BERECHNUNG DER RNA-PARTITIONSFUNKTION MIT ALGEBRAISCHER DYNAMISCHER PROGRAMMIERUNG

KERSTIN SCHMATZER

Masterarbeit

Lehrstuhl für Bioinformatik
Technische Fakultät
Albert-Ludwigs-Universität Freiburg

23. Februar 2012

Kerstin Schmatzer: *Berechnung der RNA-Partitionsfunktion mit Algebraischer Dynamischer Programmierung,* Masterarbeit, © 23. Februar 2012

BETREUER:
Professor Dr. Rolf Backofen
Dr. Mathias Möhl
GUTACHTER:
Professor Dr. Rolf Backofen
Professor Dr. Robert Giegerich

ABSTRACT

Prediction of secondary RNA structures is a important subarea of Bioinformatics. McCaskill developed a dynamic programming algorithm for calculating the base-pair probabilities using the partition function. It is possible to draw conclusions from this probabilities to the secondary structure of the RNA sequence.

Instead of the traditional dynamic programming, we use the new technique of algebraic dynamic programming. This technique is more modular, more flexible and based on context-free grammars. In our work, we introduce a grammar to calculate the partition function of an RNA sequence. This grammar divides the RNA sequence into possible secondary RNA structure elements and calculates for each element the corresponding partition function using the energy model of Turner. For determining the base-pair probabilities, we make use of the calculated partition functions for the secondary structure elements.

ZUSAMMENFASSUNG

Die Bioinformatik bietet eine Reihe von effizienten Algorithmen für die Vorhersage von sekundären RNA-Strukturen an. Der Algorithmus von McCaskill gibt die Basenpaar-Wahrscheinlichkeiten einer RNA-Sequenz an, die Rückschlüsse auf die wahrscheinlichste sekundäre RNA-Struktur zu lässt. Für die Berechnung der Basenpaar-Wahrscheinlichkeiten wird die Partitionsfunktion, ermittelt mit Hilfe von loop-basierte Energien, verwendet. Diese lassen sich mittels dynamischer Programmierung berechnen.

Mittels algebraischer dynamischer Programmierung lässt sich das Problem modularer lösen. Wir stellen eine Grammatik vor, welche die Partitionsfunktion für eine gegebene RNA-Sequenz nach der Vorgehensweise von McCaskill berechnet. Innerhalb der Grammatik werden einzelne möglichen sekundäre RNA-Strukturen betrachtet und deren Partitionsfunktionen anhand des Energiemodells von Turner ermittelt. Zur Berechnung der Basenpaar-Wahrscheinlichkeiten werden die schon bestimmten Partitionsfunktionen der Strukturelemente hinzu gezogen.

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

— Donald E. Knuth [4]

DANKSAGUNG

Zuerst möchte ich meinem Betreuer, Dr. Mathias Möhl, für seine tatkräftige Unterstützung meiner Masterarbeit danken. Vielen Dank für die hilfreichen Anregungen und die Engelsgeduld mit mir!

Vielen Dank an Professor Dr. Rolf Backofen für die Möglichkeit mich mit diesem interessanten Themengebiet zu beschäftigen.

Besonderen Dank schulde ich auch Georg Sauthoff und Stefan Janssen für die Bereitschaft, meine vielen Fragen zur Implementierung von Bellman's Gap zu beantworten. Vielen Dank für die Unterstützung!

Ich danke meiner Familie für ihre Liebe, ihre Unterstützung und ihre vielen liebevollen Aufmunterungen. Besonders möchte ich meinen Großeltern für die finanzielle Unterstützung meiner Ausbildung danken, die mir vieles ermöglicht hat. Mein Dank gilt auch meinem besten Freund, Dr. Achim Brucker, der mich durch Höhen und Tiefen begleitet hat. Danke für deine Freundschaft!

Herzlichen Dank an all meine Lektoren für die hilfreichen Anmerkungen und den Mut meine Arbeit zu lesen.

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

Teil I

EINFÜHRUNG

1 EINLEITUNG

1.1 MOTIVATION

Ribonukleinsäure (RNA) ist eine Kette von Nukleotiden. In der RNA kommen die folgenden organischen Basen vor: Adenin, Guanin, Cystein und Uracil, abgekürzt mit A, G, C und U. Eine RNA kann sich in komplexe Strukturen falten. Der Grund dafür liegt in der Fähigkeit der Basen untereinander Wasserstoffbrücken und Stapelwechselwirkungen ("stacking") auszubilden. RNA-Moleküle besitzen verschiedene Funktionen in einem sehr breiten biologischen Bereich. Im Folgenden sind einige Funktionen von RNA-Molekülen aufgezählt:

- RNA-Moleküle können selbst enzymatische Funktionen übernehmen. In den Ribosomen besitzen die sogenannten ribosomalen RNAs diese Eigenschaft.
- RNA-Moleküle können katalytisch wirksam sein.
- Bei der Genregulierung spielen RNA-Moleküle, die sogenannten microRNAs, eine wichtige Rolle und können dabei verschiedene biologische Prozesse beeinflussen, wie Entwicklung, Differenzierung und programmierten Zelltod.
- Der bekannteste RNA-Typ ist die messenger RNA, die als Matrize zur Proteinsynthese dient.
- Small nuclear RNAs sind kleine RNA-Moleküle, die in Assoziation mit Proteinen insbesondere an der Prozessierung der messenger RNA beteiligt sind.
- Die unterschiedlichsten RNAs transportieren Proteine innerhalb der Zelle. Zum Beispiel holen RNAs die entsprechenden Aminosäuren, die für die Proteinsynthese benötigt werden.

In der Zelle kommt die RNA meistens als Einzelstrang vor und faltet sich in sekundäre Strukturen. Die sekundäre RNA-Struktur eines RNA-Moleküls gibt Aufschluss auf dessen Funktion.

Es stehen unterschiedliche Methoden der Biophysik und der Molekularbiologie zur Verfügung, um sekundäre RNA-Strukturen zu bestimmen. Die Methoden unterscheiden sich drastisch in ihrem Aufwand, Grad an Auflösung und Information. Beispiel für chemische und molekulare Methoden sind folgende zwei Verfahren: enzymatisches und chemisches Mapping und UV-Spektroskopie. Alle chemischen und molekularen Verfahren sagen nur etwas über die aktuell

entstandende RNA-Struktur in der gegebene Zelle aus. Aber die RNA verändert sich je nach Temperatur ihre sekundäre RNA-Struktur. Das bedeutet es gibt nicht nur eine sekundäre RNA-Struktur, in die sich ein RNA-Molekül falten kann.

Es gibt verschiedene Verfahren zur Vorhersage von sekundären RNA-Strukturen. Durch die Vorhersage von sekundären Strukturen können experimentelle Strukturanalysen verglichen, bestätigt oder widerlegt werden. Auch kann eine Aussage über die Strukturverteilung eines RNA-Moleküls bei unterschiedlichen Temperaturen gemacht werden. Dies gibt uns eine Möglichkeit die Ergebnisse experimenteller Analysemethoden zu vergleichen. Die Vorhersage von sekundären RNA-Strukturen gibt uns Aufschluss auf die biologische Funktion des analysierten Moleküls.

In der Bioinformatik gibt es unterschiedliche Algorithmen zur Vorhersage der sekundären RNA-Struktur. Eine der bekanntesten Algorithmen sind der Algorithmus von Zuker [12] und der Algorithmus von McCaskill [6]. Der Algorithmus von Zuker berechnet die optimale sekundäre RNA-Struktur eines RNA-Molekühls mit minimaler freier Energie anhand eines gegebenen thermodynamischen Energiemodells. McCaskill gibt eine Algorithmus zur Vorhersage der sekundären Strukturverteilung eines RNA-Moleküls an. Beide Algorithmen lassen sich effizient mit dem Prinzip der dynamischen Programmierung lösen.

1.2 ZIEL UND STRUKTUR DER ARBEIT

Das Ziel der Arbeit ist es, einen Algorithmus zu entwickeln, der die Basenpaar-Wahrscheinlichkeiten für eine RNA-Sequenz mit Hilfe der Vorgehensweise des Verfahrens von McCaskill [6] berechnet. Die Berechnung geschieht mit Hilfe der Partitionsfunktionen der RNA-Strukturelemente. Zur Ermittlung der Partitionsfunktionen werden die einzelnen sekundären Strukturelemente mit ihren entsprechenden Energien betrachtet.

Zusätzlich soll für die Berechnung der RNA-Partitionsfunktion die neu entwickelte Programmiertechnik, die sogenannte algebraische dynamische Programmierung von R. Giegerich und C. Meyer [2], verwendet werden. Diese Methode beruht auf kontext-freie Grammatiken und Evaluationsalgebren und ermöglicht eine modularere und dadurch flexiblere Implementierung als die Methode der traditionellen dynamischen Programmierung.

Die Implementierung erfolgt mit der Programmierumgebung Bellman's Gap entwickelt von G. Sauthoff [10]. Diese ist speziell für Optimierungsprobleme, die mit algebraischer dynamischer Programmierung gelöst werden, entwickelt.

Zuerst stellen wir die Grundlagen und Methoden der Algorithmen, sowie der algebraischen dynamischen Programmierung vor. Danach erklären wir unsere Version des Algorithmus zur Berechnung der

Basenpaar-Wahrscheinlichkeiten. Anschließend gehen wir kurz auf die Implementierung des Algorithmus ein.

Teil II

GRUNDLAGEN

2 GRUNDLEGENDE ALGORITHMEN

In diesem Kapitel werden die grundlegenden Algorithmen erklärt. Zuerst betrachten wir den Algorithmus von Zuker und Stiegler zur Berechnung sekundärer RNA-Strukturen unter Benutzung loop-basierter Energien für eine gegebene RNA-Sequenz. Als nächstes widmen wir unsere Aufmerksamkeit dem Algorithmus von McCaskill zur Berechnung der Wahrscheinlichkeiten für die jeweiligen sekundären RNA-Strukturen einer gegebenen RNA-Sequenz.

2.1 BERECHNUNG DER OPTIMALEN SEKUNDÄREN STRUKTUR

Der Algorithmus von Zuker und Stiegler [12] berechnet die optimale sekundären RNA-Strukturen für eine gegebene RNA-Sequenz unter Beachtung der freien Energien für die einzelnen Elemente der sekundären RNA-Struktur.

Sei im Weiteren $s = s_1 ... s_n$ eine RNA-Sequenz mit Länge n, wobei $s_i \in \{A, G, C, U\}$ mit $1 \leqslant i \leqslant n$.

DEFINITION (Sekundäre RNA-Struktur)
Eine *sekundäre RNA-Struktur* R ist eine Menge von Basenpaaren. Formal,

$$R = \{(i,j) | 1 \leqslant i < j \leqslant n \textit{ und } s_i, s_j \textit{ bildet ein Basenpaar}\},$$

wobei folgendes für alle (i,j) gilt:

- $\forall (i' \neq i)(i',j) \notin R$,
- $\forall (j' \neq j)(i,j') \notin R$.

Der Algorithmus von Zuker und Stiegler [12] betrachten keine Pseudoknoten, sprich es werden nur sekundäre RNA-Strukturen beachtet, für die gilt: $\nexists((i,j) \in R \wedge (i',j') \in R)$ mit $i < i' < j < j'$

Wir betrachtet nur Watson-Crick Basenpaarbindungen, sprich die Basen A und U, die Basen G und U und die Basen G und C können ein Basenpaar bilden.

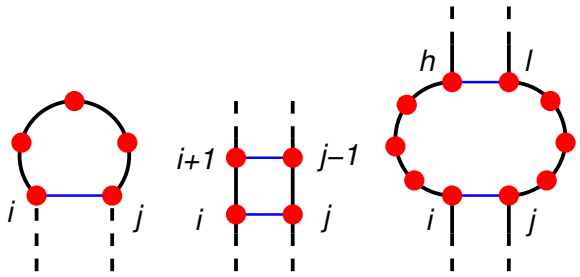
Jede sekundäre RNA-Struktur hat ihre eigene Energie. Diese Energie ist abhängig von den gebildeten sekundären RNA-Strukturelementen innerhalb der RNA-Struktur.

DEFINITION (Sekundäre RNA-Strukturelemente)
Ein *sekundäres RNA-Strukturelement* einer RNA-Struktur R mit schließendem Basenpaar (i,j) ist eines der folgenden:

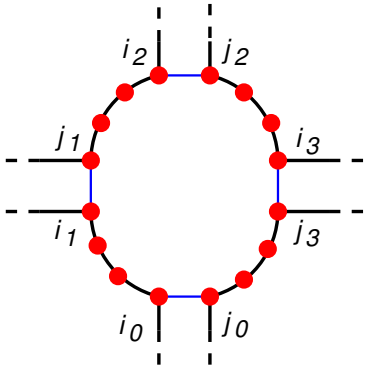
- *Hairpin*: $(i,j) \in R$, wobei $(i',j') \notin R$ für alle $(i < i' < j' < j)$.

- *Stack*: $(i,j) \in R$ und $(i+1,j-1) \in R$.
- *Interiorloop*: $(i,j) \in R$ und $(h,l) \in R$ mit $i+1 < h < l < j-1$. Die Teilsequenzen $[i+1,h-1]$ und $[l+1,j-1]$ enthalten nur ungepaarte Basen.

  Zusätzlich sind zwei spezielle Fälle möglich, genannt Bulge, falls $(h = i+1) < l < j-1$ oder falls $i+1 < h < (l = j-1)$.
- k-*Multiloop*: mit einem schließenden Basenpaar $(i_0,j_0) \in R$ und k weiteren Basenpaare $(i_1,j_1),\ldots,(i_k,j_k) \in R$, wobei noch folgende Eigenschaften gelten:
  - $i_0 < i_1 < j_1 < \cdots < i_k < j_k < j_0$
  - $\forall (l,l') \notin R$, falls $(l,l') \neq (i_1,j_1),\ldots,(i_k,j_k)$, wobei gilt, dass $i_0 < l < l' < j_0$.
  - Jedes (i_l,j_l) mit $l \in \{1,\ldots,k\}$ ist selber eine schließendes Basenpaar einer sekundären RNA-Struktur.



(a) Hairpin (b) Stack (c) Interiorloop



(d) Multiloop

Abbildung 1: Sekundäre Strukturelemente

Abbildung 1 zeigt die sekundären RNA-Strukturen - Hairpin, Stack, Interiorloop und Multiloop.

DEFINITION (Energie eines sekundären RNA-Strukturelements)
Die *Energie eines sekundären RNA-Strukturelements* mit schließendem Basenpaar (i, j) ist für:

- einen Hairpin: $eH(i, j)$,
- einem Stack: $eS(i, j, i+1, j+1)$ mit $(i+1, j+1) \in R$,
- einem Interiorloop: $eL(i, j, h, l)$ mit $(h, l) \in R$ und
- einem k-Multiloop: $eM(k, k') = a + b \cdot k + c \cdot k'$ mit den Gewichten a, b und c, wobei a die Energie für das schließende Basenpaar (i, j), k die Anzahl der Basenpaar innerhalb des k-Multiloops und k' die Anzahl der ungepaarten Basen innerhalb des k-Multiloops ist.

DEFINITION (Energie einer sekundären RNA-Struktur)
Die freie *Energie einer sekundären RNA-Struktur* R ist die Summe der Energien aller sekundären Strukturelemente für jedes Basenpaar $(i, j) \in R$:

$$E(R) = \sum_{(i,j)\in R} E^R_{i,j},$$

wobei $E(R)$ die Energie der sekundären RNA-Struktur R und $E^R_{i,j}$ die Energie der sekundären RNA-Strukturelemente mit schließendem Basenpaar (i, j) ist.

Die Berechnung der freien Energie einer sekundären RNA-Struktur kann durch rekursive Formeln beschrieben werden. Dafür werden alle möglichen Fälle von sekundären RNA-Strukturen betrachtet. Die Matrizen zur Berechnung werden durch die nachfolgende Definition erklärt.

DEFINITION (Zuker-Matrizen)

1. Die *Matrix* W_i berechnet für die Sequenz $s_0...s_i$ die sekundäre RNA-Struktur mit minimaler freier Energie.

$$W_i = \min \begin{cases} W_{i-1} \\ \min_{0 \leqslant j-1 < i}\{W_j + V_{j,i}\} \end{cases}$$

   wobei $W_0 = 0$.

2. Die *Matrix* $V_{i,j}$ berechnet für die Sequenz $s_i...s_j$ die sekundäre RNA-Struktur mit minimaler freier Energie, wobei s_i und s_j ein Basenpaar bilden.

$$V_{i,j} = \min \begin{cases} eH(i, j) \\ eS(i, j, i+1, j-1) + V_{i+1,j-1} \\ \min_{i+1<i'<j'<j-1}\{eL(i, j, i', j') + V_{i',j'}\} \\ \min_{i+1<k<j}\{M_{i+1,k} + M_{k+1,j-1} + a\} \end{cases}$$

   wobei $V_{i,i} = 0$.

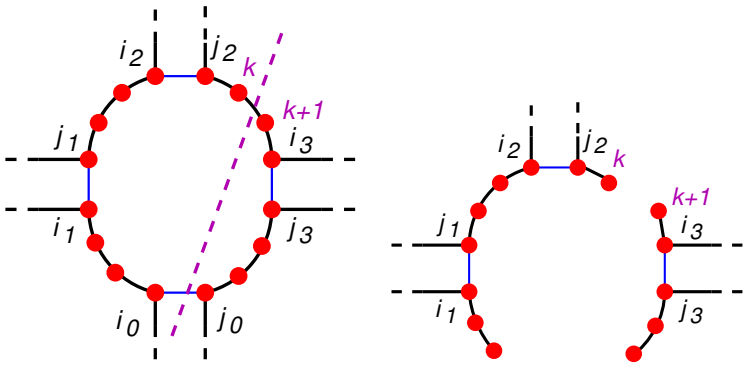
3. Die *Matrix* $M_{i,j}$ berechnet für die Sequenz $s_i ... s_j$ die sekundäre RNA-Struktur mit minimaler freier Energie, wobei s_i und s_j Teile eines Multiloops sind.

$$M_{i,j} = \min \begin{cases} M_{i+1,j} + c \\ M_{i,j-1} + c \\ \min_{0<k\leqslant i}\{M_{i,k} + M_{k+1,j}\} \\ V_{i,j} + b \end{cases}$$

wobei $M_{i,i} = 0$.

Eine Base s_i in der RNA-Sequenz s kann nicht gebunden oder mit einer anderen Base s_j gebunden sein. Dieser Basisfall wird in einer Rekursionsmatrix W_i betrachtet. Falls die Base s_i mit einer anderen Base s_j gebunden ist, werden alle möglichen Energien der einzelnen sekundären RNA-Strukturen berechnet. Die Base s_i kann mit einer anderen Base s_j einen Hairpin, einen Stack, einen Loop oder einen Multiloop bilden. Die Energie für einen Hairpin, einen Stack, einen Loop können leicht berechnet werden. Im Falle eines Stack oder Loops wird die Berechnung rekursiv auf die Basenpaare $s_{i'}$ und $s_{j'}$ angewandt, die mit dem Basenpaar s_j und s_i den Stack oder Loop bilden. Im Fall eines Multiloops ist es notwendig eine zusätzliche Rekursionsmatrix für die Berechnung seiner Energie zu erstellen. Der Multiloop wird an einer Base s_k geteilt, wobei $i < k < j$ gilt und zwischen $s_{i+1} ... s_k$ und zwischen $s_{k+1} ... s_{j-1}$ eine weiteres Basenpaar existiert um keinen einfachen Interiorloop zu erhalten. Die Berechnung der Energie wird dann rekursiv jeweils auf die RNA-Sequenzen $s_{i+1} ... s_k$ und $s_{k+1} ... s_{j-1}$ angewandt. Abbildung 2 zeigt die Teilung eines Multiloops an einer Stelle der Base s_k. Die Berechnung der Multiloop Energie geschieht in einer zusätzlichen Matrix. Die aktuell betrachteten Basen i und j können gepaart oder ungepaart sein. Eine weitere Möglichkeit ist, an einer Stelle $s_{k'}$ den Multiloop rekursiv zu teilen, wobei $i < k' < j$ gilt.

