technische universität
dortmund

Entwurf von Oligonukleotid-Bibliotheken für die DNA-Nanotechnologie

Marianna D'Addario

Algorithm Engineering Report
TR11-4-003
Mai 2011
ISSN 1864-4503

Fakultät für Informatik
Algorithm Engineering (LS 11)
44221 Dortmund / Germany
http://ls11-www.cs.uni-dortmund.de/

technische universität
dortmund

Diplomarbeit

Entwurf von Oligonukleotid-Bibliotheken für die DNA-Nanotechnologie

Marianna D'Addario
10. März 2011

Betreuer:
Prof. Dr. Sven Rahmann
Dipl.-Inf. Nils Kriege

Fakultät für Informatik
Algorithm Engineering (Ls11)
Technische Universität Dortmund
http://ls11-www.cs.uni-dortmund.de

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation und Hintergrund

Der Begriff Nanotechnologie gewinnt zunehmend an Bedeutung und findet in immer mehr wissenschaftlichen Gebieten Beachtung und Anwendung. Die Herstellung von Materialien im Nanometerbereich wird allgemein als *Nanotechnologie* bezeichnet. Zur Verdeutlichung: Ein Nanometer ist ein milliardstel Meter (1 nm = 10^{-9} m). Diese Größenordnung stellt für die Wissenschaft eine der bedeutsamsten Herausforderungen dar, da viele physikalische Eigenschaften auf dieser Ebene nicht mehr gelten. Ein Beispiel dafür ist das Element Gold, welches für seine Reaktionsträgheit bekannt ist. Werden aber Goldpartikel im Nanometerbereich, sogenannte Goldcluster, betrachtet, so stellt sich heraus, dass diese sehr reaktionsfreudig sind und sogar als Katalysator eingesetzt werden können [Sei08]. Während Chemie und Physik die geforderten Grundlagen zur gezielten Synthetisierung dieser Materialien liefern, ist unter anderem in der Elektrotechnik und der Medizin die Anwendung von Nanomaterialien zu finden [PCF+04]. Die Nanotechnologie betrifft laut [BJ07] die Spitzenforschung in weit gefächerten, verschiedenen Wissenschaftsbereichen. Im wesentlichen sind das die Wissenschaftsbereiche Quantenphysik, Materialwissenschaften, Elektronik, Informatik, Chemie, Mikrobiologie, Molekularbiologie, Zellbiologie und Medizin. Da nicht immer gemeinsame Ziele verfolgt wurden, haben sich in den vergangenen Jahren die verschiedenen wissenschaftlichen Gebiete unabhängig voneinander entwickelt. Deshalb ist die Kommunikation zwischen den einzelnen Disziplinen nicht einfach. Die jeweiligen Arbeits- und Denkweisen, verstärkt durch die unterschiedlichen Fachsprachen, sind ein weiterer Grund für die schwierige Kommunikation. Wenn die Kommunikation zwischen diesen Bereichen schon nicht einfach ist, dann entsteht ein noch größeres Kommunikationproblem zwischen Wissenschaft und der Öffentlichkeit, die sich über Nanotechnologie informieren möchte.

Nanomaterialien werden aus verschiedenen Atomen synthetisiert. Zum Beispiel können Schaltkreise im Nanometerbereich synthetisiert werden, die zur Herstellung für hochwertige Mikrochips verwendet werden. Ein besonderer Anwendungsbereich findet sich in der Prothetik, wo bioverträglichere Oberflächen für Prothesen und Organe geschaffen werden können [BJ07].

Die vier Nukleotide der DNA, benannt nach ihren Basen (A) Adenin, (C) Cytosin, (G) Guanin und (T) Thymin, sind unter anderem auf Grund ihrer Größenordnung für die Nanotechnologie interessant. Die bekannte DNA-Doppelhelix formt sich aus diesen vier Einheiten und kodiert durch die Reihenfolge der Einheiten die gesamte Erbinformation eines Organismus. Der Durchmesser einer DNA-Doppelhelix beträgt zwei Nanometer und jedes Nukleotid trägt $0,34$ Nanometer zur Länge bei [Kni06]. Die überaus gute Eignung

von DNA als Nanomaterial liegt aber vor allem an ihrer hohen Stabilität in verschiedenen Umgebungen, den einfachen Regeln der Basenpaarung, der Fähigkeit eines einzelnen DNA-Stranges einen komplementären Strang mit hoher Affinität zu binden und der reversiblen Hybridisierung der Basen durch gezielte Temperaturveränderungen [Shi09]. Die DNA-Nanotechnologie beschäftigt sich damit Nanostrukturen aus den Nukleotiden der DNA zu synthetisieren. Zuerst muss bestimmt werden welche Strukturen synthetisiert werden sollen, um anschließend die Kriterien zur Auswahl der einzelnen Bauteile festzulegen. Diese Bauteile sind DNA-Stränge, die gezielt modelliert werden müssen. Der Entwurf von Oligonukleotid-Bibliotheken für die DNA-Nanotechnologie vereinfacht die Auswahl elementarer Bausteine, die zur Herstellung synthetischer DNA-Nanostrukturen oder auch DNA-Strukturen notwendig sind.

1.2 Ziele und Aufbau der Arbeit

Der Forschungsbereich *Biologisch-Chemische Mikrostrukturtechnik* der Fakultät Chemie der TU Dortmund beschäftigt sich mit der Synthetisierung einzelner DNA-Kacheln. Diese Kacheln bestehen aus neun bis zehn Oligonukleotiden. Ziel der vorliegenden Arbeit ist es, in Zusammenarbeit mit der Fakultät Chemie, ein geeignetes Verfahren zu entwickeln, welches komplette Oligonukleotid-Bibliotheken erzeugt, verwaltet und Ergebnisse der DNA-Synthese festhält. Dabei steht die Verarbeitung von bereits existierenden Bibliotheken und ihren einzelnen Komponenten im Vordergrund. Die Bibliotheken sollen unter Angabe bestimmter, für die Synthetisierung wichtiger, Eigenschaften erstellt werden. Diese Bibliotheken bestehen aus den einzelnen Oligonukleotiden (relativ kurze DNA-Sequenzen), welche die Bausteine für komplexere Strukturen sind. Weiterhin sollen die Bibliotheken in einem passenden Format gespeichert werden können, um Ergebnisse zu archivieren und zukünftige Syntheseversuche dadurch zu vereinfachen.

1.2.1 Ziele der Arbeit

In einer graphischen Oberfläche sollen die einzelnen Oligonukleotide der Bibliothek aufgelistet und wichtige biochemische Eigenschaften angezeigt werden. Einzelne Oligonukleotide sollen aus den geladenen Bibliotheken ausgewählt und unter Berücksichtigung bestimmter Regeln variiert werden können. Es existieren zwei Arten der Modifikation von Oligonukleotiden, die von besonderem Interesse sind: Die Modifikation der *sticky ends* einer gegebenen Struktur und das Erstellen eines *Internals*. Modifikation bedeutet in diesem Zusammenhang das Finden von alternativen Oligonukleotiden. Im Falle der sticky ends werden Oligonukleotide gesucht, die alternativ zum gewählten Oligonukleotid dessen Platz in der Struktur einnehmen können. Für die Internals werden aus einem Oligonukleotid zwei verschiedene Oligonukleotide erstellt. Diese zwei Oligonukleotide sind durch bestimmte Teile voneinander abhängig und ersetzen zusammen das ursprüngliche Oligonukleotid in der DNA-Struktur. Zu einem Oligonukleotid existieren mehrere Internals, die verwendet werden könnten. Diese alternativen Oligonukleotide, die wiederum als Liste dargestellt werden sollen, können nach abgeschlossenen Laborversuchen bewertet werden. Die Bewertung der alternativen Oligonukleotide soll seitens der Fakultät Chemie bestimmt und in dafür vorgesehenen Feldern festgehalten werden. Weiterhin ist eine Analyse-Komponente wichtig, die es dem Benutzer ermöglichen soll den zu bearbeitenden Kontext hinsichtlich des Aufbaus der einzelnen Oligonukleotide zu analysieren. Die Analyse soll vor allem bei vorhandenen DNA-Strukturen Aufschluss über die Ursachen für Instabilität geben. Da ei-

nige DNA-Strukturen stabiler als andere sind, könnte die Zusammensetzung der einzelnen Oligonukleotide neue Erkenntnisse bringen.

1.2.2 Aufbau der Arbeit

Die Arbeit ist in sechs Kapitel gegliedert. In Kapitel 2 werden zunächst biochemische Grundlagen erklärt, die zum Verständnis der weiterführenden Problematik benötigt werden. Weiter werden auch graphentheoretische Grundlagen behandelt, die zur Lösung der Problematik benötigt werden. Bei dem Entwurf von Bibliothek schon bekannte Mängel von Oligonukleotiden zu vermeiden wirft einige Probleme auf, die zu einem Problem zusammengefasst werden. Kapitel 3 definiert genau diese Problematik als *DNA-Sequenzdesign-Problem* und stellt drei verwandte Arbeiten vor, die in Hinsicht auf Analogien und Unterschiede zum definierten Problem genauer untersucht werden. Im Einzelnen handelt es sich um die Arbeiten von *Seeman und Kallenbach* [SK83], *Anderson et. al* [AFN06] und *Udo Feldkamp* [FRB03]. Das DNA-Sequenzdesign-Problem wird im Zuge der Arbeit in fünf Teilprobleme unterteilt, die noch näher beschrieben werden. Kapitel 4 beschreibt einen ersten Ansatz zur Lösung des DNA-Sequenzdesign-Problems mittels *Integer linear Programming* und diskutiert für jedes der fünf Teilprobleme eine geeignete Lösung. Kapitel 5 stellt einen zweiten Ansatz zur Erstellung der Bibliotheken vor. Auch in diesem Kapitel werden für die einzelnen Teilprobleme Verfahren zur Lösung vorgestellt. Die aus Kapitel 4 und 5 resultierenden Lösungen zu den einzelnen Teilproblemen des DNA-Sequenzdesign-Problems werden im Tool *"ONlibrary - Oligonukleotid Bibliothek für die DNA-Nanotechnologie"* umgesetzt und in Kapitel 6 beschrieben. In diesem Kapitel werden die Teilprobleme aufgegriffen und anhand von Beispielen gelöst. Abschließend werden in Kapitel 7 die Ergebnisse dieser Arbeit zusammengefasst und ein Ausblick auf weitere Arbeiten gegeben.

Kapitel 2

Biochemische und graphentheoretische Grundlagen

Zum besseren Verständnis dieser Arbeit werden in diesem Kapitel Definitionen erläutert und Konventionen festgelegt. Es müssen biochemische Grundlage geschaffen und graphentheoretische Sachverhalte geklärt werden. Zunächst wird in Abschnitt 2.1 die biochemische Struktur der DNA näher erläutert. Wie sich die Arbeit im Rahmen der DNA-Nanotechnologie positioniert, wird in Abschnitt 2.3 beschrieben. Abschnitt 2.4 beinhaltet allgemeine Definitionen aus der Graphentheorie und Konventionen, die während dieser Arbeit verwendet werden.

2.1 DNA - Deoxyribonucleic acid

Die Desoxyribonukleinsäure, im Folgenden auch kurz DNA (von engl. “deoxyribonucleic acid“) genannt, ist ein Kettenmolekül. Ein DNA-Kettenmolekül wird auch Sequenz genannt. Dieses Kettenmolekül entsteht durch das Aneinanderreihen von Nukleotiden und kodiert die gesamten Erbinformationen eines Organismus. Die einzelnen Nukleotide bestehen wiederum aus drei Bestandteilen: Ein Phosphat, ein Monosaccharid (Zucker, bestehend aus fünf Kohlenstoffatomen) und eine der vier Basen [PRS98]. Der Zucker in einem Nukleotid der DNA ist die Desoxyribose und der Grund für das Präfix „Desoxyribo“. Die Basen sind Adenin (**A**), Cytosin (**C**), Guanin (**G**) und Thymin (**T**). Im Weiteren werden die einzelnen Nukleotide der Einfachheit halber nur noch mit den jeweiligen Abkürzungen A, C, G und T der Basen bezeichnet. Abbildung 2.1 schematisiert die allgemeine Struktur eines Nukleotids.

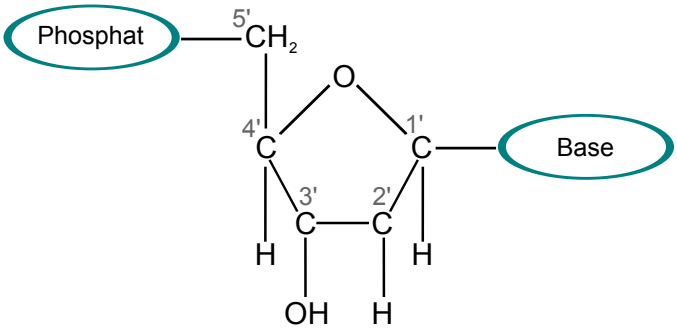


Abbildung 2.1: Nukleotidaufbau

Die Kohlenstoffatome der Desoxyribose sind, wie in Abbildung 2.1, für jedes Nukleotid gleich nummeriert. So können Verbindungen über die Nummerierung der Kohlenstoffatome, innerhalb der Desoxyribose, referenziert werden. Die Base ist immer an das erste Kohlenstoffatom gebunden, während das Phosphat am fünften Kohlenstoffatom befestigt ist. Ein Nukleotid unterscheidet sich von einem anderen nur anhand der jeweiligen Base. Die einzelnen Nukleotide einer Sequenz sind untereinander durch sogenannte **Phosphodiesterbindung** verbunden. Zwei Sequenzen können dann durch **Wasserstoffbrücken** verbunden sein, wenn diese komplementär zueinander sind. Die beiden Bindungsarten werden zunächst genauer beschrieben.

