Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Bachelorarbeit

Effiziente Methoden zur Erweiterung von Seeds bei der Berechnung lokaler Alignments

Vorgelegt von

Merle Stahl
Matrikelnummer 7297185

im Studiengang

B.Sc. Computing in Science
am Zentrum für Bioinformatik Hamburg
und Fachbereich Informatik
der MIN-Fakultät

11. Dezember 2023

1. Prüfer/in: Prof. Dr. Stefan Kurtz
2. Prüfer/in: Priv.-Doz. Dr. Peter Frommolt

Zusammenfassung

Um einen Mutationsprozess zu modellieren und Ähnlichkeiten zwischen zwei vor allem biologischen Sequenzen aufzuzeigen, werden diese typischerweise miteinander aligniert. Ein lokales Alignment bezieht sich dabei im Gegensatz zum globalen Alignment nur auf Sequenzteile und eignet sich deshalb besonders zum Auffinden ähnlicher Abschnitte. Da bei einigen biologischen Mechanismen, die zur Veränderung der DNA führen, ganze Substrings in die Sequenzen eingefügt oder daraus gelöscht werden, ist es sinnvoll, diese bei der Bewertung von Alignments in einem affinen Kostenmodell als Einheit zu betrachten. Der höhere Berechnungaufwand für solch ein Kostenmodell und die große Menge der zu vergleichenden Daten erfordert die Nutzung eines effizienten Seed-and-Extend Ansatzes zur Berechnung lokaler Alignments. Die Grundidee dieser Arbeit ist es, den WFA-Algorithmus zur effizienten Berechnung globaler Alignments unter affinen Gap-Kosten auf den Kontext der Seed-Erweiterung zu übertragen und mit einem Abbruchkriterium basierend auf Polished Points zu kombinieren. Dafür wurde der WFA Algorithmus in C++ Tempate-basiert neu implementiert. Durch die Nutzung fortgeschrittener Konzepte der C++ Programmierung, wie beispielsweise der partiellen Template-Spezifikation, entstand ein flexibler, gut strukturierter und einfach wiederverwendbarer Programmcode. Vor allem für kurz Sequenzen und kleine Fehlerraten erreicht der C++ basierte Code eine ähnliche oder geringere Laufzeit bei der Berechnung globaler Alignments als die originale C Implementierung. Durch eine Variante, die nur Distanzen berechnet, konnte sogar eine Verbesserung der Laufzeit bis zu einem Faktor von 14 erzielt werden. Vor allem für lange, unterschiedliche Sequenzen konnte dann gezeigt werden, dass der WFA-Algorithmus mit Verwendung der Polished Points in der C++ Implementierung an den passenden Stellen die Berechnung abbricht. Daher erscheint diese Implementierung sehr gut für den Einsatz in Seed-Extend Verfahren geeignet.

Inhaltsverzeichnis

Kapitel 1

1 Einleitung

Der paarweise Vergleich von Protein-, RNA- und DNA-Sequenzen ist eine der wichtigsten Methoden in der Bioinformatik, um funktionelle, strukturelle und evolutionäre Eigenschaften biologischer Sequenzen aufzuklären. Dabei wird ausgenutzt, dass ähnliche Sequenzen häufig zu ähnlichen biochemischen Funktionen und im Fall von Proteinen auch zu ähnlichen dreidimensionalen Strukturen führen. Ähnlichkeiten von Sequenzen aus verschiedenen Organismen können Aufschluss über den Grad ihrer Verwandtschaft geben. Auf diese Weise kann beispielsweise ein phylogenetischer Baum über die stammesgeschichtliche Entwicklung von Lebewesen rekonstruiert werden [1, 2, S. 54].

Um die Ähnlichkeit von Sequenzen zu beschreiben, wird typischerweise ein Alignment konstruiert. Dieses repräsentiert den Mutationsprozess mit Einfügungen, Löschungen und Ersetzungen einzelner Zeichen [1]. Man unterscheidet zwei grundlegende Formen von Alignments.

Beispiel 1 Folgend sind Alignments für zwei Sequenzen dargestellt. Beim Ersten handelt es sich um globales Alignment, während das Zweite ein lokales Alignment ist.

```
TTAATCGGGA-GAGAGGT     (global)
|    ||  | |  || |
T-CTGCGATACGGCAGCT

   TTAAT-CGGGAGAGAGGT  (lokal)
      || ||| ||
TCTGCGATACGGCAGCT
```

Beim globalen Alignment werden beide Sequenzen vollständig durch das Alignment repräsentiert. Dies ist besonders für ähnliche Sequenzen mit einer vergleichbaren Länge geeignet. Beim lokalen Alignment wiederum werden nur Sequenzabschnitte mit einer großen Ähnlichkeit betrachtet. Dabei haben diese Bereiche eine höhere Priorität als weitere benachbarte Sequenzteile. Besonders geeignet ist dies für verschiedene und unterschiedlich lange Sequenzen zum Auffinden ähnlicher Sequenzmotive [2, S. 53f]. So haben beispielsweise verwandte DNA-Sequenzen Ähnlichkeiten in den Genen, also in den kurzen codierenden Bereichen. Diese Ähnlichkeiten lassen sich durch globale Alignments aufgrund von langen nicht-codierenden Bereichen nur schwer identifizieren. Sie sind aber bei lokalen Alignments gut sichtbar [1].

Um unter der großen Zahl von Alignments die biologisch sinnvollen zu identifizieren, werden Alignments nach unterschiedlichen Modellen bewertet. Die Modelle unterscheiden sich beispielsweise darin, wie aufeinanderfolgende Löschungen oder Einfügungen (auch Gaps genannt) bewertet werden. Während das lineare Kostenmodell die Operationen unabhängig vom Kontext bewertet, also einen Wert proportional zur Länge des Gaps bildet, werden beim affinen Kostenmodell Gaps als Einheit betrachtet. Die Bewertung erfolgt dann durch Addition einer Konstanten, die unabhängig von der Länge des Gaps ist (Initialisierung), und einer Konstanten für jedes gelöschte beziehungsweise eingefügte Zeichen (Erweiterung) [1].

Beispiel 2 Die folgenden Alignments wurden durch die in dieser Arbeit entwickelte Software berechnet. Die Bewertung des ersten Alignments erfolgte nach dem linearen Modell. Die Bewertung des zweiten Modells erfolgte nach dem affinen Modell.

```
AAACGAACCCCTCCTTTACTAGAATTTGAATTCCGTTAACGTACGTACGT (lineare Kosten)
|||||||||||||||||||||||||| || | |  ||||||||||||
AAACGAACCCCTCCTTTACTAGAATTTGGAT-C-G--AACGTACGTACGT

AAACGAACCCCTCCTTTACTAGAATTTGAATTCCGTTAACGTACGTACGT (affine Kosten)
|||||||||||||||||||||||||| ||  ||  ||||||||||||
AAACGAACCCCTCCTTTACTAGAATTTGGAT--CG--AACGTACGTACGT
```

Beim linearen Modell gibt es keine Präferenz, nebeneinander liegende Gaps zusammenzufassen, da der Kontext der jeweiligen Löschung oder Ersetzung keinen Einfluss auf die Bewertung des Alignments hat. Das affine Modell mit passenden Bewertungen für die Gap-Initialisierung und -Erweiterung bewertet zusammengefasste Gaps hingegen besser und präferiert sie auf diese Weise. Daraus lässt sich auch die Motivation für die Bewertung mit affinen Kosten ableiten. Wie

in [3, S. 236f] gezeigt, gibt es mehrere biologische Szenarien, bei denen durch Einfügung oder Löschung ganzer Substrings die DNA verändert wird. Beispiele für solche Mechanismen sind das nicht-homologe Crossing-Over bei der Meiose, die Einfügungen von Transposons in die DNA, das Slipped Strang Mispairing (SSM) bei der DNA-Synthese, die Integration von DNA durch Retroviren oder auch die Translokation von DNA zwischen Chromosomen. Wenige, aber dafür längere Gaps im Alignment repräsentieren in solchen Kontexten besser den Veränderungsprozess als mehrere, kurze Gaps.

Da die Menge der verfügbaren biologischen Daten durch die fortlaufende Sequenzierung einer Vielzahl verschiedener Organismen stark wächst [4], nehmen auch die Größen der paarweise zu vergleichenden Sequenzmengen immer weiter zu. Der Smith-Waterman Algorithmus als klassische Lösung für die Berechnung lokaler Alignments mit einer Laufzeit von $O(|u| \cdot |v|)$ für zwei Sequenzen u und v ist in diesem Kontext nur noch bedingt anwendbar. Viele Methoden nutzen deshalb eine Seed-and-Extend Strategie für eine effizientere Berechnung lokaler Alignments. Dafür werden zuerst Seeds, also kurze, nahezu identische Sequenzbereiche, bestimmt und dann dynamisch, gesteuert durch die Sequenzinhalte, nach links und rechts erweitert [5].

Das Ziel dieser Arbeit war die effiziente, strukturierte und wiederverwendbare Implementierung eines Algorithmus zur Seed-Erweiterung unter affinen Gap-Kosten. Um alle genanten Eigenschaften der Implementierung zu erreichen, erfolgte die Implementierung in C++. Die Nutzung von Templates sollte dabei die Wiederverwendbarkeit der Codes ermöglichen, ohne aber Kompromisse bezüglich der Laufzeit einzugehen. Zudem war auch die Portabilität der Programmiersprache und die Verfügbarkeit vorhandener Template-Libraries und moderner Compiler zur Generierung von hochoptimiertem Objektcode ein wichtiger Grund für die Wahl von C++.

Kapitel 2

2 Stand der Wissenschaft

Die Sequenzen, die paarweise miteinander verglichen werden, werden in den nachfolgenden Ausführungen durch u und v bezeichnet. Sie haben die Längen $m = |u|$ und $n = |v|$.

2.1 Paarweises Sequenzalignment

Das wichtigste Modell für den Vergleich biologischer Sequenzen ist das Modell der Edit-Distanz. Dabei wird ein Alignment aus Edit-Operationen konstruiert, um Gemeinsamkeiten oder ähnliche Regionen in den Sequenzen zu identifizieren. Die Edit-Operationen werden mittels einer Kostenfunktion bewertet. Die Kosten eines Alignments ergeben sich aus der Summe der Kosten der Edit-Operationen. Statt Kostenfunktionen kann man auch Score-Funktionen verwenden. Diese erlauben auch negative Werte [1]. Die Notationen in diesem Abschnitt wurden von [1] übernommen.

Definition 1 Eine *Edit-Operation* ist ein Paar $(\alpha, \beta) \in (\mathcal{A} \cup \{\epsilon\}) \times (\mathcal{A} \cup \{\epsilon\}) \setminus \{(\epsilon, \epsilon)\}$ für das Alphabet $\mathcal{A}$.

Eine *Edit-Operation* (α, β) wird häufig auch als $\alpha \to \beta$ geschrieben. Es können dabei die drei folgenden Operationen unterschieden werden:

- $\alpha \to \epsilon$ beschreibt eine Löschung des Zeichens $\alpha \in \mathcal{A}$.
- $\epsilon \to \beta$ beschreibt eine Einfügung des Zeichens $\beta \in \mathcal{A}$.
- $\alpha \to \beta$ beschreibt eine Ersetzung des Zeichens $\alpha \in \mathcal{A}$ durch $\beta \in \mathcal{A}$.

Eine Reihe von aufeinanderfolgenden Löschungen oder Einfügungen wird auch *Gap* genannt.

Definition 2 Ein *globales Alignment* A von u und v ist eine Folge $(\alpha_1 \to \beta_1, \ldots, \alpha_h \to \beta_h)$ von Edit-Operationen, sodass $u = \alpha_1 \ldots \alpha_h$ und $v = \beta_1 \ldots \beta_h$ gilt.

Die Sequenz u wird also durch ein globales Alignment vollständig in v überführt.

Beispiel 3 Die Sequenz $u =$ `ZEITGEIST` kann durch (1) die Ersetzungen `Z` $\to$ `F` und `G` $\to$ `Z`, (2) eine Einfügung $\epsilon \to$ `R` und (3) die Löschungen `T` $\to \epsilon$ und `S` $\to \epsilon$ vollständig in $v =$ `FREIZEIT` überführt werden:

$$\texttt{ZEITGEIST} \xrightarrow{(1)} \texttt{FEITZEIST} \xrightarrow{(2)} \texttt{FREITZEIST} \xrightarrow{(3)} \texttt{FREIZEIT}$$

Hier ist das entsprechende Alignment in einer typischen Schreibweise dargestellt. Dabei steht $-$ für eine Löschung oder Einfügung und $|$ für eine Übereinstimmung.

```
Z-EITGEIST
  ||  || |
FREI-ZEI-T
```

Zur Bewertung von Alignments werden Funktionen definiert, die den Edit-Operationen Kosten zuordnen und diese jeweils für die Alignments aufsummieren. Die minimalen Kosten aller möglichen Alignments zweier Sequenzen lassen sich als Maß für deren Ähnlichkeit interpretieren.

Definition 3 Eine *(lineare Gap-)Kostenfunktion* δ ordnet allen $(\alpha, \beta) \in (\mathcal{A}^1 \cup \{\epsilon\}) \times (\mathcal{A}^1 \cup \{\epsilon\}) \setminus \{(\epsilon, \epsilon)\}$ die Kosten $\delta(\alpha \to \beta) \in \mathbb{R}$ zu, sodass $\delta(\alpha \to \beta) \geq 0$ für alle Ersetzungen und $\delta(\alpha \to \beta) > 0$ für alle Einfügungen und Löschungen gilt. Für ein Alignment $A = (\alpha_1 \to \beta_1, \ldots, \alpha_h \to \beta_h)$ ergibt sich dann:

$$\delta(A) = \sum_{i=1}^{h} \delta(\alpha_i \to \beta_i) \tag{2.1}$$

Um Gaps als Einheit zu bewerten, kann ein affines Gap-Kostenmodell verwendet werden.

Definition 4 Eine *affine Gap-Kostenfunktion* δ_{afn} liefert für alle $(\alpha, \beta) \in (\mathcal{A}^+ \times \{\epsilon\}) \cup (\{\epsilon\} \times \mathcal{A}^+)$ Kosten $\delta_{afn}(\alpha \to \beta) \in \mathbb{R}$, sodass für alle $w \in \mathcal{A}^+$ gilt:

$$\delta_{afn}(w \to \epsilon) = g + \sum_{i=1}^{|w|} \delta_{afn}(w[i] \to \epsilon)$$

$$\delta_{afn}(\epsilon \to w) = g + \sum_{i=1}^{|w|} \delta_{afn}(\epsilon \to w[i])$$

Dabei steht g für die Gap-Initialisierungskosten.

Beispiel 4 Sei δ eine Kostenfunktion mit folgenden Werten:

$$\delta(\alpha \to \beta) = \begin{cases} 0 & \text{falls } \alpha, \beta \in \mathcal{A} \text{ und } \alpha = \beta \\ 1 & \text{sonst} \end{cases}$$

Diese Kostenfunktion wird auch *Einheitskostenfunktion* genannt. Sei δ_{afn} eine affine Gap-Kostenfunktion mit

$$\delta_{afn}(\alpha \to \beta) = \begin{cases} 0 & \text{falls } \alpha, \beta \in \mathcal{A} \text{ und } \alpha = \beta \\ 4 & \text{falls } \alpha, \beta \in \mathcal{A} \text{ und } \alpha \neq \beta \\ 6 + |\alpha| \cdot 2 & \text{falls } \alpha \in \mathcal{A}^+ \text{ und } \beta = \epsilon \\ 6 + |\beta| \cdot 2 & \text{falls } \alpha = \epsilon \text{ und } \beta \in \mathcal{A}^+ \end{cases}$$

Dann gilt für das globale Alignment A aus Beispiel 3:

$$\begin{aligned} \delta(A) &= 1+1+0+0+1+1+0+0+1+0 = 5 \\ \delta_{afn}(A) &= 4+(6+1\cdot 2)+0+0+(6+1\cdot 2)+4+0+0+\ (6+1\cdot 2)+0 = 32 \end{aligned}$$

Definition 5 Die *Edit-Distanz (unter linearen Gap-Kosten)* $edist_\delta(u, v)$ von u und v sind die minimalen Kosten aller globalen Alignments von u und v bei Bewertung durch die Kostenfunktion δ:

$$edist_\delta(u, v) = \min\{\delta(A) \mid A \text{ ist ein Alignment von } u \text{ und } v\} \tag{2.2}$$

Ein globales Alignment A von u und v heißt *optimal*, falls $\delta(A) = edist_\delta(u, v)$ gilt.

Definition 6 Die *Edit-Distanz unter affinen Gap-Kosten* $edist_{\delta_{afn}}(u, v)$ von u und v wird entsprechend definiert durch:

$$edist_{\delta_{afn}}(u, v) = \min\{\delta_{afn}(A) \mid A \text{ ist ein Alignment von } u \text{ und } v\} \tag{2.3}$$

Ein globales Alignment A von u und v wird analog *optimal* genannt, falls $\delta_{afn}(A) = edist_{\delta_{afn}}(u, v)$ gilt.

Definition 7 Beim *Edit-Distanz Problem (unter linearen Gap-Kosten δ)* geht es darum, $edist_\delta(u, v)$ und alle optimalen globalen Alignments bezüglich δ von u und v zu bestimmen.

Definition 8 Das *Edit-Distanz Problem unter affinen Gap-Kosten* δ_{afn} besteht analog daraus, $edist_{\delta_{afn}}(u, v)$ und alle optimalen globalen Alignments von bezüglich δ_{afn} u und v zu finden.