Die Berechnung der *W*-Matrix benötigt eine quadratische Laufzeit $O(n^2)$, weil es nötig ist, einmal durch die gesamte RNA-Sequenz s mit der Länge n zu laufen und jede Base mit allen möglichen anderen Basen zu betrachten, um alle möglichen Basenpaare zu bestimmen. Die *V*-Matrix wird in $O(n^4)$ Zeit berechnet. Die hohe Laufzeit ist auf die Betrachtung aller möglichen Interiorloops zurückzuführen. Die Berechnung der *V*-Matrix für alle möglichen Basenpaare (i, j) führt zu einer quadratischen Laufzeit. Im Falle eines Interiorloops wird ein weiteres Basenpaar (i', j') gesucht, das mit dem Basenpaar (i, j) den Loop bildet. Die Suche eines weiteren Basenpaars (i', j') benötigt auch eine quadratische Laufzeit. So vergrößert sich die Laufzeit zu $O(n^4)$. Die *M*-Matrix benötigt $O(n^3)$ Laufzeit. Hier wird für jedes mögliche Basenpaar (i, j) die Matrix einmal berechnet. Zusätzlich wird jeder



(a) Multiloop Teilung an Position k

(b) Geteilter Multiloop

Abbildung 2: Der Multiloop wird an der Stelle k geteilt. Auf die dadurch entstandenen Teile des Multiloops wird die Berechnung der sekundären RNA-Strukturen rekursiv angewendet.

mögliche Splitpunkt k eines Multiloops gesucht, der im schlimmsten Fall an jeder Base der gesamten RNA-Sequenz gesetzt werden kann.

2.2 BERECHNUNG DER STRUKTURVERTEILUNG

McCaskill [6] stellt ein Methode zur Berechnung der Partitionsfunktion für RNA-Sequenzen basierend auf der Boltzmann Verteilung vor , um die Strukturverteilung einer RNA-Sequenz zu berechnen.

Das Problem bei Zuker ist, dass die sekundäre RNA-Struktur mit minimaler Energie nicht immer die biologisch relevanteste ist. Für sekundäre RNA-Strukturen mit geringer Energie ist es möglich, biologisch relevanter zu sein, als alle anderen möglichen sekundären RNA-Strukturen. Daher wird ein neue Möglichkeit benötigt, um diejenige RNA-Struktur heraus zu finden, die biologisch am relevantesten ist. McCaskill berechnet für eine RNA-Sequenz die Wahrscheinlichkeit der Basenpaare mit Hilfe der Boltzmann-Verteilung. Anhand der Basenpaar-Wahrscheinlichkeiten lässt sich die wahrscheinlichste sekundäre RNA-Struktur ermitteln.

DEFINITION (Boltzmann-Verteilung)

Sei $P(x)$ die Wahrscheinlichkeit für eine RNA-Struktur x. Dann ist eine *Boltzmann-Verteilung* gegeben durch:

$$P(x) = \frac{\exp(-E(x)/(k_B T))}{Z},$$

wobei $E(x)$ die Energie von x, T die Temperatur, k_B die Boltzmann-Konstante ist und $Z = \sum_y \exp(-E(y)/(k_B T))$, wobei y eine mögliche sekundäre RNA-Struktur ist.

DEFINITION (Partitionsfunktion für eine RNA-Sequenz)
Die *Partitionsfunktion für eine RNA-Sequenz* s ist:

$$Z = \sum_{R} \exp(-E(R)/(k_B T)),$$

wobei R eine sekundäre RNA-Struktur von s, T die Temperatur und k_B die Boltzmann-Konstante ist.

Der Algorithmus von McCaskill berechnet die Partitionsfunktion Z für eine RNA-Sequenz s effizient. Als Energiemodell für sekundäre RNA-Strukturen wird das Modell von Zuker benutzt. Das bedeutet, dass die Energie einer sekundären RNA-Struktur der Summe aller möglichen sekundären RNA-Strukturelemente entspricht.

Um eine effiziente Berechnung der Partitionsfunktion zu garantieren, ist eine Zerlegung des Problems in Teilprobleme notwendig. RNA-Sequenzen werden daher in sogenannte Strukturmengen unterteilt.

DEFINITION (Strukturmengen einer RNA-Sequenz)
Gegeben sei eine RNA-Sequenz s. Diese wird in vier *Strukturmengen* unterteilt:

- $\mathbf{R}_{all}$: Menge aller sekundären RNA-Strukturen von s.
- $\mathbf{R}_{i,j}$: Menge aller sekundären RNA-Strukturen, die nur zwischen i und j Basenpaare enthalten:

$$\mathbf{R}_{i,j} = \{R \in \mathbf{R} | \forall (i',j') \in R : i \leqslant i' < j' \leqslant j\}$$

- $\mathbf{R}^{b}_{i,j}$: Menge aller sekundären RNA-Strukturen, die (i,j) als Basenpaar haben und nur zwischen i und j Basenpaare enthalten:

$$\mathbf{R}^{b}_{i,j} = \{R \in \mathbf{R} | (i,j) \in R \wedge \forall (i',j') \in R : i \leqslant i' < j' \leqslant j\}$$

- $\mathbf{R}^{lbd}_{i,j}$: Menge aller sekundären RNA-Strukturen, die zwischen i und j mindestens ein Basenpaar haben:

$$\mathbf{R}^{lbd}_{i,j} = \{R \in \mathbf{R} | \forall R : \exists (i',j') \in R : i < i' < j' < j\}$$

DEFINITION (Partitionsfunktion einer Strukturmenge)
Sei $\mathbf{R}$ eine Strukturmenge. Dann ist deren *Partitionsfunktion* gegeben durch:

$$Z_{\mathbf{R}} = \sum_{R \in \mathbf{R}} \exp(-E(R)/(k_B T)).$$

Damit die Berechnung der Partitionsfunktion der Strukturmengen möglich ist, muss sicher gestellt sein, dass die Zerlegung der Strukturmengen unabhängig ist.

DEFINITION (Unabhängige Zerlegung)
Sei eine Strukturmenge $\mathbf{R}$ gegeben. Zwei Strukturmengen $\mathbf{R_1}$ und $\mathbf{R_2}$ sind eine *unabhängige Zerlegung* von $\mathbf{R}$, falls eine bijektive Abbildung $f : \mathbf{R} \to \mathbf{R_1} \times \mathbf{R_2}$ existiert mit folgenden Bedingungen:

- $f : \mathbf{R} \to \mathbf{R_1} \times \mathbf{R_2} : R = R_1 \uplus R_2$ und
- $E(R) = E(R_1) + E(R_2)$

mit $R \in \mathbf{R}$, $R_1 \in \mathbf{R_1}$ und $R_2 \in \mathbf{R_2}$.

THEOREM: (Unabhängige Zerlegung)
Gegeben sei eine Strukturmenge $\mathbf{R}$. Seien nun $\mathbf{R}_1$ und $\mathbf{R}_2$ zwei strukturell unabhängige Strukturmengen. Dann gilt:

$$Z_{\mathbf{R}} = Z_{\mathbf{R}_1} \cdot Z_{\mathbf{R}_2}$$

BEWEIS: Seien $\mathbf{R}_1$ und $\mathbf{R}_2$ zwei strukturell unabhängige Strukturmengen.

$$\begin{aligned} Z_{\mathbf{R}} &= \sum_{R \in \mathbf{R}} e^{-E(R)/(k_B T)} \\ &= \sum_{R \in \mathbf{R}_1 \uplus \mathbf{R}_2} e^{-E(R)/(k_B T)} \end{aligned}$$

$\mathbf{R}_1$ und $\mathbf{R}_2$ sind strukturell unabhängig. Daher folgt:

$$\begin{aligned} Z_{\mathbf{R}} &= \sum_{R \in \mathbf{R}_1} e^{-E(R)/(k_B T)} + \sum_{R \in \mathbf{R}_2} e^{-E(R)/(k_B T)} \\ &= Z_{\mathbf{R}_1} \cdot Z_{\mathbf{R}_2} \end{aligned}$$

Die rekursive Berechnung der Partitionsfunktion erfolgt anhand von Matrizen.

DEFINITION (Matrizen zur Berechnung der Partitionsfunktion)

- Berechnung der Partitionsfunktion für $\mathbf{R}_{all}$:

$$Q = Z_{\mathbf{R}_{all}} = \sum_{R} e^{-[E(R)/(k_B T)]}$$

- Berechnung der Partitionsfunktion für $\mathbf{R}_{i,j}$:

$$Q_{i,j} = Z_{\mathbf{R}_{i,j}} = \sum_{h,l} Q_{i,h-1} \cdot Q^{b}_{h,l} + 1,$$

  wobei $i \leqslant h < l \leqslant j$.

- Berechnung der Partitionsfunktion für $\mathbf{R}^{b}_{i,j}$:

$$\begin{aligned} Q^{b}_{i,j} &= Z_{\mathbf{R}^{b}_{i,j}} = e^{-eH(i,j)/(k_B T)} \\ &+ \sum_{(h,l)} e^{-eSBI(i,j,h,l)/(k_B T)} \cdot Q^{b}_{h,l} \\ &+ \sum_{(h,l)} Q^{b}_{h,l} \cdot Q^{m}_{l+1,j-1} \cdot e^{-(a+b+c(h-i-1))/(k_B T)}, \end{aligned}$$

wobei $(i < h < l < j)$ und $eSBI$ die jeweilige Energie für einen Stack, Interiorloop oder Bulge ist.

- Berechnung der Partitionsfunktion für $\mathbf{R}_{i,j}^{lbd}$:

$$Q_{i,j}^{m} = Z_{\mathbf{R}_{i,j}^{lbd}} = \sum_{h,l} (e^{-[c(h-i-1)/kT]} + Q_{i,h-1}^{b}) \cdot Q_{h,l}^{b} e^{-[(b+c(j-l-1))/(k_B T)]},$$

wobei $(i < h < l < j)$.

In dem Eintrag $Q_{1,n}$ steht das Ergebnis für die gegebene RNA-Sequenz. Die Matrix $Q_{i,j}$ betrachtet alle sekundären RNA-Strukturen in $\mathbf{R}_{i,j}$. Hierfür werden alle RNA-Strukturen beachtet, die zusätzlich ein Basenpaar (h, l) zwischen (i, j) enthalten. Dabei wird das Produkt der Partitionsfunktion für alle RNA-Strukturen, $Q_{i,h-1}$, und der von allen RNA-Strukturen mit Basenpaar (h, l), $Q_{i,j}^{b}$, summiert. Die Matrix $Q_{i,j}^{b}$ summiert alle sekundären RNA-Strukturen mit schließendem Basenpaar (i, j) unter Berücksichtigung deren Energien. Für einen Multiloop wird die einfache Energiefunktion $a + b \cdot k + c \cdot k'$ angenommen. Die Konstante a ist die Energie für das schließende Basenpaar (i, j), b die Energie eines der k Basenpaare innerhalb des Multiloops und c die Energie einer der k' ungepaarten Base innerhalb des Multiloops. Die Matrix $Q_{i,j}^{b}$ entspricht der Partitionsfunktion $Z_{\mathbf{R}_{i,j}^{b}}$ für Strukturen in $\mathbf{R}_{i,j}^{b}$. Die Berechnung der Partitionsfunktion $Z_{\mathbf{R}_{i,j}^{lbd}}$ behandelt den Fall, dass die gegenwärtige sekundäre RNA-Struktur Teil eines Multiloops mit der oben genannten vereinfachten Energiefunktion ist.

Nachdem wir die Partitionsfunktion mit Hilfe der obigen Matrizen effizient berechnen, betrachten wir im Weiteren die Berechnung der Basenpaarwahrscheinlichkeiten. Die Wahrscheinlichkeit eines Basenpaars (i, j) entspricht:

$$p_{i,j} = p_{i,j}^{E} + \sum_{(h,l)} p_{i,j}^{SBI}(h, l) + \sum_{(h,l)} p_{i,j}^{M}(h, l),$$

wobei $h < i < j < l$. Es wird zwischen drei Fällen entschieden. Beim ersten Fall ist das Basenpaar das äußerste Basenpaar, bei dem die Teilsequenzen $[1, i-1]$ und $[j+1, n]$ nur ungepaarte Basen enthalten. Die Wahrscheinlichkeit dafür entspricht $p_{i,j}^{E}$. Der zweite und dritte Fall berücksichtigt Basenpaare, die Teil eines Stacks, Interiorloops, Bulges oder Multiloops sind. Das schließende Basenpaar dieser Strukturen ist (h, l). Die Wahrscheinlichkeiten werden aufgeteilt - Wahrscheinlichkeit für Stack, Bulges und Interiorloops $p_{i,j}^{SBI}(h, l)$ und Wahrscheinlichkeit für Multiloops $p_{i,j}^{M}(h, l)$.

Die drei benötigten Wahrscheinlichkeiten zur Berechnung der Basenpaarwahrscheinlichkeit $p_{i,j}$ greifen auf die Martizen zur Berechnung der Partitionsfunktion zu.

DEFINITION (Berechnung der Basenpaarwahrscheinlichkeit)
Im Folgenden ist (h, l) ein Basenpaar für das gilt $i < h < l < j$.

- Die Wahrscheinlichkeit $p^E_{i,j}$ ist:

$$p^E_{i,j} = \frac{Q_{1,i-1} \cdot Q^b_{i,j} \cdot Q_{j+1,n}}{Q_{1,n}}$$

- Die Wahrscheinlichkeit $p^{SBI}_{i,j}(h,l)$ entspricht:

$$p^{SBI}_{i,j}(h,l) = \frac{p_{h,l} \cdot e^{-eSBI(i,j,h,l)/(k_B T)} \cdot Q^b_{i,j}}{Q^b_{h,l}}$$

- Die Wahrscheinlichkeit $p^M_{i,j}(h,l)$ ist:

$$\begin{aligned} p^M_{i,j}(h,l) &= p_{h,l} \cdot \frac{1}{Q^b_{h,l}} \cdot Q^b_{i,j} \cdot e^{-(a+b)/(k_B T)} \\ &\cdot [e^{-(i-h-1)c/k_B T} \cdot Q^m_{j+1,l-1} \\ &+ Q^m_{h+1,i-1} \cdot e^{-(l-j-1)c/k_B T} \\ &+ Q^m_{h+1,i-1} \cdot Q^m_{j+1,l-1}] \end{aligned}$$

Für die Wahrscheinlichkeit $p^E_{i,j}$ wird die Partitionsfunktion $Q^b_{i,j}$ benötigt, die alle sekundären Strukturen mit Basenpaar (i,j) berücksichtigt. Zusätzlich werden die Partitionsfunktionen $Q_{1,i-1}$ und $Q_{j+1,n}$ der Teilsequenzen $[1, i-1]$ und $[j+1, n]$ verwendet und multipliziert. Das Ganze wird durch die Partitionsfunktion $Q_{1,n}$ aller sekundären RNA-Strukturen geteilt. Um die Wahrscheinlichkeit $p^{SBI}_{i,j}(h,l)$ zu ermitteln, wird das Produkt der Partitionsfunktionen $Q^b_{i,j}$ der sekundären Strukturen mit Basenpaar (i,j) und deren Energie $e^{-eSBI(i,j,h,l)}$ berechnet. Zusätzlich wird das Ergebnis mit der Wahrscheinlichkeit des Basenpaars $p_{h,l}$ multipliziert und durch die Partitionsfunktion $Q^b_{h,l}$ aller sekundären Strukturen mit schließendem Basenpaar (h,l) geteilt. Die letzte Wahrscheinlichkeit $p^M_{i,j}$ ist die Wahrscheinlichkeit, dass (i,j) ein Teil eines Multiloops ist. Das schließende Basenpaar für den Multiloop ist (h,l). Die Wahrscheinlichkeitsberechnung wird in drei Fälle aufgeteilt, die nach der Position des Basenpaars (i,j) bestimmt werden. Entweder ist (i,j) das Basenpaar links außen, rechts außen oder in der Mitte, sprich rechts und links gibt es mindestens noch ein weiteres Basenpaar. Die Partitionsfunktion der entsprechenden sekundären Strukturen stehen in der Matrix Q^m.

Nach der Addition aller Fälle benötigen wir noch die Wahrscheinlichkeit des Basenpaars (h,l). Am Schluss teilen wir die Summe durch die Partitionsfunktion $1/Q^b_{h,l}$ aller sekundären Strukturen mit schließendem Basenpaar (h,l). Wir haben nun eine rekursive Formel für $p_{i,j}$. Durch Gebrauch des dynamischen Programmierprinzips wird dies effizient ermittelt.

Die Laufzeit des Algorithmus von McCaskill beträgt $O(n^4)$ bei der Länge n der gegebenen RNA-Sequenz. Es ist nötig, die ganze Sequenz einmal zudurchlaufen. Dies benötigt lineare Laufzeit. Zusätzlich sehen

wir uns jedes Basenpaar (h, l) mit $i < h < l < j$ an. Dies entspricht auch einem linearen Aufwand. So erhalten wir eine quadratische Laufzeit. Die Berechnung der Matrizen Q benötigen zusätzliche quadratischen Aufwand. So kommen wir zu einer Laufzeit von $O(n^4)$. Durch die Beschränkung der Länge von Loops kann die Laufzeit auf $O(n^3)$ gebracht werden.

Der Speicherplatzbedarf ist quadratisch.

2.3 ZUSAMMENFASSUNG

Sekundäre RNA-Strukturen werden zwischen Hairpins, Stacks, Interiorloops, Bulges und Multiloops unterschieden. Der Algorithmus von Zuker nimmt die Energien der einzelnen sekundären RNA-Strukturelemente und summiert diese auf. Dadurch wird die beste sekundäre RNA-Struktur ermittelt. Dies ist nicht immer die biologisch relevanteste. So benötigen wir ein anderes Kriterium. McCaskill berechnet die Wahrscheinlichkeit für jedes Basenpaar. Damit lassen sich Rückschlüsse auf sekundäre RNA-Strukturen schliessen. Beide Algorithmen werden durch dynamische Programmierung effizient implementiert.

3

ALGEBRAISCHEN DYNAMISCHEN PROGRAMMIERUNG

In diesem Kapitel führen wir das Konzept der algebraischen dynamischen Programmierung (ADP) ein, das eine neue Variante der dynamischen Programmierung ist. Desweiteren stellen wir die Programmierumgebung Bellman's Gap entwickelt von G. Sauthoff vor. Diese erlaubt eine effiziente Lösung von ADP-Problemen und benutzt eine einfache und intuitive Programmiersprache.

3.1 ALGEBRAISCHE DYNAMISCHE PROGRAMMIERUN – EIN NEUES PROGRAMMIERKONZEPT

3.1.1 *Vergleich zur klassischen dynamischen Programmieren*

Die Entwicklung des Prinzips der algebraischen dynamischen Programmierung (ADP) begann 1998. Verwandt ist dieses Prinzip, sowohl mit dem algebraischen Pfad Problem in der Informatik als auch mit der algebraischen Geometrie in der Mathematik. Wir halten uns an die Definitionen und Methoden, vorgestellt von Robert Giegerich und Carsten Meyer [2].

Mit ADP lassen sich kombinatorische Optimierungsprobleme in polynomieller Zeit lösen, indem das Problem rekursiv zerlegt wird und Zwischenergebnisse in Tabellen gespeichert werden. Voraussetzung ist, dass das Problem das Bellman'sche Prinzip der Optimalität erfüllt [1]. Der große Unterschied zur einfachen dynamischen Programmierung (DP) liegt darin, dass ADP zur Lösung des Problems kontext freie Grammatiken, sowie Algebren und Signaturen verwendet. Das Optimalitätsproblem wird durch Anwendung der Algebra auf die entsprechende Grammatik unter Beachtung der gegebenen Signatur gelöst. Dadurch lassen sich einige Nachteile der DP lösen. Ein Überblick liefert die Tabelle 1. Ein entscheidender Vorteil von ADP ist die Möglichkeit zu einer flexibleren Implementierung als in DP.