Die Phosphodiesterbindung ist die stärkere von beiden Bindungen. Jedes Nukleotid kann eine solche Bindung mit einem beliebigen anderen Nukleotid eingehen. An diese Bindung sind das Phosphat am Kohlenstoffatom $5'$ des ersten Nukleotids und die Hydroxylgruppe (OH) am Kohlenstoffatom $3'$ des zweiten Nukleotids beteiligt. Bei der Bildung dieser Verbindung verliert die Hydroxylgruppe das Wasserstoffatom und es formt sich ein sogenannter DNA-Einzelstrang. Die Phosphodiesterbindung ist das Rückgrat des Kettenmoleküls. Die Richtung eines DNA-Einzelstrangs $5' \rightarrow 3'$ sagt aus, dass am linken Nukleotid der Sequenz das Phosphat noch bindungsfähig ist. Die $5'$ bezieht sich dabei auf das fünfte Kohlenstoffatom des Zuckers, an dem das Phosphat gebunden ist. Das Nukleotid am rechten Ende der Kette ist noch durch seine Hydroxilgruppe am dritten Kohlenstoffatom bindungsfähig. Die Richtung $3' \rightarrow 5'$ beschreibt den umgekehrten Fall. Um diesen Sachverhalt zu verdeutlichen, wird jeweils $3'$ oder $5'$ vor die Sequenz oder hinter die Sequenz geschrieben. Ist das Nukleotid A der Sequenz ACT mit dem Nukleotid C über seine Hydroxylgruppe verbunden, so ist sein Phosphat noch bindungsfähig und die Sequenz wird wie folgt dargestellt: $5'$-ACT-$3'$

Wasserstoffbrücken verbinden zwei DNA-Einzelstränge, wenn diese antiparallel zueinander liegen. Zwei DNA-Einzelstränge liegen antiparallel zueinander, wenn das $5'$ Ende des einen Strangs beim $3'$ Ende des anderen liegt. Die Verbindungen zwischen den DNA-Einzelsträngen werden Hybridisierungen genannt. Zu jeder Base existiert ein **Komplement**. Das Komplement zu einem A ist ein T und umgekehrt. Für die Base G ist C das Komplement und umgekehrt. Das Komplement zu einer gewissen Base wird auch Watson-Crick Komplement genannt (nach seinen Entdeckern James D. Watson und Francis Crick). In der gesamten DNA-Doppelhelix sind nur zwischen diesen vier Paarungen Wasserstoffbrücken möglich, während die schon beschriebene Phosphodiesterbindung zwischen allen Paarungen möglich ist. Zwischen den Basen A und T zweier Nukleotide bilden sich stets zwei Wasserstoffbrücken. Die zwei komplementären Nukleotide C und G werden hingegen durch drei Wasserstoffbrücken verbunden. Die Bindung zwischen G und C ist deshalb auch stabiler als die zwischen A und T.

Die beiden Bindungsarten sind ursächlich für die bekannte Doppelhelix-Form der DNA. Die Einzelstränge entstehen durch die Phosphodiesterbindungen, die wiederum durch die Wasserstoffbrücken zu Doppelsträngen vereinigt werden. Die Stabilität der Doppelhelix wird dabei hauptsächlich durch die Summe der einzelnen Basenpaarungen gewährleistet. In Abbildung 2.2 sind beide Bindungstypen in einem Ausschnitt einer DNA-Doppelhelix zu sehen. Die gestrichelten Linien zwischen den Basen stellen die Wasserstoffbrücken und die durchgezogenen Linien zum Phosphat hin die Phosphodiesterbindungen dar. Wichtig ist, dass an jedem Ende der Doppelhelix immer eine Hydroxylgruppe und ein Phosphat zur weiteren Bindung frei stehen.

In den folgenden Abschnitten und Kapiteln wird eine vereinfachte Darstellung der DNA verwendet. Die Einzelstränge werden als String über dem Alphabet $\Sigma = \{A, C, G, T\}$ interpretiert und stellen somit die DNA-Sequenzen als Zeichenkette dar. Ein Einzelstrang

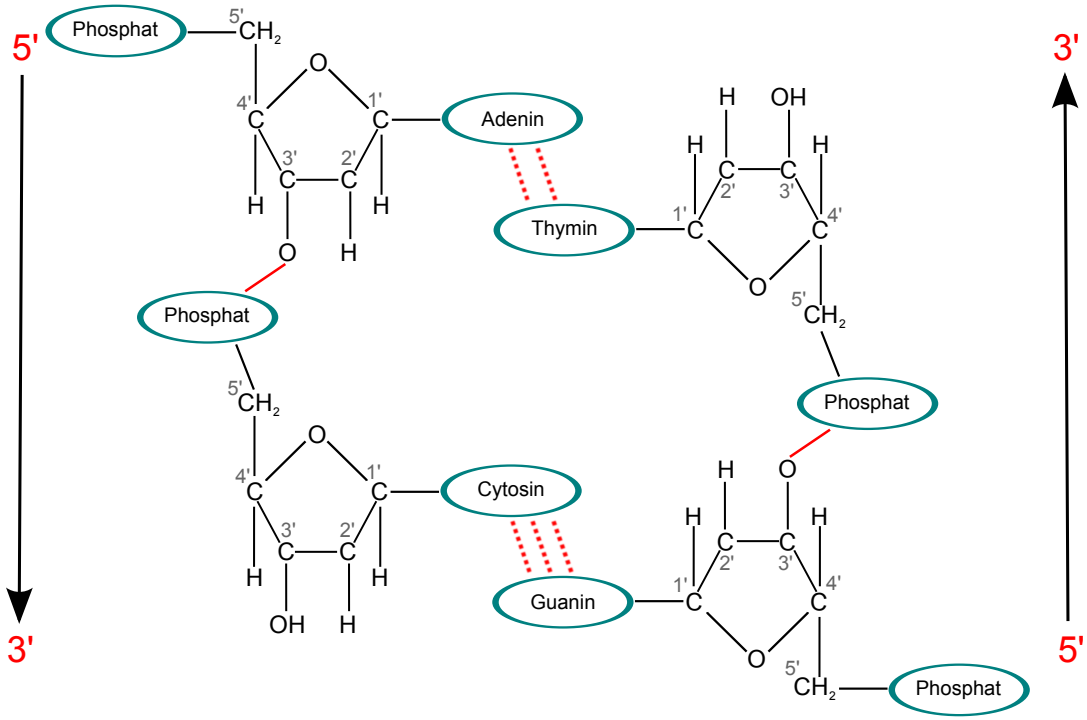


Abbildung 2.2: Ausschnitt einer DNA-Doppelhelix

kann in mehrere Subsequenzen aufgetrennt werden. Mit Subsequenz eines Einzelstrangs ist ein zusammenhängender Teilstring des Einzelstrangs gemeint. Im Folgenden wird eine Subsequenz auch Teilsequenz genannt. Sei $r = 5'$-CGTTGA-$3'$ der String zum betrachteten Sequenzabschnitt. Dann ist $\overline{r} = 3'$-GCAACT-$5'$ sein **Komplement** im Doppelstrang einer DNA-Doppelhelix. Zur Veranschaulichung der Komplementarität, können die Strings untereinander gestellt werden: $\begin{smallmatrix} 5'\text{-CGTTGA-}3' \\ 3'\text{-GCAACT-}5' \end{smallmatrix}$. Das **reverse Komplement** s^{C} einer beliebigen Sequenz $s \in \Sigma^*$ ist das Komplement gelesen vom $5'$ zum $3'$ Ende des Einzelstrangs. Für die obige Sequenz r ist $r^C = 5'$-TCAACG-$3'$. Während der weiteren Arbeit wird auf die Kennzeichnung des $5'$ und $3'$ Endes verzichtet und das linke Ende einer Sequenz wird immer als das $5'$ Ende interpretiert. Besteht eine Sequenz aus einer geraden Anzahl von Nukleotiden, kann das reverse Komplement auch die Sequenz selbst sein. Solche Sequenzen werden **selbstkomplementär** genannt.

2.2 Eigenschaften von DNA-Doppelsträngen

Für DNA-Doppelstränge Eigenschaften, die Aussagen über Reaktionsbereitschaft und Stabilität des vorliegenden Doppelstranges treffen. In den nächsten Unterabschnitten werden folgende drei Eigenschaften vorgestellt und erläutert: Der *prozentuale GC-Anteil*, die *Gibbs Energie* und die *Schmelztemperatur*.

2.2.1 Prozentualer GC-Anteil

Der prozentuale GC-Anteil (oder auch GC-Gehalt) einer Sequenz gibt an, wie viel Prozent der Sequenz aus G's und C's bestehen. Diese Eigenschaft ist ein Indikator für die Stabilität eines DNA-Doppelstanges, da wie in Abbildung 2.2 das Basenpaar GC durch drei statt zwei Wasserstoffbrücken verbunden ist. [FS06]. Mit Gleichung 2.1 lässt sich der GC-Gehalt

bei vorhandener Sequenz s berechnen. Das Zeichen # gibt die Anzahl der nachfolgenden Base in der betrachteten Sequenz s und $|s|$ die Länge von s an.

$$\%\mathrm{GC}_s = \frac{(\#\mathrm{G} + \#\mathrm{C})}{|s|} \cdot 100 \tag{2.1}$$

2.1 Beispiel. Berechnung des prozentualen GC-Anteils

Für die Sequenz s = 5′-CGTCGA-3′ ergibt die Formel einen GC-Gehalt von:

$$\%\mathrm{GC}_{\mathrm{CGTCGA}} = \frac{(2+2)}{6} \cdot 100 = \frac{4}{6} \cdot 100 \approx 67\%$$

Ein GC-Gehalt von 67% impliziert einen AT-Gehalt von 33%.

2.2.2 Gibbs Energie

Die Veränderung der Gibbs Energie, dargestellt mit ΔG, misst in kcal/mol den Austausch von Energie zwischen dem System, in diesem Fall der Doppelhelix, und dessen Umgebung. Ist ΔG negativ, so bedeutet dies, dass die Reaktion spontan abläuft, während bei positivem ΔG keine spontane Reaktion stattfindet. Die Veränderung ΔG bestimmt somit, ob eine Reaktion angestoßen werden muss oder ob diese spontan abläuft. Zur Bestimmung der Veränderung der Gibbs Energie wird das Nearest-Neighbor-Modell von *John Santa Lucia Jr.* [SJAS96] verwendet, welches im Folgenden näher erläutert wird.

Zunächst werden zu einer betrachteten Sequenz alle Teilstrings der Länge 2 in einer Liste L gespeichert. Für jede dieser Subsequenzen, listet Tabelle 2.1 die Werte ΔH°, ΔS° und ΔG°_{37} auf.

Sei $\Delta G^\circ_{37\mathrm{init}}$ der initiale Wert, $\Delta G^\circ_{37\mathrm{sym}}$ die symmetrische Korrektur, $\Delta G^\circ_{\mathrm{term\ AT}}$ die Strafe für eine AT-Basenpaar am Ende der Doppelhelix und $\Delta G^\circ_{37\ r}$ der Wert aus der Tabelle für das aktuell betrachtete 2-gramm r der Sequenz. Anhand der Werte aus Tabelle 2.1 kann mit Gleichung (2.2) die Gibbs Energie $\Delta G^\circ_{37}(\mathrm{total})$ der betrachteten Sequenz bei einer Temperatur von 37 °C berechnet werden.

$$\Delta G^\circ_{37}(\mathrm{total}) = \Delta G^\circ_{37\mathrm{init}} + \Delta G^\circ_{37\mathrm{sym}} + \sum_{r \in L} \Delta G^\circ_{37\ r} + \Delta G^\circ_{\mathrm{term\ AT}} \tag{2.2}$$

Die symmetrische Korrektur $\Delta G^\circ_{37\mathrm{sym}}$ wird nur dann aufsummiert, wenn es sich bei der betrachteten Sequenz um eine selbstkomplementäre Sequenz handelt. Die Strafe $\Delta G^\circ_{\mathrm{term\ AT}}$ wird hingegen nur dann aufsummiert, wenn ein Sequenz mit einem A oder T anfängt oder aufhört. Dann kommt am Ende oder am Anfang des Doppelstranges das Basenpaar AT vor. Ist an beiden Enden des Doppelstranges das Basenpaar AT vorhanden, dann wird die Strafe verdoppelt.

Tabelle 2.1: Thermodynamische Parameter für das Nearest-Neighbor-Modell für DNA Watson-Crick Paare in 1M NaCl, übernommen aus [SJH04]

Propagationssequenz	ΔH° $[\frac{\text{kcal}}{\text{mol}}]$	ΔS° $[\frac{\text{cal}}{\text{mol K}}]$	ΔG°_{37} $[\frac{\text{kcal}}{\text{mol}}]$
AA / TT	−7.6	−21.3	−1.00
AT / TA	−7.2	−20.4	−0.88
TA / AT	−7.2	−21.3	−0.58
CA / GT	−8.5	−22.7	−1.45
GT / CA	−8.4	−22.4	−1.44
CT / GA	−7.8	−21.0	−1.28
GA / CT	−8.2	−22.2	−1.30
CG / GC	−10.6	−27.2	−2.17
GC / CG	−9.8	−24.4	−2.24
GG / CC	−8.0	−19.9	−1.84
init	+0.2	−5.7	+1.96
term AT	+2.2	+6.9	+0.05
symmetrische Korrektur	+0	−1.4	+0.43

Die „Slashes“ geben an, dass die aufgeführten Sequenzen antiparallel gerichtet sind (z.B., AC/TG bedeutet: 5′−AC−3′ ist das Watson-Crick Basenpaar zu 3′−TG−5′). Die symmetrische Korrektur wird nur für selbstkomplementäre Sequenzen benötigt. Die „term AT“ Strafe wird angewandt, sobald ein Duplexende AT ist (hat der betrachtete Helixabschnitt an beiden Enden AT, so beträgt die Strafsumme +0.1 kcal/mol für ΔG°_{37}).