Definition 9 Ein *lokales Alignment* A von u und v ist eine Alignment von u' und v', wobei u' ein Substring von u und v' ein Substring von v ist.

Da ϵ ein Substring jeder Sequenz ist und die Distanz von ϵ und ϵ 0 ist, würde eine Minimierung der Kosten aller lokalen Alignments immer zu Kosten von 0 führen [1]. Deshalb wird in diesem Kontext ein sogenannter Score maximiert.

Definition 10 Eine *Score-Funktion* σ liefert für alle $(\alpha, \beta) \in (\mathcal{A}^1 \cup \{\epsilon\}) \times (\mathcal{A}^1 \cup \{\epsilon\}) \setminus \{(\epsilon, \epsilon)\}$ einen Score $\sigma(\alpha \to \beta) \in \mathbb{R}$, wodurch sich die Bewertung eines Alignments $A = (\alpha_1 \to \beta_1, \ldots, \alpha_h \to \beta_h)$ analog zur Kostenfunktion auf folgende Weise ergibt:

$$\sigma(A) = \sum_{i=1}^{h} \sigma(\alpha_i \to \beta_i) \tag{2.4}$$

Definition 11 Der *maximale Score* $loc_\sigma(u, v)$ von u und v für alle Substrings für die Score-Funktion σ berechnet sich durch:

$$loc_\sigma(u, v) = \max\{score_\sigma(u', v') \mid u' \text{ ist ein Substring von } u \text{ und } v' \text{ ist ein Substring von } v\} \tag{2.5}$$

mit $score_\sigma(u', v') = \max\{\sigma(A) \mid A \text{ ist ein Alignment von } u' \text{ und } v'\}$

Ein lokales Alignment A von u und v heißt *optimal*, falls $\sigma(A) = loc_\sigma(u, v)$ gilt.

Definition 12 Beim *Lokalen Alignment Problem* geht es darum, $loc_\sigma(u, v)$ und ein optimales lokales Alignment von u und v zu bestimmen.

2.2 DP-Algorithmus zur Lösung des Edit-Distanz Problems

Dieser Abschnitt nutzt ebenfalls die Notationen und Erläuterungen aus [1]. Um das Edit-Distanz Problem für die Kostenfunktion δ zu lösen, kann dynamische Programmierung und eine $(m+1) \times (n+1)$-Matrix verwendet werden.

Definition 13 Die *Edit-Distanz DP-Matrix* E_δ wird für $0 \leq i \leq m$ und $0 \leq j \leq n$ wie folgt definiert:

$$E_\delta(i, j) = edist_\delta(u[1 \ldots i], v[1 \ldots j]) \tag{2.6}$$

Um diese Matrix zu berechnen, wird das Problem auf ein Problem des kürzesten Weges in einem gewichteten, gerichteten und azyklischen Graphen, auch Edit-Graph genannt, übertragen. Der kürzeste Pfad entspricht dabei dem Pfad mit dem minimalen Gewicht. Diese Technik wird im Folgenden skizziert und durch Abbildung 2.1 verdeutlicht. Für jeden Matrixeintrag wird ein Knoten definiert.

Entsprechend der drei Edit-Operationen werden für jeden Knoten bis zu drei Kanten eingefügt, welche den Wert der Kostenfunktion für die jeweilige Operation als Gewicht erhalten. Die Pfade im Graphen und die zugehörigen Gewichte können dann bijektiv auf die Alignments und deren Kosten abgebildet werden. Ein kürzester Pfad von $(0,0)$ nach (m,n) korrespondiert mit einem optimalen Alignment der Sequenzen und das Gesamtgewicht dieses Weges entspricht der Edit-Distanz von u und v.

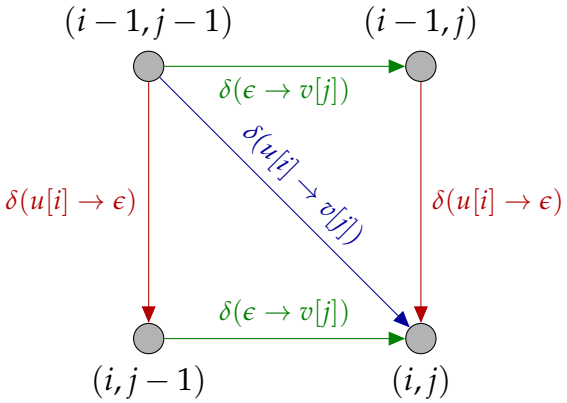

Abbildung 2.1: Ausschnitt des Edit-Graphen zur Berechnung von E_δ (in Anlehnung an [1]).

Daraus lässt sich eine Rekurrenz ableiten, welche ermöglicht, die Werte in topologischer Reihenfolge der Knoten des Edit-Graphen zu berechnen. Dafür ist beispielsweise eine spaltenweise Berechnung möglich und üblich.

Lemma 1 Für $0 \leq i \leq m$ und $0 \leq j \leq n$ gilt:

$$E_\delta(i,j) = \begin{cases} 0 & \text{falls } i = 0 \text{ und } j = 0 \\ E_\delta(0,j-1) + \delta(\epsilon \to v[j]) & \text{falls } i = 0 \text{ und } j > 0 \\ E_\delta(i-1,0) + \delta(u[i] \to \epsilon) & \text{falls } i > 0 \text{ und } j = 0 \\ \min \left\{ \begin{array}{l} E_\delta(i-1,j) + \delta(u[i] \to \epsilon) \\ E_\delta(i,j-1) + \delta(\epsilon \to v[j]) \\ E_\delta(i-1,j-1) + \delta(u[i] \to v[j]) \end{array} \right\} & \text{falls } i > 0 \text{ und } j > 0 \end{cases} \tag{2.7}$$

Der Eintrag (m,n) in E_δ entspricht $edist_\delta(u,v)$. Zur Berechnung eines optimalen globalen Alignments wird ein Backtracing vom Eintrag (m,n) bis $(0,0)$ entlang eines minimierenden Pfades durchgeführt.

Der Algorithmus erreicht eine Laufzeit von $O(m \cdot n)$ und einen Speicherplatzbedarf von $O(\min\{m,n\})$ bei der Distance-Only Variante und von $O(m \cdot n)$ mit der Berechnung eines Alignments.

2.3 Algorithmus von Gotoh

Der Algorithmus von Gotoh [6] modifiziert das vorherige Verfahren und berechnet so das Edit-Distanz Problem unter affinen Gap-Kosten δ_{afn} durch dynamische Programmierung.

Die Verwendung eines affinen Gap-Kostenmodells wird ermöglicht, indem ein Element der $(m+1) \times (n+1)$-Matrix in drei Komponenten zerlegt wird [1].

Definition 14 Die *Gotoh DP-Matrix* wird für $0 \leq i \leq m$ und $0 \leq j \leq n$ wie folgt definiert:

$$\begin{aligned} D(i,j) = \min(\{\infty\} \cup \{\delta_{afn}(A) \mid A \text{ ist ein Alignment von } u[1 \ldots i] \text{ und } v[1 \ldots j] \\ \text{und } A \text{ endet mit einer Löschung}\}) \qquad (2.8) \\ I(i,j) = \min(\{\infty\} \cup \{\delta_{afn}(A) \mid A \text{ ist ein Alignment von } u[1 \ldots i] \text{ und } v[1 \ldots j] \\ \text{und } A \text{ endet mit einer Einfügung}\}) \\ M(i,j) = \min(\{\infty\} \cup \{edist_{\delta_{afn}}(u[1 \ldots i], v[1 \ldots j])\}) \end{aligned}$$

Dabei werden bei D und I nur die Fälle betrachtet, in welchen ein Alignment mit einem Gap endet. Die M Komponente minimiert über alle möglichen Alignments und enthält so die Edit-Distanz unter affinen Gap-Kosten des jeweiligen Präfixpaares der Sequenzen.

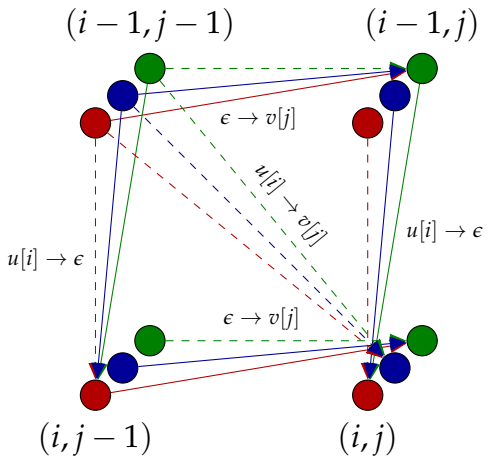

Abbildung 2.2: Ausschnitt des affinen Edit-Graphen zur Berechnung von M, D und I. Dabei steht ● für eine Einfügung, ● für eine Ersetzung und ● für eine Löschung. Eine Gap-Intialisierung wird durch —— dargestellt, während - - - - eine normale Kante beschreibt (in Anlehnung an [1]).

Um eine Rekurrenz zur Berechnung der Werte herzuleiten, werden die Knoten im Edit-Graphen in drei Komponenten aufgeteilt, die den Edit-Operationen

entsprechen [1]. Dieses Prinzip wird in Abbildung 2.2 dargestellt. Dadurch wird ermöglicht, dass zwischen einer Gap-Intialisierung und -Erweiterung unterschieden werden kann. Die Aufteilung des Graphen verändert nicht dessen Eigenschaften, sodass die Pfade im Graphen und dessen Gewichte wieder mit Alignments und deren diesmal affinen Gap-Kosten assoziert werden können [1]. Daraus lässt sich mit zusätzlichen Optimierungen eine Rekurrenz ableiten [1,6].

Lemma 2 Die *Gotoh DP-Matrix* kann für $0 \leq i \leq m$ und $0 \leq j \leq n$ durch folgende Rekurrenz berechnet werden [1,4]:

$$D(i,j) = \begin{cases} g & \text{falls } i = 0 \text{ und } j = 0 \\ \infty & \text{falls } i = 0 \text{ und } j > 0 \\ \min \left\{ \begin{array}{l} M(i-1,j) + g + \delta(u[i] \to \epsilon) \\ D(i-1,j) + \delta(u[i] \to \epsilon) \end{array} \right\} & \text{falls } i > 0 \end{cases} \tag{2.9}$$

$$I(i,j) = \begin{cases} g & \text{falls } i = 0 \text{ und } j = 0 \\ \infty & \text{falls } i > 0 \text{ und } j = 0 \\ \min \left\{ \begin{array}{l} M(i,j-1) + g + \delta(\epsilon \to v[j]) \\ I(i,j-1) + \delta(\epsilon \to v[j]) \end{array} \right\} & \text{falls } j > 0 \end{cases} \tag{2.10}$$

$$M(i,j) = \begin{cases} 0 & \text{falls } i = 0 \text{ und } j = 0 \\ D(i,j) & \text{falls } i > 0 \text{ und } j = 0 \\ I(i,j) & \text{falls } i = 0 \text{ und } j > 0 \\ \min \left\{ \begin{array}{l} M(i-1,j-1) + \delta(u[i-1] \to v[j-1]) \\ D(i,j) \\ I(i,j) \end{array} \right\} & \text{falls } i > 0 \text{ und } j > 0 \end{cases} \tag{2.11}$$

Eine spaltenweise Berechnung der Elemente der Gotoh Matrix ist erneut eine mögliche Lösung, wobei aber jeweils die Werte der D und I Komponente vor der M Komponente berechnet werden müssen.

Es gilt $edist_{\delta_{afn}}(u,v) = M(m,n)$ [1,4]. Zur Berechnung eines optimalen Alignments kann analog zur vorherigen Methode ein Backtracing von (m,n) bis $(0,0)$ durchgeführt werden.

Auf diese Weise erreicht diese Technik ebenfalls eine Laufzeit von $O(m \cdot n)$ und einen Speicherplatzbedarf von $O(\min\{m,n\})$ bei der Distance-Only Variante und von $O(m \cdot n)$ mit der Berechnung eines Alignments [1]. Da aber die Anzahl der zu berechnenden Werte zugenommen hat, wird dies bei der realen Laufzeit und dem Speicherplatzbedarf eines Programms ebenfalls der Fall sein [1].

2.4 Wavefront Algorithmus

Die Darstellung in diesem Abschnitt basiert auf [7]. Der Wavefront Algorithmus wurde unabhängig voneinander zuerst in [8] und [9] beschrieben. Er berechnet die Edit-Distanz unter dem Einheitskosten-Modell δ und betrachtet die Edit-Distanz Matrix E_δ dafür diagonalweise.

Definition 15 Zu $h \in [-m, n]$ sei eine *(Vorwärts-)Diagonale* h die Menge aller Paare (i, j) in E_δ, $0 \leq i \leq m$, $0 \leq j \leq n$, für die $j - i = h$ gilt.

Die Abbildung 2.3a zeigt beispielhaft die Diagonalen in der Edit-Distanz Matrix für zwei Sequenzen `FREIZEIT` und `ZEITGEIST`.

Definition 16 Ein *Front-Wert* $front(h, d)$ für die Kosten $d \in \mathbb{N}_0$ und die Diagonale $-d \leq h \leq d$ wird wie folgt definiert:

$$front(h, d) = \max\{i \mid 0 \leq i \leq m, E_\delta(i, h + i) = d\} \tag{2.12}$$

Definition 17 Eine *Wavefront* für die Kosten $d \in \mathbb{N}_0$ enthält alle Front-Werte $front(h, d)$.

Bei Front-Werten handelt es sich also um die maximale Zeile einer Diagonale in der Edit-Distanz Matrix für definierte Kosten. Um diese Werte effizient zu bestimmen, kann erneut eine Übertragung des Problems auf den Edit-Graphen verwendet werden. Hierfür werden sogenannte d-Pfade definiert. Dabei handelt es sich um Pfade im Edit-Graphen mit genau d Kanten, die jeweils Kosten von 1 haben. Der Front-Wert für die Kosten d und die Diagonale h entspricht der maximalen Zeile i aller Endpunkte (i, j) der von $(0, 0)$ ausgehenden d-Pfade, welche eine maximale Länge haben und genau auf h enden. Durch das Aufteilen dieses d-Pfades in 3 Teile mit

(1) einem maximalen $(d - 1)$-Pfad,

(2) einer Kante mit Kosten von 1, also einer Einfügung, Löschung oder Nicht-Übereinstimmung, und

(3) einem maximalen 0-Pfad aus Übereinstimmungen.

lässt sich eine Rekurrenz für die Berechnung der Front-Werte ableiten.

Lemma 3 Für $d \in \mathbb{N}_0$ und $-d \leq h \leq d$ gilt:

$$front(h,d) = \ell + |lcp(u[\ell+1 \ldots m],\ v[h+\ell+1 \ldots n])| \tag{2.13}$$

$$\text{mit } \ell = \begin{cases} 0 & \text{falls } d = 0 \\ \max \left\{ \begin{array}{l} front(h-1, d-1) \\ front(h+1, d-1)+1 \\ front(h, d-1)+1 \end{array} \right\} & \text{falls } d > 0 \end{cases}$$

Das Ziel dieser Technik ist es, die minimalen Kosten d zu bestimmen, bei denen ein maximaler d-Pfad von $n-m$ auf (m,n) endet. $E_\delta(m,n)$ entspricht in diesem Kontext also $front(h,d) = m$ mit $h = n-m$ und $d = edist_\delta(u,v)$.

In Algorithmus 1 werden dafür die erlaubten Kosten sukzessive erhöht (siehe Zeile 2), bis das genannte Terminationskriterium erfüllt ist (siehe Zeile 5). Dabei wird in jedem Schritt über die zu betrachtenden Diagonalen iteriert (siehe Zeile 3) und der jeweilige Front-Wert entsprechend Gleichung (2.13) berechnet (siehe Zeile 4).

Algorithmus 1 Wavefront Algorithmus

Eingabe: Sequenzen u und v, Einheitskostenfunktion δ
Ausgabe: $edist_\delta(u,v)$

```
1: (m, n) ← (|u|, |v|)
2: for d ← 0 upto max{m, n} do
3:     for h ← −d upto d do
4:         Berechne front(h, d) nach Gl. 2.13
5:     if −d ≤ n − m ≤ d and front(n − m, d) = m then
6:         break
7: return d
```

Die Wavefronten für zwei Sequenzen sind in Abbildung 2.3b zu sehen. Hier lässt sich besonders die iterative Entwicklung der Wavefronten erkennen, bis der Algorithmus mit dem Front-Wert $front(n-m,d) = front(1,5) = 8$ für die Kosten 5 terminiert.

Diese Technik lässt sich um die Berechnung eines optimalen Alignments erweitern, indem ebenfalls ein Backtracing rückwärts von (m,n) bis zu $(0,0)$ durchgeführt wird.

Die Effizienz des Algorithmus ist abhängig von der Eingabe. Er erreicht eine Laufzeit von $O(m+n+edist_\delta(u,v)^2)$ und einen Speicherplatzbedarf von $O(edist_\delta(u,v))$ bei der Distance-Only Variante und von $O(edist_\delta(u,v)^2)$ mit der Berechnung eines Alignments. Somit ist diese Technik besonders für ähnliche Sequenzen geeignet und effizienter als ein DP-Ansatz.