Im folgenden erklären wir das Prinzip der algebraischen dynamischen Programmierung und die dazu benötigten Elemente.

3.1.2 *Das Prinzip der Algebraischen Dynamischen Programmierung*

DEFINITION (ADP-Problem)
Ein *Problem der algebraischen dynamischen Programmierung* besteht aus folgenden Teilen:

Problem in DP	Lösung in ADP
Unvollständige Scoring-Abstraktion	Evaluation Algebren
Ausgabe nur einer einzelnen Antwort	Ausgabe von Antwortlisten
Indexfehler leicht möglich	Baumgrammatiken
Redundanz des Suchraumes	Verfeinerung der Grammatik
Über-Tabellierung	Annotierte Grammatiken
Backtracking	Pretty printing Algebras
Geringe Mächtigkeit	Produktalgebren

Tabelle 1: Probleme der dynamischen Programmierung (DP) und deren Lösungen in der algebraischen dynamischen Programmierung (ADP)

- einem Alphabet A,
- einer Signatur Σ und
- einer Evaluationsalgebra E.

Das Problem muss das Bellman'sche Prinzip der Optimalität erfüllen.

DEFINITION (Alphabet)
Ein *Alphabet* A ist eine Menge von Symbolen, auf denen es meistens eine alphabetische Ordnung gibt.

Das Alphabet ist die Menge von möglichen Eingabezeichen des ADP-Problems, zum Beispiel für RNA-Sequenzen die Basen G, A, U und C. Zu diesem Alphabet wird eine Signatur definiert.

DEFINITION (Signatur)
Eine *Signatur* Σ ist eine Menge von Funktionsdeklarationen. Hierbei gibt es eine Basismenge B und ein Alphabet A und eine Menge von Funktionsnamen. Die Funktionen nehmen als Parametertypen A oder B. Der Ausgabetyp jeder Funktion ist B.

Eine Signatur Σ kann mit Termen beschrieben werden, die aus Symbolen in A und Funktionsnamen in der Signatur Σ gebildet werden. Die Sprache von Termen für eine Signatur Σ ist T_Σ. Wenn zusätzlich eine Variablenmenge V als Argumente von Termen zulässig ist, dann ist die Sprache mit $T_\Sigma(V)$ definiert. Terme können als Bäume gesehen werden. Dabei enthalten die Knoten der Bäume die Operatoren und die Blätter der Bäume enthalten die Eingabeargumente aus A. Der Baum wird generiert, indem der äußerste Operator die Wurzel ist und dessen Eingabeargument die Kinder sind. Mit einer Baumgrammatik werden Terme formal beschrieben. In ADP werden reguläre Baumgrammatiken verwendet.

DEFINITION (Reguläre Baumgrammatik)
Eine *reguläre Baumgrammatik* G enthält:

- eine Menge von Nichtterminalsymbole V,
- eine Menge von Terminalsymbole A,
- ein Axiom $s \in V$ und
- eine Menge von Produktionen $v \to t$, wobei $v \in V$ und $t \in T_\Sigma(V)$ sind.

Die Lösungsmenge eines ADP-Problems wird durch die Sprache der dazugehörigen Baumgrammatiken beschrieben. Die folgende für eine Baumgrammatik G ist:

$$L(G) = \{g \in T_\Sigma | s \Rightarrow^* g\}.$$

Zuletzt wird eine Evaluationsalgebra E verwendet um das Verhalten der Funktion, die in der Signatur Σ deklariert sind, zu beschreiben.

DEFINITION (Evaluationsalgebra)
Sei Σ die gegebene Signatur. Eine *Evaluationsalgebra* E besitzt:

- den Definitionsbereich der Werte für die Basismenge B in Σ,
- jeweils eine Funktion f mit entsprechenden Ausgabetypen und Parametertypen, wie in Σ beschrieben, und
- mindestens eine Auswahlfunktion $h : [B] \to [B]$, wobei $[B]$ eine Liste von Elementen in B ist. Diese wählt die geeigneten Kandidaten aus.

Die Abbildung 3 stellt die Rolle von Signatur, reguläre Baumgrammatik und Evaluationsalgebra da.

DEFINITION (Bellman'sches Prinzip der Optimalität) [2]
Eine Evaluationsalgebra E mit Signatur Σ erfüllt das *Bellman'sche Prinzip der Optimalität*, falls

1. $\forall f \in \Sigma : h[f(x_1, ..., x_n)|x_1 \leftarrow X_1, ..., x_n \leftarrow X_n] = h[f(x_1, ..., x_n|x_1 \leftarrow h(X_1), ..., x_n \leftarrow h(X_n)]$,
   wobei jedes X_i $(1 \leqslant i \leqslant n)$ eine Liste über den Wertebereich von E.
2. $\forall (Y, Z) : h[Y + +Z] = h(h(Y) + +h(Z))$,
   wobei Y und Z Liste über den Wertebereich von E sind und die Verkettung der gegebenen Listen mit $++$ angegeben ist.

Wie oben beschrieben kann die Ableitung der resultierenden Kandidaten einer Baumgrammatik durch Bäume beschrieben werden. Wenn wir die Zeichen an den Blättern zusammensetzen, bekommen wir die Eingabesequenz, dessen Ableitung anhand der Baumgrammatik den entsprechenden Baum generiert. Die Sequenz aus den zusammengesetzten Zeichen, die an den Blättern stehen, wird Yield genannt. Durch Parsing des Yield wird die Kandidatenmenge von der Eingabesequenz konstruiert.

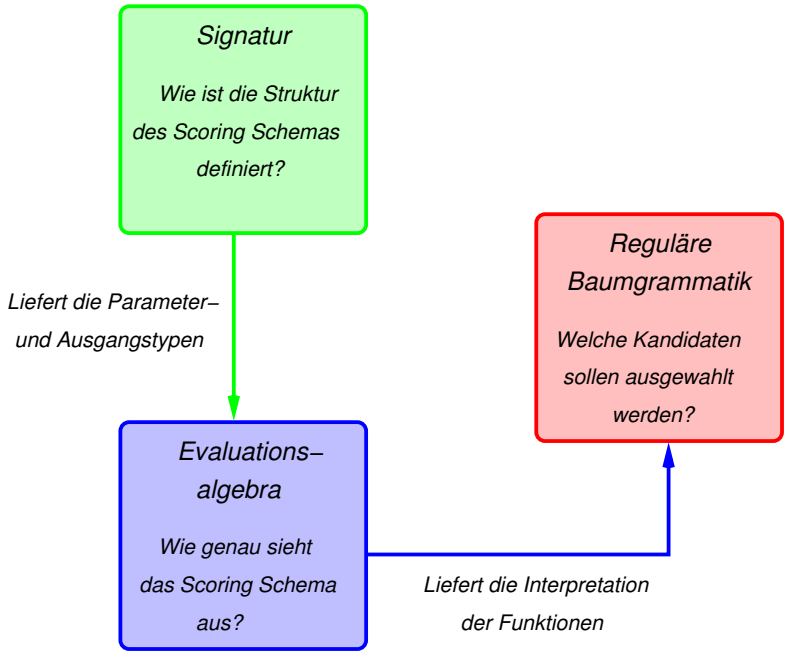


Abbildung 3: Die einzelnen Komponenten eines ADP-Problems: Grammatik, Algebra und Signatur

DEFINITION (Yield Sprache)
Die *Yield-Sprache* einer Baumgrammatik G ist:

$$L_y(G) = \{yield(t) | t \in T_\Sigma\}$$

Die Grammatik, die die Yield Sprache beschreibt, ist eine kontext-freie Grammatik.

DEFINITION (Lösung eines ADP-Problems)
Sei G die Yieldgrammatik, E die Evaluationsalgebra und w die Eingabe. Dann ist die *Lösung eines ADP-Problems* gegeben durch:

$$G(E, w) = h[\{E(w) | yield(t) = w, t \in L_y(G)\}]$$

3.2 BELLMAN'S GAP

Das Programm Bellman's Gap [10] ist von G. Sauthoff entwickelt worden. Es ermöglicht dem Programmierer eine Implementation eines ADP-Programms. Die Beschreibung von Signatur, Algebra und Grammatik ist einfach und intuitiv zu verstehen.

3.2.1 *Überblick*

Bellman's Gap besteht aus drei Teilen, den GAP-L, GAP-C und GAP-M. GAP-L ist die Programmiersprache, deren Syntax ähnlich der von Java/C ist. Es liefert dabei eine Implementation der APD Konzepte,

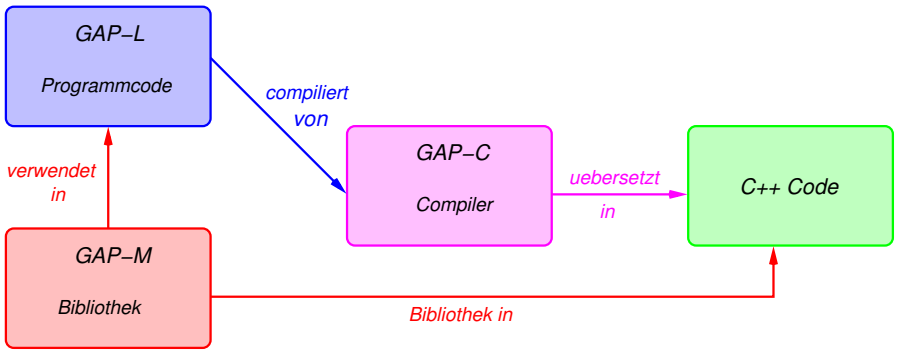


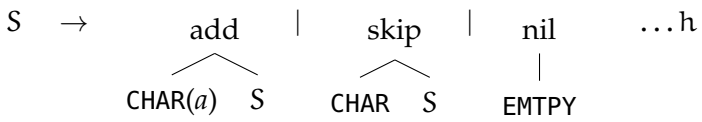
Abbildung 4: Die Funktionsweise von Bellman's Gap mit Hilfe von GAP-M, GAP-C und GAP-L

wie Alphabet, Signatur, Algebra, Baumgrammatik. Der Compiler GAP-C übersetzt den Programmiercode in einen effizienten C++-Code. GAP-M ist eine Bibliothek für das kompilierte GAP-L Programm. Diese liefert Datenstrukturen und eine Ansammlung von Funktionen für das GAP-L Programm, wie den Datentyp einer Base, die in einer RNA-Sequenz vorkommen kann, oder Energiefunktionen für die Vorhersage von sekundären RNA Strukturen. Eine Übersicht davon liefert Abbildung 4.

3.2.2 *Kurze Einführung in die Programmiersprache*

Die Implementation von Signatur, Grammatik und Algebren ist intuitiv. Anhand eines kleinen Beispiels geben wir einen Einblick in die Programmiersprache. Die Aufgabe ist: Berechne die Anzahl des Zeichens a in einem Wort. Für diese Aufgabe enthält das Alphabet alle Buchstaben des Alphabets der deutschen Sprache ausgenommen der Umlaute und dem Scharf-S, sprich $\texttt{Alphabet} = \{a, b, c, ..., z\}$.

Betrachten wir zuerst die Grammatik für dieses Problem. Der Charakter an der aktuellen Position ist entweder ein a oder ein anderer Charakter in A. Im Folgenden werden Charakter in der Grammatik mit `CHAR` bezeichnet. Der Charakter a wird durch $\texttt{CHAR}(a)$ ausgewählt. Zusätzlich ist die leere Sequenz `EMTPY` von Charakteren möglich. Wir erhalten folgende Grammatik:



Wenn ein a gesehen wird, führen wir die Funktion `add` aus, bei einem anderen Charakter `skip` und bei einer leeren Sequenz `nil`. Auf die Werte, die von der Grammatikregel berechnet werden, wird die Auswahlfunktion h angewandt.

Wir benennen die Grammatik `Gram`. Angenommen die dazugehörige Signatur zu `Gram` sei `Sign`. Dann wird `Gram`, wie folgt, in Bellman's Gap implementiert:

```
grammar Gram uses Sign (axiom = S){
  S = add( CHAR(a), S ) | skip( CHAR, S ) | nil( EMPTY ) #h;
}
```

Die dazugehörige Signatur `Sign` zu `Gram` definiert die Ausgabe- und Parametertypen der benötigten Funktionen mit Hilfe von `Alphabet` und der Basismenge `Basis`. Sowohl die Funktion `add` als auch die Funktion `skip` nimmt ein Element aus `Alphabet` und eines aus `Basis` und gibt ein Element aus `Basis` zurück. Die Funktion `nil` nimmt eine leere Sequenz als Eingabe. Dieser Eingabetyp ist mit `void` definiert. Der Ausgabetyp von `nil` ist ein Element aus `Basis`. Die Auswahlfunktion nimmt immer als Parameter ein Liste von Elementen aus `Basis` und gibt den gleichen Typ zurück. `Sign` ist folgendermaßen implementiert:

```
signature Sign (Alphabet, Basis){
  Basis add( Alphabet, Basis );
  Basis skip( Alphabet, Basis );
  Basis nil( void );

  choice [Basis] h([Basis]);
}
```

Als letztes wird eine Algebra benötigt. Jede Funktion in der Grammatik braucht eine Interpretation. Die Parameter- und Ausgabetypen, in der Signatur mit `Alphabet` und `Basis` beschrieben, sind in der Algebra Characters und Integers, sprich `char` und `int`. Die Funktion `add` zählt eins zur schon berechnenden Anzahl hinzu. Bei der Funktion `skip` wird eine Charakter, der nicht a ist, eingelesen und dabei wird nur die schon berechnende Anzahl von a in dem Eingabewort zurückgegeben. Die Funktion `nil` betrachtet den Fall, wenn eine leere Sequenz gegeben ist. Dies ist der elementarste Fall, bei dem Null zurückgegeben wird. Zum Schluß wird noch die Auswahlfunktion interpretiert. Diese zählt hier die Ergebnisse der einzelnen Werte in der Liste für die Charakters zusammen. Außerdem wird noch angegeben, welche Signatur diese Algebra implementiert, und welchen Namen die Algebra hat. Die Signatur ist hier `Sign` und der Name `count`. Die Algebra sieht, wie folgt, aus:

```
algebra count implements Sign (Alphabet = char, Basis = int)
{
  int add( char a, int i ){
    return i+1;
  }
```

```
  int skip( char a, int i ){
    return i;
  }

  int nil(void){
    return 0;
  }

  choice [int] h( [int] l ){
    return list( sum(l) );
  }
}
```

Es ist möglich in einem Programm mehrere Algebren zu definieren und diese, wie von P. Steffen und R. Giegerich [11] vorgestellt, zu kombinieren.

3.2.3 *Im Vergleich mit manueller Implementierung*

Im Folgenden gehen wir auf die Vorteile und Besonderheiten von Bellman's Gap ein. Im Gegensatz zur manuellen Implementierung eines DP-Problems direkt in C++ hat Bellman's Gap viele Vorteile. Einige davon sind folgende:

- Eine Algebra kann andere Algebren erweitern.
- Es können mehrere Instanzen für ein Programm existieren. Dabei bezeichnet eine Instanz folgendes: Für ein Programm gibt es mehre Algebren, zum Beispiel seien diese x und y, die jeweils eine andere Interpretation der Funktionen haben. Die eine Instanz ist die Ausführung der Grammatik mit der Interpretation durch x und die andere Instanz mit der Interpretation durch y. Diese beiden Instanzen sind in demselben Programm möglich.
- Algebren können mit einander kombiniert werden. Dies wird auch als Produkt von Algebren bezeichnet.
- Die Menge der Kandidaten kann durch syntaktische oder semantische Filter innerhalb der Grammatik eingeschränkt werden. Auch bei Produkte von Algebren sind semantische Filter möglich.
- Parsers der Baumgrammatik kann ein oder mehrere Eingabesequenzen verarbeiten. Ein Beispiel für zwei Eingabesequenzen ist die Berechnung des Aligments der Sequenzen.

Zusätzlich liefert Bellman's GAP eine Möglichkeit, die Eingabe nur auf die RNA-Basen A, C, G und U zu beschränken. Dadurch ist keine Überprüfung der Eingabesequenz auf ihre Korrektheit nötig.

Bellman's Gap ist noch eine sehr neu entwickelte Programmierumgebung. Dies hat sowohl Vorteile wie auch Nachteile. Die Vorteile sind oben beschrieben. Der Nachteil, wie bei allen neuen Programmen, ist, dass es noch fehleranfällig ist und die Programmiersprache den Umfang gängiger Programmiersprachen, wie C++ oder Java, nicht erreicht. Viele gängige Methoden sind nicht anzuwenden, wie das Ausgeben von Variablenwerten während der Laufzeit des Programms. Dies macht vor allem das Debuggen sehr zeitaufwändig. Aber im Gegensatz zu anderen Programmen, die mit ADP umgehen können, ist Bellman's Gap klar im Vorteil.

3.3 ZUSAMMENFASSUNG

Wie wir gezeigt haben, bietet die algebraische dynamische Programmierung modularere Lösungen für kombinatorische Optimierungsprobleme als die klassische dynamische Programmierung an. Die Grundlage in ADP ist eine reguläre Baumgrammatik neben einer oder mehreren Algebren, sowie deren Signatur. Damit es möglich ist ein ADP-Problem zu lösen, ist es notwendig, dass die Evaluationsalgebra das Bellman'sche Prinzip der Optimalität erfüllt. Eine neu entwickelte Programmierumgebung zum Lösen von ADP-Problemen ist Bellman's Gap, das sich aus drei Bestandteilen - GAP-M, GAP-C und GAP-L, zusammensetzt. GAP-M bietet eine Reihe von Bibliotheken, GAP-C einen Compiler und GAP-L eine Programmiersprache mit ähnlicher Syntax von Java/C. Die Vorteile von Bellman's Gap im Gegensatz zu anderen ADP-Programmierumgebungen überwiegen.

Teil III

RNA-PARTITIONSFUNKTION

4

DIE RNA-PARTITIONSFUNKTION VON TEILSEQUENZEN

G. Sauthoff stellt ein Programm zur Berechnung der Partitionsfunktion von Teilsequenzen zur Verfügung. Sei s eine RNA-Sequenz und n deren Länge. Das Programm ermittelt für jede RNA-Teilsequenz $[i, j]$ unter Berücksichtigung der Energien einer sekundären RNA-Struktur die Partitionsfunktion, wobei $1 \leqslant i < j \leqslant n$. Hierbei gilt, dass kein Basenpaar (i', j') existiert, das ein schließendes Basenpaar einer sekundären RNA-Struktur ist, und bei dem $(1 \leqslant i' < i < j < j' <\leqslant n)$ gilt. Solche sekundären RNA-Strukturen der Teilsequenz $[i, j]$ werden im Weiteren durch $\mathit{inner}(i, j)$ beschrieben.

In den nächsten Abschnitten erklären wir die Grammatik zur Bestimmung der Partitionsfunktion von Teilsequenzen detailliert.

4.1 GRAMMATIK ZUR BERECHNUNG DER PARTITIONSFUNKTION

Wie in Abschnitt 2.2 besprochen, brauchen wir Strukturmengen, die unabhängig zerlegbar sind, um die Partitionsfunktion nach McCaskill zu berechnen. Die entsprechenden Strukturmengen sind hier $\mathbf{R}_{\mathit{inner}}$, $\mathbf{R}_{\mathit{inner}(i,j)}$, $\mathbf{R}^{b}_{\mathit{inner}(i,j)}$ und $\mathbf{R}^{l\,b\,d}_{\mathit{inner}(i,j)}$.