2.2 Beispiel. Berechnung der Gibbs Energie ΔG°_{37}

Sei 5′-CGTTGA-3′ der String zum betrachteten Sequenzabschnitt. Wird die Gleichung (2.2) entsprechend der Parameter aus Tabelle 2.1 angewandt, so ist:

$$\begin{aligned}\Delta G^\circ_{37\ \text{CGTTGA}} &= \Delta G^\circ_{37\text{init}} + \Delta G^\circ_{37\ \text{CG/GC}} + \Delta G^\circ_{37\ \text{GT/CA}} \\ +\Delta G^\circ_{37\ \text{TT/AA}} &+ \Delta G^\circ_{37\ \text{TG/AC}} + \Delta G^\circ_{37\ \text{GA/CT}} + \Delta G^\circ_{\text{term AT}} \\ &= 1.96 - 2.17 - 1.44 - 1.00 - 1.45 - 1.30 + 0.05 \\ &= -5.35\frac{\text{kcal}}{\text{mol}}\end{aligned}$$

Die symmetrische Korrektur $\Delta G^\circ_{37\text{sym}}$ entfällt in diesem Beispiel, da es sich nicht um eine selbstkomplementäre Sequenz handelt.

Die Gibbs Energie bei beliebiger Temperatur T wird mit Gleichung (2.3) berechnet und bedarf zwei weiterer Parameter, der Enthalpie und der Entropie. Die Enthalpie ΔH ist das Maß an Energie in kcal/mol in einem System und ist abhängig von der inneren Energie, dem Volumen und dem herrschenden Druck des Systems [LK08]. Die Entropie ΔS ist der Gesamtaustausch an Energie, oder in diesem Fall auch Wärme, zwischen System

und Umgebung cal/(mol K) [LK08]. Die Temperatur ist hierbei in Kelvin (K) angegeben.

$$\Delta G = \Delta H - T \cdot \Delta S \tag{2.3}$$

Die Enthalpie ΔH und die Entropie ΔS berechnen sich analog zu ΔG mit den Parametern aus Tabelle 2.1.

$$\Delta H^\circ = \Delta H^\circ_{\text{init}} + \Delta H^\circ_{\text{sym}} + \sum_{r \in L} \Delta H^\circ_r + \Delta H^\circ_{\text{term AT}} \tag{2.4}$$

$$\Delta S^\circ = \Delta S^\circ_{\text{init}} + \Delta S^\circ_{\text{sym}} + \sum_{r \in L} \Delta S^\circ_r + \Delta S^\circ_{\text{term AT}} \tag{2.5}$$

Wie schon für die Gibbs Energie erwähnt, fallen $\Delta H^\circ_{\text{sym}}$ und $\Delta S^\circ_{\text{sym}}$ nur an, wenn es sich um eine selbstkomplementäre Sequenz handelt. Die Strafen $\Delta H^\circ_{\text{term AT}}$ und $\Delta S^\circ_{\text{term AT}}$ werden auch hier vernachlässigt, wenn die Enden des Doppelstranges kein AT Basenpaar sind.

2.2.3 Schmelztemperatur

Die Schmelztemperatur T_{m} eines DNA-Doppelstranges ist direkt abhängig von dessen GC-Gehalt und ist definiert als die Temperatur, bei der 50% der Doppelhelix in denaturiertem Zustand (also einzelsträngig) vorliegt. Wie schon der GC-Ghalt ist auch die Schmelztemperatur ein Indikator für stabile Hybridisierungen und deswegen eine wichtige Eigenschaft zur Auswahl geeigneter Oligonukleotide [FS06]. Auch hier können die Parameter aus dem Nearest-Neighbor-Modell aus [SJH04] verwendet werden, um mit Gleichung (2.6) die Schmelztemperatur für beliebige Sequenzen zu errechnen.

$$T_{\text{m}} = \frac{\Delta H^\circ \cdot 1000}{\Delta S^\circ + R \cdot \ln\left(\frac{C_{\text{T}}}{x}\right)} - 273.15 \tag{2.6}$$

Hier ist C_{T} die Gesamtmolarität eines Stranges und R die Gaskonstante 1.9872 cal/(mol K). Die Konstante -273.15 ist zur Umrechnung von Kelvin in Grad Celsius. Der Wert der Variable x ist abhängig von der betrachteten Sequenz:

$$x = \begin{cases} 1, \text{ wenn Sequenz selbstkomplementär} \\ 4, \text{ sonst} \end{cases}$$

2.3 Beispiel. Berechnung der Schmelztemperatur T_{m}

Für eine nicht selbstkomplementäre Sequenz mit $\Delta H^\circ = -43.5 \frac{\text{kcal}}{\text{mol}}$, $\Delta S^\circ = -122.5 \frac{\text{cal}}{\text{mol K}}$ und einer Molarität von 0.2 mM für jeden Einzelstrang ergibt die Gleichung (2.6):

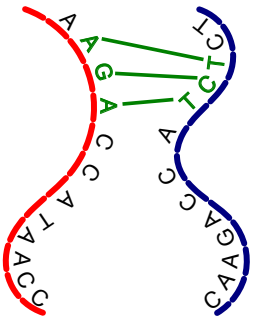
$$T_{\text{m}} = \frac{-43.5 \cdot 1000}{-122.5 + 1.9872 \cdot \ln\left(\frac{0.0004}{4}\right)} - 273.15 = 35.8\,^\circ\text{C}$$

2.3 DNA - Nanotechnologie

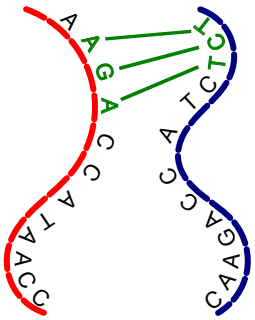
Nanotechnologie bezeichnet die gezielte Herstellung und/oder Manipulation einzelner Strukturen im Nanometerbereich. Ein Nanometer entspricht einem milliardstel Meter:
$1\ \text{nm} = 10^{-9}\ \text{m}$.
In Abschnitt 2.1 wurden die Grundlagen zu der Zusammensetzung einer DNA-Sequenz erläutert. Dieser Abschnitt klärt, wie sich die Arbeit in den Kontext der DNA-Nanotechnologie einordnen lässt. Ziel ist es, unübliche DNA-Motive über genau definierte Hybridisierungen miteinander zu komplexeren Strukturen zu kombinieren [Ami08]. Die Realisierung solcher Strukturen bedarf einer präzisen Planung der Zusammensetzung der elementaren Bausteine. Die Oligonukleotide müssen so konstruiert werden, dass sich diese zum Aufbau komplexerer Strukturen eignen. Der Artikel *Rationaler Entwurf von DNA-Nanoarchitekturen* [FN06] beschreibt einige DNA-Nanostrukturen, die durch Selbstorganisation von gezielt entworfenen Oligonukleotiden entstehen können. Die Selbstorganisation von DNA-Strukturen wird Assemblierungsprozess genannt. Die Oligonukleotide hybridisieren nur an den vorgesehenen Stellen und formen eine DNA-Struktur. Diese Struktur kann durch noch ungebundene Teile seiner Oligonukleotide mit weiteren Strukturen hybridisieren. Inhalt dieser Arbeit ist es, Verfahren und Methoden zu entwickeln, die es ermöglichen Oligonukleotide zur Erzeugung von DNA-Strukturen zu modellieren und schon vorhandene Oligonukleotide zu modifizieren. Die einzelnen Nukleotide lassen sich in beliebiger Reihenfolge und Anzahl zusammensetzen. Besteht eine Sequenz aus zehn Nukleotiden, wird in dieser Arbeit von einer Sequenz der Länge 10 gesprochen. Ein Nukleotid ist somit die kleinste Einheit und stellt einen elementaren Baustein dar, welcher zum Aufbau der nächstgrößeren Strukturen benötigt wird. Diese sind Oligonukleotide und werden wiederum weiter zu den komplexeren Strukturen kombiniert. Dieses Vorgehen wird Bottom-Up Methode genannt und ist das Gegenstück zur Top-Down Methode, die als Ausgangspunkt eine komplexe Struktur hat und daraus die elementaren Bausteine gewinnt.

2.3.1 Oligonukleotide zur gezielten Synthetisierung von DNA-Nanostrukturen

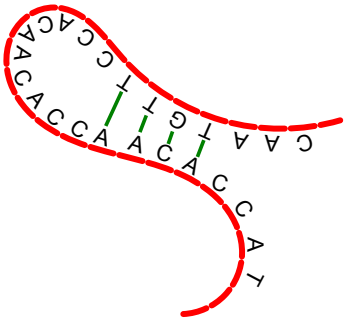
Oligonukleotide (oligo, aus dem griechischen „wenige“) sind relativ kurze Sequenzen aus ca. 10 bis 100 Nukleotiden. Zum Vergleich: Das menschliche Genom besteht aus 3 Milliarden Basenpaaren. Zur Erstellung von DNA-Nanostrukturen werden mehrere Oligonukleotide als Bausteine verwendet. Wie schon erwähnt ist ein Nukleotid $0,34$ nm lang und ein Oligonukleotid deshalb bis zu 34 nm lang. Das Modellieren von solchen Sequenzen besteht darin, die Reihenfolge der einzelnen Nukleotide gezielt zu bestimmen. Die Oligonukleotide müssen so modelliert werden, dass unerwünschte Hybridisierungen ausgeschlossen werden können. Die einzelnen Oligonukleotide einer DNA-Nanostruktur dürfen nur an vorgesehenen Abschnitten miteinander hybridisieren. Dazu ist es notwendig bestimmten Regeln beim Design der Oligonukleotide zu folgen. Zum einen soll innerhalb eines Oligonukleotids keine Subsequenz einer definierten Länge doppelt vorkommen, so dass ein bestimmter Abschnitt des Oligonukleotids gezielt zur Hybridisierung ausgewählt werden kann. Zum anderen soll das Komplement der vorhandenen Subsequenzen nicht in dem Oligonukleotid selbst vorkommen. Eine dritte Regel fordert den Ausschluss selbstkomplementärer Subsequenzen. Ist ein Oligonukleotid genau nach diesen Regeln modelliert, kann ein zweites Oligonukleotid in Abhängigkeit des Ersten modelliert werden, dass an gewünschter Stelle hybridisiert. Dazu muss das Komplement mindestens einer Subsequenz des ersten Oligonukleotids im zweiten Oligonukleotidvorkommen. Nun stellt sich die Frage: „Wieso müssen genau diese Regeln erfüllt werden?“. In Abbildung 2.3(c) ist ein Oligonukleotid mit kom-



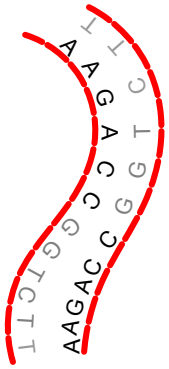
(a) Bindung zweier Oligonukleotide



(b) Alternative Bindung zweier Oligonukleotide



(c) Oligonukleotid mit komplementären Subsequenzen



(d) Selbstkomplementäre Oligonukleotid

Abbildung 2.3: Unerwünschte Hybridisierungen von Oligonukleotide in einer Bibliothek

plementären Subsequenzen der Länge 4 dargestellt. Dabei entsteht ein Nadelöhr, welches für eine Weiterverarbeitung problematisch ist. In Abbildung 2.3(a) und 2.3(b) sind zwei verschiedene Bindungsmöglichkeiten zweier Oligonukleotide aufgezeigt. Diese alternativen Bindungsstellen entstehen durch zwei identische Subsequenzen der Länge 3. Unter solchen alternativen Bindungsmöglichkeiten leidet die Stabilität der Bindung selbst und kann im Weiteren keine stabilen Strukturen ergeben. Abbildung 2.3(d) veranschaulicht die Verbindung zwischen zwei gleichen selbstkomplementären Oligonukleotiden. Die Verbindungen zwischen den einzelnen Basen wurden hierbei nicht skizziert. Das dargestellte Oligonukleotid besteht aus zwölf Basen. Die ersten sechs Basen sind komplementär zu den letzten sechs und farblich hervorgehoben. Eine Sequenz ist genau dann selbstkomplementär, wenn diese aus einer geraden Anzahl an Basen besteht und die erste Hälfte das Komplement zur zweiten ist. Da im Assemblierungsprozess mit Konzentrationen von Oligonukleotiden gearbeitet wird, also dasselbe Oligonukleotid mehrmals vorkommt, sind selbstkomplementäre Subsequenzen einer bestimmten Länge verboten. Diese Eigenschaften sind die Basis der später in Abschnitt 3.1 formulierten Regeln zum Entwurf von Oligonukleotiden.

Liegen zwei größere Strukturen bereit und sollen durch ein Oligonukleotid miteinander verbunden werden, kann unter Einhaltung der ersten beiden Eigenschaften sichergestellt werden, dass diese an genau bestimmbaren Stellen hybridisieren. Die vorliegenden Strukturen sind der Kontext und müssen beachtet werden. Unter Berücksichtigung der genannten Eigenschaften soll eine Auswahlmöglichkeit an Oligonukleotiden berechnet werden, die als Bindungsglieder fungieren. Diese Oligonukleotide dürfen den Kontext nicht durch zusätzlichen Bindungsmöglichkeiten destabilisieren.

