSAGUNG

Danksagung aus Gründen des Datenschutzes entfernt.

LEBENSLAUF

Lebenslauf aus Gründen des Datenschutzes entfernt

Lebenslauf aus Gründen des Datenschutzes entfernt

Lebenslauf aus Gründen des Datenschutzes entfernt

PUBLIKATIONEN

Veröffentlichungen

Axel Fürstberger, Markus Maucher and Hans A. Kestler. Extended pairwise local alignment of wild card DNA/RNA-sequences using dynamic programming. *Journal of Statistical Computation and Simulation*, 85(1):3–13, 2015.

Vorträge

Axel Fürstberger and Hans A. Kestler. Extended pairwise local alignment of wild card DNA/RNA-sequences using dynamic programming. *Präsentation auf der 45th Arbeitstagung der Statistical Computing in Günzburg*, 2013.

Axel Fürstberger and Hans A. Kestler. Interactive zoom-able alignment graphs for pairwise wild base nucleotide protein alignment. *Präsentation auf der 46th Arbeitstagung der Statistical Computing in Günzburg*, 2014.

Poster

Christoph Müssel, Johann Kraus, Steffen Priebe, **Axel Fürstberger**, Torsten Thalheim, Jürgen Sühnel and Hans A. Kestler. An integrated database for data management, SOP analyses and metadata integration in ageing research. *Präsentation auf der 5th Conference on Systems Biology of Mammalian Cells in Berlin*, 2014.

Ludwig Lausser, Florian Schmid, Johann Kraus, **Axel Fürstberger** and Hans A. Kestler. Semantic gene set construction for multi classifier systems. *Präsentation auf der 5th Conference on Systems Biology of Mammalian Cells in Berlin*, 2014.

Axel Fürstberger, Marius Felder, Matthias Platzer and Hans A. Kestler. Pairwise local alignment of wild base containing nucleotide sequences and protein sequences. *Präsentation auf der Jährlichen Konferenz der Deutschen Gesellschaft für Alternsforschung in Jena*, 2015.

Axel Fürstberger, Johann M. Kraus and Hans A. Kestler. Integration of uncertain knowledge in an aging-related ontology. *Präsentation auf dem Gerontosys Konsortialtreffen in Köln*, 2015.

Herausgeber

Axel Fürstberger, Ludwig Lausser, Johann M. Kraus, Matthias Schmid and Hans A. Kestler. Ulmer Informatik-Berichte Nr. 2015-04: Abstracts der 47. Arbeitstagung der Statistical Computing. *Universität Ulm*, 2015.

Axel Fürstberger, Ludwig Lausser, Johann M. Kraus, Matthias Schmid and Hans A. Kestler. Ulmer Informatik-Berichte Nr. 2016-04: Abstracts der 48. Arbeitstagung der Statistical Computing. *Universität Ulm*, 2016.

Axel Fürstberger, Johann M. Kraus, Hans A. Kestler. Ulmer Informatik-Berichte Nr. 2016-05: Classification 2016 - Abstracts of the 5th German-Japanese Symposium. *Universität Ulm*, 2016.