Für die Berechnung von $Z_{\mathbf{R}_{\mathit{inner}}}$ betrachten wir alle RNA-Strukturen, die durch die gegebene RNA-Sequenz s entstehen und in der Strukturmenge $\mathbf{R}_{\mathit{inner}}$ enthalten sind. Grundsätzlich gibt es zwei verschiedene Strukturen - eine RNA-Sequenz mit nur ungepaarten Basen oder eine RNA-Sequenz mit einer sekundären RNA-Struktur mit schließendem Basenpaar (h, l), wobei $i \leqslant h < l \leqslant j$ gilt. Die Partitionsfunktion $Z_{\mathbf{R}_{\mathit{inner}(i,j)}}$ berechnet sich folgendermaßen:

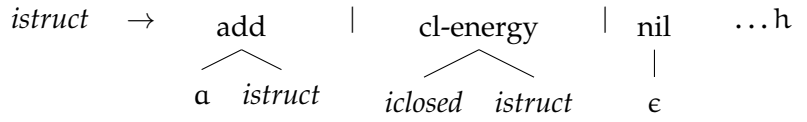
$$Z_{\mathbf{R}_{\mathit{inner}(i,j)}} = \sum_{h,l} 1 + Z_{\mathbf{R}_{\mathit{inner}(h,l)}} \cdot Z_{\mathbf{R}^{b}_{\mathit{inner}(l+1,j)}},$$

wobei $1 \leqslant i \leqslant h < l \leqslant j \leqslant n$.

Die Partitionsfunktion $Z_{\mathbf{R}_{\mathit{inner}(i,j)}}$ wird in der Grammatik durch die Nichtterminale *istruct* beschrieben. Für die Nichtterminale *istruct* benötigen wir eine Grammatikregel, die es erlaubt, ungepaarte Basen hinzu zu fügen. Zusätzlich ist auch die leere Teilsequenz als elementarster Fall zu berücksichtigen. Die Partitionsfunktion einer Sequenz von nur ungepaarten Basen oder einer leeren Sequenz ist 1. Auch die Möglichkeit einer sekundären RNA-Struktur mit schließendem Basenpaar $(l + 1, j)$ ist zu beachten, deren Partitionsfunktion $Z_{\mathbf{R}^{b}_{\mathit{inner}(l+1,j)}}$ ist. Diese Partitititionsfunktion wird durch die Nichtterminale *iclosed* berechnet.

Außer bei der Hinzunahme einer leeren Teilsequenz erfolgt außerdem ein rekursiver Zugriff auf die schon berechnete Partitonsfunktion der Teilsequenz $[h, l]$, die in der Nichtterminalen *istruct* gespeichert ist.

Dies führt zu folgender Grammatikregel, die die Partitionsfunktion von $Z_{[\mathbf{R}]_{inner(i,j)}}$ ermittelt:



Die Funktion `add` fügt eine Base hinzu und betrachtet die restliche einzulesende Sequenz. Hingegen nimmt die Funktion `cl-energy` eine sekundäre Struktur, die in der Nichtterminalen *iclosed* beschrieben wird, und die Teilsequenz. Die leere Sequenz wird durch ϵ dargestellt, auf die die Funktion `nil` angewandt wird. Die Auswahlfunktion h wird auf die berechneten Werte der Grammatik angewendet, wobei h die berechneten Werte aufsummiert.

Die Nichtterminale *iclosed* erfasst alle sekundären Strukturelemente der Teilsequenz, in denen die äußersten Basen ein Basenpaar bilden. Diese Strukturelemente sind in der Strukturmenge $\mathbf{R}^b_{inner(i,j)}$ enthalten, deren Partitionsfunktion $Z_{\mathbf{R}^b_{inner(i,j)}}$ ist.

Als Nächstes widmen wir unsere Aufmerksamkeit der rekursiven Berechnung der Partitionsfunktion $Z_{\mathbf{R}^b_{inner(i,j)}}$ mit Hilfe von Grammatikregeln, sprich:

$$\begin{aligned} \textit{iclosed} = Z_{\mathbf{R}^b_{inner(i,j)}} =& e^{-eH(i,j)/(k_B T)} \\ &+ e^{-eS(i,j,i+1,j-1)} \cdot Z_{\mathbf{R}^b_{inner(i+1,j-1)}} \\ &+ \sum_{h,l} e^{-eI(i,j,h,l)} \cdot Z_{\mathbf{R}^b_{inner(h,l)}} \\ &+ \Big(\sum_{h,l} Z_{\mathbf{R}^b_{inner(h,l)}} \cdot Z_{\mathbf{R}^{lbd}_{inner(l+1,j-1)}} \\ &\cdot e^{-(a+b+c(h-i-1)/(k_B T))} \Big). \end{aligned}$$

Der erste Summand betrachtet einen Hairpin, der zweite einen Stack, der dritte Interiorloops inklusive den Bulges und der letzte Multiloops.

Die dazugehörige Grammatikregel ist folgende:

$$\textit{iclosed} \rightarrow \textit{ihairpin} \mid \textit{istack} \mid \textit{iloop} \mid \textit{ileftB} \mid \textit{irightB} \mid \textit{imultiloop} \quad \ldots h$$

Hier bezeichnen *ihairpin*, *istack*, *iloop*, *ileftB*, *irightB* und *imultiloop* die sekundären RNA-Strukturen. Auch hier wird die Auswahlfunktion h

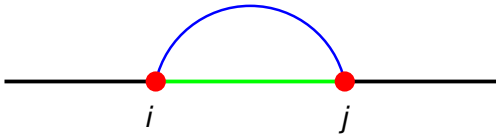


Abbildung 5: Hairpin

auf die berechneten Werte der Grammatik angewendet, wobei h die berechneten Werte aufsummiert.

Im Weiteren ist die Auswahlfunktion h die Summe aller Werte, die durch die Grammatikregel berechnet werden.

4.1.1 *Hairpin*

Der einfachste Fall einer sekundären Struktur ist der Hairpin, der in Abbildung 5 veranschaulicht ist. Die Bedingung ist ein Basenpaar (i, j) und eine Region von ungepaarten Basen zwischen dem Basenpaar. In Abbildung 5 wird diese Region durch eine grüne Linie gekennzeichnet. Um nur biologisch relevante Hairpins zu berücksichtigen, wird verlangt, dass ein Hairpin mindestens drei ungepaarten Basen enthält. Diese Bedingung wird in der Grammatik durch einen Filter gewährleistet.

Für einen Hairpin brauchen wir nur die Energie zu berechnen, um die Partitionsfunktion zu bestimmen, sprich:

$$\mathit{ihairpin} = e^{-eH(i,j)/(k_B T)}$$

Durch folgende Grammatikregel wird diese Berechnung sichergestellt:

$$\mathit{ihairpin} \quad \rightarrow \quad \underset{\alpha \quad \texttt{REGION} \quad \hat{\alpha}}{\text{hl-energy}} \quad \text{with length}(\texttt{REGION}) \geqslant 3$$

Das Basenpaar wird durch α und $\hat{\alpha}$ beschrieben und die Region von ungepaarten Basen durch `REGION`. Die Funktion `hl-energy` berechnet anhand des gegebenen Basenpaars und der Region von ungepaarten Basen die Energie des Hairpins. Zusätzlich garantiert der Filter, dass nur Hairpins betrachtet werden, die mindestens drei ungepaarte Basen enthalten.

Da kein weiteres Basenpaar für einen Hairpin benötigt wird, reicht es aus, nur die Energie eines Hairpins zu berechnen.

4.1.2 *Stack*

Bei einem Stack sind zwei Basenpaare vorhanden, für die gilt, dass sie direkt aufeinander folgen, sprich ist das Basenpaar (i, j) zu sehen,

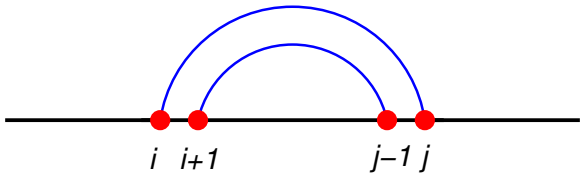


Abbildung 6: Stack

dann auch das Basenpaar $(i+1, j-1)$. Abbildung 6 stellt einen Stack dar. Um dessen Partitionsfunktion zu bestimmen, wird sowohl die jeweilige Energie des Stacks als auch die Partitionsfunktion $Z^{b}_{\mathbf{R}_{inner}(i+1,j-1)}$ benötigt. Die Partitionsfunktion eines Stacks wird dann folgendermaßen ermittelt:

$$istack = e^{-eS(i,j,i+1,j-1)} \cdot Z^{b}_{\mathbf{R}_{inner}(i+1,j-1)}$$

Die entsprechende Grammatikregel ist gegeben durch:

$$istack \rightarrow \text{sr-energy}\ a\ iclosed\ \hat{a}$$

Die Energie des Stacks wird mit der Funktion `sr-energy` ermittelt. Das Basenpaar (i, j) ist durch a und $\hat{a}$ gegeben. Die Nichtterminale *iclosed* beschreibt die Partitionsfunktion der sekundären RNA-Strukturen, bei denen die äußeren Basen miteinander gebunden sind. Dadurch ist garantiert, dass auch das Basenpaar $(i+1, j-1)$ existiert.

4.1.3 *Interiorloop und Bulge*

Es gibt drei Varianten von Interiorloops - den einfachen Interiorloop, den linken Bulge und den rechten Bulge. Bei dem einfachen Interiorloop existiert neben den Basenpaar (i, j) ein zweites Basenpaar (h, l) mit $i+1 < h < l < j-1$. Sowohl zwischen den Basen i und k als auch zwischen den Basen l und j befindet sich eine Basenregion mit mindestens einer ungepaarten Base. Abbildung 7 zeigt einen einfachen Interiorloop, wobei die ungepaarte Basenregion sowohl zwischen i und k als auch zwischen l und j grün dargestellt ist.

Die Partitionsfunktion für einen einfachen Interiorloop ist folgende:

$$iloop = \sum_{h,l} e^{-eI(i,j,h,l)} \cdot Z^{b}_{\mathbf{R}_{inner}(h,l)}$$

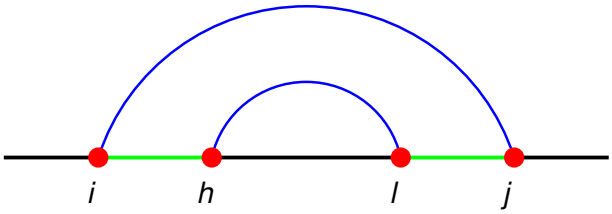
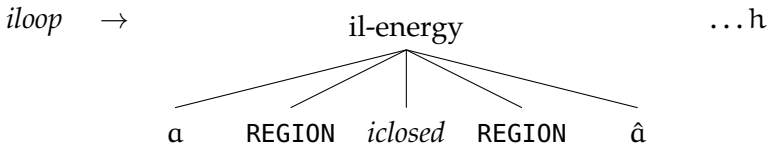


Abbildung 7: Interiorloop

wobei $i+1 < k < l < j-1$. Diese Berechnung ist durch die nachfolgende Grammatikregel beschrieben:



Ein Interiorloop wird aufgeteilt in das Basenpaar (i,j), dargestellt durch a und $\hat{a}$, in zwei ungepaarte Basenregionen `REGION` und irgendeine andere sekundäre Struktur *iclosed*, bei der das äußerste Basenpaar gebunden ist. Die Funktion `il-energy` in der Regel multipliziert die Energie des Interiorloops mit der Partitionsfunktion beschrieben durch *iclosed*.

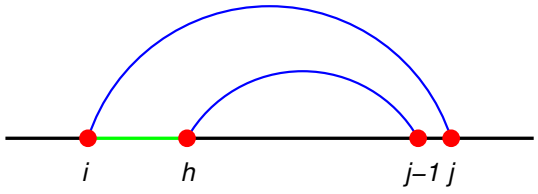
Als Nächstes sind die Spezialfälle des Interiorloops zu betrachten. Zuerst beschäftigen wir uns mit dem linken Bulge und danach mit dem rechten Bulge. Beim linken Bulge gilt, dass das Basenpaar (i,j) und das Basenpaar $(h,j-1)$ existieren. Zwischen i und h befindet sich eine Region von ungepaarten Basen, die mindestens eine Base besitzt. Abbildung 8a zeigt einen linken Bulge. Auch hier ist die Region von ungepaarten Basen zwischen i und k grün dargestellt.

Der rechte Bulge enthält eine Region von ungepaarten Basen zwischen l und j und zwei Basenpaare (i,j) und $(i+1,l)$. In Abbildung 8b ist der rechter Bulge dargestellt. Ungepaarte Regionen bei den Bulges werden durch eine grüne Linie markiert.

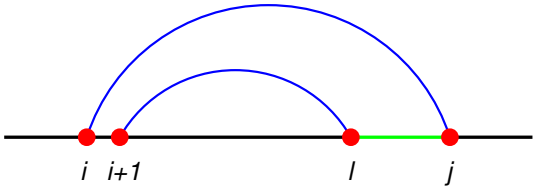
Partitionsfunktionen von Bulges werden anhand derselben Formel wie für einfache Interiorloops berechnet:

$$\mathit{ileftB} = \sum_{h} e^{-eI(i,j,h,j-1)} \cdot Z_{\mathbf{R}^b_{\mathit{inner}(h,j-1)}},$$
$$\mathit{irightB} = \sum_{l} e^{-eI(i,j,i+1,l)} \cdot Z_{\mathbf{R}^b_{\mathit{inner}(i+1,l)}}.$$

Der einzige Unterschied ist, dass für einen linken Bulge $l = j-1$ und $i+1 < h < j-1$ oder für einen rechten Bulge $h = i+1$ und $i+1 < l < j-1$ gilt.



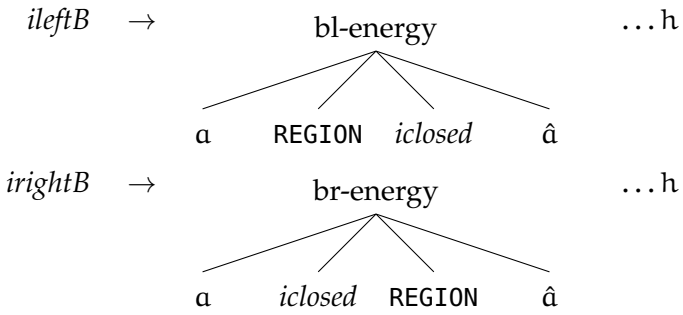
(a) Linker Bulge



(b) Rechter Bulge

Abbildung 8: Linker Bulge (a), rechter Bulge (b)

Die Ermittlung der Partitionsfunktionen für Bulges ist durch die folgende Grammatikregel bewerkstelligt:



Die erste Regel generiert einen linken Bulge und die zweite einen rechten Bulge. Auch hier wird die sekundäre RNA-Struktur - die Bulges - in Komponenten zerlegt. Diese sind das Basenpaar (i, j) gegeben durch a und $\hat{a}$, die Region der ungepaarten Basen `REGION` und der sekundären RNA-Struktur mit dem Basenpaar $(h, j-1)$ für den linken Bulge und mit Basenpaar $(i+1, l)$ für den rechten Bulge. Die Energiefunktionen sind `bl-energy` für den linken Bulge und `br-energy` für den rechten Bulge. Zusätzlich wird rekursiv auf den Eintrag $Z^{b}_{\mathbf{R}_{inner(h,l)}}$, gespeichert in *iclosed*, zugegriffen.

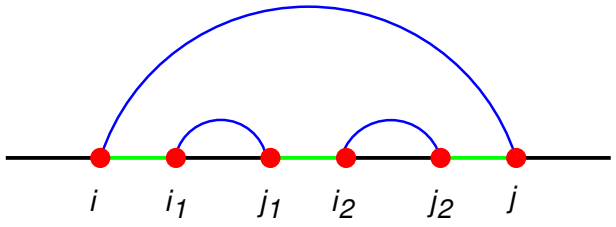


Abbildung 9: Multiloop

4.1.4 *Multiloop*

Als letztes untersuchen wir die Grammatikregeln für die Partitionsfunktion eines Multiloops. Abbildung 9 stellt einen Multiloop dar. Die Partitionsfunktion eines Multiloops lautet:

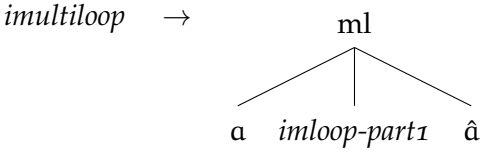
$$imultiloop = \sum_{h,l} Z_{\mathbf{R}^{b}_{inner(h,l)}} \cdot Z_{\mathbf{R}^{1bd}_{inner(l+1,j-1)}} \cdot e^{-(a+b+c(h-i-1)/(k_B T))}$$

Der Multiloop wird nach dem ersten Basenpaar (h, l) innerhalb des Multiloops geteilt, dessen Partitionsfunktion $Z_{\mathbf{R}^{b}_{inner(h,l)}}$ ist. Der Rest des Multiloops ist in der Strukturmenge $\mathbf{R}^{1bd}_{l+1,j-1}$ zu finden, die garantiert, dass es noch mindestens ein Basenpaar innerhalb des Multiloops gibt. Als Letztes wird noch die freie Energie des Multiloops mit schließendem Basenpaar (i, j) benötigt. Hierfür nehmen wir folgende einfache Energiefunktion an:

$$a + b \cdot k + c \cdot k'.$$

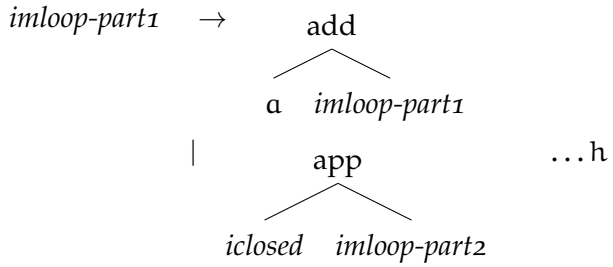
Die Konstante a ist die Energie des schließenden Basenpaars, b die Energie eines Basenpaars innerhalb des Multiloops und c die Energie einer ungepaarten Base innerhalb des Multiloops. Die Anzahl der Basenpaare innerhalb des Multiloops ist k und die Anzahl der ungepaarten Basen k'.

Auch innerhalb der Grammatik teilen wir den Multiloop stückchenweise auf. In der ersten Regel gehen wir sogar noch etwas einfacher vor. In dieser gehen wir von dem schließenden Basenpaar a und $\hat{a}$ aus und definieren den inneren Teil eines Multiloops nicht exakt, sondern ersetzen ihn durch eine Nichtterminale, genannt *imloop-part1*. Die erste Regel ist daher:



Die Funktion `ml` berechnet die Partitionsfunktion anhand des schließenden Basenpaars und den Multiloopteil geschlossen durch a und $\hat{a}$.

Die Nichtterminale *imloop-part1* dient als Platzhalter für einen Multiloopteil mit mindestens zwei Basenpaaren. Wie auch bei der Partitionsfunktion *imultiloop*, die oben beschrieben wurde, wird dafür den Multiloop in zwei Teile geteilt. Die Teilung geschieht genau nach dem ersten vorkommenden Basenpaar. Die nachfolgende Grammtikregel beschreibt diese Teilung:



Der erste Teil des Multiloops ist *imloop-part1*, der mindestens zwei Basenpaare enthält. Es gibt genau zwei Möglichkeiten für diesen Multiloopteil. Beim ersten Fall ist die aktuelle Base a ungepaart und der noch zu betrachtende Teil enthält noch mindestens zwei Basenpaare. Der zweite Fall besteht darin, dass die aktuelle Base ein Teil einer sekundären RNA-Struktur innerhalb des Multiloops ist. Diese sekundäre RNA-Struktur ist in *iclosed* beschrieben. Für den Rest des Multiloops gilt, dass nur noch ein weiteres Basenpaar folgen muss. Dies wird mit Hilfe der Nichtterminalen *imloop-part2* beschrieben. Die Funktion `add` addiert eine ungepaarte Base hinzu und die Funktion `app` eine sekundäre RNA-Struktur.