2.3.2 Die DNA-Kacheln A^2 und B^3

In dem Artikel von *Saccá et al.* [SMF+08] wird beschrieben, wie sich Oligonukleotide zunächst zu größeren DNA-Strukturen selbstorganisieren. Diese DNA-Strukturen lassen sich kachelförmig darstellen und werden deshalb auch DNA-Kacheln genannt. Diese DNA-Kacheln formen anschließend ein supramolekulares Gitter, indem sie sich abwechselnd aneinanderreihen. Solche Gitter werden auch als DNA-Nanoarrays bezeichnet. Es werden dabei zwei Kacheln verwendet: Kachel A^2 und B^3. Diese bestehen jeweils aus neun Oligonukleotiden und sind in Abbildung 2.4 dargestellt.

Die Anbindung an die jeweils nächste Kachel geschieht über die sogenannten *sticky ends* (klebrige Enden), die ungebunden aus den vier Armen der Kachel ragen. Der Assemblierungsprozess der beiden genannten Kacheln kann auf zwei Wegen stattfinden. Es können zunächst die einzelnen Kacheln assembliert werden und anschließend ein neuer Prozess mit den schon fertigen Kacheln gestartet werden. Oder die insgesamt 18 Oligonukleotide können sich in nur einem Prozess zu einem DNA-Nanoarray assemblieren.

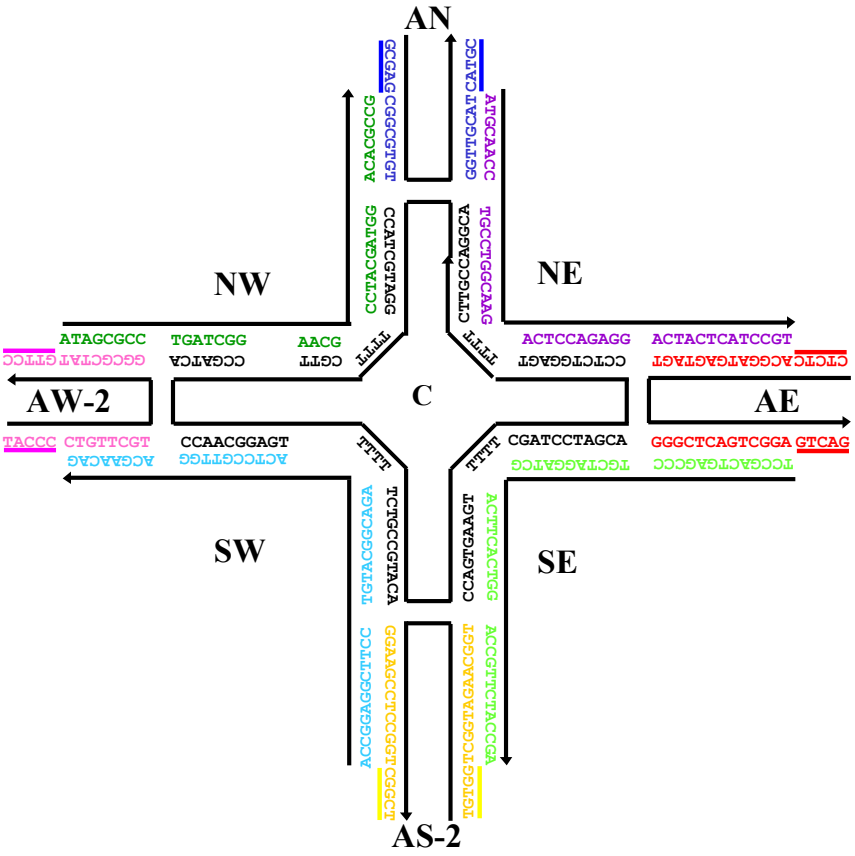
Die Kachel B^3 aus Abbildung 2.4(b) ist nicht so stabil wie sie theoretisch sein sollte. Ein Ziel ist es, die „sticky ends“ durch Alternativen auszutauschen. Die sticky ends einer Kachel sind ungebunden und suchen deshalb eine Möglichkeit zu hybridisieren. Dieser Fakt könnte eine Ursache für die Instabilität der gesamten Kachel sein. Alternative sticky ends könnten der veränderten Eigenschaften des gesamten Oligonukleotids weniger störend im Assemblierungsprozess sein. Die alternativen Oligonukleotide müssen entsprechend bestimmter Regeln erstellt werden. Diese Regeln werden in Kapitel 3 näher beschrieben und sorgen dafür, dass innerhalb der DNA-Struktur keine unerwünschten Hybridisierungen stattfinden. Es ist vorteilhaft die maximale Anzahl an alternativen Oligonukleotiden zur Ersetzung zu finden, da so unter den verschiedenen Möglichkeiten die stabilste ausgewählt werden kann. Eine weitere Möglichkeit ist es komplette Oligonukleotide auszutauschen. Auch hier ist es vorteilhaft alle möglichen Alternativen zu erhalten, um anschließend zu testen, mit welchem Oligonukleotid die DNA-Kachel stabil bleibt.

2.4 Graphentheorie: Definitionen und Konventionen

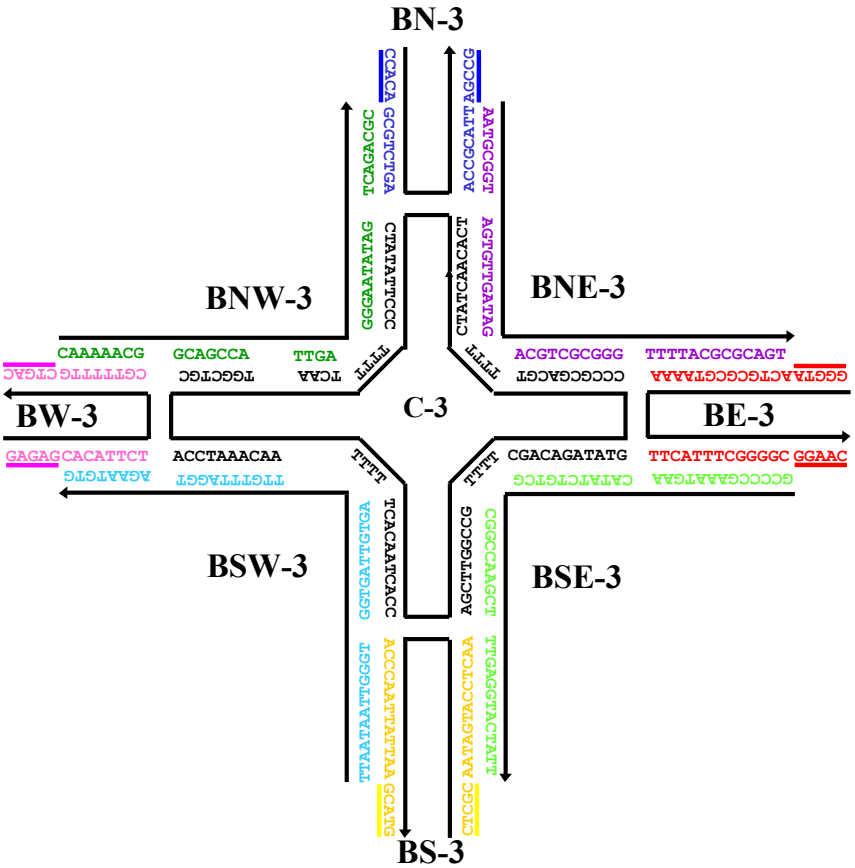
In verschiedensten Arbeiten zum Thema Sequenzdesign ist der Zusammenhang zwischen dem Erzeugen einer Sequenz und einem eigens dafür konstruierten Graphen ein wichtiges Konzept. So auch in den Arbeiten von *Udo Feldkamp* [FBR+00] [FRB03] [Fel09] und der Arbeit von *James W. Anderson et al.* [AFN06], welche in Abschnitt 3.3 näher erläutert werden. Im folgenden Unterabschnitt werden zunächst grundlegende Begriffe der Graphentheorie definiert. Anschließend werden zwei graphentheoretische Probleme vorgestellt. Das Durchlaufen eines Graphen kann in direktem Zusammenhang mit dem Aufbau einer Sequenz gesehen werden und wird in einem weiteren Unterabschnitt genauer beschrieben.

2.4.1 Graphen

Einleitend wird der Begriff Graph im Allgemeinen eingeführt. Es existieren zwei Arten von Graphen, die im Folgenden verwendet werden: Der *ungerichtete Graph* und der *gerichtete Graph*. Ein Graph besteht aus einer Menge von Knoten und einer Menge von Kanten. Dabei sind Kanten Verbindungen zwischen zwei Knoten. Mehrere Kanten zwischen zwei Knoten sind nicht zulässig. In dieser Arbeit wird für jeden Graphen erlaubt, dass eine Kante einen Knoten mit sich selbst verbindet. Diese Kanten werden Schleifen genannt. Formal werden die beiden Graphen wie folgt definiert.



(a) Kachel A^2



(b) Kachel B^3

Abbildung 2.4: Die Kacheln aus dem Artikel [SMF+08]

2.4 Definition. Ungerichteter Graph

Ein ungerichteter Graph $G = (V, E)$ besteht aus zwei Mengen: $V = \{v_1, v_2, \ldots\}$, die Menge der Knoten und $E = \{e_1, e_2, \ldots\}$, die Menge der Kanten.
Für die Menge der Kanten gilt:

$$E \subseteq \{\{\{u, v\}\} | u, v \in V\} \tag{2.7}$$

2.5 Definition. Gerichteter Graph

Ein gerichteter Graph ist ähnlich zu einem ungerichteten Graph definiert: Der Unterschied liegt darin, dass die Kanten zwischen zwei Knoten gerichtet sind und deswegen die Menge $\{u, v\}$ aus Relation (2.7) geordnet ist.

$$E \subseteq \{(u, v) | u, v \in V\} \tag{2.8}$$

Die Richtung einer Kante e wird graphisch durch $\longrightarrow$ gekennzeichnet: $u \xrightarrow{e} v$ ist von u nach v gerichtet, u ist der Quell- und v der Zielknoten der Kante e.

In dieser Arbeit häufig verwendete Begriffe sind *Pfad* und *Kreis*. Die Verknüpfung mehrerer Kanten wird Pfad genannt. Pfade haben einen Start- und einen Endknoten. Sind Start- und Endknoten identisch, dann wird ein Pfad als Kreis bezeichnet. Zu beachten ist, dass bei Kanten von Quell- und Zielknoten gesprochen wird und bei Pfaden und Kreisen von Start- und Endknoten. Pfad und Kreis sind formal wie folgt definiert.

2.6 Definition. Pfad in einem Graphen nach *A. Brandstädt* [Bra94]

Ein **ungerichteter** Pfad P in einem ungerichteten Graphen ist eine Folge von Kanten $e_1, e_2, \ldots, e_k$ mit der Eigenschaft:

1. e_i und e_{i+1} haben einen gemeinsamen Knoten $v_i \in \{v_1, \ldots, v_{k-1}\}$.
2. Falls e_i keine Schleife ist und nicht e_1 oder e_k ist, so hat e_i einen Knoten mit e_{i-1} und den anderen Knoten mit e_{i+1} gemeinsam, $i \in \{2, \ldots, k-1\}$

P hat dann die Form $v_0 \overset{e_1}{—} v_1 \overset{e_2}{—} v_2 \ldots v_{k-1} \overset{e_k}{—} v_k$ (Pfad zwischen v_0 und v_k). Dabei heißen $v_0, \ldots, v_k$ die Knoten von P, $v_1, \ldots, v_{k-1}$ die inneren Knoten von P und $e_1, \ldots, e_k$ die Kanten von P.

Ein **gerichteter** Pfad P ist eine Folge von Kanten $e_1, e_2, \ldots, e_k$ und Knoten $v_0, v_1, \ldots, v_k$, so dass der Quellknoten v_{i-1} von e_i der Zielknoten von e_{i-1} ist, $i \in \{2, \ldots, k\}$:

$$v_0 \overset{e_1}{\rightarrow} v_1 \overset{e_2}{\rightarrow} v_2 \ldots v_{k-1} \overset{e_k}{\rightarrow} v_k$$

Die Länge $|P|$ eines Pfades P ist die Anzahl k seiner Kanten.

In der vorliegenden Arbeit ist ein Pfad immer kantendisjunkt. Ein Pfad beinhaltet jede Kante höchstens einmal.

2.7 Definition. Kreis in einem Graphen nach *A. Brandstädt* [Bra94]

Ein **ungerichteter** Kreis ist ein ungerichteter Pfad $P = (e_1, \ldots, e_k)$, für den $v_0 = v_k$ ist.

Ein **gerichteter** Kreis ist ein gerichteter Pfad $P = (e_1, \ldots, e_k)$ mit gleichen Start- und Endknoten $v_0 = v_k$.

Interessante Eigenschaften von Graphen sind der *Zusammenhang* eines Graphen und der *Grad* eines Knotens. Formal sind diese beiden Begriffe wie folgt definiert.

2.8 Definition. Zusammenhang eines Graphen nach *A. Brandstädt* [Bra94]

Ein Graph G heißt zusammenhängend genau dann, wenn für alle Knoten $u, v \in V, u \neq v$, ein Pfad zwischen u und v existiert.

2.9 Definition. Grad eines Knotens nach *A. Brandstädt* [Bra94]

Der Grad („degree“) $deg(v)$ eines Knotens $v \in V$ ist die Anzahl der zu v inzidenten Kanten. Dabei zählen Schleifen doppelt.
Für gerichtete Graphen existieren zwei Grade: Der Eingangsgrad („indegree“) $indeg(v)$ und der Ausgangsgrad („outdegree“) $outdeg(v)$ des Knotens v.

- $indeg(v)$ = Anzahl der Kanten $e \in E$ mit Zielknoten v.
- $outdeg(v)$ = Anzahl der Kanten $e \in E$ mit Quellknoten v.

Im Weiteren wird von zusammenhängenden Graphen ausgegangen, anderenfalls wird explizit darauf hingewiesen.