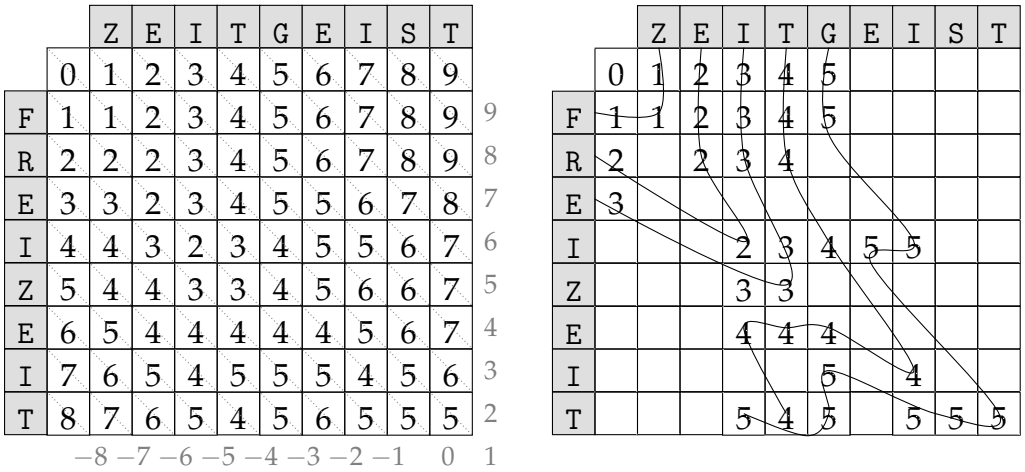

Abbildung 2.3: Edit-Distanz-Matrix unter den Einheitskosten mit eingezeichneten Diagonalen und Wavefronten für die Kosten 1 bis 5 für die Sequenzen $u =$ `FREIZEIT` und $v =$ `ZEITGEIST` [7].

2.5 WFA Algorithmus

Der WFA (Wavefront Alignment) Algorithmus von [4] ist eine Verallgemeinerung des Wavefront Algorithmus. Dabei überträgt er die Idee des vorherigen Algorithmus auf den Algorithmus von Gotoh, sodass globale Alignments von zwei Sequenzen auch nach dem affinen Gap-Kostenmodel bewertet werden können.

Im Folgenden wird dieses Kostenmodell durch $\delta_{afn} = (x, o, e)$ dargestellt. Dabei beschreiben x die Kosten für eine Nicht-Übereinstimmung und o und e die Gap-Kosten (Initialisierung und Erweiterung). Dieses Verfahren nimmt immer Kosten von 0 für eine Übereinstimmung an [4].

Definition 18 Ein *affiner Front-Wert* für die Kosten $d \in \mathbb{N}_0$ und die Diagonale $lo(d) \leq h \leq hi(d)$ wird wie folgt definiert:

$$\widetilde{D}(d,h) = \max\{i \mid 0 \leq i \leq m, D(i,h+i) = d\} \tag{2.14}$$

$$\widetilde{I}(d,h) = \max\{i \mid 0 \leq i \leq m, I(i,h+i) = d\} \tag{2.15}$$

$$\widetilde{M}(d,h) = \max\{i \mid 0 \leq i \leq m, M(i,h+i) = d\} \tag{2.16}$$

Definition 19 Eine *affine Wavefront* für die Kosten $d \in \mathbb{N}_0$ enthält alle affinen Front-Werte von d.

Um die affinen Front-Werte durch eine Rekurrenz zu bestimmen, werden analog zum Wavefront Algorithmus Pfade in der Gotoh-Matrix betrachtet.

Lemma 4 Ein *affiner Front-Wert* für die Kosten $d \in \mathbb{N}_0$ und die Diagonale $lo(d) \leq h \leq hi(d)$ kann durch folgende Rekurrenz berechnet werden [4]:

$$\widetilde{D}(d,h) = \max\left\{\widetilde{M}(d-o-e,h+1), \widetilde{D}(d-e,h+1)\right\} \tag{2.17}$$

$$\widetilde{I}(d,h) = \max\left\{\widetilde{M}(d-o-e,h-1), \widetilde{I}(d-e,h-1)\right\} + 1 \tag{2.18}$$

$$\widetilde{M}(d,h) = \ell + |lcp(u[\ell+1\ldots m], v[h+\ell+1\ldots n])| \tag{2.19}$$

$$\text{mit } \ell = \begin{cases} 0 & \text{falls } d = 0 \\ \max\left\{\widetilde{M}(d-x,h)+1, \widetilde{D}(d,h), \widetilde{I}(d,h)\right\} & \text{falls } d > 0 \end{cases}$$

Die Berechnung der Wavefront für ein gegebenes d ist dabei nicht wie beim einfachen Wavefront Algorithmus nur von der Wavefront für $d-1$ abhängig, sondern bedingt durch das Kostenmodell von beliebigen Wavefronten für $d' < d$. So werden die kleinste und größte Diagonale für eine Distanz d nicht durch Verringern beziehungsweise Erhöhen des vorherigen Wertes um 1, sondern nach genau diesen Abhängigkeiten berechnet [4].

Definition 20 Für alle $d \in \mathbb{N}_0$ seien $lo(d)$ und $hi(d)$ der kleinste beziehungsweise größte zu berechnende Diagonalindex der Wavefront für d.

Lemma 5 Für $d \in \mathbb{N}_0$ gilt

$$lo(d) = \begin{cases} 0 & \text{falls } d < \min\{x, o+e, e\} \\ \min\{lo(d-x), lo(d-o-e), lo(d-e)\} - 1 & \text{falls } d \geq \min\{x, o+e, e\} \end{cases}$$

$$hi(d) = \begin{cases} 0 & \text{falls } d < \min\{x, o+e, e\} \\ \max\{hi(d-x), hi(d-o-e), hi(d-e)\} + 1 & \text{falls } d \geq \min\{x, o+e, e\} \end{cases}$$

$edist_{\delta_{afn}}(u,v) = M(m,n)$ entspricht in diesem Kontext $\widetilde{M}(h,d) = m$ mit $h = n-m$ und $d = edist_{\delta_{afn}}(u,v)$. Das Ziel dieser Technik ist es also, die minimalen Kosten d zu bestimmen, bei denen die $\widetilde{M}$-Komponente des affinen Front-Wertes der Diagonalen $n-m$ der aktuellen Wavefront m ist [4].

In Algorithmus 2 wird die $\widetilde{M}$-Komponent im ersten Schritt mit der Anfangsbedingung initialisiert (siehe Zeile 2). Dann werden die erlaubten Kosten analog zum Wavefront Algorithmus sukzessive erhöht (siehe Zeile 3) und die jeweilige Wavefront diagonalweise nach den Gleichungen (2.17)-(2.19) berechnet (siehe Zeile 5). Falls mit dieser Wavefront das beschriebene Terminationskriterium erfüllt wird (siehe Zeile 6), bricht der Algorithmus mit Berechung eines globalen Alignments (siehe Zeile 8) ab, indem in jedem Schritt beim Backtracing der minimale Vorgänger ermittelt wird [4].

Der Algorithmus erreicht auf diese Weise eine Laufzeit von $O(\max\{m,n\} \cdot edist_{\delta_{afn}}(u,v))$ und einen Speicherplatzbedarf von $O(edist_{\delta_{afn}}(u,v)^2)$ [4].

Algorithmus 2 WFA Algorithmus [4]

Eingabe: Sequenzen u und v, Kosten $\delta_{afn} = (x, o, e)$
Ausgabe: globales Alignment A von u und v unter δ_{afn}

1: $(m, n) \leftarrow (|u|, |v|)$
2: $\widetilde{M}(0,0) \leftarrow |lcp(u, v)|$
3: **for** $d \leftarrow 1$ **upto** ∞ **do**
4: **for** $h \leftarrow lo(d)$ **upto** $hi(d)$ **do**
5: Berechne $\widetilde{I}(d,h)$, $\widetilde{D}(d,h)$, $\widetilde{M}(d,h)$ nach Gl. (2.17)-(2.19)
6: **if** $lo(d) \leq n - m \leq hi(d)$ and $\widetilde{M}(d, n-m) \geq m$ **then**
7: **break**
8: **return** $A \leftarrow$ Backtracing von $\widetilde{M}(d, n-m)$ bis $\widetilde{M}(0,0)$

2.5.1 Implementierung des WFA Algorithmus

Die Autoren von [4] stellen den Algorithmus in einer Bibliothek [10] als C Implementierung bereit.

		Z			E			I			T			G			E			I			S			T		
	6	6	0	∞	8	8	∞	10	10	∞	12	12	∞	14	14	∞	16	16	∞	18	18	∞	20	20	∞	22	22	∞	24	24	
F	8	∞	8	16	16	4	18	12	12	20	14	14	22	16	16	24	18	18	26	20	20	28	22	22	30	24	24	32	26	26	9
R	10	∞	10	12	18	12	20	20	8	22	16	16	24	18	18	26	20	20	28	22	22	30	24	24	32	26	26	34	28	28	8
E	12	∞	12	14	20	14	16	22	12	24	20	12	26	20	20	28	22	22	30	24	20	32	26	26	34	28	28	36	30	30	7
I	14	∞	14	16	22	16	18	24	18	20	26	12	28	20	16	30	22	22	28	24	24	34	26	20	36	28	28	38	30	30	6
Z	16	∞	16	18	24	14	20	22	20	20	24	20	24	26	16	30	24	20	30	26	26	28	28	28	36	30	24	38	32	32	5
E	18	∞	18	20	26	20	22	28	14	22	22	22	24	24	24	28	26	20	32	28	20	30	28	28	32	30	30	40	32	28	4
I	20	∞	20	22	28	22	22	30	22	24	30	14	26	22	22	28	24	24	28	26	24	32	28	20	34	28	28	36	30	30	3
T	22	∞	22	24	30	24	24	32	24	22	32	22	28	30	14	30	22	22	30	24	24	28	26	26	36	28	24	38	30	**28**	2
		−8			−7			−6			−5			−4			−3			−2			−1			0			1

Abbildung 2.4: Gotoh DP-Matrix (D, I, M) mit eingezeichneten Diagonalen für die Sequenzen u = `FREIZEIT` und v = `ZEITGEIST` und die Kostenfunktion $\delta_{afn} = (4, 6, 2)$.

Die Gotoh DP-Matrix für zwei Sequenzen ist in Abbildung 2.4 dargestellt. Hier lässt sich erkennen, dass abhängig vom Kostenmodell einige Wavefronten nicht benötigt werden [4]. So können bei den Kosten $\delta_{afn} = (4, 6, 2)$ beispielsweise alle ungeraden Wavefronten bei der Berechnung übersprungen werden. Dies wurde in der Implementierung ausgenutzt, indem die Wavefronten, die keinen definierten Vorgänger haben, auf `NULL` gesetzt und nicht berechnet werden [4]. Im anderen

Fall existiert eine Fallunterscheidung, um nur die benötigten Komponenten eines Front-Wertes zu berechnen [4]. Um bei den Berechnung aufbauender Wavefronten Verzweigungen zu vermeiden, wird eine `null_wavefront` mit Standardwerten und ein Loop Unrolling zur eigenständigen Handhabung von Grenzwerten verwendet [10]. Dadurch vereinfacht die Implementierung Datenabhängigkeiten und Fallunterscheidungen, die ein moderner Compiler ausnutzen kann, um effizienter ausführbaren Code (beispielsweise durch Pipelining und die Verwendung von SIMD-Operationen) zu generieren [4]. Andere Optimierungen werden bei der Berechnung von *lcp* genutzt, indem dieser in 64 bit-Blöcken parallel auf Bit-Ebene berechnet wird.

Die Ergebnisse aus [4] zeigen, dass die Technik effizienter als viele verbreitete Bibliotheken ist. So konnten kurze Sequenzen $20 - 300\times$ und lange, verrauschte Sequenzen $10 - 100\times$ schneller als durch konkurrierende Verfahren verarbeitet werden.

2.6 Seed-and-Extend Ansatz zur Lösung des lokalen Alignment Problems

Um den maximalen Score und ein optimales lokales Alignment zu berechnen, verwendet der Smith-Waterman Algorithmus [11] eine DP-Matrix analog zu den früher beschriebenen Algorithmen, wobei jedoch ein Score maximiert und das Backtracing ausgehend von einem maximalen Eintrag bis zu einem 0-Eintrag innerhalb der Matrix durchgeführt wird [1]. Auf diese Weise werden ebenfalls eine Laufzeit und ein Speicherplatzbedarf von $O(m \cdot n)$ erreicht [1].

Dieser Ansatz ist aber vor allem mit vielen Vergleichen langer Sequenzen zu ineffizient, weshalb viele Methoden eine Seed-and-Extend Strategie nutzen, um Kandidaten für optimale lokale Alignments zu bestimmen [1]. Bekannte Bespiele dafür sind Programme wie BLAST [12] oder FASTA [13].

Im ersten Schritt werden sogenannte Seeds bestimmt, wobei es sich um kurze, sehr ähnliche Sequenzbereiche handelt. Bei vielen Techniken werden dafür k-mere verwendet, also Substrings der Länge k. Diese werden gefiltert, in Datenstrukturen effizient gespeichert und gegebenenfalls komprimiert [14]. Bei BLAST werden bespielsweise die Positionen von k-meren (typischerweise für $k = 11$) in einer Hash-Tabelle gespeichert [15]. Bei anderen Methoden werden maximale exakte Treffer als Seeds verwendet. Diese Treffer können beispielsweise durch eine Baumstruktur repräsentiert werden [15]. Im nächsten Schritt werden die Seeds nach links und rechts durch dynamische Programmierung erweitert. Dies wird

durch die Sequenzinhalte gesteuert und nicht durch die Sequenzlängen wie bei globalen Alignments.

2.7 Technik zum Polieren der Enden von Alignments

In [16] wurde ein Seed-and-Extend Verfahren mit einer Seed-Erweiterung auf Basis des Wavefront-Algorithmus zur Berechnung von lokalen Alignments beschrieben. Dabei werden verschiedene Trimming-Strategien zur Reduzierung der Anzahl der Front-Werte in einer Wavefront entwickelt. Diese Strategien basieren auf einer Mindestanzahl an Übereinstimmungen am Ende eines Alignments beziehungsweise auf dem Abstand eines Front-Werts zum Front-Wert mit der maximal alignierten Länge der beiden Sequenzen. Der Alignment-Prozess wird dabei abgebrochen, falls

(a) das Ende beider Sequenzen erreicht ist oder

(b) keine Front-Werte diese Kriterien erfüllen.

Das Ziel sind Alignments mit einer Übereinstimmungsrate von mindestens $1 - 2\xi$, wobei ξ eine benutzerdefinierte Fehlerrate ist [16]. Da im Fall (b) die Enden der Alignments allerdings oft viele Fehler enthalten, ist es sinnvoll, diese soweit zu kürzen, dass sie nur aus wenigen Differenzen bestehen. Diese Verkürzung erfolgt auf der Basis von Polished Points und Suffix-positiven Alignments, wie im Folgenden definiert [16].

Definition 21 Ein *Polished Point* mit der Fehlerrate ξ ist ein Front-Wert, bei dem die Suffixe der letzten k Spalten eine Fehlerrate $\leq 1 - 2\xi$ haben.

Definition 22 Ein *Suffix-positives Alignment* mit der Fehlerrate ξ ist ein lokales Alignment, das mit einem Polished Point endet.

Dies führt dazu, dass Suffix-positive Alignments an den Enden wenige Fehler haben und so die Wahrscheinlichkeit sehr groß ist, dass sie Teil eines optimalen lokalen Alignments sind [16].

Um die Polished Points bei der iterativen Berechnung der Wavefronten zu bestimmen, wird ein Bitvektor b der Länge k verwendet, der bei einer Übereinstimmung eine 1 und sonst eine 0 für die letzten k Edit-Operationen speichert. Dabei kann diese Match History durch einen Bit-Shift und das Addieren der jeweiligen Zahl aus dem Bitvektor des Vorgängers gebildet werden. Um aus diesem Bitvektor zu bestimmen, ob es sich bei dem zugehörigen Front-Wert um einen Polished Point handelt, wird ein Array *SP* eingeführt [16].

Definition 23 Das Array SP wird für den Bitvektor b mit dem Score $\alpha = 2\xi$ für eine Übereinstimmung und $\beta = 1 - \alpha = 1 - 2\xi$ für eine Nicht-Übereinstimmung wie folgt definiert:

$$\begin{aligned} SP(b) &= \min\{SC(b') \mid b' \text{ ist ein Suffix von } b\} \\ \text{mit } SC(b) &= \alpha \cdot |b|_1 - \beta \cdot |b|_0 \end{aligned} \tag{2.20}$$

$|b|_1$ bezeichnet dabei die Anzahl an Einsen und $|b|_0$ die Anzahl an Nullen im Bitvektor b.

Für einen Bitvektor b' gilt $SC(b') \geq 0$ genau dann, wenn $\frac{|b'|_1}{|b'|_1 - |b'|_0} \geq 1 - 2\xi$ ist. Das heißt, dass das Alignment von b' eine Fehlerrate von höchstens $1 - 2\xi$ hat [16]. Daraus lässt sich Folgendes ableiten:

Lemma 6 Ein Front-Wert mit dem Bitvektor b ist genau dann ein Polished Point, wenn $SP(b) \geq 0$ gilt.

Falls das Array SP für alle 2^k Bitkombinationen berechnet wird, kann für einen Front-Wert mit der Match History b in $O(1)$ Zeit bestimmt werden, ob dieser ein Polished Point ist, indem auf das Element $SP(b)$ zugegriffen wird [16].