Dass Teil des Multiloops, dargestellt durch *imloop-part1*, mindestens zwei sekundäre RNA-Strukturen enthält, wird gewährleistet, indem wir solange rekursiv auf die Nichtterminale zugreifen, bis die erste RNA-Struktur vorkommt. Sobald eine sekundäre RNA-Struktur gesehen wird, gehen wir zu der Nichtterminalen *imloop-part2*, in der mindestens eine weitere sekundäre RNA-Struktur vorkommt.

Die Nichtterminale *imloop-part2* entspricht folgender Partitionsfunktion:

$$\textit{imloop-part2} = Z_{\mathbf{R}^{lbd}_{\textit{inner}(l+1,j-1)}}$$

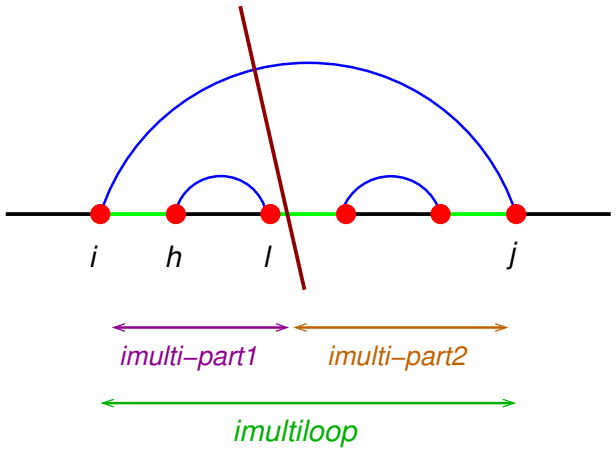
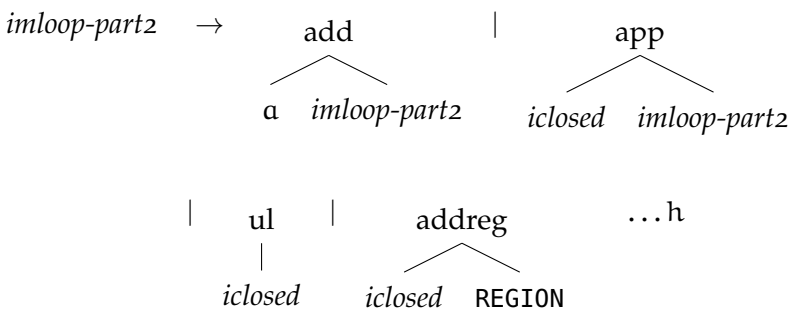


Abbildung 10: Teilung des Multiloops in die Grammatikregeln *imultiloop*, *imloop-part1* und *imloop-part2*

Das heißt, dass die Teilsequenz $[l+1, j-1]$ noch mindestens ein Basenpaar enthält. Die folgende Grammatik entspricht dieser Partitionsfunktion:



Hier werden vier verschiedene Fälle betrachtet. Im ersten Fall ist die aktuell betrachtete Base `a` ungepaart. Für den Rest des Multiloops gilt, dass dieser noch mindestens ein Basenpaar enthält. Dies wird durch die Nichtterminale *imloop-part2* beschrieben. Die Funktion, die die beiden Teile zusammenfügt, ist `add`. Im zweiten Fall nimmt die Funktion `app` eine sekundäre RNA-Struktur und einen Multiloopteil, in dem noch mindestens ein Basenpaar enthalten ist. Der dritte Fall ist, dass der Teil nur eine sekundäre RNA-Struktur, beschrieben in *iclosed*, enthält, deren Partitionsfunktion mit `ul` berechnet wird. Der letzte Fall betrachtet eine sekundäre RNA-Struktur *iclosed* und eine folgende Region von ungepaarten Basen, dargestellt durch `REGION`. Die Funktion `addreg` ermittelt dafür die Partitionsfunktion.

Das Kriterium für die Beendigung des Multiloopteils ist erst dann gegeben, wenn eine sekundäre RNA-Struktur aufgetreten ist. Dies garantiert, dass mindestens eine sekundäre RNA-Struktur in dem Teil des Multiloops, bezeichnet mit *imloop-part2*, enthalten ist.

Abbildung 10 veranschaulicht noch mal die Aufteilung eines Multiloops durch die Grammatikregeln, wobei ungepaarte Basenregionen grün dargestellt sind und die Teilung anhand der dunkelroten Linie geschieht.

4.2 ZUSAMMENFASSUNG

Die Partitionsfunktion für Teilsequenzen $[i, j]$ kann mit Hilfe der in diesem Kapitel beschriebenen Grammatikregeln berechnet werden. Diese Berechnung betrachtet nur Teilsequenzen, die kein Basenpaar (i', j') enthalten mit $1 \leqslant i' < i < j < j' \leqslant n$. Dafür wird die Teilsequenz in Strukturmengen aufgeteilt. Die Strukturmenge $Z_{\mathbf{R}^b_{inner(i,j)}}$ enthält alle sekundären Strukturen mit schließendem Basenpaar i und j, sprich Hairpins, Interiorloops, Bulges und Multiloops. Die Berechnung der Partitionsfunktion dieser Strukturmenge erfolgt durch die Berechnung der Partitionsfunktion der einzelnen sekundären RNA-Strukturen - Hairpin, Interiorloop, Bulge und Multiloop.

5

BERECHNUNG DER PARTITIONSFUNKTION VON RNA-SEQUENZ

In Kapitel 4 haben wir uns mit der Berechnung der Partitionsfunktion von Teilsequenzen $[i,j]$ einer Sequenz s beschäftigt. Dabei berücksichtigten wir nur Strukturen, die kein Basenpaar (i',j') mit $1 \leqslant i' < i < j < j' \leqslant n$ besitzen. Zur Berechnung der Partitionsfunktion $\mathbf{Z}$ der kompletten Sequenz s werden auch diese Strukturen benötigt. Die Frage ist nun, welche konkreten Strukturen müssen noch berücksichtigt werden, um die korrekte Berechnung der Partitionsfunktion von s zu gewährleisten.

Im Weiteren sei s eine RNA-Sequenz der Länge n.

5.1 VON TEILSEQUENZEN ZU VOLLSTÄNDIGEN SEQUENZEN

Die Berechnung der Partitionsfunktion $\mathbf{Z}$ für s geschieht anhand folgender Vorschrift:

$$\mathbf{Z}(s) = \sum_{R} \exp^{-E(R)/(k_B T)} = \sum_{0 \leqslant i < j \leqslant n} \mathit{inner}(i,j) \cdot \mathit{outer}(i,j),$$

wobei $\mathit{inner}(i,j) = \mathbf{Z}_{\mathbf{R}_{\mathit{inner}}}(i,j)$ und $\mathit{outer}(i,j) = \mathbf{Z}_{\mathbf{R}_{\mathit{outer}}}(i,j)$. Diese beiden Strukturmengen $\mathit{inner}(i,j)$ und $\mathit{outer}(i,j)$ geben eine unabhängige Zerlegung an. Alle sekundären Strukturen der Teilsequenz $[i,j]$ sind Strukturelemente von $\mathbf{R}_{\mathit{inner}}(i,j)$, d.h.:

$$\mathbf{R}_{\mathit{inner}}(i,j) = \{R \in \mathbf{R}_{\mathit{inner}} | (i,j) \in R \wedge \forall (i',j') \in R : i \leqslant i' < j' \leqslant j\}.$$

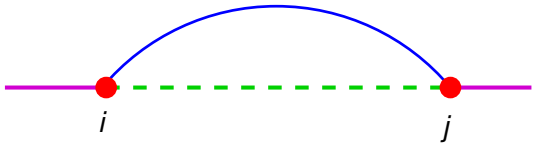
Alle sekundären Strukturen zwischen den Teilsequenzen $[0, i-1]$ und $[j+1, n]$ enthält die Menge $\mathbf{R}_{\mathit{outer}i,j}$, d.h.:

$$\begin{aligned}\mathbf{R}_{\mathit{outer}}(i,j) = &\{R \in \mathbf{R}_{\mathit{outer}} | (i,j) \in R \\ &\wedge \forall (i',j') \in R : 1 \leqslant i' < i < j < j' \leqslant n\}\end{aligned}$$

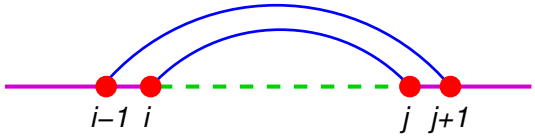
Die Berechnung folgt direkt aus der Beschreibung zur Berechnung der Partitionsfunktion $\mathbf{Z}$ für RNA-Sequenzen im Algorithmus von McCaskill, wie im Abschnitt 2.2 angegeben.

Im Folgenden bezeichnen wir sekundäre RNA-Strukturen beschrieben durch $\mathbf{R}_{\mathit{outer}}(i,j)$ als Outerfragmente und durch $\mathbf{R}_{\mathit{inner}}(i,j)$ als Innerfragmente. Die möglichen Innerfragmente haben wir in Kapitel 4 schon vorgestellt.

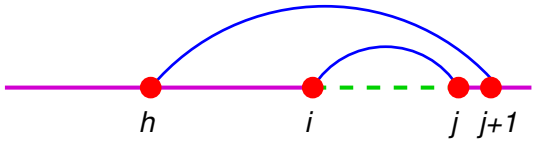
Die unabhängige Zerlegung der Outerfragmente in Strukturmengen geschieht, wie bei den Innerfragmenten, durch die sekundären Strukturelemente. Betrachten wir nun die Outerfragmente für $[i,j]$, die (i,j) als Basenpaar enthalten, sprich die Strukturmenge $\mathbf{R}^{b}_{\mathit{outer}(i,j)}$.



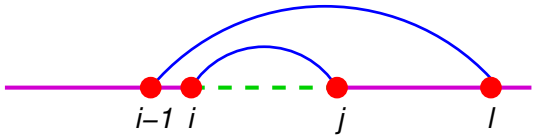
(a) Pair



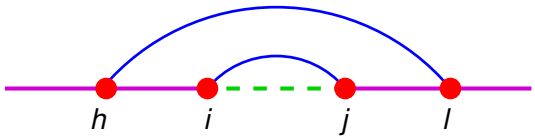
(b) Stack



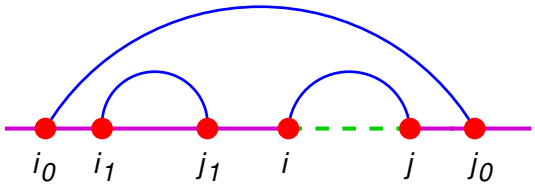
(c) Rechter Bulge



(d) Linker Bulge



(e) Interiorloop



(f) Multiloop

Abbildung 11: Sekundäre Strukturelemente der Outerfragmente: Das aktuell betrachtende Basenpaar ist i und j. Die magenta-farbene Sequenz entspricht der Sequenz für die Outerfragmente, die grüne Sequenz der Sequenz für die Innerfragmente.

DEFINITION (Sekundäre Strukturelemente des Outerfragments)
Im Weiteren entspricht R_{outer} einer sekundären Struktur eines Outerfragments. Die Menge der sekundären Strukturen $\mathbf{R}^{b}_{outer(i,j)}$ für Outerfragmente enthält folgende Strukturelemente:

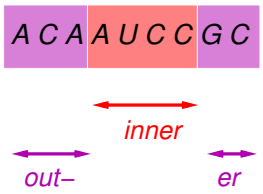
- *Pair*: $(i,j) \in R_{outer}$, wobei gilt, dass $(i',j') \notin R_{outer}$, wenn $(1 \leqslant i' < i < j < j' \leqslant n)$ gilt,
- *Stack*: $(i,j) \in R_{outer}$ und $(i-1,j+1) \in R_{outer}$,
- *Interiorloop*: $(i,j) \in R_{outer}$ und $(h,l) \in R_{outer}$ mit $h < i-1$ und $l > j+1$ und die Teilsequenzen $[h+1,i-1]$ und $[j+1,l-1]$ enthalten nur ungepaarte Basen,
- *Rechter Bulge*: $(i,j) \in R_{outer}$ und $(i-1,l) \in R_{outer}$ mit $l > j+1$ und die Teilsequenz $[j+1,l-1]$ besitzt nur ungepaarte Basen,
- *Linker Bulge*: $(i,j) \in R_{outer}$ und $(h,j+1) \in R_{outer}$ mit $h < i-1$ und die Teilsequenz $[h+1,i-1]$ enthalten nur ungepaarte Basen,
- *Multiloop* mit schließendem Basenpaar $(i_0,j_0) \in R_{outer}$ und $k+1$ weiteren Basenpaare $(i_1,j_1),...,(i,j),...,(i_k,j_k) \in R_{outer}$ mit den Eigenschaften:
  - $i_0 < i_1 < j_1 < ... < i < j < ... < i_k < j_k < j_0$
  - $\forall (l,l') \notin R_{outer}$, falls $i_0 < l < l' < j_0$ und $(l,l') \neq (i_1,j_1),...,(i,j),...,(i_k,j_k)$
  - Jedes Basenpaar (i_l,j_l) mit $l \in \{1,...,k\}$ ist selbst ein schließendes Basenpaar einer eigenen sekundären RNA-Struktur.

Abbildung 11 visualisiert die einzelnen sekundären RNA-Strukturen der Outerfragmente.

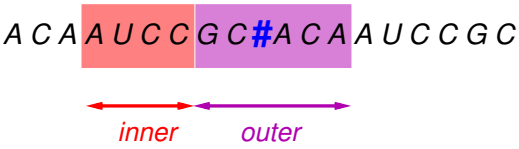
5.2 KODIERUNG DER EINGABE

Sei die RNA-Sequenz s auch die Eingabe des Programms. Algebraische dynamische Programmierung basiert auf kontext-freien Grammatiken. Für jede Teilsequenz $[i,j]$ von s werden die Werte anhand der gegebenen Grammatikregeln berechnet. Zu jedem Nichtterminal gehören nur zusammenhängende Teilsequenzen $[i,j]$. Innerfragmente repräsentieren nur zusammenhängende Teilsequenzen, aber Outerfragmente sind nicht zusammenhängend, sondern haben ein Loch in der Teilsequenz an der Stelle $]i,j[$. Um auch für Outerfragmente zusammenhängende Teilsequenzen zu bekommen, wird eine Kodierung der Sequenz vorgenommen. Wir verdoppeln s und schreiben dazwischen ein Trennungssymbol. Dies führt zu folgender Definition:

DEFINITION (Kodierte Eingabesequenz)
Sei s die Eingabesequenz mit Länge n und $s_i \in \{A,G,C,U\}$,



(a) Ursprüngliche Sequenz



(b) Kodierte Sequenz

Abbildung 12: Darstellung der Inner- und Outerfragmente in der ursprünglichen und kodierten Sequenz

wobei $(1 \leqslant i \leqslant n)$ gilt. Desweiteren sei # ein Symbol, wobei $\# \notin \{A, G, C, U\}$. Dann ist $s\#s$ die *kodierte Eingabesequenz* mit Länge von $2n + 1$.

Die Kodierung der Eingabe s führt zur Betrachtung zweier möglicher Fälle:

1. Sekundäre RNA-Strukturen der Teilsequenz $[i, j]$, sprich Innerfragmente
2. Sekundäre RNA-Strukturen der Teilsequenz $[j + 1, n + i + 1]$, sprich Outerfragmente

DEFINITION (Innerfragment)
Angenommen wir berechnen für die Teilsequenz $[i, j]$ mit $\# \notin [i, j]$ die Werte für eine kodierte Eingabe, dann wird diese Teilsequenz als das *Innerfragment* für $(i, j) \in R$ bezeichnet.

DEFINITION (Outerfragment)
Ein *Outerfragment* für $(i, j) \in R$ ist die Teilsequenz $[j + 1, n + i]$ in der kodierten RNA-Sequenz. In der ursprünglichen RNA-Sequenz überdeckt das Outerfragment das Präfix $[0, i - 1]$ und das Suffix $[j + 1, n]$.

Ein Outerfragment $[j + 1, n + i]$ der kodierten Sequenz $s\#s$ entspricht in der ursprünglichen Sequenz s dem Fragment $[0, j - n - 1] \cup [i, n]$.

Die Abbildung Abbildung 12 repräsentiert diese Sichtweise. Hier ist $s_1 ... s_n = ACAAUCCGC$ die ursprüngliche RNA-Sequenz mit Länge n. Das Innerfragment ist die Teilsequenz $s_4 ... s_7 = AUCC$ und das Outerfragment setzt sich zusammen aus der Teilsequenzen $s_1 ... s_3 = ACA$ und $s_8 s_9 = GC$. Das Outerfragment korrespondiert mit der Teilsequenz $s_8 ... s_{13} = GC\#ACA$, welche das Trennungssymbol # enthält.

5.3 BERECHNUNG DER PARTITIONSFUNKTION FÜR OUTERFRAGMENTE

Widmen wir unsere Aufmerksamkeit jetzt der rekursiven Berechnung der Outerfragmente anhand einer kontext-freien Grammatik. Dafür sei im Weiteren # das Trennungssymbol. Zusätzlich sei $[i, n]$ die aktuelle zu betrachtende Teilsequenz vor dem Trennungssymbol und $[n+2, j]$ die zu betrachtende Teilsequenz nach dem Trennungssymbol, wobei $i < j$ in der kodierten Sequenz und $(j - (n+1)) < i$ in der ursprünglichen Sequenz ist. Die Auswahlfunktion h definieren wir als die Summe aller berechneten Werte für die Grammatikregel, auf die h angewandt wird. Als Basen werden die Symbole a und $\hat{a}$ benutzt. Basen sind in der Grammatik Terminale. Erscheinen in der gleichen Grammatikregel a und $\hat{a}$, dann sind diese beiden gepaart.

5.3.1 *Outerfragmente*

Die Partitionsfunktion $Z_{\mathbf{R}_{outer}}$ entspricht der Partitionsfunktion für Outerfragmente. Um eine rekursive Berechnung anzugeben, werden die Energien für jede sekundäre Struktur der Outerfragmente benötigt. In ADP werden diese mit Hilfe einer Algebra berechnet. Im Folgenden nehmen wir an, dass wir die Energiefunktionen schon definiert haben.

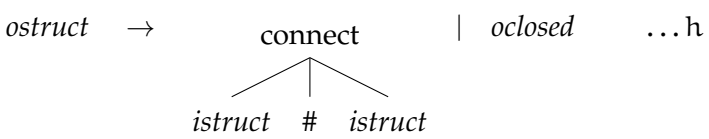
Die Partitionsfunktion $Z_{\mathbf{R}_{outer}}$ sieht wie folgt aus:

$$Z_{\mathbf{R}_{outer}} = \sum_{R_{outer}} e^{-E(R_{outer})/(k_B T)},$$

wobei R_{outer} alle sekundären RNA-Strukturelemente der Outerfragmente der gegebenen RNA-Sequenz sind.

Für ein Outerfragment der Teilsequenz $[i, j]$ gibt es zwei sekundäre RNA-Strukturen, die zu beachten sind. Entweder ist i mit j gepaart oder die Teilsequenzen $[i, n]$ und $[n+2, j]$ bilden jeweils eigene sekundäre Strukturen. Im zweiten Fall entsprechen die sekundären Strukturen den Strukturen eines Innerfragments. Die Grammatik für Innerfragment haben wir schon in Kapitel 4 besprochen. Die Nichtterminale für Innerfragmente ist daher *istruct*. Zwischen den zwei Strukturen steht das Trennungssymbol. Falls i mit j gepaart ist, werden sekundäre Strukturen für ein Outerfragment betrachtet. Diese Strukturen sind in der Nichtterminale *oclosed* beschrieben und enthalten noch das Trennungssymbol.