2.4.2 Eulerkreisproblem und Hamiltonkreisproblem

Das Eulerkreisproblem und das Hamiltonkreisproblem sind zwei klassische algorithmische Probleme der Graphentheorie. Dabei besteht das Eulerkreisproblem darin, Kreise in einem Graphen zu finden, die alle Kanten des Graphen enthalten. Im Gegensatz dazu, sucht das Hamiltonkreisproblem nach Kreisen im Graphen, die alle Knoten beinhalten. Die beiden Probleme werden wie folgt formalisiert.

2.10 Definition. Eulerkreis nach *A. Brandstädt* [Bra94]

Ein Eulerpfad eines Graphen mit $E = \{e_1, e_2, \ldots, e_k\}$ ist ein Pfad $P = (e_{i_1}, \ldots, e_{i_k})$, in dem jede Kante aus E in P genau einmal vorkommt.
Ein Eulerkreis eines Graphen ist ein Eulerpfad des Graphen, der ein Kreis ist.

2.11 Definition. Hamiltonkreis nach *A. Brandstädt* [Bra94]

Ein Kreis eines Graphen heißt Hamiltonkreis genau dann, wenn dieser alle Knoten des Graphen genau einmal enthält.

Ein ungerichteter Graph enthält nach dem Satz von Euler genau dann einen Eulerkreis, wenn jeder Knoten einen geraden Grad hat.

> **Satz.** *(Euler 1736)*
> *Ein ungerichteter Graph G=(V, E) hat genau dann einen Eulerpfad, wenn G bis auf isolierte Knoten zusammenhängend und die Zahl z der Knoten mit ungeradem Grad 0 oder 2 ist. Ist dabei z = 0, so hat G einen Eulerkreis und umgekehrt.*

Auf Basis dieses Satzes ist in [Bra94] ein Linearzeitalgorithmus zur Konstruktion von Eulerkreisen beschrieben. Einen Hamiltonkreis zu einem gegebenen Graphen zu finden ist **NP**-vollständig, was in [Bra94] bewiesen wird.

2.4.3 Zusammenhang zwischen Graphen und Sequenzdesign

Eine Graphenstruktur, sei diese gerichtet oder ungerichtet, kann zum Entwurf von Oligonukleotiden genutzt werden. Ein Eulerkreis ist eine geordnete Menge von Kanten. Sind diese Kanten mit DNA-Basen beschriftet, kann aus der Reihenfolge der Kanten eine Sequenz gelesen werden. Analog dazu können auch die Knoten mit den Basen beschriftet und ein Hamiltonkreis gesucht werden. Da aber für das Eulerkreisproblem bereits effiziente Algorithmen existieren, beschränkt sich diese Arbeit auf das Beschriften der Kanten. Das Wählen einer Kante entspricht dem Lesen eines DNA-Zeichens. So entsteht der Zusammenhang zwischen Pfad oder Kreis in einem Graphen und einer DNA-Sequenz durch das Ablaufen von Kanten, die mit Basen beschriftet sind. Auf Basis solcher Graphenstrukturen werden in den Kapiteln 4 und 5 zwei Vorgehensweisen vorgestellt, mit denen Sequenzen und Oligonukleotide entworfen werden können. Ein Pfad oder Kreis wird im Weiteren immer auch als DNA-Sequenz interpretiert.

Des Weiteren gelten folgende Konventionen:

- Eine DNA-Sequenz ist eine Zeichenkette aus den Buchstaben A, C, G und T. Im Weiteren sei $\Sigma = \{\text{A, C, G, T}\}$ das DNA-Alphabet.
- Einzelne **Zeichenketten** werden von Kleinbuchstaben wie r, s und t repräsentiert.
- Der griechische Kleinbuchstabe σ steht für ein **Zeichen**.
- Eine Zeichenkette besteht aus der Konkatenation mehrerer Zeichen: $\sigma_1 \circ \sigma_2 \circ \ldots \circ \sigma_n$.
- Die Konkatenation zweier Zeichenketten s und t ergibt eine neue Zeichenkette r: $r = s \circ t$.
- Das Konkatenationszeichen $\circ$ wird im Weiteren weggelassen.
- Eine **Menge** von Zeichenketten wird stets mit großen Buchstaben wie X, Y und Z dargestellt.
- Eine Zeichenkette wird im Folgenden String oder Sequenz genannt.
- Ein Teilstring einer Sequenz ist gleichbedeutend mit dem Begriff Subsequenz.

Kapitel 3

Das DNA-Sequenzdesign-Problem

Die folgenden Abschnitte fassen die in der vorliegenden Arbeit zu behandelnde Problematik zusammen. In Abschnitt 3.1 wird zunächst das DNA-Sequenzdesign-Problem im Allgemeinen definiert. Der Entwurf von Oligonukleotid-Bibliotheken für die DNA-Nanotechnologie beinhaltet mehrere Aufgabenstellungen, die als Teilprobleme des DNA-Sequenzdesign-Problems beschrieben werden können. Dazu werden anschließend in Abschnitt 3.2 die einzelnen Teilprobleme identifiziert und beschrieben. Abschnitt 3.3 stellt abschließend verwandte Arbeiten vor und zeigt Analogien sowie Unterschiede zum DNA-Sequenzdesign-Problem auf.

3.1 Definition des DNA-Sequenzdesign-Problems

Bei dem Entwurf von Oligonukleotiden müssen die in Abschnitt 2.1 beschriebenen Regeln eingehalten werden, da anderenfalls die erstellten Sequenzen unkontrolliert hybridisieren. Dazu muss zunächst die Länge der zu betrachtenden Subsequenzen festgelegt werden. Der Parameter q sei im Folgenden immer die Länge der betrachteten Subsequenzen. Der Entwurf einer Bibliothek von Oligonukleotiden erfordert auf Grund der in Abschnitt 2.1 erwähnten Fälle von unerwünschten Hybridisierungen die Einhaltung folgender Sequenzdesign-Regeln:

1. Jede Subsequenz der Länge q darf höchstens einmal vorkommen.
2. Das Komplement einer Subsequenz der Länge q darf keinmal vorkommen.
3. Selbstkomplementäre Subsequenzen sind verboten.

Im Artikel [Fel09] werden die ersten beiden Regeln zur Definition der **q-Eindeutigkeit** von DNA-Sequenzen genutzt. Eine DNA-Sequenz ist q-eindeutig, wenn jede Teilsequenz der Länge q einzigartig ist und das Komplement dieser Teilsequenz nicht vorkommt. Diese Eigenschaft lässt sich auch auf eine Menge von Sequenzen erweitern. Dabei darf eine Teilsequenz der Länge q höchstens einmal in der gesamten Menge auftauchen und das Komplement darf nicht in der gesamten Menge vorkommen. Die dritte Regel vermeidet, dass im Organisationsprozess von DNA-Strukturen gleiche Oligonukleotide hybridisieren. Beim Erzeugen wird mit Konzentrationen von Oligonukleotiden und nicht mit einzelnen Oligonukleotiden gearbeitet, weswegen es üblich ist, dass mehrere gleiche Oligonukleotide vorkommen. Das hier untersuchte Problem ist, unter Berücksichtigung der vorgestellten Regeln, die maximale Anzahl q-eindeutiger Oligonukleotide zu entwerfen. Dieses Problem kann mit Hilfe einer noch zu bestimmenden Graphenstruktur modelliert werden. So können verschiedene Lösungsansätze aus der Graphentheorie genutzt werden.

In Anlehnung an die Arbeiten von Udo Feldkamp aus Abschnitt 3.3 wird das hier behandelte Problem *DNA-Sequenzdesign-Problem* genannt. Im Folgenden wird die Abkürzung *DSD-Problem* verwendet. Das DSD-Problem besteht aus mehreren Teilproblemen, die in Abschnitt 3.2 identifiziert und detailliert erläutert werden. Wie die neu entworfenen Oligonukleotide aussehen und welche Länge diese haben sollen, hängt von dem jeweiligen Teilproblem ab, welches gelöst werden soll.

3.2 Identifikation einzelner Teilprobleme

Aus sequenzdesigntheoretischer Sicht gibt es zwei Teilprobleme, die sich aus der Definition des DSD-Problems ergeben. Sind keine Oligonukleotide vorgegeben, besteht das erste Teilproblem darin die längste q-eindeutige Sequenz zu suchen. Diese Sequenz kann anschließend in Oligonukleotide gewünschter Länge aufgeteilt werden. Bei bereits vorgegebenen Oligonukleotiden entsteht ein sogenannter Kontext, der beachtet werden muss. Das zweite Teilproblem sucht daher die längste Sequenz, ohne die vorgegebenen Subsequenzen der Länge q und ihre Komplemente beim Entwurf zu verwenden.

Aus Anwendersicht können drei weitere Teilprobleme identifiziert werden. Die in Abschnitt 2.3 beschriebenen DNA-Kacheln bestehen aus neun Oligonukleotiden, die den Kontext bilden. Das dritte Teilproblem besteht darin die sticky ends eines schon vorgegebenen Oligonukleotids aus dem Kontext zu ändern und den restlichen Kontext beizubehalten. Das vierte Teilproblem ist, ein Oligonukleotid aus dem Kontext so zu modifizieren, dass bei der Selbstorganisation der Kachel eine Arm senkrecht aus der Ebene hervorragt. Das Erstellen von kompletten DNA-Kacheln, wie sie in Abbildung 2.4 zu sehen sind, ist das fünfte Teilproblem.

In der Praxis sind bei der Modellierung von DNA-Strukturen $q \geq 7$ nicht denkbar. Bei $q = 7$ müssen q-gramme mit $q < 7$ nicht den Sequenzdesign-Regeln entsprechen und könnten gleichzeitig mit ihren Komplementen vorkommen. Auf Grund ihrer Länge und Häufigkeit könnten diese den Assemblierungsprozess erheblich beeinträchtigen. Deswegen ist es in der Praxis vorteilhaft die Länge der Subsequenzen so niedrig wie möglich zu wählen.

In den folgenden zwei Unterabschnitten werden diese fünf Teilprobleme der beiden Sichten genauer beschrieben.

3.2.1 Sequenzdesigntheoretische Probleme

Die längste q-eindeutige Sequenz, die bei einer Subsequenzlänge von q möglich ist, ist die Lösung zum ersten Teilproblem. Das Finden dieser Sequenz, unter Berücksichtigung der drei Regeln, erfordert eine Datenstruktur, die jede mögliche Subsequenz der Länge q enthält. Damit sind aber auch alle Komplemente der Subsequenzen vorhanden. Das Wählen einer bestimmten Subsequenz ist stets verbunden mit dem Löschen der komplementären Subsequenz. Diese Relation stellt ein Problem bei der Suche nach einem Pfad dar, da sich die Menge der noch wählbaren Subsequenzen bei jeder Wahl verändert.

Das zweite Teilproblem ist abgeleitet vom Ersten und entsteht durch eine zusätzliche Einschränkung. Es soll ein Kontext beachtet werden, der aus vorgegebenen Oligonukleotiden besteht. Jedes der vorgegebenen Oligonukleotide besteht aus einer bestimmten Anzahl an Subsequenzen der Länge q, die nun nicht mehr genutzt werden dürfen. Auch die Komplemente der Subsequenzen können nicht mehr genutzt werden, da die Sequenzdesign-Regeln

auch den Kontext mit einbeziehen. Ein Oligonukleotid der Länge l besteht aus maximal $n_{\max}$ verschiedenen Subsequenzen der Länge q.

$$n_{\max} = l - q + 1 \tag{3.1}$$

Aus n verschiedenen Subsequenzen der Länge q entsteht ein Oligonukleotid maximaler Länge $l_{\max}$.

$$l_{\max} = n + q - 1 \tag{3.2}$$

Die beiden sequenzdesigntheoretischen Probleme können wie folgt zusammengefasst werden:

1. Suche die längste q-eindeutige Sequenz ohne die Sequenzdesign-Regeln zu verletzen.
2. Suche, bei vorgegebenen Sequenzen beliebiger Länge, die längste q-eindeutige Sequenz. Die gefundene Sequenz mit den schon vorgegebenen Sequenzen darf keiner Sequenzdesign-Regel widersprechen.

3.2.2 Probleme aus Anwendersicht

Die Teilprobleme aus Anwendersicht ergeben sich aus bestimmten Aufgabenstellungen. Zudem sind die Sequenzdesign-Regeln nicht für alle zu entwerfende Oligonukleotide gültig, da sich die DNA-Kacheln durch gezielte Hybridisierungen selbstorganisieren sollen.

Wie schon in Abschnitt 2.3 beschrieben, ist die Modifikation der sticky ends eine der Hauptaufgaben. Das Austauschen der sticky ends ohne einen Regelverstoß verlangt eine Überprüfung der Überlappungen der Länge q zwischen den neu angefügten sticky ends und dem schon vorgegebenen Oligonukleotid. Das Ersetzen eines Zeichens innerhalb des zu ändernden sticky ends ist schon eine Modifikation, die ausgegeben werden soll. Die Länge der sticky ends ist für die in dieser Arbeit behandelten Kacheln immer fünf Nukleotide. In Abbildung 3.1 werden die sticky ends eines Oligonukleotids der Kachel B^3 näher betrachtet.