Mit steigendem k würde dieses Array aber sehr groß werden. Deshalb ist es in diesem Fall sinnvoll, die Match History in $\lambda = k/l$ einzelnen l-Bit Abschnitten zu betrachten [16]:

$$b = \underbrace{b[1 \ldots l]}_{b_\lambda} \cdot \underbrace{b[l+1 \ldots 2 \cdot l]}_{b_{\lambda-1}} \cdot \ldots \cdot \underbrace{b[(\lambda-1) \cdot l + 1 \ldots k]}_{b_1}$$

Definition 24 *Polish* wird für den Bitvektor $b = b_\lambda \cdot b_{\lambda-1} \cdot \ldots \cdot b_1$ durch folgende Rekurrenz berechnet [16]:

$$Polish(\lambda) = \begin{cases} Polish(\lambda - 1) \text{ und } Score(\lambda - 1) + SP(b_\lambda) \geq 0 & \text{falls } \lambda \geq 1 \\ true & \text{falls } \lambda = 0 \end{cases} \tag{2.21}$$

$$\text{mit } Score(\lambda) = \begin{cases} Score(\lambda - 1) + SC(b_\lambda) & \text{falls } \lambda \geq 1 \\ 0 & \text{falls } \lambda = 0 \end{cases}$$

Lemma 7 Der Front-Wert mit dem Bitvektor $b = b_\lambda \cdot b_{\lambda-1} \cdot ... \cdot b_1$ ist genau dann ein Polished Point, wenn $Polish(\lambda) = true$ gilt [16].

Um in konstanter Zeit zu bestimmen, ob ein Front-Wert poliert ist, werden die Arrays SP und SC vorher für alle 2^l Bitkombinationen von einen l-Bit Abschnitt der Match History berechnet. Dann lässt sich $Polish(\lambda)$ für einen Front-Wert in $O(\lambda)$ Zeit bestimmen [16].

Kapitel 3

3 Methoden

3.1 Grundidee

Der WFA++ Algorithmus kombiniert den WFA Algorithmus mit der Technik zum Polieren der Alignment-Enden als Abbruchkriterium im Kontext der Seed-Erweiterung bei der Berechnung lokaler Alignments.

Der initiale WFA Algorithmus berechnet ein paarweises globales Alignment und ist dabei durch seine Ausgabe-abhängige Leistung besonders für ähnliche Sequenzen effizient. Aus diesem Grund eignet er sich für die Seed-Erweiterung, die nur für solche Sequenzabschnitte stattfindet. Dafür wird diese Technik ausgehend vom Seed nach rechts und links angewendet und für ein frühzeitiges Ende des Alignment-Prozesses mit einem Abbruchkriterium kombiniert, sodass lokale Alignments erzielt werden.

Aus der Technik der Polished Points und Suffix-positiven Alignments lässt sich ein solches Abbruchkriterium konstruieren, indem die Anzahl der Polished Points in jeder Wavefront bestimmt und die Seed-Erweiterung abgebrochen wird, falls

(a) das Ende beider Sequenzen erreicht ist oder

(b) der Abstand zur letzten Wavefront mit Polished Points größer als eine Grenze $L(\gamma)$ wird.

Diese Schranke für die Anzahl aufeinanderfolgender Wavefronten ohne Polished Points $L(\gamma)$ ergibt sich dabei aus der Fehlerrate γ und nimmt Werte zwischen 1 und 5 an:

$$L(\gamma) = \begin{cases} 1 & \text{falls } \gamma \leq 2.0 \\ 2 & \text{falls } 2.0 < \gamma \leq 4.0 \\ 4 & \text{falls } 4.0 < \gamma \leq 8.0 \\ 5 & \text{falls } 8.0 < \gamma \end{cases} \tag{3.1}$$

Zur Bestimmung der Polished Points berechnen sich die Scores für eine Übereinstimmung und eine Nicht-Übereinstimmung zusätzlich noch mit einem Bias-Wert ζ:

$$\begin{aligned} \alpha &= 20.0 \cdot \gamma \cdot \zeta \\ \beta &= 1\,000.0 - \alpha \end{aligned} \tag{3.2}$$

Dabei muss $\alpha \leq 1\,000.0$ gelten.

Dies ist sinnvoll, da auf diese Weise neben dem Kürzen von Differenzen an den Enden des lokalen Alignments der Prozess abgebrochen wird, ohne die Komplexität der Methode zu erhöhen.

3.2 Implementierung

Der WFA++ Algorithmus wurde als eine effiziente, strukturierte und flexible C++ Implementierung auf Basis der GTTL-Software Bibliothek [17] umgesetzt.

3.2.1 Optionen

Dabei hat die Implementierung verschiedene Varianten, die durch den Optionsparser ausgewählt und spezifiziert werden können:

```
./wfaxx.x [options] <u-inputfile v-inputfile>

  --penalties arg specify penalties <MXOE> (default: 0111)
  --cigar-string  show cigar string
  --alignment     show alignment
  --polished arg  use polished points as termination criterion
                  <polishing_error_percentage,matchscore_bias>
  --wildcards     consider wildcards
  -h, --help      print usage
```

Als Eingabe dienen zwei FASTA-Dateien, deren Sequenzen jeweils paarweise miteinander verglichen werden.

Das Kostenmodell wird durch einen String „`MXOE`“ repräsentiert, wobei „`M`“ und „`X`“ die Kosten für eine Ersetzung (Übereinstimmung und Nicht-Übereinstimmung) und „`O`“ und „`E`“ die Gap-Kosten (Intialisierung und Erweiterung) beschreiben. Dieser String kann mit einer Option ausgewählt werden. Als Standardwert wird das affine Kostenmodell „`0111`“ verwendet.

Der Algorithmus bietet die Auswahl zwischen einer Distance-Only Variante und der Berechnung eines Alignments durch ein Backtracing. Dabei gibt es jeweils die Option, die Berechnung der Wavefronten vorzeitig mit dem in Abschnitt 3.1 beschriebenen Kriterium abzubrechen. Die Alignments können in der klassischen Darstellung oder als Cigar-String ausgegeben werden. Ein Cigar-String ist eine weit verbreitete und kompakte Beschreibung von Alignments in Form von Edit-Operationen, aber ohne Beschreibung der alignierten Sequenzen. Außerdem kann mit einer Option ausgewählt werden, wie Wildcards in den Sequenzen beim Test auf Zeichengleichheit berücksichtigt werden sollen.

3.2.2 Umsetzung der Optionen

Die verschiedenen Varianten der Software wurden durch Template-basierte Klassen und Funktionen in C++ implementiert.

Die Klasse `Wavefront` bildet die Grundlage der Implementierung (siehe Abbildung 3.1). Diese enthält die Werte einer Wavefront für die Diagonalen in dem durch *lo* und *hi* definierten Indexbereich. Die Klasse implementiert Methoden zur Manipulation und zum Zugriff auf die genannten Werte. Durch Verwendung der Template-Parameter `track_eops` und `track_match_history` können verschiedene Code-Abschnitte bereits zur Compilezeit ausgewählt werden. Die Front-Werte werden ebenfalls durch eine eigene Klasse `FrontValue` repräsentiert. Sie enthält vor allem die drei Komponenten $\widetilde{M}$, $\widetilde{I}$, $\widetilde{D}$, die zu einer Struktur `FrontValueBase` zusammengefasst werden. Hier wird der Mechanismus der partiellen Template-Spezifikation [18] genutzt, um abhängig von den Template-Parametern neben einem der genannten drei Werte selbst auch ein Objekt der Klasse `Backreference` zum Speichern von Rückwärtsverweisen (für das Backtracing) und/oder eine Match History (für das Abbruchkriterium) hinzufügen zu können.

Das WFA++ Programm (siehe Algorithmus 3) verallgemeinert den WFA Algorithmus abhängig von der Alignment- und Polishing-Option. So werden bei der Berechnung der Front-Werte einer Wavefront in Zeile 9 abhängig von diesen ebenfalls die Edit-Operationen und Match Histories aus den Vorgänger-Werten

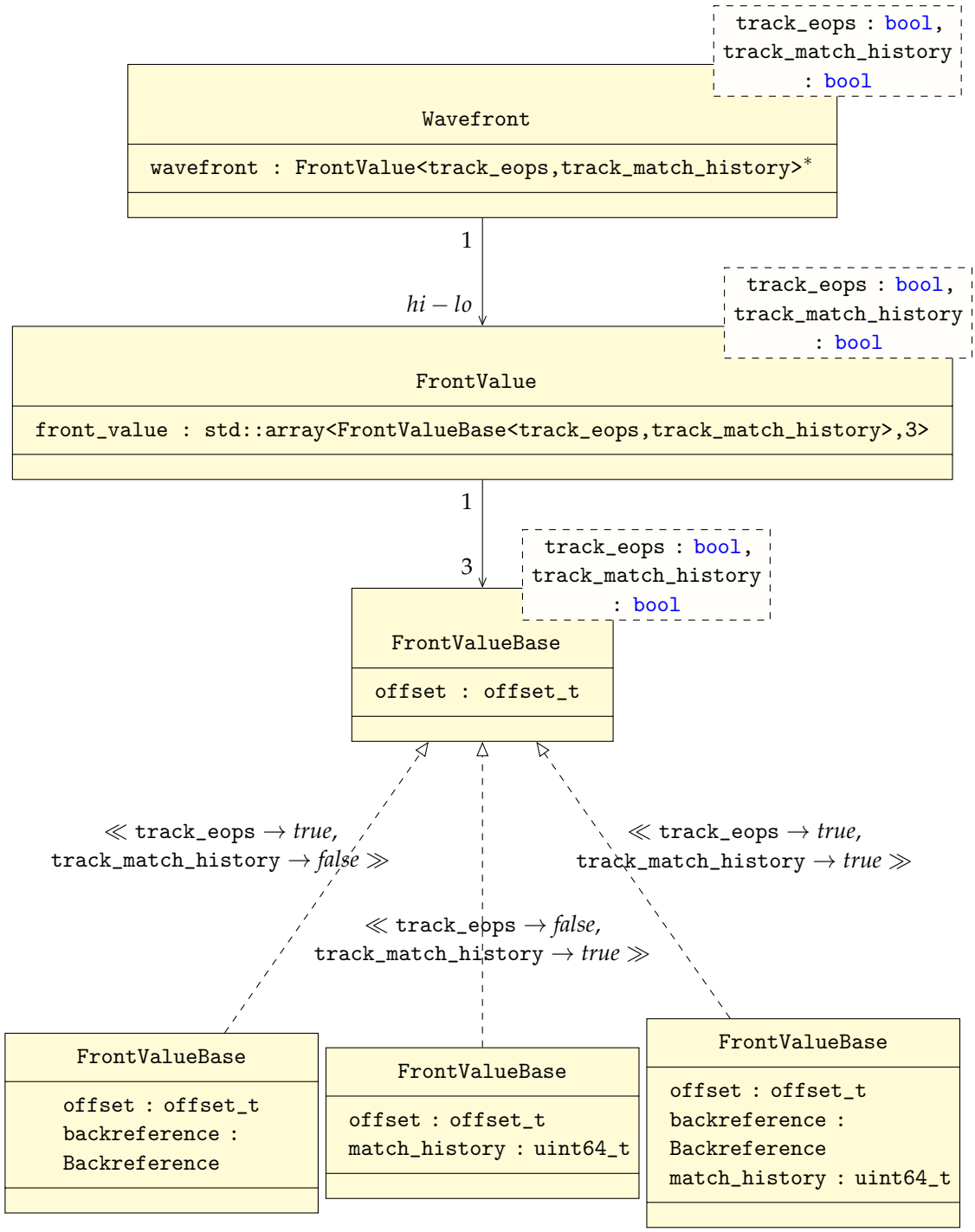

Abbildung 3.1: Klassendiagramm der WFA++ Implementierung.

bestimmt oder aktualisiert. Die Funktion zur Berechnung von maximalen gemeinsamen Substrings wird ebenfalls flexibel über einen Template-Parameter übergeben. Dadurch kann der paarweise Sequenzvergleich im Kontext der Seed-Erweiterung nach links mithilfe einer Methode zur Bestimmung des längsten

Algorithmus 3 WFA++ Algorithmus

Eingabe: Sequenzen u und v, Kostenmodell $\delta_{afn} = (x, o, e)$, Funktion `lcs_or_lcp`, Alignment-Option, Polishing-Option (mit Fehlerrate γ und Bias-Wert ζ)
Ausgabe: Distanz oder Alignment A von u und v unter δ_{afn} abhängig von Alignment- und Polishing-Option

```
1:  (m, n) ← (|u|, |v|)
2:  // d = 0
3:  M̃(0,0) ← |lcs_or_lcp (u[1...m], v[1...n])|
4:  if M̃(0,(n−m)).offset ≥ m then
5:      break
6:  for d ← 1 upto ∞ do
7:      // Berechnung der Wavefront
8:      for h ← lo(d) upto hi(d) do
9:          Berechne Ĩ(d,h), D̃(d,h), M̃(d,h) nach Gl. (2.17)-(2.19) mit lcs_or_lcp
10:     // Termination?
11:     if Polishing-Option then
12:         for h ← lo(d) upto hi(d) do
13:             if Polish(M̃(d,h).match_history) nach Gl. (2.21) mit Gl. (3.2) then
14:                 aligned_len ← 2 · M̃(d,h).offset + h
15:                 PolishedPoints += PolishedPoint(d, h, aligned_len)
16:         if d − PolishedPoints.last_d > L(γ) nach Gl. (3.1) then
17:             break
18:     if M̃(d,(n−m)).offset ≥ m then
19:         break
20: if Polishing-Option then
21:     best_polished_point ← Polished Point mit größter aligned_len
22:     if Alignment-Option then
23:         return A ← Backtracing von M̃(best_polished_point.d, best_polished_point.h)
24:                                  bis M̃(0,0)
25:     else
26:         return best_polished_point.d
27: else
28:     if Alignment-Option then
29:         return A ← Backtracing von M̃(d, n−m) bis M̃(0,0)
30:     else
31:         return d
```

gemeinsamen Suffixes der Sequenzen erfolgen. Im Kontext der Seed-Erweiterung nach rechts wird stattdessen eine Methode zur Bestimmung des längsten gemeinsamen Präfixes benötigt. Beim Überprüfen des Terminationskriteriums werden bei der Polishing-Option zusätzlich die Polished Points der aktuellen Wavefront

berechnet und auf das in Abschnitt 3.1 beschriebene Abbruchkriterium getestet (siehe Zeile 16). Abhängig von der Alignment-Option kann nach Termination des Algorithmus ein Backtracing mithilfe der gespeicherten Edit-Operationen zum Bestimmen des lokalen oder globalen Alignments durchgeführt werden (siehe Zeilen 29 und 23) oder die jeweilige Distanz zurückgegeben werden (siehe Zeilen 31 und 26).

Um diesen Algorithmus für die Seed-Erweiterung zur Berechnung lokaler Alignments zu verwenden, müssen die jeweiligen Sequenzteile ab dem berechneten Seed und eine passende Funktion für `lcs_or_lcp` übergeben werden.

3.3 Qualitätsanforderungen

Für die Implementierung standen vor allem die Qualitätsmerkmale Effizienz und Wartbarkeit mit Schwerpunkt auf Verständlichkeit und Veränderbarkeit im Fokus. Da es bei diesen Eigenschaften gewisse Zielkonflikte gibt, mussten bei der Softwareentwicklung Kompromisse gefunden werden.

Der Code wurde in kleinere, in sich abgeschlossene und voneinander möglichst unabhängige Funktionen zerlegt. Durch die Verwendung von Templates konnte eine hohe Flexibilität erreicht und trotz der vielen unterschiedlichen Funktionalitäten redundanter Code vermieden werden, ohne Effizienz bei der Ausführung zu verlieren. Auf diese Weise ist der Code oder Teile davon leicht austauschbar, anpassbar und auch in anderen Kontexten verwendbar. Beispielsweise wurde das Abbruchkriterium über Template-Parameter umgesetzt, weswegen an dieser Stelle auch andere Kriterien verwendet werden könnten. Die Benennungen und Gliederungen des Programmcodes sind so gewählt, dass dieser möglichst selbsterklärend und verständlich ist. Dabei war die Objektorientierung von C++ eine wichtige Strategie, wodurch die Klasse `Wavefront` eine zentrale Komponente des WFA++ Algorithmus ist. Beispielsweise wurde vorzugsweise auf globale Definitionen von Konstanten verzichtet und diese stattdessen in Klassen durch öffentliche `static constexpr const`-Bezeichner implementiert. Diese Ansätze sorgen für eine hohe Verständlichkeit und Änderbarkeit.

Die Template-basierten Klassen und Funktionen haben dabei den Vorteil, dass die verschiedenen Optionen zur Kompilierungszeit bekannt sind, sodass der Compiler viel Optimierungspotential hat und ein hocheffizientes Programm entsteht.