Dies führt uns zur folgender Regel für Outerfragmente:



Die Funktion `connect` multipliziert die Energien der zwei Innerfragmente. Diese Multiplikation entspricht der Vorschrift zur Berechnung der Partitionsfunktion mit Hilfe des McCaskill Algorithmus, wie in Abschnitt 2.2 gezeigt.

Die sekundären RNA-Strukturen, die durch *oclosed* beschrieben werden, sind die gleichen wie die der Outerfragmente, dargestellt im Abschnitt 5.1. Die Partitionsfunktion $Z_{\mathbf{R}^b_{outer(i,j)}}$ entspricht der Berechnung von *oclosed*. Daraus folgt:

$$\begin{aligned} oclosed = Z_{\mathbf{R}^b_{outer(i,j)}} &= e^{-eP(i,j)/(k_B T)} \cdot Z_{\mathbf{R}_{inner(i+1,n)}} \cdot Z_{\mathbf{R}_{inner(n+2,j-1)}} \\ &+ e^{-eS(i,j,i+1,j-1)/(k_B T)} \cdot Z_{\mathbf{R}^b_{outer(i+1,j-1)}} \\ &+ \sum_{h,l} e^{-eI(i,j,h,l)/(k_B T)} \cdot Z_{\mathbf{R}^b_{outer(h,l)}} \\ &+ \Big(\sum_{(h,l)} Z_{\mathbf{R}^b_{inner(h,1)}} \cdot Z_{\mathbf{R}^{lbd}_{outer(l+1,j-1)}} \cdot e^{-(a+b+c(h-i-1))/(k_B T)} \\ &+ Z_{\mathbf{R}^b_{outer(h,l)}} \cdot Z_{\mathbf{R}^{lbd}_{inner(l+1,j-1)}} \cdot e^{-(a+b+c(h-i-1))/(k_B T)} \Big). \end{aligned}$$

Hier berechnet der erste Summand die Partitionsfunktion für einen Pair, der zweite die für einen Stack, der dritte die eines Interiorloops inklusive den Bulges und der letzte die eines Multiloops. In $\mathbf{R}_{outer(i,j)}$ ist, wie in Abschnitt 2.2 beschrieben, das äußere Basenpaar gepaart.

Wir erhalten also folgende Grammatikregel:

$$\begin{aligned} oclosed \quad &\rightarrow \quad opair \mid ostack \mid oloop \mid orightB \mid oleftB \\ &\mid \quad omultiloop \qquad \dots h \end{aligned}$$

Hier bezeichnen *opair*, *ostack*, *oloop*, *orightB*, *oleftB* und *omultiloop* die einzelnen Partitionsfunktionen der sekundären RNA-Strukturen des Outerfragments.

In den nächsten Abschnitten werden wir genauer auf die Berechnung der einzelnen Summanden, benötigt für die Partitionsfunktion von *oclosed*, eingehen.

5.3.2 *Pair*

Bei einem Pair sind i und j gepaart. In Abbildung 13 wird diese Struktur für die kodierte Sequenz dargestellt. Die Partitionsfunktion für einen Pair berechnet sich folgendermaßen:

$$opair = e^{-eP(i,j)/(k_B T)} \cdot Z_{\mathbf{R}_{inner(i+1,n)}} \cdot Z_{\mathbf{R}_{inner(n+2,j-1)}}$$

Es existiert also kein Basenpaar (i', j'), für das $i' \in [i+1, n]$ und $j' \in [n+2, j-1]$ gilt. Allerdings muss die Partitionsfunktion für sekundären Strukturen der jeweiligen Teilsequenzen $[j+1...n]$ und

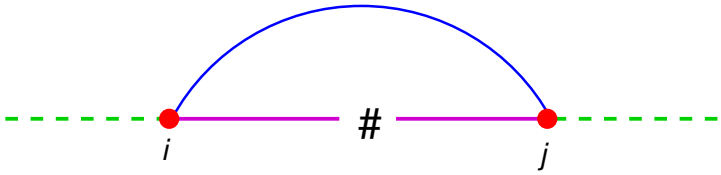


Abbildung 13: Pair in der kodierten RNA-Sequenz

$[n+2...i-1]$ beachtet werden. Die Partitionsfunktion dieser sekundären RNA-Strukturen entspricht der Partitionsfunktion der Innerfragmente der Teilsequenz $[i+1, n]$ und der Teilsequenz $[j+1, n]$. An der Stelle $n+1$ steht das Trennungssymbol.

Die Partitionsfunktion für einen Pair wird mit folgender Grammatikregel ermittelt:

$$\textit{opair} \rightarrow \text{pair-energy}\ (a, \textit{istruct}, \#, \textit{istruct}, \hat{a}) \quad \ldots h$$

Die Energie des Pairs wird mit der Funktion `pair-energy` ermittelt.

5.3.3 *Stack*

In einem Stack eines Outerfragments sind sowohl die Basen i und j also auch die Basen $i+1$ und $j-1$ gepaart. Abbildung 14 zeigt den Stack für die kodierte Sequenz. Das Trennungssymbol liegt zwischen dem Basenpaar $(i+1, j-1)$. Dies führt zur Ermittlung der Partitionsfunktion:

$$\textit{ostack}(i,j) = e^{-eS(i,j,i+1,j-1)/(k_B T)} \cdot Z_{\mathbf{R}^b_{\textit{outer}}(i+1,j-1)}$$

Dies wird durch folgende Grammatikregel beschrieben:

$$\textit{ostack} \rightarrow \text{sr-energy}\ (a, \textit{oclosed}, \hat{a}) \quad \ldots h$$

Die Nichtterminale *oclosed* beschreibt die Partitionsfunktion für Outerfragmente, deren äußere Basenpaare gebunden sind. Dadurch erreichen wir, dass das Basenpaar $(j+1, i-1)$ existiert.

Die verwendete Funktion `sr-energy` in der Grammatikregel berechnet die Energie des Stacks.

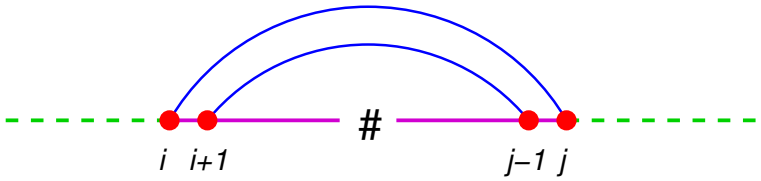


Abbildung 14: Stack in der kodierten RNA-Sequenz

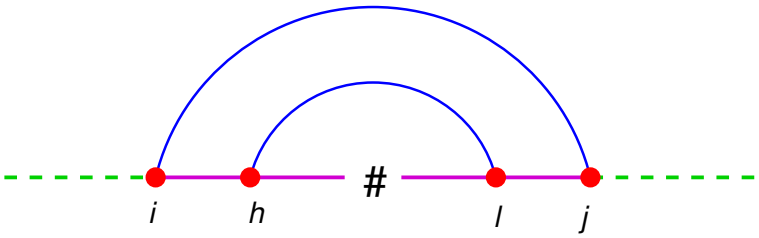


Abbildung 15: Interiorloop in der kodierten RNA-Sequenz

5.3.4 *Interiorloop und Bulge*

Als Nächstes beschäftigen wir uns mit dem Interiorloop eines Outerfragments.

Für einen Interiorloop berechnen wir folgende Partitionsfunktion:

$$oloop = \sum_{h,l} e^{-eI(i,j,h,l)/(k_B T)} \cdot Z_{\mathbf{R}^b_{outer}(h,l)}$$

In einem Interiorloop ist (h, l) auch ein Basenpaar. Hier steht das Trennungssymbol in der Teilsequenz $[h+1, h-1]$. Es sind drei mögliche Interiorloops zu beachten. Für den einfachen Interiorloop gilt, $i+1 < h < l < j-1$ und die Teilsequenzen $[i+1, h-1]$ und $[l+1, j-1]$ enthalten nur ungepaarte Basen. In Abbildung 15 sehen wir für die kodierte Sequenz den Interiorloop.

Falls $h = i+1$, $l < j-1$, erhalten wir in der kodierten Sequenz einen rechten Bulge. In der ursprünglichen Sequenz ist dies allerdings ein linker Bulge. Dies spielt vor allem eine Rolle bei der Berechnung der Energie dieser RNA-Struktur. Entsprechend gilt dies für einen linken Bulge in der kodierten Sequenz, für den $i+1 < h$ und $l = j-1$ gilt. In der ursprünglichen Sequenz ist dieser linke Bulge ein rechter Bulge. Wir berechnen also für den linken Bulge in der kodierten Sequenz die Energie für einen rechten Bulge und für den rechten Bulge die Energie für einen linken Bulge. Abbildung 16 zeigt jeweils den Bulge in der kodierten wie in der ursprünglichen Sequenz.

Wir brauchen drei Grammatikregeln für die Errechnung der Partitionsfunktion eines Interiorloops, wenn wir auch die Bulges betrachten. Die Basenregionen, die ungepaart sind, werden mit `REGION` bezeichnet.

Eine `REGION` enthält mindestens eine Base. Wir erhalten die folgenden Grammatikregeln:

oloop → ilenergy(a, REGION, *oclosed*, REGION, â) ... h

orightB → bl-energy(a, REGION, *oclosed*, â) ... h

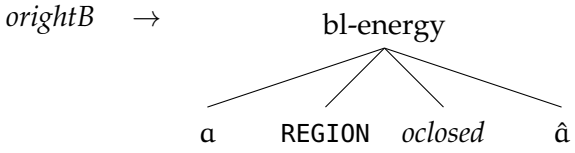
oleftB → br-energy(a, *oclosed*, REGION, â) ... h

Die Funktion `il-energy` berechnet die Energie eines Interiorloops, der kein Bulge ist. Die Energie für einen rechten Bulge in der kodierten Sequenz, der ein linker Bulge in der ursprünglichen Sequenz ist, wird mit der Funktion `bl-energy` ermittelt und die Energie für einen linken Bulge in der kodierten Sequenz, der einem rechten Bulge in der ursprünglichen Sequenz entspricht, mit der Funktion `br-energy`. Die Garantie, dass das Basenpaar (h, l) existiert, wird durch *oclosed* erreicht.

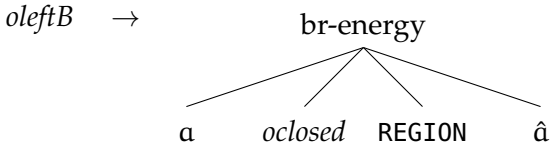
Um die Laufzeit gering zu halten, werden meistens nur Interiorloops und Bulges mit ungepaarten Basenregionen betrachtet, die höchstens dreißig Basen enthalten. Dies kann durch einen Filter erreicht werden, der wie folgt gesetzt wird:

oloop → il-energy(a, REGION, *oclosed*, REGION, â)

$$\text{with } \text{length}(\text{REGION}) \leqslant 30 \qquad \ldots h$$



with length(REGION) $\leqslant 30$ ... h



with length(REGION) $\leqslant 30$ ... h

5.3.4.1 *Multiloop*

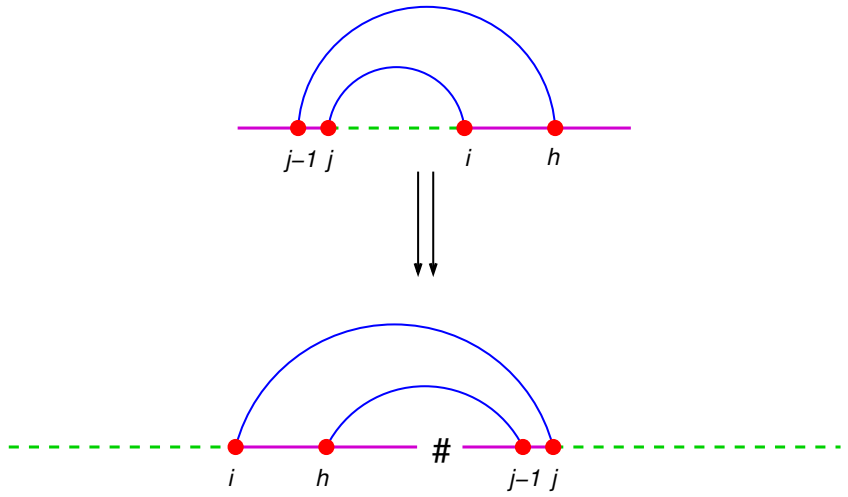
Als letztes erläutern wir die Berechnung der Partitionsfunktion eines Multiloops, die folgendermaßen aussieht:

$$\begin{aligned} \mathit{omultiloop} = \sum_{(h,l)} & Z_{\mathbf{R}^{b}_{\mathit{inner}(h,l)}} \cdot Z_{\mathbf{R}^{lbd}_{\mathit{outer}(l+1,j-1)}} \cdot e^{-(a+b+c(h-i-1))/(k_B T)} \\ & + Z_{\mathbf{R}^{b}_{\mathit{outer}(h,l)}} \cdot Z_{\mathbf{R}^{lbd}_{\mathit{inner}(l+1,j-1)}} \cdot e^{-(a+b+c(h-i-1))/(k_B T)} \end{aligned}$$

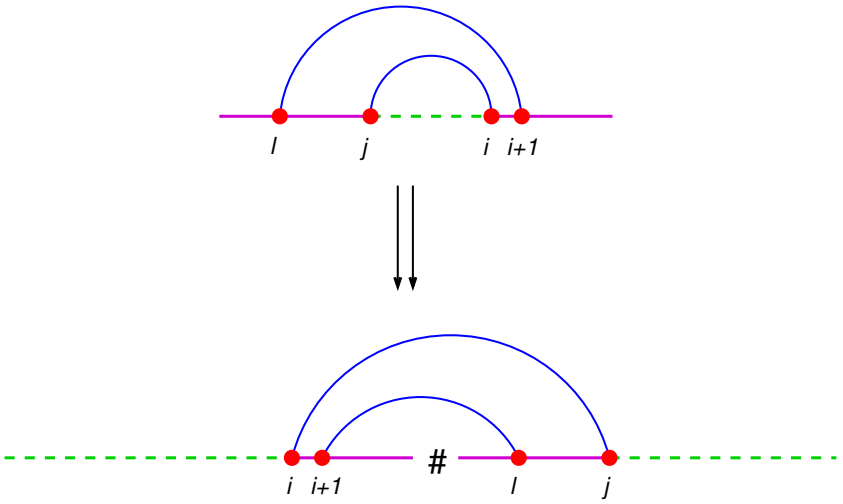
Der Multiloop wird aufgeteilt genau nach der ersten sekundären RNA-Struktur. Danach folgt noch mindestens eine weitere sekundäre RNA-Struktur. Garantiert wird dies durch die Strukturmenge $\mathbf{R}^{lbd}_{i,j}$. So gewährleisten wir, dass unterhalb des Multiloops mindestens zwei sekundäre RNA-Strukturen stehen.

Sowohl die Strukturmenge $\mathbf{R}^{lbd}_{\mathit{outer}(i,j)}$ als auch die Strukturmenge $\mathbf{R}^{lbd}_{\mathit{inner}(i,j)}$ beinhalten alle sekundäre RNA-Strukturen, die zwischen der Teilsequenz $[i,j]$ mindestens ein Basenpaar enthalten. Der Unterschied der beiden Strukturmengen besteht darin, dass $\mathbf{R}^{lbd}_{\mathit{outer}(i,j)}$ das Trennungssymbol enthält und $\mathbf{R}^{lbd}_{\mathit{inner}(i,j)}$ nicht. Die Ermittlung der Partitionsfunktion für einen Multiloop ist im Gegensatz zu derjenigen, beschrieben in Abschnitt 2.2, erweitert. Da hier die Beachtung der Position des Trennungssymbols eine Rolle spielt. Es ist möglich, dass das Trennungssymbol in der ersten Strukturmenge $\mathbf{R}^{lbd}_{i+1,h-1}$ oder auch in der zweiten Strukturmenge $\mathbf{R}^{b}_{h,l}$ liegt.

Um dies in einer Grammatikregel zu beschreiben, brauchen wir eine



(a) Linker Bulge

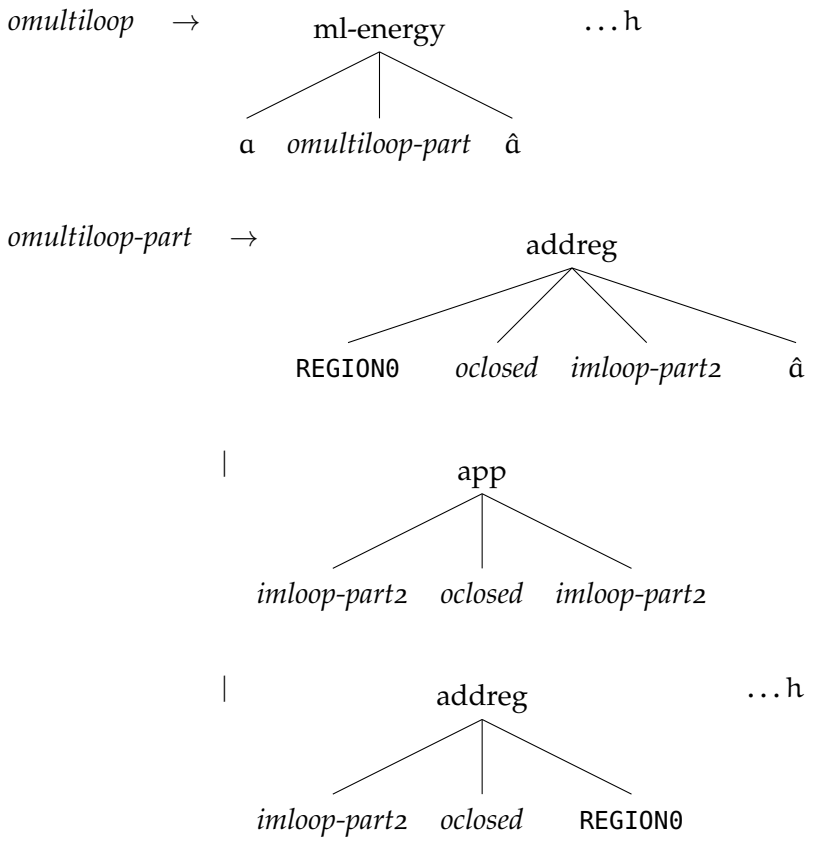


(b) Rechter Bulge

Abbildung 16: Falls in der ursprünglichen Sequenz ein rechter Bulge ist, dann ist dies ein linker Bulge in der kodierten Sequenz **(a)**. Wenn in der ursprünglichen Sequenz ein linker Bulge ist, dann ist dies ein rechter Bulge in der kodierten Sequenz **(b)**

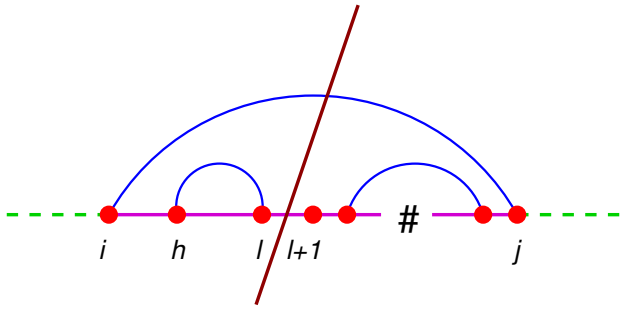
Aufteilung der Strukturmenge $\mathbf{R}^{lbd}_{outer(i,j)}$ für die Teilsequenz $[i,j]$ in zwei Teilmengen. Die erste Teilmenge enthält die Strukturen ohne Trennungssymbol, sprich Innerfragmente. Diese Strukturen sind in der Strukturmenge $\mathbf{R}^{lbd}_{inner(i,j)}$ enthalten. Die Partitionsfunktion dieser Strukturen steht in *imloop-part2*. Die zweite Teilmenge besteht aus sekundären RNA-Strukturen, die das Trennungssymbol zwischen einem Basenpaar (i',j') enthalten. Diese sind in der Strukturmenge $\mathbf{R}^{b}_{outer(i',j')}$ enthalten, deren Partitionsfunktion in *oclosed* steht.