Einen Arm zu erstellen bedeutet ein Oligonukleotid an einer bestimmten Stelle in zwei Teilsequenzen zu teilen. Anschließend wird der ersten Teilsequenz eine Sequenz der Länge 28 angehängt. Diese Sequenz wird sich in der fertigen DNA-Kachel senkrecht zur Ebene der Kachel aufstellen. Vor den Anfang des zweiten Teils werden sechs Zeichen, die komplementär zu den ersten sechs Zeichen des Arms sind, gesetzt. Zudem werden zwei Thyminbasen und anschließend die zweite Teilsequenz angehängt. Die Anzahl der Thyminbasen sorgen durch ihre Verbindung für das stabile Aufrichten des Arms. Ein solcher

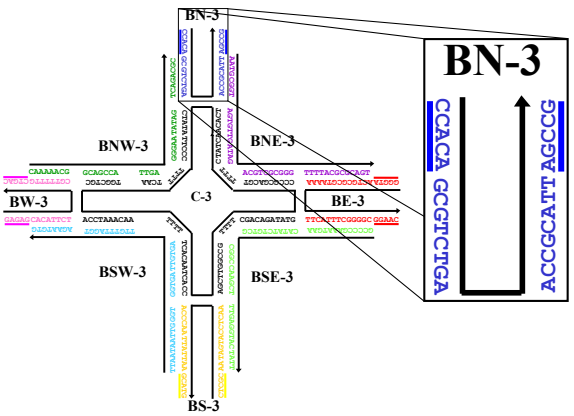


Abbildung 3.1: sticky ends der Kacheln B^3

Arm wird *Internal* genannt. Die zwei entworfenen Oligonukleotide entsprechen nicht den Regeln, da die Hybridisierung zwischen den bestimmten Stelle erzwungen werden. Da zur Bildung des Internal zwei Oligonukleotide notwendig sind, besteht eine DNA-Kachel mit Internal aus zehn Oligonukleotiden statt aus neun.

Ein weitere Aufgabenstellung aus Anwendersicht ist es, eine DNA-Kachel mit und ohne Internal zu erstellen. Dazu werden jeweils neun oder zehn Oligonukleotide benötigt, die an vorgegebenen Stellen hybridisieren. Die Vorgehensweise besteht immer aus zwei Schritten: Zunächst wird eine Kachel ohne Internal erstellt und dann kann mit der Lösung zum vierten Teilproblem das Internal hinzugefügt werden. In Abbildung 3.1 ist die Kachel B^3 zu sehen. Nach der Vorlage dieser Kachel sollen sollen weitere erstellt werden. Als Erstes muss zentrale Oligonukleotid der Kachel entworfen werden, das für die Kachel B^3 mit der Beschriftung C-3 versehen ist. Dieses ist genau 100 Zeichen lang. Das erste Teilproblem aus sequenzdesigntheoretischer Sicht hat als Lösung eine Sequenz, die alle Regeln beachtet. Diese Sequenz wird auf die Länge der zentralen Komponente gekürzt und ist somit der Ausgangspunkt zum Entwurf einer DNA-Kachel. Auch die Oligonukleotide mit den sticky ends werden auf diese Weise erstellt, da sie unabhängig von der zentralen Komponente sind und zusammen die drei Regeln respektieren müssen. Diese vier Oligonukleotide bestehen jeweils aus 26, 36, 26 und 36 Zeichen. Insgesamt muss die erzeugte Sequenz eine Länge von 208 Zeichen besitzen, um in diese fünf Komponenten aufgeteilt zu werden. Die weiteren Oligonukleotide werden in Abhängigkeit zu den vorgesehenen Hybridisierungen entworfen. Diese sollen bewusst nicht den Regeln entsprechend entworfen werden.

Um die Auflistung der Teilprobleme zu komplettieren, werden die eben beschriebenen drei Teilprobleme aus Anwendersicht wie folgt zusammengefasst:

3. Verändern der sticky ends. Alle noch möglichen Kombinationen für ein oder beide sticky ends ausgeben.
4. Verändern bzw. Hinzufügen eines Internals. Auch hier sollen alle verbleibenden Möglichkeiten ausgegeben werden.
5. Eine komplette Kachel den Sequenzdesign-Regeln entsprechend erstellen.

3.3 Verwandte Arbeiten

Zunächst werden drei wichtige verwandte Arbeiten vorgestellt, um im Weiteren eine genauere Abgrenzung zu dieser Arbeit zu schaffen. Das grundlegende Ziel ist es, einen Pool an Sequenzen zu schaffen, die eine hohe Unähnlichkeit aufweisen, damit eindeutig sind und keine unerwünschten Hybridisierungen stattfinden können. Als erstes ist die Arbeit von *Seeman und Kallenbach* zu nennen, die Sequenzen mit überlappenden Subsequenzen einer festen Länge entworfen haben und somit die Einzigartigkeit der Sequenz selbst verstärken konnten [SK83]. Des Weiteren ist in der Arbeit von *James W. Anderson et al.* ein wichtiges Konzept herausgearbeitet worden [AFN06], mit dem es möglich ist minimale Sequenzen zu generieren, die jede Subsequenz einer bestimmten Länge mindestens einmal beinhalten. *Udo Feldkamp* beschreibt in seinen Arbeiten die Programme **DNASequenceCompiler** und **DNASequenceGenerator**, die analog zu *Seeman und Kallenbach* eine Überlappung von Subsequenzen nutzen, um eine hohe Unähnlichkeit zwischen den entworfenen Sequenzen zu erreichen [Fel09] [FRB03].

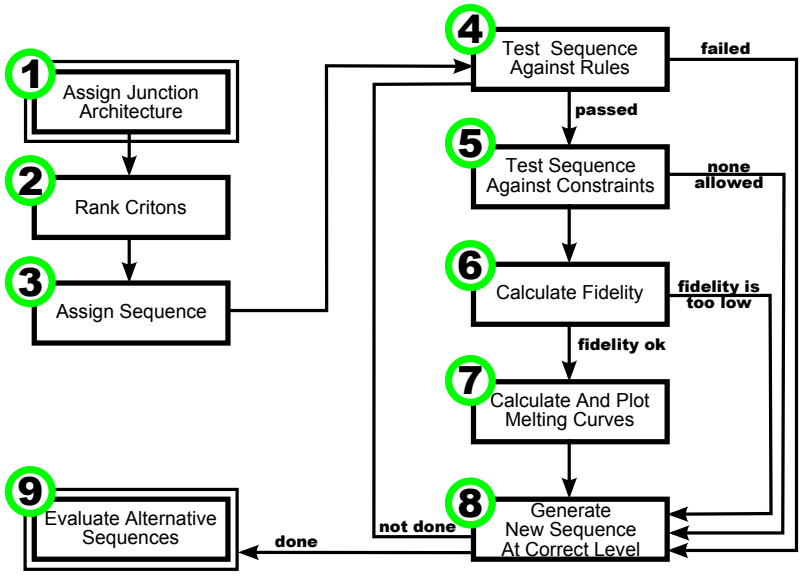


Abbildung 3.2: Schritte für die Optimierung von Sequenzen einer Kachel nach Seeman und Kallenbach [SK83]

3.3.1 *Seeman und Kallenbach* - Design Of Immobile Nucleic Acid Junctions

Seeman, ein Pionier im Bereich der DNA-Nanotechnologie, folglich auch im Entwurf von DNA-Sequenzen, setzte mit seinen Arbeiten neue Maßstäbe und entwickelte das Programm **SEQUIN**. Hier wird den Benutzern Hilfestellung beim Entwurf mehrarmiger Strukturen geboten. Der Ablauf des Entwurfs wird in neun Schritte schematisiert, siehe Abbildung 3.2. Die doppelt umrandeten Boxen sagen dabei aus, dass dieser Schritt nicht automatisch ausgeführt werden kann. In Schritt eins wird bestimmt, wie viele Oligonukleotide gebraucht werden, aus wie vielen Basen sie bestehen sollen, die Regionen, die hybridisieren sollen, welche Basen unabhängig und wie lang die Subsequenz sein sollen. Im zweiten Schritt werden Ränge an alle möglichen Subsequenzen der vorgegebenen Länge vergeben. Ist die Länge der Subsequenzen q, so existieren 4^q verschiedene Subsequenzen. Die initiale Sequenz wird im dritten Schritt gesetzt. Diese kann entweder standardmäßig eingestellt sein oder muss vom Benutzer gewählt werden. Der vierte Schritt besteht darin, die erzeugten Sequenzen den Regeln entsprechend zu testen. Jede Subsequenz darf einmal in den Sequenzen vorkommen, das Komplement einer Subsequenz darf nicht vorkommen, selbstkomplementäre Subsequenzen sind verboten und gleiche Basenpaare können höchstens zweimal an der inneren Kreuzung der Struktur auftauchen. Wird gegen eine der Regeln verstoßen, so werden neue Sequenzen erzeugt, indem die Subsequenz mit dem höchsten Rang, der in Schritt zwei festgelegt wurde, geändert wird. Der fünfte Schritt prüft, ob die vom Benutzer eingegebenen Einschränkungen eingehalten werden. Die „*fidelity*“ ist die relative Wahrscheinlichkeit der Bildung einer komplexeren Struktur in Relation zu alternativen Paarungen und wird im sechsten Schritt berechnet. Der siebte Schritt ist die Berechnung der Schmelztemperatur. Desto höher diese ist, desto stabiler sind die Bindungen der einzelnen Paarungen. Im achten Schritt wird wieder auf Schritt vier verwiesen, solange es noch weitere Subsequenzen niedrigeren Rangs gibt. Diese Schleife wird so oft wiederholt, bis es keine in Schritt zwei gelistete Subsequenzen mehr gibt. Dann wird zu Schritt neun übergegangen, in dem der Benutzer die Wahl trifft, welche Sequenzen seinen Anforderungen entsprechen und ausreichend stabil sind.

Analogien und Unterschiede zum DSD-Problem
Die Gemeinsamkeit der Arbeit von *Seeman und Kallenbach* mit dem DSD-Problem liegt in der Suche nach q-eindeutigen Oligonukleotiden zum gezielten Entwurf von DNA-Strukturen. Außerdem kann der Benutzer auch hier unter den verschiedenen Alternativen, die für ihn geeigneten auswählen. Doch es lassen sich auch einige Unterschiede zur Problemstellung identifizieren. Während im DSD-Problem nur Teile eines Oligonukleotids aus einer schon existierenden DNA-Struktur modifiziert werden sollen, werden hier unter Vorgabe einer initialen Sequenz komplette DNA-Strukturen entworfen.

3.3.2 *James W. Anderson et al.* - A Fast Algorithm For The Construction Of Universal Footprinting Templates

Die Autoren beschreiben einen Ansatz, der das Problem löst eine minimale Sequenz zu finden, in der jede Subsequenz einer bestimmten Länge mindestens einmal vorkommt. Auch hier wurde mit Subsequenzen einer festen Länge gearbeitet, die in einem selbstdefinierten Graphen dargestellt werden. Die Kanten des Graphen stellen dabei die Subsequenzen und gleichzeitig deren Komplemente dar und das Durchlaufen des Graphen erzeugt die gewünschte Sequenz, indem jede Kante mindestens einmal gelesen wurde. Es wird also ein Eulerkreis in dem selbstdefinierten Graphen gesucht. Der Algorithmus läuft generell in vier Schritten ab. Zunächst wird jeder Kante eines Knotens ihre „Partnerkante“ zugewiesen. Diese Zuweisung wird später bestimmen, welche Kantenabfolge beim Durchlaufen des Graphen existieren kann. Wird ein Knoten durch eine bestimmte Kante erreicht, so wird der Knoten durch die „Partnerkante“ wieder verlassen. Im zweiten Schritt wird der Graph durchlaufen, indem zunächst eine Kante zufällig gewählt und dann gemäß der „Partnerkante“ fortgeschritten wird. Die abgelaufenen Kanten werden als benutzt markiert. Es wird solange fortgefahren bis ein Knoten erreicht ist, der für die eingehende Kante keine „Partnerkante“ aufweist. An diesem Punkt muss überprüft werden, ob noch unbenutzte Kanten existieren. Ist dies der Fall, so wird ein zweiter Durchlauf gestartet und wie oben beschrieben fortgefahren, anderenfalls sind alle Kanten schon benutzt worden und der Schritt ist abgearbeitet. Dabei entsteht eine Menge von Kreisen, die im dritten Schritt über gemeinsame Knoten vereinigt werden. Der vierte Schritt besteht darin die Kanten abzulaufen und Zeichen um Zeichen die Sequenz zu erstellen. Beispiele zu dieser Vorgehensweise sind in Kapitel 5 beschrieben. Die Autoren unterscheiden zwei Fälle: Subsequenzen gerader und ungerader Länge. Für beiden Fälle wurde die Länge der gewünschten Sequenz bestimmt und bewiesen. Sei q die Länge der Subsequenzen. Für ungerade q und $q \geq 3$ hat die gesuchte Sequenz eine Länge n:

$$n = \frac{1}{2}4^q + q - 1$$

Für gerade q und $q \geq 4$ werden folgende Schranken angegeben:

$$min(n) = \frac{1}{2}4^q + 4^{n/2} + q - 4$$

$$max(n) = \frac{1}{2}4^q + q \cdot 4^{n/2-1}$$

Die selbstkomplementären Kanten im Graphen sind der Grund für diese Schranken. Diese Kanten sind dafür verantwortlich, dass Knoten mit ungeradem Grad entstehen und kein Eulerkreis im Graphen mehr möglich ist. Die Autoren beschreiben eine Möglichkeit Kreise

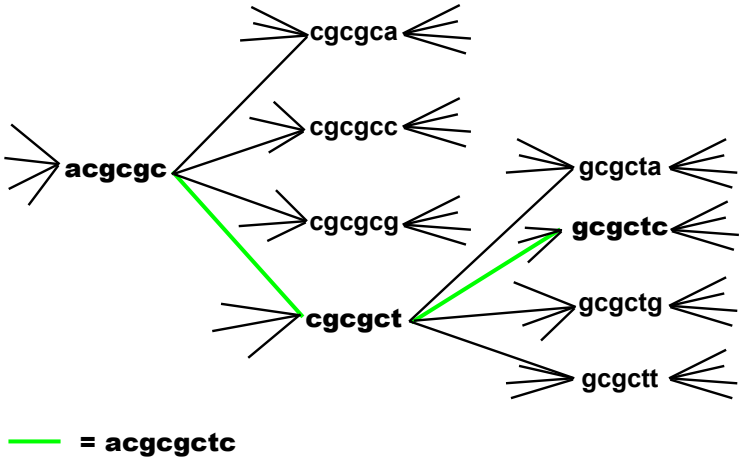


Abbildung 3.3: Graph mit Subsequenzen der Länge 6 aus dem Artikel [Fel09]

zu bilden, die einige Kanten doppelt wählen. Diese Kreise werden anschließend entfernt und es ist möglich in dem restlichen Graphen einen Eulerkreis zu finden. Der vorgestellte Algorithmus kann dann wie schon beschrieben fortfahren.