Da sich beim Ablauf des Algorithmus die Anzahl der Wavefronten stetig erhöht und die Gesamtanzahl nicht vorhersehbar ist, wurde zum Speichern von diesen `std::vector<Wavefront>` als Datentyp gewählt. Die Laufzeit bei der Verwendung

dieser Klasse wurde entsprechend der Hinweise aus [19] optimiert. Zum Einen wurde beim Zugriff auf Objekte der Indexzugriffsoperator `[]`-Operation verwendet. Dieser führt im Gegensatz zur Funktion `std::vector::at()` keine Überprüfungen der Gültigkeit von Indizes durch und ist daher effizienter als die genannte Funktion [20]. Außerdem wurde beim Hinzufügen eines neuen Elements die Funktion `std::vector::emplace_back()` statt `std::vector::push_back()` genutzt. Das führt dazu, dass ein Objekt In Place erzeugt und nicht zunächst erstellt und dann in den Vektor kopiert wird. Beim Hinzufügen von bereits existierenden Objekten wird das Kopieren von diesen durch die Verwendung von Move-Kontruktoren verhindert. Falls nämlich ein Kopierkonstruktor verwendet werden würde, würden beim Hinzufügen eines `Wavefront`-Objektes zum Vektor zunächst ein neuer Array allokiert, die Elemente dann einzeln kopiert und der Speicher des alten Arrays freigegeben werden. Bei Verwendung des Move-Konstruktors wird nur ein Zeiger auf den gleichen Speicherbereich erzeugt und der alte Zeiger erhält den Wert `nullptr`. Dieser Weg ist Laufzeit-effizienter. Der Nachteil der Verwendung der Klasse `Wavefront` in Kombination mit einem Vektor ist, dass, anders als in der WFA Implementierung, Zeiger von nicht benötigten Wavefronten nicht auf `NULL` gesetzt werden können. Der Vektor benötigt nämlich vollständige Objekte von einem festen Typen. In einer experimentellen Variante der WFA++ Implementierung wurde ein Vektor von Zeigern auf `Wavefront` Objekte verwendet. Dies hat aber nicht zur gewünschten Laufzeitverbesserung geführt, was wahrscheinlich an der schlechteren Lokalität der sonst im Speicher direkt hintereinander liegenden Objekte lag [21]. Eine andere Strategie bestand darin, die Klassen-Templates `std::optional<>` oder `std::variant<>` der C++ Standardbibliothek zu nutzen. Dieser Ansatz führt jedoch zu unübersichtlichem Programmcode und verbesserte ebenso nicht die Laufzeit. Deshalb wurde letztendlich in der Klasse `Wavefront` ein Konstruktor für eine Art `NULL`-Wavefront hinzugefügt, wessen Objekte nur aus ihren Instanzvariablen bestehen. Diese Objekte können ebenfalls zum Vektor hinzugefügt und durch eine Funktion `Wavefront::is_defined()` erkannt werden. Analog zur Implementierung von WFA wurde ein `Wavefront`-Objekt `null_wavefront` verwendet, das für den maximal möglichen Indexbereich von $-(\max\{m,n\}+1)$ bis $\max\{m,n\}+1$ jeweils undefinierte Werte enthält. Durch den Zugriff auf die `null_wavefront` können Verzweigungen im Programm durch den Aufruf der Methode `Wavefront::is_defined()` vermieden werden. Außerdem wurde bei der Berechnung der Front-Werte ein Loop Unrolling durchgeführt, wobei das Vorhandensein beim Zugriff auf die vorigen Front-Werte nur an den Grenzen der Diagonalindizes getestet wird. Um die direkte Verwendung von negativen Indexwerten für die Diagonalen zu erlauben, wird der Pointer auf die Front-Werte für d in der Klasse `wavefront` um $|lo(d)|$ Elemente nach vorne verschoben.

Der Sequenzvergleich ohne Wildcards wird analog zur WFA Implementierung mithilfe von Bitoperationen in Blöcken von 8 Zeichen durchgeführt. Dafür

benötigen die zu vergleichenden Sequenzen am Ende 7 Padding Zeichen, um fehlerhafte Speicherzugriffe zu vermeiden. Diese werden in der C++ Implementierung hinzugefügt, indem der Padding String an das Ende der Sequenz angehängt wird. Anschließend wird, wie im folgenden Codestück gezeigt, für diesen Speicherbereich ein `std::string_view`-Objekt erzeugt. Dessen Ende wird mit Hilfe der `std::string_view::remove_suffix()`-Methode auf das ursprüngliche Ende der Sequenz (vor dem Padding) verschoben. Auf diese Weise liegen die Padding-Zeichen direkt hinter der Sequenz im Speicher, aber WFA++ kann auf die Sequenzen ohne weitere Anpassungen im Code zugreifen.

```
const std::string padding_string(7,static_cast<char>(UINT8_MAX));
std::string_view seq_padded = seq.append(padding_string);
seq_padded.remove_suffix(7);
```

Um die Berechnungen möglichst Speicher-effizient zu halten, wird beispielsweise die vor allem bei langen Sequenzen speicherintensive `null_wavefront` nur einmal erzeugt und bei Bedarf eine Referenz verwendet.

3.4 Generierung von Datensätzen

Die Grundlage für die Tests und das Benchmarking ist ein C++ Generator für Sequenzpaare, der den Mutationsprozess einer Sequenz als Markov-Prozess simuliert. Auf diese Weise kann der Datensatz spezifisch für die Problemstellung durch die Bevorzugung von Gaps und optionaler Generierung lokaler Ähnlichkeiten erstellt und effizient In Place verwendet werden, da keine Ein- und Ausgabe von Dateien benötigt wird.

Definition 25 Eine *Zeit-homogene Markovkette* ist ein Paar (S, Π) mit

- einer endlichen Menge an Zuständen $S = \{s_1, s_2, ..., s_h\}$ und
- einer Übergangsmatrix $\Pi = (\pi_{s,t})_{s,t \in S}$ mit $P(x_i = t | x_{i-1} = s) = \pi_{s,t}$ für alle i [7, 22].

Eine Verteilung μ auf S für eine solche Markovkette heißt *stationäre Verteilung*, falls Folgendes gilt [22]:

$$\mu = \mu\Pi$$

Zur Simulation eines Mutationsprozesses wird die Zustandsmenge durch die Edit-Operationen mit Unterscheidung von Übereinstimmung und Nicht-Übereinstimmung als $S = \{M \widehat{=} 1, X \widehat{=} 2, D \widehat{=} 3, I \widehat{=} 4\}$ definiert und M als Startzustand gewählt. Die Übergangsmatrix Π wird aus Benutzer-definierten Wahrscheinlichkeiten

gebildet. Abbildung 3.2 verdeutlicht das Prinzip als ein Zustandsdiagramm mit einem Beispiel.

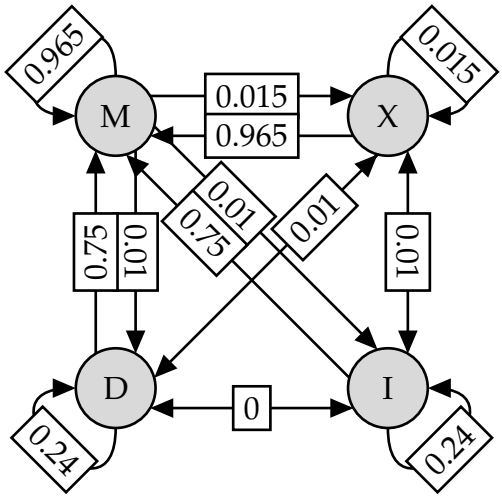

Abbildung 3.2: Darstellung des Modells als Graph mit Knoten für die Zustände und mit Wahrscheinlichkeiten als Markierungen der Kanten (Prozessdiagramm).

Wenn die Markovkette nur auf die durch M erreichbaren Zustände eingeschränkt wird und der Zustand M aus diesen wiederum erreichbar ist, konvergiert die Verteilung der Zustände gegen eine eindeutige stationäre Verteilung [22]. Die Komponente M dieser stationären Verteilung lässt sich dann als durchschnittliche Wahrscheinlichkeit interpretieren, sich in einem Schritt des Mutationsprozesses bei einer Übereinstimmung zu befinden [22]. Das heißt, dass man aus der Gegenwahrscheinlichkeit der M-Komponente eine Fehlerrate ϵ für den Mutationsprozess ableiten kann:

$$\epsilon = 1 - \mu[1]$$

Mit ϵ lässt sich ein Erwartungswert für die Edit-Distanz unter Einheitskosten für zwei Sequenzen u und v abschätzen:

$$\epsilon \cdot \frac{m+n}{2}$$

Die Implementierung benötigt als Eingabe eine tsv-Datei mit den Wahrscheinlichkeiten des Markovmodells. Dabei können Gaps durch hohe Übergangswahrscheinlichkeiten für die Gap-Erweiterungen (D zu D beziehungsweise I zu I) ermöglich werden. Außerdem ist es sinnvoll, aufeinanderfolgende Einfügungen und Löschungen zu verhindern, indem die Übergange zwischen D und I auf

0 gesetzt werden. Im ersten Schritt wird eine zufällige DNA-Sequenz u mit einer durch den Benutzer-definierten festen Länge m generiert. Dann werden nacheinander mittels Pseudo-Zufallszahlen und entsprechend der Übergangswahrscheinlichkeiten des Markovmodells Edit-Operationen generiert und diese auf die nächste Position in u anwendet. Das Ergebnis dieser Anwendung ist eine neue Sequenz v mit der erwarteteten Fehlerrate. Anschließend kann optional an die Sequenzen jeweils eine Benutzer-definierte Länge von zufälligen Basenpaaren angefügt werden. Auf diese Weise können lokale Ähnlichkeiten in den Sequenzen erzeugt werden.

Für die Tests wurden insbesondere die folgenden Übergangsmatrizes verwendet, um annähernd Fehlerraten von 2 %, 4 %, 6 % und 10 % zu erzielen:

$$\Pi_{2\,\%} = \begin{pmatrix} 0.982 & 0.012 & 0.003 & 0.003 \\ 0.982 & 0.012 & 0.003 & 0.003 \\ 0.78 & 0.005 & 0.215 & 0 \\ 0.78 & 0.005 & 0 & 0.215 \end{pmatrix} \tag{3.3}$$

$$\mu_{2\,\%} \approx \begin{pmatrix} 0.9805 & 0.0119 & 0.0038 & 0.0038 \end{pmatrix}$$

$$\epsilon_{2\,\%} \approx 0.0195$$

$$\Pi_{4\,\%} = \begin{pmatrix} 0.965 & 0.015 & 0.01 & 0.01 \\ 0.965 & 0.015 & 0.01 & 0.01 \\ 0.75 & 0.01 & 0.24 & 0 \\ 0.75 & 0.01 & 0 & 0.24 \end{pmatrix} \tag{3.4}$$

$$\mu_{4\,\%} \approx \begin{pmatrix} 0.9595 & 0.0149 & 0.0128 & 0.0128 \end{pmatrix}$$

$$\epsilon_{4\,\%} \approx 0.0405$$

$$\Pi_{6\,\%} = \begin{pmatrix} 0.948 & 0.02 & 0.016 & 0.016 \\ 0.948 & 0.02 & 0.016 & 0.016 \\ 0.75 & 0.01 & 0.24 & 0 \\ 0.75 & 0.01 & 0 & 0.24 \end{pmatrix} \tag{3.5}$$

$$\mu_{6\,\%} \approx \begin{pmatrix} 0.9400 & 0.0196 & 0.0202 & 0.0202 \end{pmatrix}$$

$$\epsilon_{6\,\%} \approx 0.0600$$

$$\Pi_{10\,\%} = \begin{pmatrix} 0.907 & 0.057 & 0.018 & 0.018 \\ 0.907 & 0.057 & 0.018 & 0.018 \\ 0.75 & 0.01 & 0.24 & 0 \\ 0.75 & 0.01 & 0 & 0.24 \end{pmatrix} \tag{3.6}$$

$$\mu_{10\,\%} \approx \begin{pmatrix} 0.8999 & 0.0549 & 0.0226 & 0.0226 \end{pmatrix}$$

$$\epsilon_{10\,\%} \approx 0.1001$$

Die Verteilung der Fehler, also der Edit-Operationen, die keine Übereinstimmung sind, lassen sich für die verschiedenen Übergangsmatrizes in Abbildung 3.3 erkennen und bestätigen die vorherigen Berechnungen. Die Histogramme der Fehleranzahl lassen sich durch Normalverteilungen darstellen, dessen Parameter in Tabelle 3.1 angegeben sind.

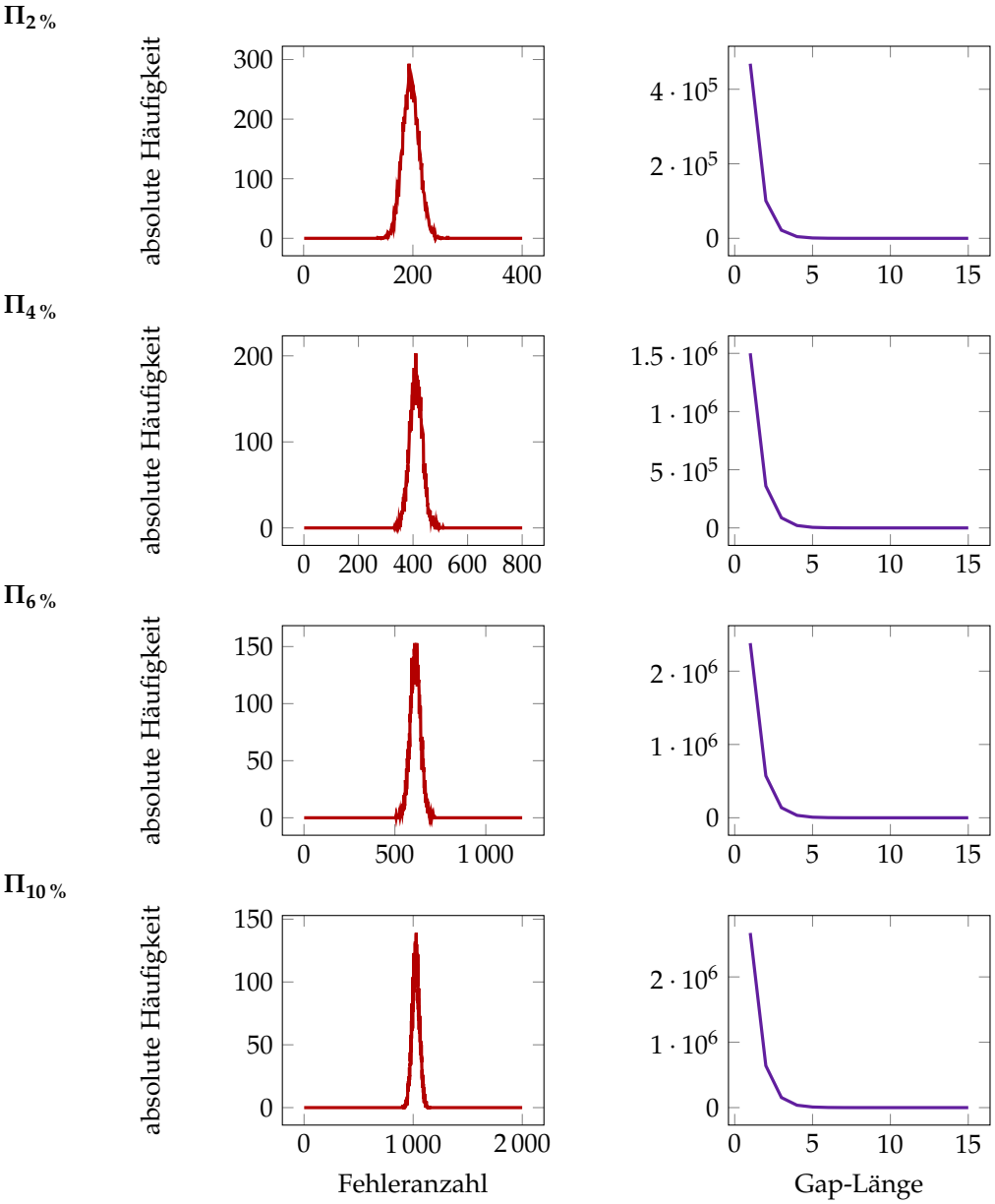

Abbildung 3.3: Histogramm der Fehleranzahl und Gap-Längen in 10 000 generierten Sequenzpaaren mit $m = 10\,000$ und $\Pi_{2\,\%}$, $\Pi_{4\,\%}$, $\Pi_{6\,\%}$ oder $\Pi_{10\,\%}$ aus den Gleichungen (3.3) bis (3.6) als Übergangsmatrix.

Abbildung 3.3 zeigt außerdem, dass praktisch keine Gap-Länge > 10 erreicht wird. Um Übergangsmatrizes zu erstellen, die bei diesen Fehlerraten längere Gaps

Übergangsmatrix	Erwartungswert	Standardabweichung
$\Pi_{2\,\%}$	196.1773	15.3676
$\Pi_{4\,\%}$	410.6185	23.3697
$\Pi_{6\,\%}$	612.8218	28.4688
$\Pi_{10\,\%}$	1024.0088	33.7420

Tabelle 3.1: Parameter für die Normalverteilungen der Fehleranzahl beim durch eine Markovkette simulierten Mutationsprozess einer Sequenz mit der Übergangsmatrix $\Pi_{2\,\%}$, $\Pi_{4\,\%}$, $\Pi_{6\,\%}$ oder $\Pi_{10\,\%}$ aus den Gleichungen (3.3) bis (3.6).

generieren, kann man das Problem als ein lineares Gleichungssystem ausgehend von der stationären Verteilung formulieren [22]. Hier wird das Prinzip an einem Beispiel verdeutlicht, welches zur Vereinfachung mit $S' = \{M\widehat{=}1, D\widehat{=}2, I\widehat{=}3\}$ keine Nicht-Übereinstimmungen erlaubt und Einfügungen und Löschungen gleich behandelt, um eine Übergangsmatrix mit $\epsilon = 0.4$ zu bestimmen:

$$\Pi' = \begin{pmatrix} x_1 & (1-x_1)/2 & (1-x_1)/2 \\ x_2 & 1-x_2 & 0 \\ x_2 & 0 & 1-x_2 \end{pmatrix}$$
$$\mu'_{4\,\%} = \begin{pmatrix} 0.96 & 0.02 & 0.02 \end{pmatrix}$$

Es muss gelten:

$$\mu'_{4\,\%} = \mu'_{4\,\%}\Pi'$$
$$\Leftrightarrow x_1 = 1 - \frac{1}{24} \cdot x_2$$

Für $x_2 \in [0,1]$ ist $0 \leq x_1 = 1 - \frac{1}{24} \cdot x_2 \leq 1$. Zwei mögliche Lösungen mit einer höheren Wahrscheinlichkeit für weniger, aber längere Gaps sind $x_2^{(1)} = 0.2$ oder $x_2^{(2)} = 0.1$:

$$\Pi_{4\,\%}^{(1)} \approx \begin{pmatrix} 0.992 & 0.004 & 0.004 \\ 0.2 & 0.8 & 0 \\ 0.2 & 0 & 0.8 \end{pmatrix} \tag{3.7}$$

$$\Pi_{4\,\%}^{(2)} \approx \begin{pmatrix} 0.996 & 0.002 & 0.002 \\ 0.1 & 0.9 & 0 \\ 0.1 & 0 & 0.9 \end{pmatrix} \tag{3.8}$$

Die Ergebnisse in Abbildung 3.4 zeigen, dass mit diesen Werten längere Gaps erzeugt werden. Auf der anderen Seite lässt sich erkennen, dass die Standardabweichung der Normalverteilung der Fehleranzahl in der mutierten Sequenz deutlich größer ist (siehe Tabelle 3.2).