Die Grammatikregeln für die Partitionsfunktion eines Mulitloops lautet:

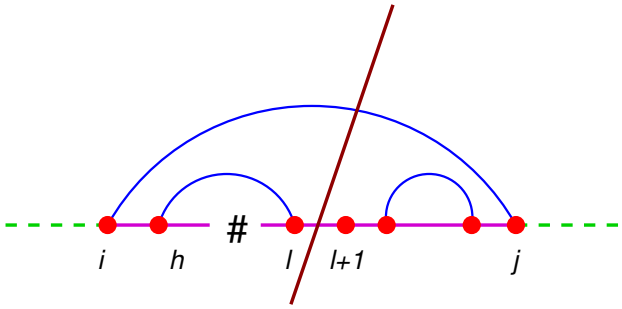


Die erste Regel, genannt *omultiloop*, betrachtet das schließende Basenpaar des Multiloops und den Multiloopteil *omultiloop-part*, der mindestens zwei Basenpaare enthält.

Die zweite Grammatikregel spezifiziert einen Multiloopteil eines Outerfragments, den sogenannten *omultiloop-part*. Wie in Unterabschnitt 4.1.4 erklärt, steht *imloop-part2* für $Z_{\mathbf{R}^{lbd}_{inner(i,j)}}$, sprich *imloop-part2* enthält mindestens ein Basenpaar und nicht das Trennungssymbol. Das Trennungssymbol ist in der sekundären RNA-Struktur, beschrieben durch *oclosed*, enthalten. Eine Region von ungepaarten Basen, die auch leer sein kann, ist dargestellt durch `REGION0`. Wie auch schon oben bemerkt, muss die Position des Trennungssymbol beachtet werden. In der Grammatik sind die Fälle etwas anders beschrieben. Die



(a) 1. Fall: Trennungssymbol nach dem Basenpaar



(b) 2. Fall: Trennungssymbol innerhalb des Basenpaars

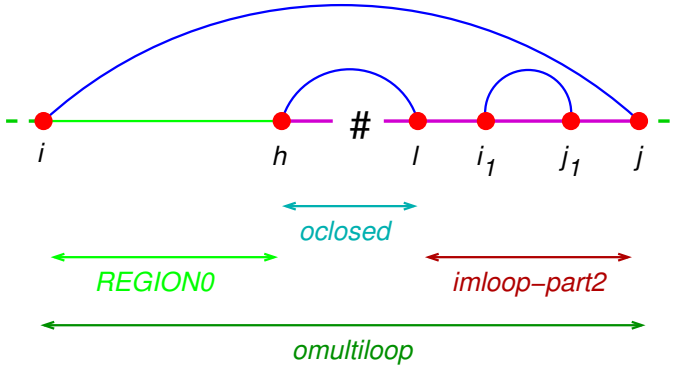
Abbildung 17: Teilung des Multiloops zur Berechnung der Partitionsfunktion nach dem ersten Basenpaar

RNA-Struktur, die das Trennungssymbol enthält, kann entweder ganz rechts stehen, zwischen anderen sekundären RNA-Strukturen stehen oder ganz links stehen. Wenn diese RNA-Struktur ganz rechts steht, folgen danach nur noch eine Region von ungepaarten Basen oder die leere Sequenz. Im Fall, bei dem diese RNA-Struktur ganz links steht, enthält die Teilsequenz vor der RNA-Struktur nur ungepaarte Basen oder die leere Sequenz. Abbildung 18 verdeutlicht noch mal die Aufteilung nach der Grammatik. Der erste Fall zeigt, dass die sekundäre RNA-Struktur, die das Trennungssymbol enthält, ganz links steht. Beim zweiten Fall ist diese sekundäre RNA-Struktur von zwei anderen sekundären RNA-Strukturen umgeben und beim dritten Fall ist diese sekundäre RNA-Struktur ganz rechts.

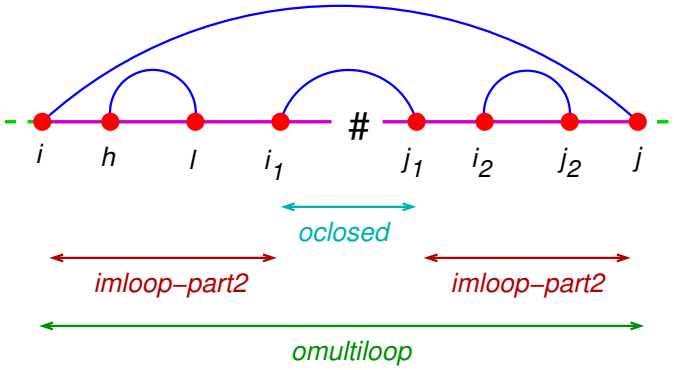
5.4 BERECHNUNG DER VOLLSTÄNDIGEN PARTITIONSFUNKTION

Um die vollständige Partitionsfunktion zu berechnen, wird die Grammatik für die Innerfragmente und die Grammatik für die Outerfragmente zusammengesetzt. Wir benötigen hierfür eine Grammatikregel, die sowohl die Partitionsfunktion für die Innerfragmente als auch die Partitionsfunktion für die Outerfragmente ermittelt.

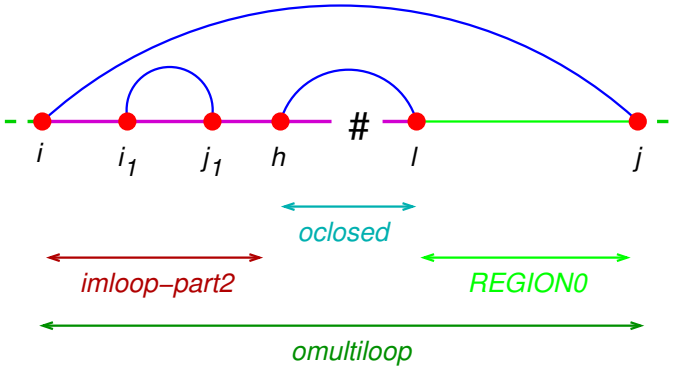
Wie schon in Abschnitt 5.2 angegeben, wird die Eingabesequenz kodiert. Betrachten wir die Teilsequenz $[i, j]$. Zusätzlich sind die Teilsequenzen $[1, i-1]$ und $[j+1, n]$ für die Berechnung der Outerfragmente



(a) 1. Fall



(b) 2. Fall

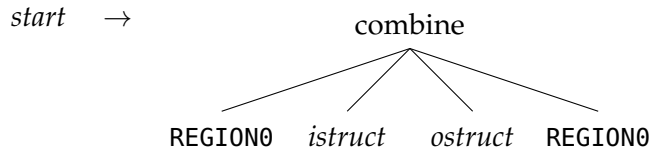


(c) 3. Fall

Abbildung 18: Teilung des Multiloops innerhalb der Gramatik

zu betrachten. Diese Teilsequenzen entsprechen auch der Teilsequenz $[j+1, n+i]$ in der kodierten Sequenz. Das bedeutet, dass wir immer eine Sequenz mit der Länge $n+1$ ansehen. Die Grammatikregel zur Kombination der Partitionsfunktionen der Innerfragmente und Outerfragmente muss dies gewährleisten. Dies kann durch einen Filter erreicht werden.

Die Grammatikregel lautet:



$$\text{with } \text{length}(\textit{istruct} + \textit{ostruct}) = n+1$$

Die Funktion `combine` kombiniert die Partitionsfunktion der Innerfragmente *istruct* und die Partitionsfunktion der Outerfragmente *ostruct*. Die Rückgabe der Funktion ist nicht relevant, da wir für die Berechnung der Basenpaar-Wahrscheinlichkeiten auf die Tabellen *istruct*, *iclosed* und *oclosed* zugreifen. Es ist nur relevant, dass alle Innerfragmente und Outerfragmente betrachtet werden. Die Variable `REGION0` bezeichnet eine Region von ungepaarten Basen oder die leere Sequenz. Dadurch kann das Betrachtungsfenster entsprechend verschoben werden. Dass das Betrachtungsfenster immer eine Größe von $n+1$ besitzt, gewährleistet der Filter, definiert durch $\text{with } \text{length}(\textit{istruct} + \textit{ostruct}) = n+1$.

Diese Grammatikregel *start* ist gleichzeitig das Axiom unserer Grammatik.

5.5 ZUSAMMENFASSUNG

In diesem Kapitel haben wir die sekundären RNA-Strukturen für Outerfragmente vorgestellt - Pair, Stack, Interiorloop, Bulges und Multiloop. Outerfragmente für die Teilsequenz $[i, j]$ sind die sekundären RNA-Strukturen, die $[i, j]$ innerhalb der Struktur enthalten sind, sprich es existiert ein Basenpaar (i', j') mit $1 \leqslant i' < i < j < j' \leqslant n$. Um die Partitionsfunktion für solche Outerfragmente zu berechnen, wird eine Kodierung für die Eingabe benötigt. Diese verdoppelt die Eingabesequenz und schreibt dazwischen ein Trennungssymbol. Danach haben wir die Grammatikregeln für die Berechnung der Partitionsfunktion der Outerfragmente erklärt. Die Grammatikregeln korrespondieren mit den einzelnen Partitionsfunktion auf den einzelnen Strukturmengen entsprechend denen des Algorithmus von McCaskill. Für jede einzelne sekundäre RNA-Struktur eines Outerfragments - Pair, Stack, Interiorloop, Bulge, Multiloop - wird eine eigene Grammatikregel zur Berechnung der Partitionsfunktion benötigt. Zuletzt

zeigten wir, wie die Berechnung der Partitionsfunktion für die Inner- und Outerfragmente kombiniert wird.

Teil IV

BASENPAAR-WAHRSCHEINLICHKEITEN

6

BERECHNUNG DER SEKUNDÄREN RNA-STRUKTURVERTEILUNG

In diesem Kapitel erklären wir für eine RNA-Sequenz s der Länge n die Berechnung der Basenpaar-Wahrscheinlichkeiten. Dafür betrachten wir zuerst die Ermittlung der Basenpaar-Wahrscheinlichkeiten anhand unserer Berechnung der Partitionsfunktion, siehe Kapitel 4 und Kapitel 5, im Detail. Danach gehen wir noch kurz auf die Implementierung ein.

6.1 BERECHNUNG DER BASENPAAR-WAHRSCHEINLICHKEITEN

In Abschnitt 2.2 haben wir schon allgemein die Berechnung der Basenpaar-Wahrscheinlichkeiten erläutert. Im Weiteren beschreiben wir detailliert die Berechnung der Basenpaar-Wahrscheinlichkeiten für unseren Algorithmus. Dafür werden die Partitionsfunktionen der Inner- und Outerfragmente benutzt, die mittels algebraischer dynamischer Programmierung, wie in Kapitel 4 und Kapitel 5 gezeigt, ermittelt werden.

Die Wahrscheinlichkeit für das Auftreten eines Basenpaars (i, j) kann mit Hilfe der relativen Häufigkeit berechnet werden. Damit ist die Basenpaar-Wahrscheinlichkeit definiert durch:

$$\frac{\textit{Anzahl der Strukturen mit } (i,j)}{\textit{Anzahl aller Strukturen}}$$

Wir gewichten die RNA-Strukturen mit Hilfe der Partitionsfunktion. Wir erhalten die Basenpaar-Wahrscheinlichkeit $p_{i,j}$ durch die Gewichtung:

$$p_{i,j} = \frac{\textit{Partitionsfunktion der Strukturen mit } (i,j)}{\textit{Partitionsfunktion für alle Strukturen}}$$

Die Ermittlung der Partitionsfunktion aller RNA-Strukturen ist in der Berechnung der Innerfragmente enthalten. Für eine RNA-Sequenz der Länge n ist das entsprechende Innerfragment, das alle RNA-Strukturen enthält, die Teilsequenz $[1, n]$. Die Partitionsfunktion der Sequenz $[1, n]$ steht in $Z_{\mathbf{R}_{inner(1,n)}}$, die in der Grammatik in der Nichtterminale *istruct* gespeichert wird.

Die Partitionsfunktion aller RNA-Strukturen mit dem Basenpaar (i, j) setzt sich aus der Partitionsfunktion des Innerfragments $[i, j]$ und des Outerfragments, bei denen jeweils gilt, dass (i, j) ein Basenpaar ist. Diese sekundären RNA-Strukturen sind Strukturelemente von $\mathbf{R}^{b}_{inner(i,j)}$ und $\mathbf{R}^{b}_{outer(i,j)}$. Wir benötigen also die Partitionsfunktionen

$Z_{\mathbf{R}^b_{inner(i,j)}}$ und $Z_{\mathbf{R}^b_{outer(i,j)}}$. Für die Wahrscheinlichkeit des Basenpaars (i,j) werden diese beiden Partitionsfunktionen zu multipliziert.

Wir erhalten folgende Formel für die Wahrscheinlichkeit eines Basenpaars (i,j):

$$p_{i,j} = \frac{Z_{\mathbf{R}^b_{inner(i,j)}} \cdot Z_{\mathbf{R}^b_{outer(i,j)}}}{Z_{\mathbf{R}_{inner(1,n)}}}.$$

In unserer Grammatik steht die Partitionsfunktion $Z_{\mathbf{R}^b_{inner(i,j)}}$ in der Nichtterminale *iclosed* und die Partitionsfunktion $Z_{\mathbf{R}^b_{outer(i,j)}}$ in der Nichtterminale *oclosed*. Die Partitionsfunktion $Z_{\mathbf{R}_{inner(1,n)}}$ für die ganze Sequenz $[1,n]$ ist in der Berechnung der Partitionsfunktion der Innerfragmente enthalten und zwar in der Nichtterminale *istruct*. Daraus folgt:

$$p_{i,j} = \frac{Z_{\mathbf{R}^b_{inner(i,j)}} \cdot Z_{\mathbf{R}^b_{outer(i,j)}}}{Z_{\mathbf{R}_{inner(1,n)}}} = \frac{iclosed(i,j) \cdot oclosed(i,j)}{istruct(1,n)}.$$

6.2 IMPLEMENTIERUNG

Die Berechnung der Partitionsfunktion geschieht mittels algebraischer dynamischer Programmierung innerhalb eines ADP-Programms. Für die Implementierung verwenden wir die Programmierumgebung, die von G. Sauthoff [10] entwickelt wurde. Ein zusätzliches Programm für die Berechnung der Wahrscheinlichkeiten der jeweiligen Basenpaare (i,j), die in s vorkommen, wird benötigt. Schließlich ist eine geeignete grafische Darstellung der Wahrscheinlichkeiten zu finden. Wahrscheinlichkeiten lassen sich gut anhand eines Dotplots darstellen.

Abbildung 19 liefert einen Überblick, wie die Implementierung erfolgt. Die Energiefunktionen für die RNA-Strukturen werden in einer Bibliothek vorgegeben. Die Berechnung der Energiefunktionen geschehen anhand des Modells von Turner [5]. Das ADP-Programm zur Berechnung der Partitionsfunktionen verwendet diese Energiefunktionen in der Algebra. Diese Algebra spezifiziert die Funktionen, die in der Grammatik benötigt werden. Die Grammatik liefert die detaillierte Anweisung, wie die Partitionsfunktion ermittelt wird. Die Basenpaar-Wahrscheinlichkeiten werden mit Hilfe eigenes Programms in C++ berechnet. Dafür werden die Tabellen der Nichtterminalen *iclosed*, *oclosed* und *istruct* benötigt, die innerhalb des ADP-Programms generiert werden. Die Ausgabe der Wahrscheinlichkeiten erfolgt in einem Dotplot, das als Postscript vorliegt.

6.2.1 *Implementierung mit einfachen Energiefunktionen*

Unsere Implementierung verwendet eine vereinfachte Energiefunktion, die die Basenpaare der sekundären RNA-Strukturen zählt und

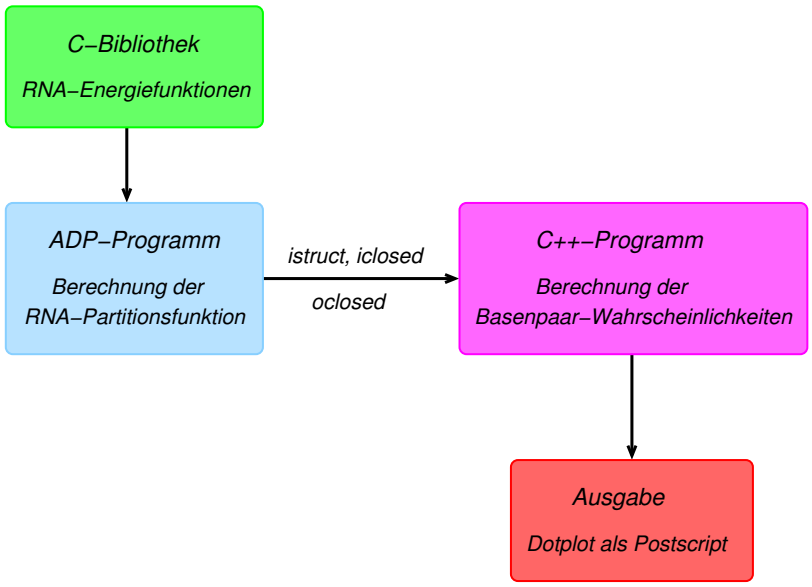


Abbildung 19: Überblick der Implementierung zur Berechnung der Basenpaar-Wahrscheinlichkeiten

jedes Basenpaar gleich gewichtet. Dabei werden auch isoliert stehende Basenpaare betrachtet ("lonely base-pairs"). Unser vereinfachtes Energiemodell entspricht dem des Algorithmus von Nussinov [7].

6.2.1.1 *Grund für vereinfachte Energiefunktionen*

Der Algorithmus von McCaskill verwendet ein thermodynamisches Energiemodell, wie zum Beispiel das von Turner [5]. Seit Kurzem besitzt Bellman's Gap [10] eine Bibliothek für ein thermodynamisches Energiemodell implementiert von S. Janssen, gleich dem Energiemodell von RNAfold [3].

Die implementierten Energiefunktionen berücksichtigen die Leserichtung der RNA-Sequenz, die vom $5'$-Ende zum $3'$-Ende geht. Wir nehmen an, dass s eine RNA-Sequenz der Länge n ist. Dann gilt für die Teilsequenz $[i, j]$ mit Basenpaar (i, j):

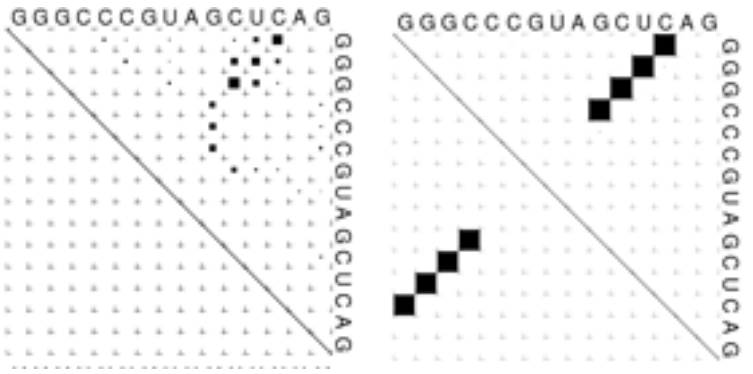
$$5' - i...j - 3',$$

wobei $1 \leqslant i < j \leqslant n$. Dies ist für die Innerfragmente kein Problem. Betrachten wir nun die Outerfragmente. Die Grammatik betrachtet immer die Outerfragmente in der kodierten Sequenz, siehe Kapitel 5. Für das Basenpaar (i, j) gilt in der kodierten Sequenz:

$$5' - j...\#...i + n + 1 - 3',$$

wobei $1 \leqslant i < j \leqslant n$. Aber in der ursprünglichen Sequenz entspricht dies:

$$5' - i...j - 3'.$$



(a) Unsere Implementierung (b) RNAfold

Abbildung 20: Dotplots für die RNA-Sequenz `GGGCCCGUAGCUCAG`

Dies ist auch die Sichtweise, die für die Energiefunktionen notwendig ist.