Analogien und Unterschiede zum DSD-Problem
Findet dieser Ansatz die minimale Sequenz zu einer gegebenen Länge, dann entspricht diese den ersten beiden Sequenzdesign-Regeln. Der verwendete Graph ist für das DSD-Problem geeignet, da zwei von den drei Regeln schon durch die Struktur respektiert werden. Der Unterschied ist, dass hier gefordert wird, dass jede Subsequenz der Länge q mindestens einmal enthalten ist. Im DSD-Problem hingegen wird gefordert, dass die Subsequenz höchstens einmal enthalten sind. Weiterhin ist in dieser Arbeit die Problematik der Selbstorganisation von DNA-Strukturen nicht behandelt. Auch sind selbstkomplmentäre Subsequenzen zur Konstruktion der geforderten Sequenz zulässig, während die dritte Regel des DSD-Problems diese explizit verbietet.

3.3.3 *Udo Feldkamp* - Software Tools For DNA Sequence Design

Der **DNASequenceGenerator** arbeitet auf einem gerichteten Graphen, der alle Subsequenzen der Länge q als Knoten darstellt. Zwischen Knoten existiert genau dann ein Kante, wenn die Subsequenzen der Länge $q-1$ der betrachteten Knoten identisch sind, siehe Abbildung 3.3. Der Algorithmus startet mit einem beliebigen Knoten und läuft einen beliebigen Pfad entlang, der die Sequenzdesign-Regeln respektiert. Die besuchten Knoten werden als benutzt markiert. Ist ein Knoten erreicht, der zu keiner unbenutzten Kante mehr adjazent ist, so kann durch „Backtracking“ ein anderer Pfad weiter verfolgt werden. Das Konzept des „Backtracking“ kann auch dazu genutzt werden die Schmelztemperatur oder den prozentualen GC-Gehalt zu beeinflussen. Ist die bisher gefundene Sequenz über oder auch unter einem bestimmten Wert, so kann zurück gesprungen werden, um einen neuen Pfad zu wählen, der die Bedingung nicht verletzt. Soll eine Subsequenz in der erzeugten Sequenz vorkommen, so kann der Knoten, welcher die Subsequenz darstellt, als erste Wahl für den Vorgängerknoten gesetzt werden.

Der **DNASequenceCompiler** nutzt formale Grammatiken, um Sequenzen zu erzeugen [FBR+00]. Die elementaren Bauteile des Compilers sind sogenannte *algomere*, deren Mittelstück ein Doppelstrang ist und zwei sticky ends besitzen. Mit regulären Ausdrücken der Form $A \to xB$ (mit A, B als Variablen und x als Terminal) werden aus den einzel-

nen algomere längere Sequenzen, *logomere* genannt, konstruiert, welche einem Wort der benutzten Grammatik entsprechen. Ein Beispiel ist in Tabelle 3.1 aufgeführt. Im Beispiel können vier algomere als Start gewählt werden, um dann mit den weiteren Regeln verschiedene logomere zu konstruieren, die alle mit dem selben algomer enden. Der im **DNASequenceGenerator** benutzte Algorithmus kommt hier beim Erzeugen von eindeutigen Sequenzen zum Einsatz, da die sticky ends der einzelnen algomere anderenfalls unkontrolliert hybridisieren könnten.

Analogien und Unterschiede zum DSD-Problem
Das grundlegende Ziel der vorgestellten Tools ist identisch mit der Aufgabenstellung des DSD-Problems. Gesucht ist eine Menge von q-eindeutigen Oligonukleotiden zum Entwurf von selbstorganisierenden DNA-Strukturen. Zusätzlich sind die Tools darauf ausgelegt beim Entwurf der Oligonukleotide auch biochemische Eigenschaften, wie beispielsweise das Erreichen eines festgelegten GC-Gehaltes, mit einzubeziehen. Die Berücksichtigung biochemischer Eigenschaften ist beim Entwurf von Oligonukleotiden in der Aufgabenstellung des DSD-Problems nicht vorgesehen.

Tabelle 3.1: Beispiel einer Grammatik zur Erstellung von logomeren aus [FBR+00]

Regel		*algomere*			
		Hind III	s_0	$\overline{A}$	
$S := s_0 A$	$5'$	agctt	caacacatggagttacacgc		$3'$
	$3'$	a	gttgtgtacctcaatgtgcg	gcctttgtag	$5'$
		Hind III	s_1	$\overline{A}$	
$S := s_1 A$	$5'$	agctt	gaaaaaattggactcggggc		$3'$
	$3'$	a	ctttttttaacctgagccccg	gcctttgtag	$5'$
		Hind III	s_2	$\overline{A}$	
$S := s_2 A$	$5'$	agctt	gctcctagaagtctacaagc		$3'$
	$3'$	a	cgaggatcttcagatgttcg	gcctttgtag	$5'$
		Hind III	s_3	$\overline{A}$	
$S := s_3 A$	$5'$	agctt	cttctgccatacaactaggc		$3'$
	$3'$	a	gaagacggtatgttgatccg	gcctttgtag	$5'$
		I	e	$\overline{BamHI}$	
$I \to e$	$5'$	gtcttgtgtc	cttgtttaatacaggggcgc	g	$3'$
	$3'$		gaagacggtatgttgatccg	cctag	$5'$
		A	0	$\overline{B}$	
$A \to 0A$	$5'$	cggaaacatc	ggatttggcaacaacctgag		$3'$
	$3'$		cctaaaccgttgttggactc	gaaaatcggg	$5'$
		A	1	$\overline{B}$	
$A \to 1A$	$5'$	cggaaacatc	caaccaggattaagccatgc		$3'$
	$3'$		gttggtcctaattcggtacg	gaaaatcggg	$5'$
		B	0	$\overline{C}$	
$B \to 0C$	$5'$	cttttagccc	ggatttggcaacaacctgag		$3'$
	$3'$		cctaaaccgttgttggactc	cctctaatgg	$5'$
		B	1	$\overline{C}$	
$B \to 1C$	$5'$	cttttagccc	caaccaggattaagccatgc		$3'$
	$3'$		gttggtcctaattcggtacg	cctctaatgg	$5'$
		C	0	$\overline{D}$	
$C \to 0D$	$5'$	ggagattacc	ggatttggcaacaacctgag		$3'$
	$3'$		cctaaaccgttgttggactc	ggcgtttatc	$5'$
		C	1	$\overline{D}$	
$C \to 1D$	$5'$	ggagattacc	caaccaggattaagccatgc		$3'$
	$3'$		gttggtcctaattcggtacg	ggcgtttatc	$5'$
		D	0	$\overline{E}$	
$D \to 0E$	$5'$	ccgcaaatag	ggatttggcaacaacctgag		$3'$
	$3'$		cctaaaccgttgttggactc	gtctcgtatg	$5'$
		D	1	$\overline{E}$	
$D \to 1E$	$5'$	ccgcaaatag	caaccaggattaagccatgc		$3'$
	$3'$		gttggtcctaattcggtacg	gtctcgtatg	$5'$
		E	0	$\overline{F}$	
$E \to 0F$	$5'$	cagagcatac	ggatttggcaacaacctgag		$3'$
	$3'$		cctaaaccgttgttggactc	gcatcttgac	$5'$
		E	1	$\overline{F}$	
$E \to 1F$	$5'$	cagagcatac	caaccaggattaagccatgc		$3'$
	$3'$		gttggtcctaattcggtacg	gcatcttgac	$5'$
		F	0	$\overline{G}$	
$F \to 0G$	$5'$	cgtagaactg	ggatttggcaacaacctgag		$3'$
	$3'$		cctaaaccgttgttggactc	ctgccaatag	$5'$
		F	1	$\overline{G}$	
$F \to 1G$	$5'$	cgtagaactg	caaccaggattaagccatgc		$3'$
	$3'$		gttggtcctaattcggtacg	ctgccaatag	$5'$
		G	0	$\overline{H}$	
$G \to 0H$	$5'$	gacggttatc	ggatttggcaacaacctgag		$3'$
	$3'$		cctaaaccgttgttggactc	gacttcactg	$5'$
		G	1	$\overline{H}$	
$G \to 1H$	$5'$	gacggttatc	caaccaggattaagccatgc		$3'$
	$3'$		gttggtcctaattcggtacg	gacttcactg	$5'$
		H	0	$\overline{I}$	
$H \to 0I$	$5'$	ctgaagtgac	ggatttggcaacaacctgag		$3'$
	$3'$		cctaaaccgttgttggactc	cagaacacag	$5'$
		H	1	$\overline{I}$	
$H \to 1I$	$5'$	ctgaagtgac	caaccaggattaagccatgc		$3'$
	$3'$		gttggtcctaattcggtacg	cagaacacag	$5'$

Kapitel 4

Ein De Bruijn Graph zur Lösung des DNA-Sequenzdesign-Problems

Ein erster Lösungsansatz für das DSD-Problem ist auf Basis eines De Bruijn Graphen ein ganzzahligen linearen Programms (integer linear Programming, kurz ILP) aufzustellen. Abschnitt 4.1 definiert zunächst den verwendeten De Bruijn Graphen und beschreibt dann Aufbau und Eigenschaften des Graphen. Anschließend wird in Abschnitt 4.2 das ILP formuliert und mögliche Lösungen aufgezeigt. Die Graphenstruktur des De Bruijn Graphen und das darauf aufgestellte ILP lassen sich für die einzelnen Teilprobleme des DSD-Problems anpassen. Für zwei Teilprobleme ist das ILP nicht nötig, da die Struktur des De Bruijn Graph ausreicht um diese zu lösen. Abschnitt 4.3 beschreibt die Vorgehensweisen zur Lösung der einzelnen Teilprobleme.

4.1 Ein De Bruijn Graph zur Erzeugung von DNA-Sequenzen: Aufbau und Eigenschaften

Der De Bruijn Graph ist ein gerichteter Graph, dessen Kanten und Knoten mit Wörtern beschriftet sind. Ist $\Sigma = \{A, C, G, T\}$ das Alphabet und q ein Parameter, dann sind die Knoten mit Wörtern aus Σ^{q-1} und die Kanten mit Wörtern aus Σ^q beschriftet. Im Zusammenhang mit dem Sequenzdesign sei q weiter die Länge der betrachteten Subsequenzen der zu erzeugenden q-eindeutigen Sequenz. Dazu sind die Kanten des De Bruijn Graphen dann mit allen möglichen Subsequenzen beschriftet. Ein Pfad in diesem De Bruijn Graphen, also eine Reihenfolge von Kanten, kann auch als Reihenfolge der Beschriftungen der Kanten interpretiert werden. Diese Reihenfolge von Subsequenzen wird zur Erzeugung einer Sequenz genutzt. Auf Basis dieser Idee soll im Folgenden ein erster Ansatz zur Lösung des DSD-Problems vorgestellt werden. Dabei soll ein Pfad im Graphen gefunden werden, der eine Sequenz gemäß der Sequenzdesign-Regeln erzeugt. Zunächst folgt die Definition des De Bruijn Graphen zur Erzeugung von DNA-Sequenzen, um anschließend die Struktur näher zu erläutern. Anhand eines Beispiels wird dann der Zusammenhang zwischen einem Pfad im De Bruijn Graph und einer erzeugten Sequenz verdeutlicht. Im Weiteren werden die zu betrachtenden Subsequenzen der Länge q als q-gramme bezeichnet.

4.1 Definition. De Bruijn Graph

Sei $\Sigma = \{A, C, G, T\}$ und q ein Parameter. Dann ist der De Bruijn Graph $G = (V, E)$ zu (Σ, q) durch die Menge aller Knoten V und die Menge aller Kanten E definiert.
Es existieren 4^{q-1} Knoten:

$$V = \Sigma^{q-1}.$$

Es existieren 4^q Kanten:

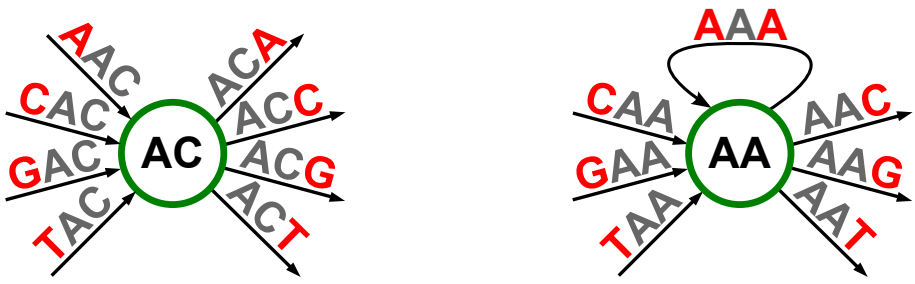
$$E = \{(\sigma_1 s, s\sigma_2) : \sigma_1, \sigma_2 \in \Sigma, s \in \Sigma^{q-2}, \sigma_1 s, s\sigma_2 \in V\}.$$

Jede Kante $e \in E$ mit Quellknoten $\sigma_1 s \in V$ und Zielknoten $s\sigma_2 \in V$ trägt die Beschriftung:

$$\sigma_1 s \sigma_2 : \sigma_1, \sigma_2 \in \Sigma, s \in \Sigma^{q-2}$$

Die Knoten des De Bruijn Graphen enthalten alle möglichen Wörter der Länge $q-1$, während die Kanten mit allen Wörtern der Länge q beschriftet sind. Eine Kante verbindet zwei Knoten genau dann, wenn das Suffix des Quellknotens der Länge $q-2$ gleich dem Präfix des Zielknotens ist. Sei der Knoten $v = \sigma_1 s$ der Quellknoten der Kante e und $u = s\sigma_2$ der Zielknoten der Kante. Dann trägt die Kante e die Beschriftung $\sigma_1 s \sigma_2$. Die so beschrifteten Kanten stellen somit alle möglichen Subsequenzen dar, die zur Erzeugung einer q-eindeutigen Sequenz zur Verfügung stehen. Jeder Knoten im Graph besitzt vier ein- und vier ausgehende Kanten, mit Ausnahme der Knoten, deren Beschriftungen ein Vielfaches eines Zeichens sind. Von diesen Knoten existieren im De Bruijn Graphen über dem Alphabet $\Sigma = \{A, C, G, T\}$ genau vier. Sie besitzen eine Schleife und jeweils drei ein- und ausgehende Kanten. Der Grad ist in beiden Fällen insgesamt acht. In Abbildung 4.1 sind beide Knotenarten beispielhaft dargestellt.