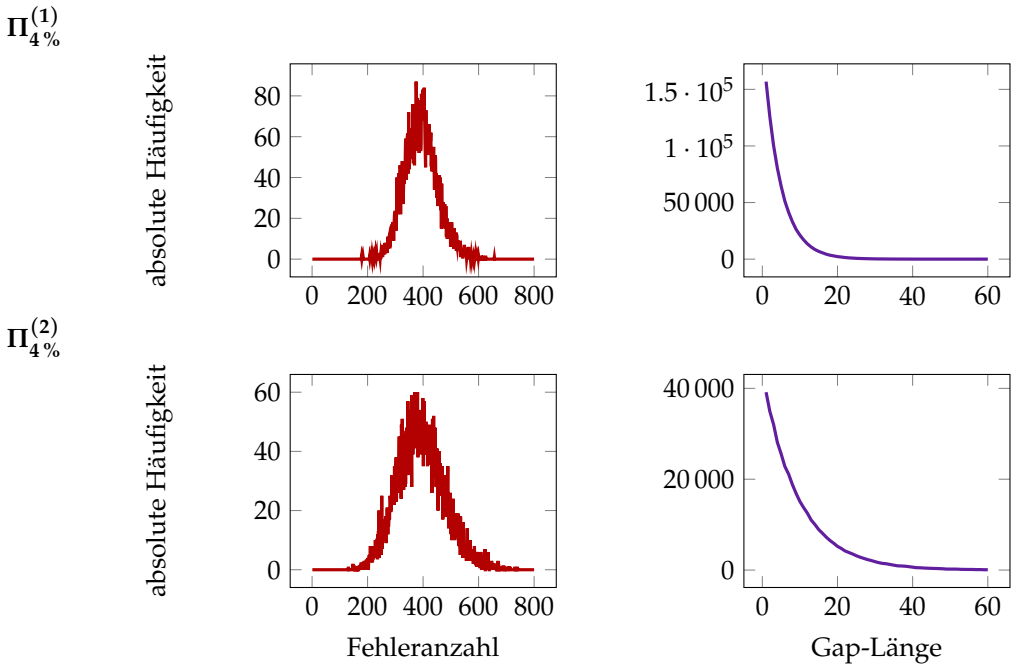

Abbildung 3.4: Histogramm der Fehleranzahl und Gap-Längen in 10 000 generierten Sequenzpaaren mit $m = 10\,000$ und $\Pi_{4\,\%}^{(1)}$ oder $\Pi_{4\,\%}^{(2)}$ aus den Gleichungen (3.7) und (3.8) als Übergangsmatrix.

Übergangsmatrix	Erwartungswert	Standardabweichung
$\Pi_{4\,\%}^{(1)}$	392.2825	59.3038
$\Pi_{4\,\%}^{(2)}$	392.1884	84.8767

Tabelle 3.2: Parameter für die Normalverteilungen der Fehleranzahl beim durch eine Markovkette simulierten Mutationsprozess einer Sequenz mit der Übergangsmatrix $\Pi_{4\,\%}^{(1)}$ oder $\Pi_{4\,\%}^{(2)}$ aus den Gleichungen (3.7) und (3.8).

3.5 Validierung

Zur Validierung der Implementierung von WFA++ diente eine Testsuite von automatisierten Tests. Diese wurde als Regressionstest verwendet. Dadurch wurde sichergestellt, dass die grundsätzliche Funktionalität bei Codeänderungen erhalten bleibt.

Tabelle 3.3 zeigt das Design der hierzu ausgeführten Tests. Die Ergebnisse wurden im ersten Schritt mittels generierter Datensätze getestet. Dabei wurde die errechnete Distanz mit der tatsächlichen des Mutationsprozesses verglichen. Außerdem wurde das Backtracing überprüft, indem der Cigar-String und das Alignment auf Korrektheit in Bezug auf die Edit-Operationen, der Darstellung der Sequen-

zen und der Distanz getestet wurden. Zusätzlich wurde die Match History mit diesen abgeglichen und so verifiziert. Diese Tests wurden mit verschiendenen Kostenmodellen durchgeführt. Damit wurde sichergestellt, dass die Ergebnisse in sich konsistent sind. Um zu testen, ob der Algorithmus tatsächlich korrekte Distanzen und davon abgeleitet auch optimale Alignments berechnet, wurden die Distanzen mit Ergebnissen anderer Programme verglichen. Dafür wurden das linspace-Programm [23] aus der GenomeTools-Software [24] und die Originalimplementierung des WFA Algorithmus [10] verwendet und die Sequenzen mit unterschiedlichen Kostenfunktionen bewertet. Hierbei wurden Testdaten aus der GTTL-Software Bibliothek [17] genutzt.

Strategie	getestete Komponenten	Datensatz	Kosten		
			x	o	e
Vergleich mit Mutationsprozess, Prüfen auf Konsistenz	Distanz, Cigar String, Alignment, Match History	250 bis 750 generierte Sequenzpaare (500bp Länge, 10 % Fehler)	1	0	1
			3	0	2
			1	0	4
			1	1	1
			4	6	2
			6	5	3
Vergleich mit GenomeTools und WFA	Distanz	Datensätze aus GTTL	1	0	1
			1	0	5
			5	0	1
			1	1	1
			1	3	1
			3	7	1
			1	1	4
			4	6	2
			6	5	3

Tabelle 3.3: Teststrategie zur Validierung der Ergebnisse von WFA++.

Kapitel 4

4 Ergebnisse

Um die WFA++ Implementierung zu evaluieren, wurde diese gemeinsam mit der C Implementierung von WFA in einem C++ Programm vereint und mit dem GCC beziehungsweise G++ Compiler compiliert. Die Laufzeit-Messungen wurden dann auf den folgenden beiden Systemen durchgeführt:

(a) Intel Core i7-4790 Prozessor mit Ubuntu 22.04.1 und

(b) ARM64 M1 mit Darwin.

Als Datensätze wurden 1 000 Sequenzpaare mit 20 bis 1 500 bp Länge durch das in Abschnitt 3.4 beschriebene Programm generiert, wobei durch die Übergangsmatrizes aus den Gleichungen (3.3) bis (3.5) Fehlerraten von 2 %, 4 % und 6 % erzielt wurden. Zur Bewertung der Sequenzen wurden die affinen Kostenmodelle $\delta_{afn} = (1, 1, 1)$, $\delta_{afn} = (4, 6, 2)$ und $\delta_{afn} = (6, 5, 3)$ verwendet, um den Einfluss des gewählten Modells auf die Laufzeit zu testen. Die letzten beiden Kostenfunktionen wurden außerdem auch für die Evaluierung der WFA Implementierung in [4] verwendet.

Da im initialen WFA Programm immer ein Alignment berechnet wird, wurden im ersten Schritt die Laufzeiten inklusive Berechnung eines Alignments gemessen. Die Ergebnisse für System (a) sind in Abbildung 4.1 und für System (b) in Abbildung 4.2 dargestellt. Hier ist zu erkennen, dass die WFA++ Implementierung besonders für kurze Sequenzen mit kleiner Fehlerrate schnell Ergebnisse liefert. Auf der anderen Seite ist das WFA Programm bei steigender Sequenzlänge und Fehlerrate effizienter. Dabei fällt auf, dass WFA++ im Vergleich zu WFA für System (b) besser performt.

Im Folgenden wurden die Laufzeiten für die Distance-Only Variante auf beiden Systemen bestimmt. Abbildung 4.3 und Abbildung 4.4 zeigen die Laufzeiten. In

dieser Variante von WFA++ konnte eine deutliche Verbesserung der Laufzeiten erzielt werden. Auf System (a) (siehe Abbildung 4.3) wurden im Vergleich zur WFA Implementierung in 120 von 140 Fällen kleinere oder gleiche Laufzeiten erreicht. Die Summe der Laufzeiten für alle Sequenzpaare betrug für WFA++ 4 357 ms und für WFA 3 812 ms. Auf System (b) (siehe Abbildung 4.4) wurde für die genannten Datensätze sogar in 143 von 144 Fällen geringere Laufzeiten erreicht. Dabei war WFA++ um bis zu einem Faktor 14 schneller als WFA. Für WFA++ ergab für alle Sequenzpaare eine Gesamtlaufzeit von 2 497 ms und für WFA von 4 001 ms.

Um das in Abschnitt 3.1 beschriebene Abbruchkriterium zu evaluieren, wurde die Distance-Only Variante von WFA++ mit und ohne dieses Kriterium ausgeführt. Die Ergebnisse für die beiden Systeme sind in Abbildung 4.5 und Abbildung 4.6 dargestellt. Hier lässt sich für das Kostenmodell $\delta_{afn} = (1, 1, 1)$ erkennen, dass für alle Fehlerraten kaum frühzeitige Abbrüche stattfinden. Aus den Messungen für dieses Kostenmodells kann man daher die Laufzeitzunahme für die zusätzlichen Berechnungen und Vergleiche bei der Verwendung dieses Abbruchkriteriums abschätzen. Für die anderen beiden Kostenmodelle lässt sich für steigende Sequenzlängen und Fehlerraten ein frühzeitiger Abbruch der Berechnungen durch einen steigenden Wert für den Quotienten $\frac{\text{Laufzeit WFA++}}{\text{Laufzeit WFA++ (PP)}}$ erkennen. Dabei brechen die Berechnungen für die Datensätze beim Kostenmodell $\delta_{afn} = (6, 5, 3)$ am schnellsten ab.

Für die letzten Messungen wurden 10 000 Sequenzpaare mit 500 bis 3 000 bp Länge und 4 % und 10 % Fehlerraten generiert, wobei jeweils die letzten 100 Basenpaare zufällig erzeugt wurden. Außerdem wurden 10 000 Sequenzpaare mit 100 bis 600 zufälligen Basenpaaren verwendet. Dann wurden die Laufzeiten für WFA mit der Berechnung optimaler globaler Alignments und für ein Seed-and-Extend Programm, welches WFA++ mit dem Abbruchkriterium aus Abschnitt 3.1 zur Seed-Erweiterung verwendet und Kandidaten für optimale lokale Alignments berechnet, gemessen. Zur Bewertung wurden die gleichen affinen Gap-Kostenmodelle wie bei den vorigen Tests gewählt. Die Ergebnisse für System (a) werden in Abbildung 4.7 präsentiert. Für das Kostenmodell $\delta_{afn} = (1, 1, 1)$ werden die Berechnungen bei einer 4 % Fehlerrate spät abgebrochen, wohingegen dieser bei 10 % Fehlerrate deutlich früher erfolgt, was sich besonders bei großen Sequenzlängen auf die Laufzeit auswirkt. Bei den zufälligen Sequenzen lässt sich durch die kleinen Laufzeiten ein direkter Abbruch erkennen. Bei den anderen beiden Kostenmodellen findet der Abbruch analog zu den Ergebnissen des vorigen Tests früher statt.

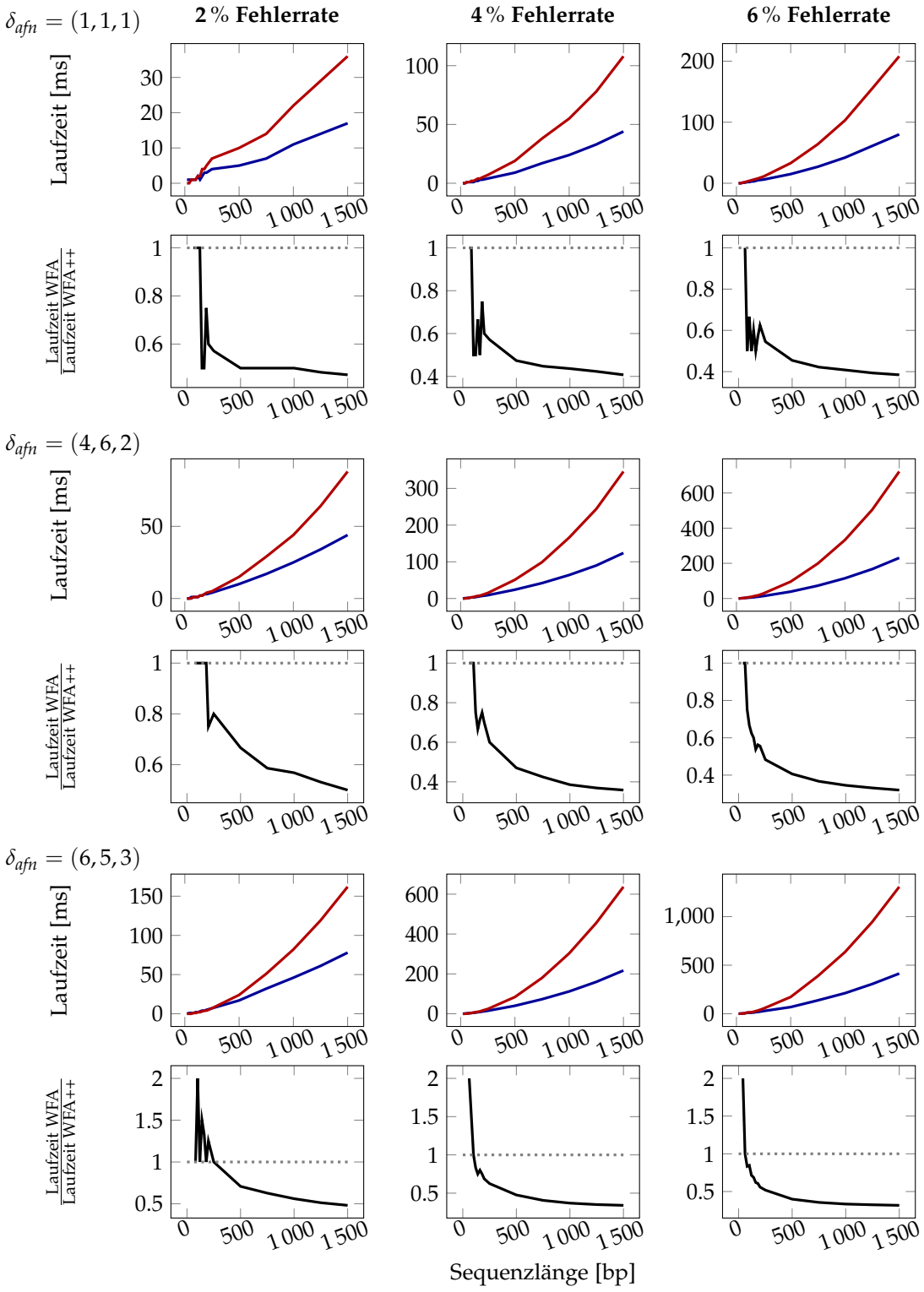

Abbildung 4.1: Laufzeiten auf System (a) für die Berechnung von globalen Alignments durch WFA — und WFA++ —. Es wurden drei Kostenmodelle und generierte Sequenzpaare mit $m \in \{20, 40, 60, 80, 100, 120, 140, 160, 180, 200, 250, 500, 750, 1\,000, 1\,250, 1\,500\}$ und drei Fehlerraten verwendet.