Unsere Idee ist, die Basen für die Energiefunktionen für die Outerfragmente an die Positionen in der ursprünglichen Sequenz zu verschieben. Dies wird als Index Hacking bezeichnet [9]. Dies ist schon implementiert.

Allerdings reicht dieses nicht ganz aus. Es sind auch andere Faktoren für die Energiefunktionen der Outerfragmente zu beachten, wie:

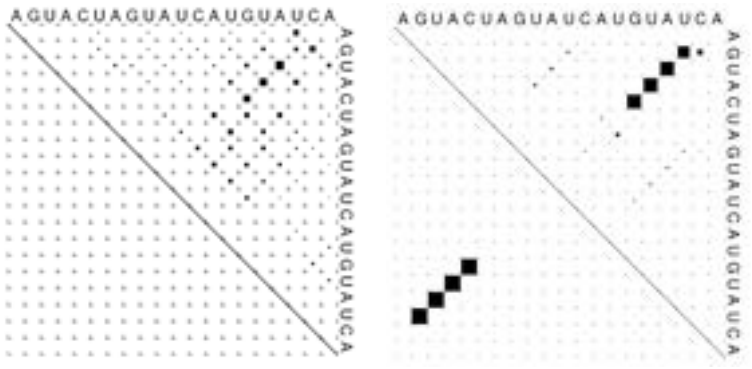
- zusätzliche Energien für schließende Basenpaare,
- die Bestimmung des richtigen Vorgänger und Nachfolger einer Base,
- ungepaarte Basen, die an den sekundären RNA-Strukturen - Interiorloop, Bulges und Multiloop - beteiligt sind,
- die Anpassung der Matrizen für die benötigten Parameter zur Bestimmung der Energiefunktion,
- die Beeinflussung der Energien der Outerfragmente für die Energien der Innerfragmente.

Bei der Implementierung der loop-basierten Energiefunktionen treten noch unerwartete Effekte auf, die einer weiteren tiefer gehenden Bearbeitung bedürfen, den zeitlichen Rahmen dieser Masterarbeit aber überschritten hätten.

6.2.2 *Analyse der Implementierung*

6.2.2.1 *Vergleich mit RNAfold*

Die Ausgabe unserer Implementierung liefert einen Dotplot der Basenpaar-Wahrscheinlichkeiten. In der Darstellung steht sowohl auf der



(a) Unsere Implementierung (b) RNAfold

Abbildung 21: Dotplots für die RNA-Sequenz `AGUACUAGUAUCAUGUAUCA`

x-Achse als auch auf der y-Achse die eingegebene RNA-Sequenz. In einem Dotplot visualisieren die Punkte, genannt Dots, die Wahrscheinlichkeit für das Basenpaar zwischen der Base in der Spalte und der Base in der Zeile. Je größer die Wahrscheinlichkeit für das Basenpaar ist umso größer der Dot. Anhand des Dotplots lassen sich auch schon wahrscheinliche sekundäre Strukturelemente ablesen. Vor allem gut lassen sich Stacks erkennen. Ein Stack existiert, falls in einer Diagonalen mehr als mehrere Dots hintereinander folgen. Allerdings sind bei unserer Implementierung die wahrscheinlichsten Strukturen nicht sehr deutlich sichtbar. Der Grund ist die Verwendung des vereinfachten Energiemodells, das alle Basenpaar gleich bewertet und dadurch sehr viele Strukturen für wahrscheinlich gehalten werden.

Beispiele für die Ausgabe unserer Implementierung und der von RNAfold [3] zeigen Abbildung 20 und Abbildung 21. Der Dotplot von RNAfold zeigt zusätzlich im linken Teil von der Hauptdiagonalen die am wahrscheinlichste sekundäre RNA-Struktur.

Widmen wir zuerst unsere Aufmerksamkeit der Abbildung 20. Hier werden die Basenpaar-Wahrscheinlichkeiten für die Sequenz `GGGCCCGUAGCUCAG` mit der Länge von 15 Basen visulisiert. Wie aus beiden Dotplots ersichtlich ist eines Stacks von der Teilsequenz $[1, 4]$, `GGGC`, und der Teilsequenz $[10, 13]$, `GCUC` sehr groß, wobei die Base an Position 1 mit der Base an Position 13 gepaart ist, und die Base an Position 2 mit der Base an Position 12, die Base an Position 3 mit der Base an Position 11 und die Base an Position 4 mit der Base an Position 10. In Abbildung 20b ist die Wahrscheinlichkeit ausgeprägter als in Abbildung 20a. Dies liegt daran, dass bei unserer Implementierung sekundäre Strukturen mit gleicher Gewichtung bewertet werden. Hier wird schon ersichtlich, dass die berechnete Strukturverteilung unserer Implementierung sich stark von der berechneten von RNAfold unterscheidet.

Für die RNA-Sequenz `AGUACUAGUAUCAUGUAUCA` der Länge 20 erhalten wir die Dotplots in Abbildung 21. Die Anzahl der möglichen sekun-

dären RNA-Strukturen ist größer als die der RNA-Sequenz im Beispiel davor. Hier wird noch deutlicher, dass unsere Implementierung alle sekundären Strukturen gleich bewertet. Die Strukturverteilung berechnet durch unsere Implementierung unterscheidet sich sehr von der Strukturverteilung berechnet durch RNAfold. Mit unserer Implementierung erhalten wir deutlich mehr wahrscheinliche Strukturen. Dadurch ist die Ausprägung der wahrscheinlichsten sekundären Struktur nicht so ersichtlich wie in RNAfold.

Wie schon durch die Beispiele ersichtlich, ist ein Vergleich mit anderen Implementierungen des Algorithmus von McCaskill, wie RNAfold, hier nicht sinnvoll. Der Grund liegt darin, dass wir nur ein vereinfachtes Energiemodell verwenden. Mit diesem Energiemodell ist eine realistische Aussage über die sekundäre Strukturverteilung einer RNA-Sequenz nicht möglich. Aber eine Aussage über die Korrektheit der Grammatik lässt sich in soweit treffen, dass unsere Implementierung auch wirklich nur Basenpaare bewertet, die nach Watson-Crick möglich sind. Für kleine Sequenzen bekommen wir durch manuelle Berechnung der Basenpaar-Wahrscheinlichkeiten mit dem vereinfachten Energiemodell zu den gleichen Ergebnissen.

6.2.2.2 *Laufzeit*

Zur Analyse der Laufzeit haben wir 500 RNA-Sequenzen zufällig generiert. Die zufälligen RNA-Sequenzen sind normal verteilt. Die Länge der Sequenzen liegt zwischen 10 und 200 Basen. Auf einem Rechner mit folgenden Komponenten wurde die Laufzeitanalyse ausgeführt:

- *Prozessor:* Intel Pentium M760 Prozessor mit 2 Hz,
- *Größe des RAMs:* 1,5 GB,
- *Technologie:* Intel Centrino.

Abbildung 22 zeigt das Resultat der Laufzeitanalyse. Die x-Achse repräsentiert die Länge der Sequenz und die y-Achse die Zeit in Sekunden, die unsere Implementierung benötigt, um die Basenpaar-Wahrscheinlichkeiten zu berechnen. Die Laufzeit für RNA-Sequenzen mit 100 Basen beträgt noch unter 5 Sekunden. Aber schon für eine Sequenz mit 200 Basen beträgt die Laufzeit ungefähr 45 Sekunden.

Bei der Laufzeit ist zu beachten, dass diese im Gegensatz zu anderen Implementierungen schlechter ist. Der Grund ist die kodierte Eingabe. Die Länge der Eingabesequenz verdoppelt sich dadurch. Wir berechnen hier viele unnötige Einträge. Dies führt zu einer schlechten Laufzeit. Dies sollte in Zukunft optimiert werden.

6.2.2.3 *Speicherplatzbedarf*

Die Nichtterminalen in der Grammatik speichern die schon berechneten Teillösungen in Tabellen. In Bellman's Gap ist es möglich anzugeben, welche Nichtterminalen eine Tabelle anlegen sollen und welche

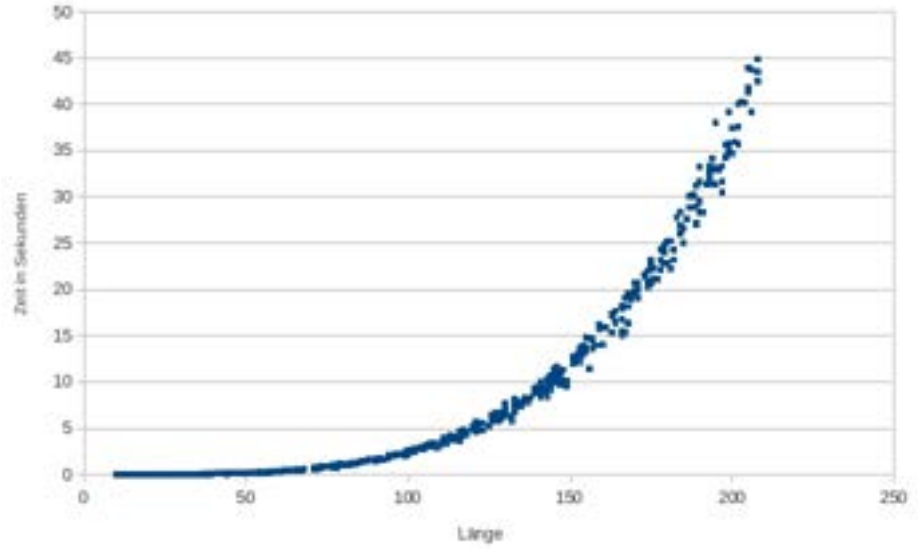


Abbildung 22: Zeitanalyse

nicht. Die Anzahl der Tabellen hängt mit der Laufzeit der Implementierung eng zusammen. Werden zu wenige Tabellen angelegt, ist der Speicherplatzbedarf niedrig, aber es erhöht sich die Laufzeit. Werden hingegen zu viele Tabellen angelegt, ist der Speicherplatzbedarf sehr groß, aber es erniedrigt sich die Laufzeit.Die Aufgabe ist nun, so wenig Tabellen wie möglich anzulegen, so dass sich die optimale Laufzeit nicht erhöht. Wir speichern die berechneten Werte für die folgende Nichtterminale in Tabellen: *istruct*, *iclosed*, *imloop-part1*, *imloop-part2*, *oclosed*. Für unsere Implementierung ist dies die optimale Anzahl der Tabellen. Jede Tabelle speichert in einem Eintrag nur einen Wert. Daraus folgt, dass der Speicherplatzbedarf hier in $O(m^2)$ liegt, wobei m hier die Länge der Eingabesequenz ist, sprich die Länge der kodierten Sequenz. Wie schon oben erwähnt, speichern wir hier unnötige Tabelleneinträge durch die Kodierung und erhalten somit eine schlechter Laufzeit als bei einer Implementierung, die nur die Tabelleneinträge für ursprüngliche Sequenz speichert. Auch hier sollte zukünftig eine optimiertere Implementierung erfolgen.

6.3 ZUSAMMENFASSUNG

Am Anfang des Kapitels haben wir die Berechnung der Basenpaar-Wahrscheinlichkeiten vorgestellt. Dabei werden die Partitionsfunktion aller sekundären RNA-Strukturen mit Basenpaar (i, j) und die Partitionsfunktion aller sekundären RNA-Strukturen benötigt. Die Partitionsfunktion aller sekundären RNA-Strukturen mit Basenpaar (i, j) setzt sich zusammen aus der Partitionsfunktion der Innerfragmente mit Basenpaar (i, j) und der Partitionsfunktion der Outerfragmente mit Basenpaar (i, j).

Die Berechnung der Basenpaar-Wahrscheinlichkeiten folgt dann aus den entsprechenden Tabellen der Inner- und Outerfragmente, die die Nichtterminalen der Grammatik speichern.

Als Nächstes gingen wir auf unsere Implementierung ein. Das verwendete Energiemodell ist vereinfacht, so dass nur die Basenpaare ohne Gewichtung aufsummiert werden. Eine Implementatierung mit loop-basierten Energien war im Rahmen dieser Masterarbeit nicht mehr möglich.

Wir haben eine Zeitanalyse für zufällige RNA-Sequenzen gemacht. Ein Vergleich mit anderen Implementierung ist nicht realistisch, da durch die Verwendung der vereinfachten Energiefunktionen keine biologisch relevante Strukturverteilung berechnet wird. Wir speichern nur die Werte der Nichttermialen - *istruct*, *iclosed*, *imloop-part1*, *imloop-part2*, *oclosed* - in Tabellen. Dies entspricht einem Speicherplatzbedarf von $O(n^2)$.

Teil V

SCHLUSS

7 ZUSAMMENFASSUNG

7.1 AUSBLICK

In diesem Abschnitt geben wir ein paar Anregungen für weitere zukünftige Projekte.

Unsere Implementierung der Berechnung der Partitionsfunktion mittels algebraischer dynamischer Programmierung verwendet eine vereinfachte Energiefunktion, die die Basenpaare der sekundären RNA-Strukturen aufsummiert. Eine Implementierung der Energiefunktionen, die das Energiemodell von Turner verwendet, wäre erstrebenswert. Dafür müssten die Energiebibliothek, die von Stefan Janssen für Bellman's Gap [10] programmiert wurde, für Outerfragmente angepasst werden. Danach wäre eine umfangreiche Analyse sinnvoll. Reale RNA-Sequenzen sollten getestet werden. Auch ein Vergleich mit anderen Implementierungen sollte erfolgen. Eins der bekanntesten Implementierungen ist Vienna RNAfold [3]. RNAfold berechnet auch die Basenpaar-Wahrscheinlichkeiten und stellt diese in einem Dotplot dar. Für einen Vergleich mit anderen Implementierungen muss jeweils sichergestellt sein, dass die verwendeten Energiemodelle gleich sind.

Eine weitere Idee ist die Erweiterung unserer Implementierung durch andere Algebren. Eine Algebra zur Darstellung der berechneten RNA-Strukturen mittels Shapes ist zum Beispiel möglich.

Unsere Implementierung benötigt eine kodierte Eingabe. Das Problem kann durch Multitracking gelöst werden. Multitracking bedeutet, dass eine Grammatik mehr als nur eine Eingabesequenz verarbeitet. Um die Kodierung der Eingabe zu umgehen, ist es nötig die selbe Sequenz gleichzeitig zweimal einzulesen. Dies erfordert eine entsprechende Veränderung der Grammatik. Das Prinzip des Multitrackings wird in der Promotionsarbeit von G. Sauthoff [9] vorgestellt.

Ein zukünftiges Projekt wäre die Möglichkeit in der Eingabesequenz auch Gaps zu erlauben. Die schon vorhandenen Energiefunktionen in Bellman's Gap berücksichtigen solche RNA-Sequenzen schon, falls Gaps keinen Einfluss auf die Energien haben.

Auch kann die Idee der Aufteilung in Inner- und Outerfragmente für andere Optimierungsprobleme angewandt werden, zum Beispiel für den Algorithmus von Sankhoff [8].

7.2 ZUSAMMENFASSUNG

Wir haben die Basenpaar-Wahrscheinlichkeiten anhand der RNA-Partitionsfunktion berechnet. Die RNA-Partitionsfunktion wird mittels

algebraischer dynamischer Programmierung [2] ermittelt. Dafür benötigen wir eine kontext-freie Grammatik, die die Vorgehensweise der Berechnung der Partitionsfunktion einer kompletten RNA-Sequenz s der Länge n beschreibt.

Die Berechnung der Partitionsfunktion der Teilsequenz $[i, j]$ ist ohne Modifikation der Eingabe durch eine Grammatik möglich. Die Grammatik für diese Teilsequenzen, auch Innerfragmente genannt, betrachtet dabei alle sekundäre RNA-Strukturelemente - Hairpin, Stack, Interiorloop, Bulges und Multiloop.

Für die Berechnung der Partitionsfunktion der kompletten Sequenz sind Basenpaare (i', j') zu berücksichtigen, für die $i' \in [1, i-1]$ und $j' \in [j+1, n]$ gilt. Die Fragmente $[1, i-1] \cup [j+1, n]$ werden Outerfragment genannt. Die Berücksichtigung der Outerfragmente erfordert eine Kodierung der Eingabe, da zu jedem Nichtterminal in der Grammatik nur zusammenhängende Teilsequenzen gehören. Die Eingabe wird kodiert, indem wir die Sequenz verdoppeln und dazwischen ein Trennungssymbol schreiben. Ein Basenpaar gehört zu einem Outerfragment, wenn unterhalb des Basenpaars das Trennungssymbol steht. Outerfragmente teilen wir in die sekundären RNA-Strukturelemente Pair, Stack, Interiorloop, linker Bulge, rechter Bulge und Multiloop ein. Die Grammatik der Outerfragmente berechnet für jede dieser sekundären RNA-Strukturelemente die Partitionsfunktion.

Die Grammatik der Innerfragmente und die der Outerfragmente können leicht zusammengesetzt werden. Die einzige Bedingung dabei ist, dass die Grammatik immer nur eine Teilsequenz der kodierten Eingabe mit Länge $n+1$ betrachtet.

Für die Berechnung der Basenpaar-Wahrscheinlichkeiten nach dem Algorithmus von McCaskill [6] werden die berechneten Werte der Partitionfunktionen benötigt. Die dafür notwendigen Werte stehen in den Nichtterminalen, die diese in Tabellen speichern. Wir greifen auf die entsprechenden Tabelleneinträge zu und berechnen so die Wahrscheinlichkeiten. Für jedes Basenpaar (i, j) wird sowohl die Partitionsfunktion des Innerfragments (i, j) als auch die Partitionsfunktion des Outerfragments (i, j) benötigt.

Unsere Implementierung ist gegenüber einer Implementierung mit dynamischer Programmierung flexibler und modularer. Sie kann leichter durch andere Funktionen erweitert werden. Dafür sind nur Algebren notwendig, die die Funktionen innerhalb der Grammatik neu spezifizieren. Die Grammatik muss für viele Erweiterungen nicht verändert werden.

LITERATURVERZEICHNIS

[1] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[2] Robert Giegerich and Carsten Meyer. Algebraic dynamic programming. *Algebraic Methodology And Software Technology*, 2002.

[3] Ivo L. Hofacker. Vienna RNA secondary structure server. *Nucleic Acids Research*, 31:3429–3431, 2003.

[4] Donald E. Knuth. Computer programming as an art. *Communications of the ACM*, 17(12):667–673, December 1974.

[5] D. H. Mathews, J. Sabina, D. H. Turner, and M. Zuker. Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. *J. Mol. Biol.*, 1999.

[6] J.S. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29:1105–1119, 1990.

[7] Ruth Nussinov Pieczenik, Jerrold R. Griggs, and Daniel J. Kleitman. Algorithms for loop matchings. *SIAM Journal on Applied Mathematics*, 1978.

[8] David Sankoff. Simultaneous solution of the RNA folding, alignment and protosequence problems. *SIAM Journal on Applied Mathematics*, 5, 1985.

[9] Georg Sauthoff. *Bellman's GAP: A 2nd Generation Language and System for Algebraic Dynamic Programming*. PhD thesis, Bielefeld University, 2011.

[10] Georg Sauthoff. Bellman's GAP. `https://gapc.eu`, 2012.

[11] Peter Steffen and Robert Giegerich. Versatile and declarative dynamic programming using pair algebras. *BMC Bioinformatics*, 6(1):224, September 2005.

[12] Micheal Zuker and Patrick Stiegler. Optimal computer folding of large RNA sequence using thermodynamics ans auxiliary information. *Nucleic Acids Reasearch*, 9, 1981.

ERKLÄRUNG

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Freiburg, 23. Februar 2012

Kerstin Schmatzer