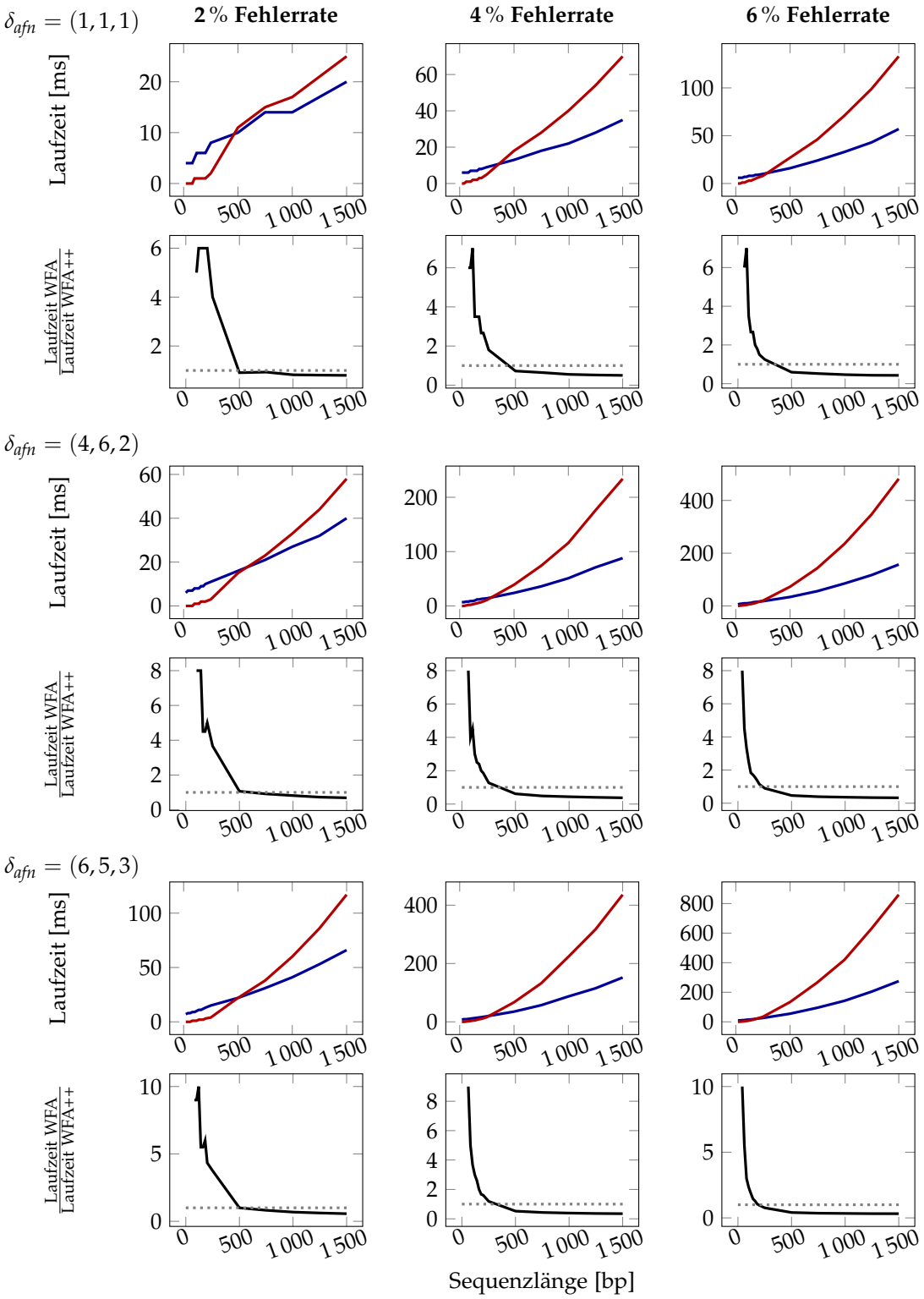

Abbildung 4.2: Laufzeiten auf System (b) für die Berechnung von globalen Alignments durch WFA — und WFA++ —. Es wurden drei Kostenmodelle und generierte Sequenzpaare mit $m \in \{20, 40, 60, 80, 100, 120, 140, 160, 180, 200, 250, 500, 750, 1\,000, 1\,250, 1\,500\}$ und drei Fehlerraten verwendet.

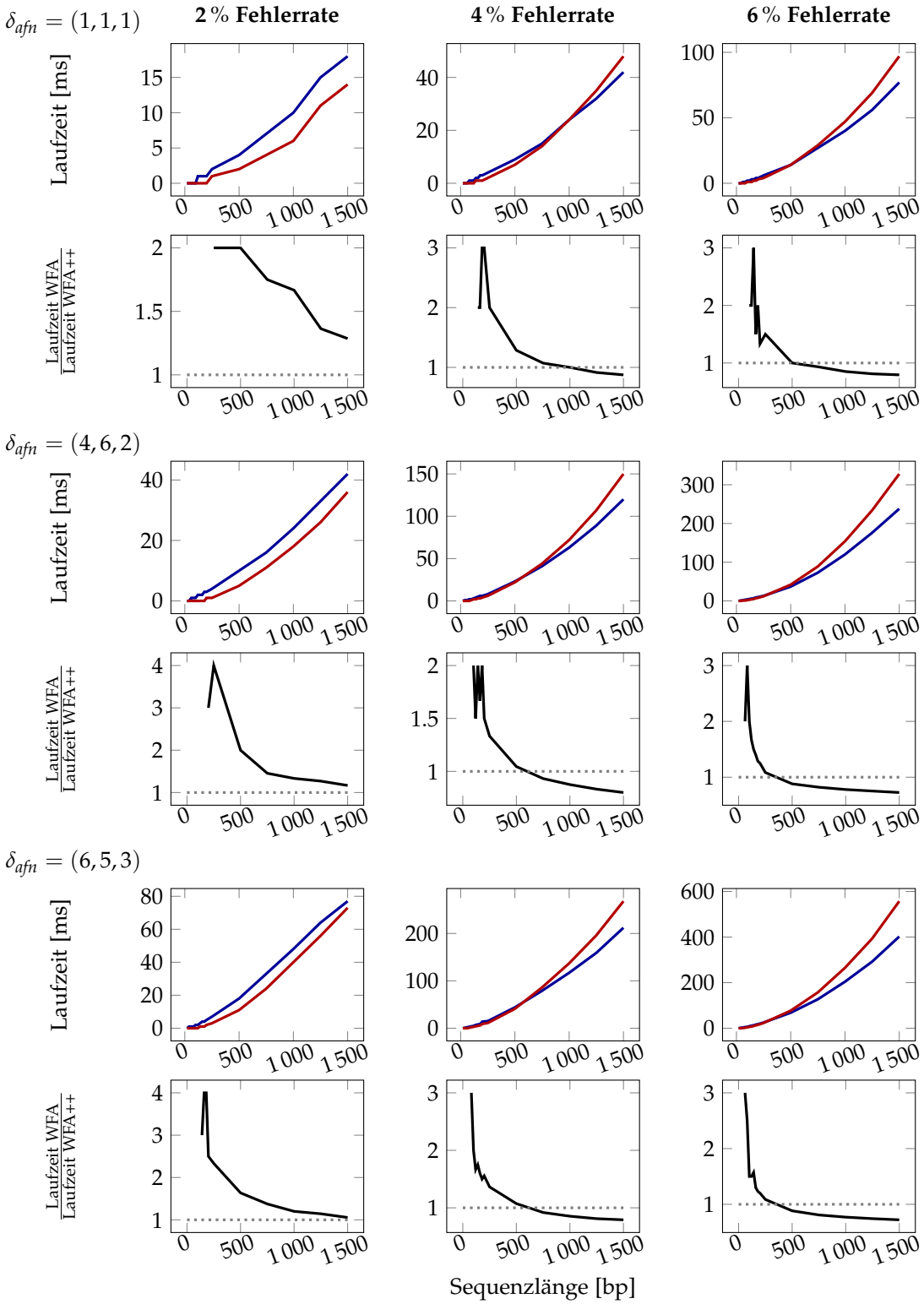

Abbildung 4.3: Laufzeiten auf System (a) für die Berechnung von globalen Alignments durch WFA — und der Edit-Distanz durch WFA++ —. Es wurden drei Kostenmodelle und generierte Sequenzpaare mit $m \in \{20, 40, 60, 80, 100, 120, 140, 160, 180, 200, 250, 500, 750, 1\,000, 1\,250, 1\,500\}$ und drei Fehlerraten verwendet.

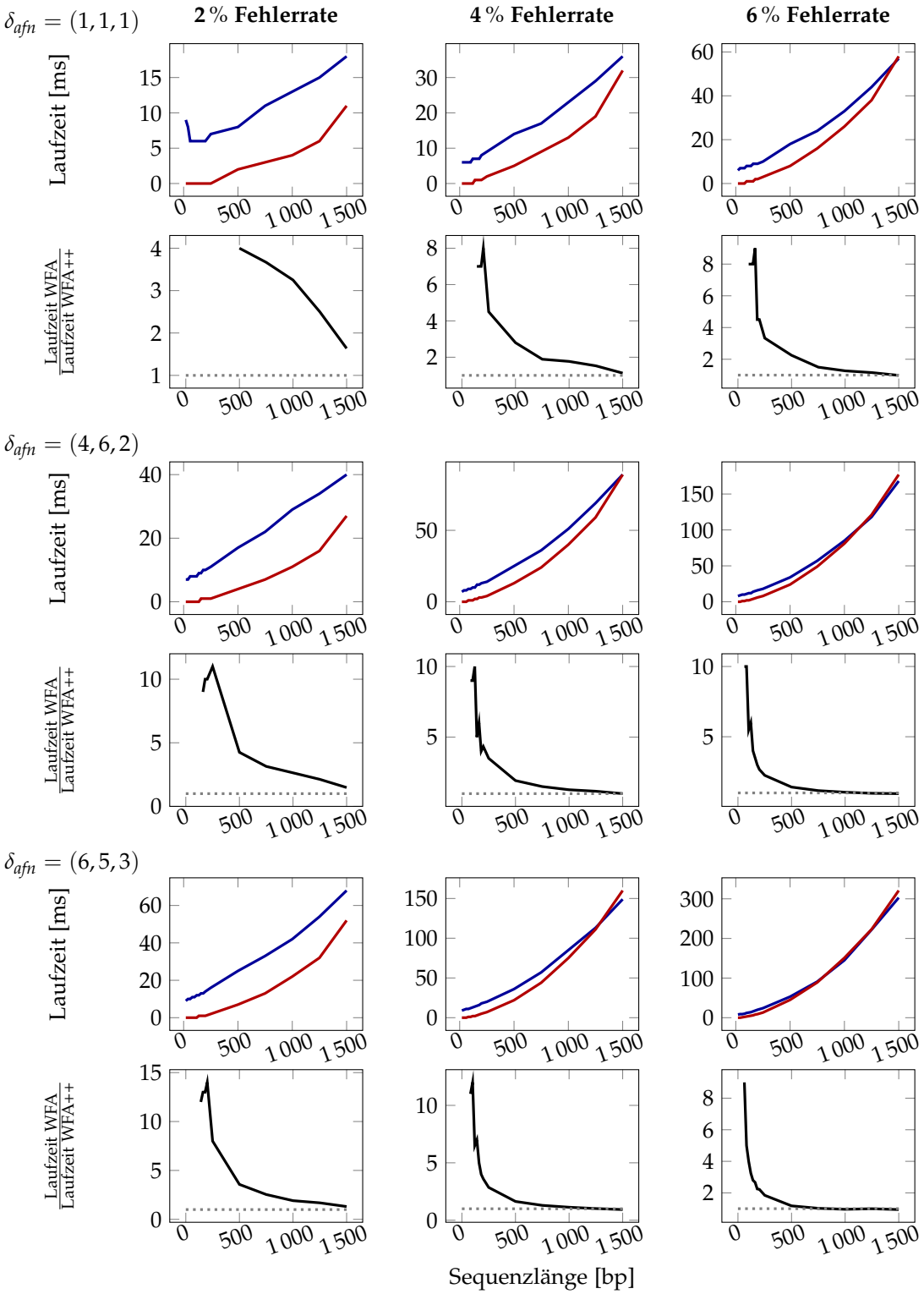

Abbildung 4.4: Laufzeiten auf System (b) für die Berechnung von globalen Alignments durch WFA — und der Edit-Distanz durch WFA++ —. Es wurden drei Kostenmodelle und generierte Sequenzpaare mit $m \in \{20, 40, 60, 80, 100, 120, 140, 160, 180, 200, 250, 500, 750, 1\,000, 1\,250, 1\,500\}$ und drei Fehlerraten verwendet.

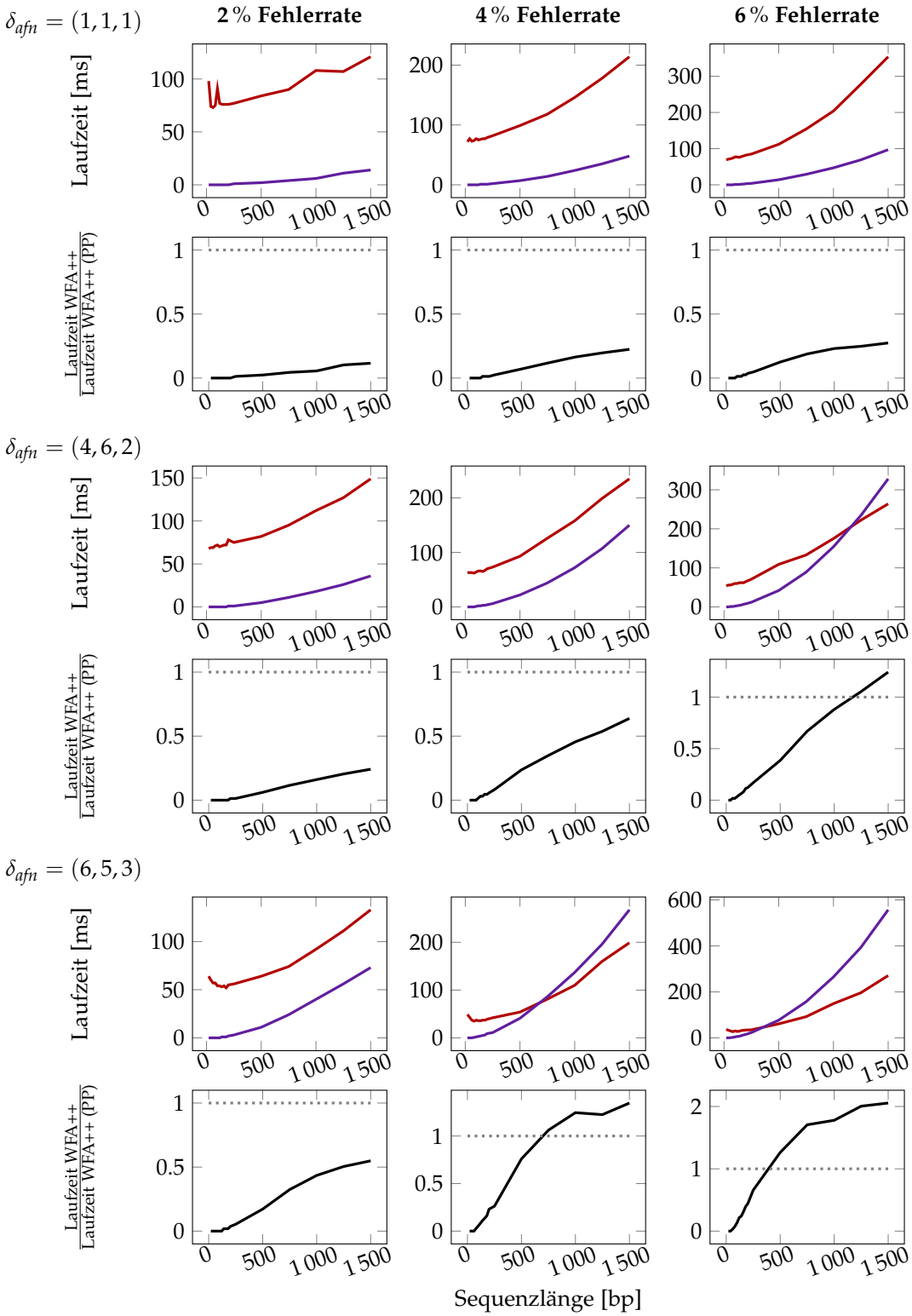

Abbildung 4.5: Laufzeiten auf System (a) für die Berechnung der Edit-Distanz durch WFA++ ohne Abbruchkriterium — und mit dem Abbruchkriterium aus Abschnitt 3.1 ($\gamma = \zeta = 5.0$) —. Es wurden drei Kostenmodelle und generierte Sequenzpaare mit $m \in \{20, 40, 60, 80, 100, 120, 140, 160, 180, 200, 250, 500, 750, 1\,000, 1\,250, 1\,500\}$ und drei Fehlerraten verwendet.

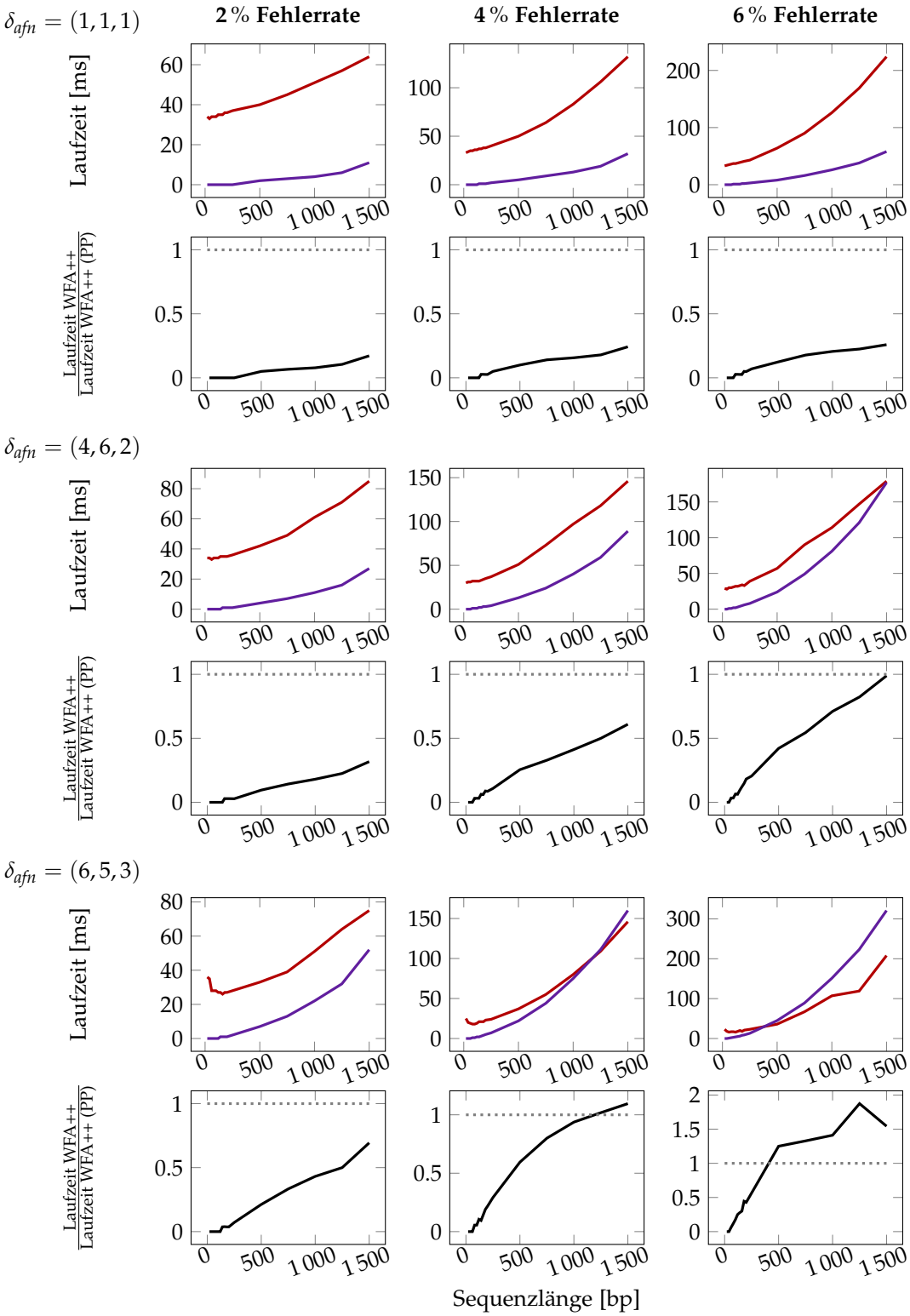

Abbildung 4.6: Laufzeiten auf System (b) für die Berechnung der Edit-Distanz durch WFA++ ohne Abbruchkriterium — und mit dem Abbruchkriterium aus Abschnitt 3.1 ($\gamma = \zeta = 5.0$) —. Es wurden drei Kostenmodelle und generierte Sequenzpaare mit $m \in \{20, 40, 60, 80, 100, 120, 140, 160, 180, 200, 250, 500, 750, 1\,000, 1\,250, 1\,500\}$ und drei Fehlerraten verwendet.

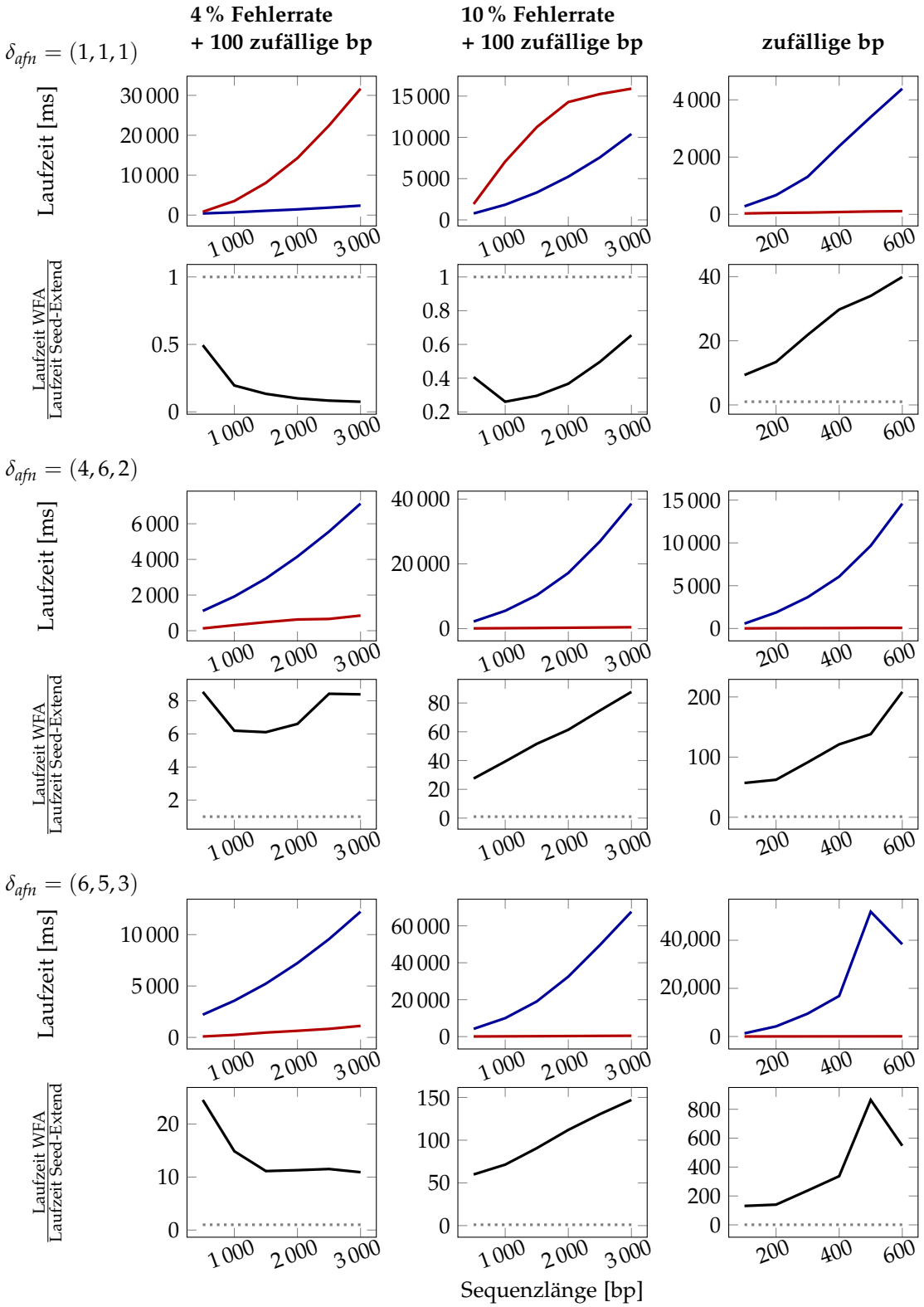

Abbildung 4.7: Laufzeiten auf System (a) für die Berechnung gobaler Alignments durch WFA —— und lokaler Alignments durch ein Seed-and-Extend Programm inkl. WFA++ ——. Es wurden drei Kostenmodelle, generierte Sequenzpaare mit $m \in \{500, 1\,000, 1\,500, 2\,000,\ 2\,500, 3\,000\}$ inkl. 100 zufälligen Basenpaaren am Ende für zwei Fehlerraten und zufällige Sequenzen der Längen $\{100, 200, 300, 400, 500, 600\}$ verwendet.

Kapitel 5

5 Diskussion

In dieser Arbeit wurde eine flexible und strukturierte C++ Implementierung des WFA Algorithmus vorgestellt. Die Implementierung, genannt WFA++, wurde um die Technik der Polished Points erweitert, sodass eine Verwendung im Kontext der Erweiterung von Seeds möglich ist.

Umfangreiche Evaluationen an simulierten Daten zeigen, dass die Laufzeiten von WFA und WFA++ stark von der Länge der Sequenzen, ihrer Fehlerrate sowie von der Wahl des Kostenmodells abhängen.

Bei der Berechnung von optimalen Alignments ergaben sich im Vergleich zur initialen C Implementierung von WFA für kurze, ähnliche Sequenzen vergleichbare oder geringere Laufzeiten. Dies ist im Kontext der Seed-Erweiterung vorteilhaft, denn es werden dabei genau solche Sequenzabschnitte verarbeitet und die Berechnungen bei der Verwendung eines passenden Kriteriums abgebrochen, sobald zu viele Differenzen auftreten. Dabei ist es sinnvoll, die Seed-Erweiterung zunächst ohne Bestimmung eines Alignments durchzuführen, da man im ersten Schritt nur daran interessiert ist, wie weit und mit welcher Fehlerzahl sich ein Seed erweitern lässt. Nur bei einer erfolgreichen Seed-Erweiterung, die für einen geringen Teil der Seeds zu erwarten ist, muss ein Alignment überhaupt konstruiert werden. Im Kontext der Seed-Erweiterung ist es sinnvoll, die Distance-Only Variante von WFA++ mit der WFA Implementierung (die keine Distance-Only Variante bietet) zu vergleichen. Hierfür konnte WFA++ gegenüber WFA eine Verbesserung der Laufzeit bis zu einem Faktor von 14 erreichen.

Außerdem konnte bei der Kombination dieser Technik mit dem auf den Polished Points basierenden Abbruchkriterium der frühzeitige Abbruch besonders bei langen Sequenzen mit einer hohen Fehlerrate gezeigt werden. Für kleine Fehlerraten konnte der erwartete Einfluss des Abbruchkriteriums auf die Laufzeit um

einen konstanten Faktor in den Messungen bestätigt werden. Dabei ist dies stark vom gewählten Kostenmodell abhängig. Bei der Verwendung von WFA++ zur Seed-Erweiterung in einem Seed-and-Extend Programm konnte in Abhängigkeit von den affinen Kosten der frühzeitige Abbruch der Berechnungen bei einer Fehlerrate von 10 % im Gegensatz zu einem Fehler von 4 % gezeigt werden. Bei vollständig zufälligen Sequenzen brach das Programm die Berechnungen, wie für lokale Alignemnts gewünscht, direkt ab. Um zu überprüfen, ob mit diesem Abbruchkriterium eine ausreichende Sensitivität und Spezifität bei der Berechnung lokaler Alignments erreicht wird, sind weitere Tests nötig. Diese könnten auch auf Basis des C++ Generators für Sequenzpaare stattfinden.

Die Implementierung von WFA++ bietet Potential für weitere Optimierungen.

Zum Einen könnte die Berechnung von $lo(d)$ und $hi(d)$ beschleunigt werden. Die aus WFA übernommene Technik berechnet diese Werte als Minima beziehungsweise Maxima der Werte von den Wavefronten für $d - x$, $d - o - e$ und $d - e$ mit Substraktion beziehungsweise Addition von 1. Da die Anzahl der zu berechnenden Diagonalen mit steigendem d zunimmt, werden die Diagonalindizes von der Vorgänger-Wavefront mit dem maximalen d bestimmt. Dies hängt von $\min(x, e)$ ab. Die Idee besteht darin, den minimalen und maximalen Diagonalindex von d durch folgende Gleichungen zu berechnen:

$$lo(d) = -\left\lfloor \frac{d}{\min(x,e)} \right\rfloor$$
$$hi(d) = \left\lfloor \frac{d}{\min(x,e)} \right\rfloor$$

Für den realistischen Fall, dass $\min\{x, o, e\} > 0$ und $x < 2(o + e)$ gilt, konnten die Gleichungen für 6 600 affine Gap-Kostenmodelle bestätigt werden.

Ein Problem der C++ Implementierung war die Behandlung der nicht benötigten Wavefronten. Ein Ansatz wäre es hier, die Eigenschaften der Kostenmodelle zur Kompilierungszeit auszunutzen. Bei dem Kostenmodell $\delta_{afn} = (4, 6, 2)$ wird beispielsweise bei der Berechnung nur jede zweite Wavefront benötigt. Hier wäre es eine Möglichkeit, die Wavefronten durch den Ausdruck $d/2$ zu indizieren, sodass unnötigen Wavefronten übersprungen werden. Dadurch würden die Berechnungen vereinfacht und Speicherplatz reduziert werden.

Eine andere Laufzeitoptimierung ist die Parallelisierung der Berechnungen. In [25] werden dazu Optimierungen des WFA-Algorithmus auf Ebene der Sequenzpaare, der Wavefronten oder der Front-Werte beschrieben, die auf den einfachen Abhängigkeiten der Wavefronten beruhen. Diese Prinzipien könnten auch in der WFA++ Implementierung verwendet werden.

Für die Distance-Only Variante lässt sich der Speicherplatzbedarf reduzieren, indem nur die für die aktuellen Berechnungen notwendigen Wavefronten gespeichert werden. In [26] wird durch eine Technik der Speicher des WFA-Algorithmus auf $O(edist_{\delta_{afn}}(u, v))$ reduziert, allerdings auf Kosten der Laufzeit. Dabei werden nur die benötigten Wavefronten gespeichert und beim Backtracing die benötigten Front-Werte ab einem Breakpoint rückwärts erneut berechnet.

Ein anderer Ansatz wäre es, analog zum WFA-Adapt Algorithmus [4] den Algorithmus mit beispielsweise den in Abschnitt 2.7 vorgestellten Trimming-Strategien aus [16] zu kombinieren, um den Suchraum zu reduzieren. Man würde also die ohnehin für den Abbruch im Kontext der Seed-Erweiterung berechneten Polished Points für einen weiteren Zweck nutzen. Bei geeigneter Wahl der Parameter würde wie in [16] nur in wenigen Fällen das optimale Ergebnis verfehlt werden.

Literaturverzeichnis

[1] Stefan Kurtz. Foundations of sequence analysis. Lecture notes for a course in the Wintersemester 2020/2021, Universität Hamburg, 2020.

[2] David W. Mount. *Bioinformatics: Sequence and Genome Analysis*. Cold Spring Harbor Laboratory Press, U.S., 2001.

[3] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

[4] Santiago Marco-Sola, Juan Carlos Moure, Miquel Moreto, and Antonio Espinosa. Fast gap-affine pairwise alignment using the wavefront algorithm. *Bioinformatics*, 37(4):456–463, 09 2020.

[5] Nauman Ahmed, Koen Bertels, and Zaid Al-Ars. A comparison of seed-and-extend techniques in modern DNA read alignment algorithms. In *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1421–1428, 2016.

[6] Osamu Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705–708, 1982.

[7] Stefan Kurtz. Genominformatik. Lecture notes for a course in the Sommersemester 2020, Universität Hamburg, 2020.

[8] Eugene W. Myers. An O(ND) difference algorithm and its variations. *Algorithmica*, 1(2):251–266, 1986.

[9] Esko Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64(1):100–118, 1985. International Conference on Foundations of Computation Theory.

[10] Santiago Marco-Sola, Juan Carlos Moure, Miquel Moreto, and Antonio Espinosa. WFA. `https://github.com/smarco/WFA`, 2021. (letzter Zugriff: 13.09.2022).

[11] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[12] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[13] David Lipman and William Pearson. Rapid and sensitive protein similarity searches. *Science (New York, N.Y.)*, 227:1435–41, 04 1985.

[14] Camille Marchet, Christina Boucher, Simon J Puglisi, Paul Medvedev, Mikaël Salson, and Rayan Chikhi. Data structures based on k-mers for querying large collections of sequencing datasets. *bioRxiv*, 2019.

[15] Heng Li and Nils Homer. A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in bioinformatics*, 11 5:473–83, 2010.

[16] Eugene W. Myers. Efficient local alignment discovery amongst noisy long reads. In *Algorithms in Bioinformatics - 14th International Workshop, WABI 2014, Wroclaw, Poland, 08.-10.09.2014, Proceedings*, pages 52–67, 2014.

[17] Stefan Kurtz. GTTL. `https://github.com/stefan-kurtz/gttl`. (letzter Zugriff: 10.09.2022).

[18] Partial template specialization. `https://en.cppreference.com/w/cpp/language/partial_specialization`. (letzter Zugriff: 21.09.2022).

[19] Deb Haldar. 6 tips to supercharge C++11 vector performance. `https://www.acodersjourney.com/6-tips-supercharge-cpp-11-vector-performance`. (letzter Zugriff: 11.09.2022).

[20] std::vector::at. `https://cplusplus.com/reference/vector/vector/at`. (letzter Zugriff: 18.09.2022).

[21] C++ - performance of vector of pointer to objects, vs performance of objects. `https://stackoverflow.com/questions/22703663/c-performance-of-vector-of-pointer-to-objects-vs-performance-of-objects`, 2014. (letzter Zugriff: 18.09.2022).

[22] Silke Rolles. Markovketten (MA2404). Notizen zur Vorlesung, Wintersemester 2015/2016, Technische Universität München.

[23] A. Seidel. Entwicklung eine Software-Bibliothek für paarweise Sequenzalignments in linearem Speicherplatz. Bachelorarbeit, Studiengang B.Sc. Computing in Science, Universität Hamburg, 2015.

[24] Gordon Gremme. GenomeTools. `https://github.com/genometools/genometools`. (letzter Zugriff: 10.09.2022).

[25] Quim Aguado-Puig, Santiago Marco-Sola, Juan Carlos Moure, Christos Matzoros, David Castells-Rufas, Antonio Espinosa, and Miquel Moreto. WFA-GPU: Gap-affine pairwise alignment using GPUs. *bioRxiv*, 2022.

[26] Santiago Marco-Sola, Jordan M. Eizenga, Andrea Guarracino, Benedict Paten, Erik Garrison, and Miquel Moreto. Optimal gap-affine alignment in O(s) space. *bioRxiv*, 2022.

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich vorliegende Bachelorarbeit im Studiengang Computing in Science selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Hamburg, den 11. Dezember 2023

_____

Merle Stahl

Ich bin mit einer Einstellung der Bachelorarbeit in den Bestand der Bibliothek des Departments Informatik einverstanden.

Hamburg, den 11. Dezember 2023

_____

Merle Stahl