Die obere Schranke für die Anzahl maximal wählbarer Kanten und gleichzeitig möglicher Subsequenzen, um einen q-eindeutigen Pfad zu beschreiben, lässt sich einfach errechnen. Für ungerade q-gramme sind maximal $(4^q)/2$ Kanten wählbar, da genau die Hälfte aller Kanten komplementär beschriftet ist und deswegen nicht gleichzeitig im gesuchten Pfad enthalten sein kann. Bei geradem q entstehen genau $4^{q/2}$ selbstkomplementäre Sequenzen, die auf Grund der dritten Sequenzdesign-Regel nicht in einer q-eindeutigen Sequenz enthal-



(a) Normaler Knoten

(b) Knoten mit Vielfachen eines Zeichens

Abbildung 4.1: Die zwei Knotenarten des De Bruijn Graphen

Tabelle 4.1: Obere Schranke der Anzahl maximal wählbarer Kanten

	Gesamt-kanten	selbstkompl.-Kanten ($4^{q/2}$)	maximal wählbare Kanten
2-gramme	16	4	6
3-gramme	64	–	32
4-gramme	256	16	120
5-gramme	1024	–	512
6-gramme	4096	64	2016
7-gramme	16384	–	8192
8-gramme	65536	256	32640
9-gramme	262144	–	131072
10-gramme	1048576	1024	523776

ten sein dürfen. Für gerade q-gramme stellen deswegen maximal $(4^q)/2 - (4^{q/2})/2$ Kanten eine gültige Lösung dar. In Tabelle 4.1 sind diese oberen Schranken für $q \leq 10$ aufgelistet.

Die Kanten eines hier beschriebenen De Bruijn Graphen, können in zwei disjunkte Mengen aufgeteilt werden. In der ersten Menge sind alle wählbaren q-gramme enthalten. Die zweite Menge besteht aus allen q-grammen, die zu denen aus der ersten Menge komplementär sind. Für den Fall, dass q gerade ist, existiert eine dritte Menge, die alle selbstkomplementären q-gramme enthält. Die selbstkomplementären Kanten werden nicht zur Erzeugung einer q-eindeutigen Sequenz verwendet und deshalb nicht weiter betrachtet. Das Durchlaufen des De Bruijn Graphen ergibt genau dann eine q-eindeutige Sequenzen, wenn alle Kanten des erzeugten Pfades sich in einer der beiden Mengen befinden. Da die längste q-eindeutiger Sequenz gesucht wird, soll die Aufteilung der Kanten in die beschriebenen Mengen optimal sein. Die Kanten, die einer Menge angehören müssen dafür genau einen Pfad ergeben, der alle Kanten der Menge genau einmal abläuft.

Das erste Teilproblem des DSD-Problems aus Abschnitt 3.2 sucht die längste q-eindeutige Sequenz. Wie beschrieben ist es möglich, einen Pfad im De Bruijn Graphen zur Erstellung einer Sequenz zu verwenden. Dieser soll die maximale Anzahl an Kanten des De Bruijn Graphen ablaufen ohne die Sequenzdesign-Regeln zu verletzten. Sind Oligonukleotide einer vorgegebenen Länge gesucht kann die erzeugte Sequenz anschließend in Teilsequenzen der gewünschten Länge unterteilt werden. Diese Teilsequenzen sind dann auch q-eindeutig.

Ein kleiner Ausschnitt eines De Bruijn Graphen mit $q = 3$ ist in Abbildung 4.2 zu sehen. Der Ausschnitt ist so gewählt, dass keine komplementäre Kante beinhaltet ist. Anhand des Ausschnittes soll gezeigt werden, wie das Durchlaufen eines De Bruijn Graphen eine q-eindeutige Sequenz erzeugt.

Knoten **AA** sei in diesem Beispiel der Startknoten. Die erste gewählte Kante entspricht den q ersten Zeichen der zu erzeugenden Sequenz. Für jede weitere Kante, die abgelaufen wird, verlängert sich die Sequenz um ein Zeichen. In Abbildung 4.2 wird zunächst die Kante **AAA** gewählt und anschließend die Kante **AAC**. Das entspricht dann der Sequenz **AAAC**. Solange jede Kante höchstens einmal gewählt wird, ist die resultierende Sequenz 3-eindeutig. In Tabelle 4.2 sind die nachfolgenden Schritte und die resultierenden Sequenzen aufgelistet. Mit geeigneten Methoden muss nun das Wählen von Kanten ausgeschlossen werden, die mit dem Komplement von bereits besuchten Kanten beschriftet

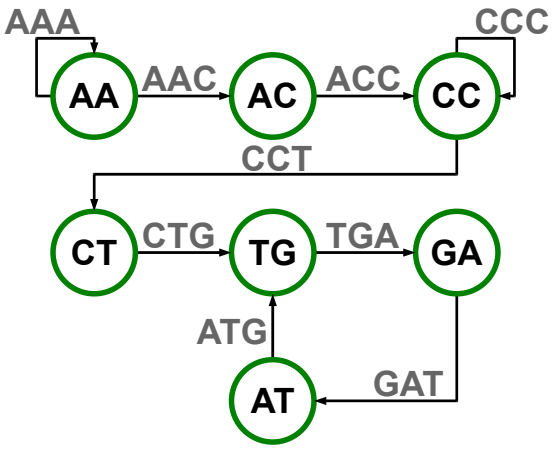


Abbildung 4.2: Ausschnitt des De Bruijn Graphen mit $q = 3$

Tabelle 4.2: Exemplarischer Durchlauf des De Bruijn Graphen aus Abbildung 4.2

Schritt	Kante	Sequenz
1	AAA	AAA
2	AAC	AAAC
3	ACC	AAACC
4	CCC	AAACCC
5	CCT	AAACCCT
6	CTG	AAACCCTG
7	TGA	AAACCCTGA
8	GAT	AAACCCTGAT
9	ATG	AAACCCTGATG

sind. Ist n dabei die Anzahl der abgelaufenen Kanten, so kann die Länge L_s der erzeugten Sequenz s unter Berücksichtigung der Formel 3.2 wie folgt bestimmt werden:

$$L_s = n + q - 1. \tag{4.1}$$

Wie schon aus Tabelle 4.1 hervorgeht, gibt es zu jedem q-gramm eine obere Schranke für die maximal wählbare Kantenanzahl. Wird n durch diese Anzahl ersetzt ergibt sich die maximale Länge einer Sequenz L_{sMax}:

$$L_{sMax} = (4^q)/2 + q - 1 \text{ , für ungerade } q \text{ und} \tag{4.2}$$

$$L_{sMax} = ((4^q)/2 - (4^{q/2})/2 + q - 1 \text{ , für gerade } q. \tag{4.3}$$

4.2 ILP zur Berechnung längster Pfade

Auf Basis der oben beschriebenen Graphenstruktur wird zur Lösung des ersten Teilproblems aus Abschnitt 3.2 ein ILP aufgestellt. Zunächst wird kurz das Konzept *integer linear programming* (Ganzzahlige lineare Optimierung) vorgestellt, um anschließend ein lineares

Programm zur Lösung des ersten Teilproblems aus Kapitel 3 aufzustellen. Im Folgenden soll die Abkürzung ILP für integer linear programming stehen.

4.2.1 Integer linear programming

ILP untersucht lineare Programme, deren Variablen ganzzahlig sein müssen. Das grundlegende Problem ist hierbei $max\{cx|Ax \leq b, x\text{ganzzahlig}\}$ zu berechnen. Ein ILP kann auch als Suche nach den Gitterpunkten in einem Polyeder oder als das Lösen eines linearen Gleichungssystems nicht negativer, ganzzahliger Variablen aufgefasst werden [Sch00]. Um ein Problem als ILP formulieren zu können, muss Folgendes möglich sein: Die Zielfunktion muss formuliert werden können und es müssen Nebenbedingungen existieren. Nebenbedingungen sind lineare (Un-)Gleichungen, die gewisse Teilbereiche des Problems modellieren und werden auch „Constraints“ genannt.

4.2 Definition. Integer linear programming nach [Sch00]

Gegeben seien eine rationale Matrix A, und die rationalen Vektoren b und c. Die Zielfunktion lautet:

$$\max\ c^T x$$

Nebenbedingung:

$$\begin{aligned} Ax &\leq b \\ x &\geq 0 \\ x &\in \mathbb{Z} \end{aligned}$$

4.2.2 Aufstellen eines ILP zur Berechnung längster Pfade

Wie zuvor schon geschildert, wird zur Lösung des DSD-Problems der längste q-eindeutige Pfad gesucht. Um die Bestimmung eines solchen Pfades durch ILP zu erreichen, werden zunächst die Variablen definiert. Für jede Kante im Graphen existiert eine Variable x_e. Wird Kante e gewählt, so ist x_e eins, ansonsten null. Weiter sei das Komplement zu Kante e durch γ(e) gegeben. Da ein Pfad einen Start- und Endknoten hat, existieren für jeden Knoten v zwei Variablen: a_v entscheidet, ob v ein Startknoten ist und z_v, ob v ein Endknoten ist. Der De Bruijn Graph ist gerichtet, so dass jeder Knoten ein- und ausgehende Kanten besitzt. Die eingehenden Kanten eines Knotens v sind in der Menge $IN(v)$ und die ausgehenden in der Menge $OUT(v)$ enthalten. Das ILP zur Lösung des ersten Teilproblems aus Abschnitt 3.2 sieht wie folgt aus:

Variablen:

$$x_e = \begin{cases} 1, \text{ wenn Kante } e \text{ gewählt,} \\ 0 \text{ sonst;} \end{cases}$$

$$\gamma(e) := \text{die komplementäre Kante zu } e;$$

$$a_v = \begin{cases} 1, \text{ wenn Knoten } v \text{ Startknoten ist,} \\ 0 \text{ sonst;} \end{cases}$$

$$z_v = \begin{cases} 1, \text{ wenn Knoten } v \text{ Endknoten ist,} \\ 0 \text{ sonst;} \end{cases}$$

$$IN(v) := \text{alle eingehenden Kanten eines Knotens } v;$$

$$OUT(v) := \text{alle ausgehenden Kanten eines Knotens } v.$$

Zielfunktion:

$$\max \sum_e x_e \tag{4.4}$$

Nebenbedingungen:

$$\forall e: \ 0 \leq x_e \leq 1, x_e \in \mathbb{Z} \tag{4.5}$$

$$\forall v: \ 0 \leq a_v \leq 1, \quad 0 \leq z_v \leq 1, \quad a_v, z_v \in \mathbb{Z} \tag{4.6}$$

$$\forall e: \ x_e + x_{\gamma(e)} \leq 1 \tag{4.7}$$

$$\sum_{e=\gamma(e)} x_e = 0 \tag{4.8}$$

$$\forall v: \ \sum_{e \in In(v)} x_e + a_v = \sum_{e \in Out(v)} x_e + z_v \tag{4.9}$$

$$\sum_v a_v \leq 1, \quad \sum_v z_v \leq 1, \quad \sum_v a_v = \sum_v z_v \tag{4.10}$$

Da der längste Pfad gesucht ist, maximiert die Zielfunktion (4.4) die Anzahl der zu wählenden Kanten im De Bruijn Graphen. Die Nebenbedingungen (4.5) und (4.6) beschränken die Variablen der Knoten und Kanten auf binäre Werte. Das heißt die Kanten und Knoten haben lediglich zwei Zustände: sie sind gewählt, dann nimmt die entsprechende Variable den Wert eins an oder sie sind nicht gewählt und die Variable nimmt den Wert null an. Diese Einschränkung der Variablenwerte formuliert außerdem die erste Regel, dass

jede Subsequenz höchstens einmal gewählt werden darf. Nebenbedingung (4.7) steht für die zweite Regel der q-eindeutigen Sequenzen. Wenn eine Subsequenz der Länge q gewählt wurde, dann darf das Komplement nicht mehr vorkommen. Selbstkomplementäre Kanten werden durch Nebenbedingung (4.8) direkt ausgeschlossen. Somit ist auch die dritte Regel als Constraint formuliert. Um einen Pfad zu finden, muss an jedem Knoten dieselbe Anzahl an ein- und ausgehenden Kanten gewählt sein. Diese Eigenschaft wird durch die Nebenbedingung (4.9) erzwungen. Zusätzlich muss bedacht werden, dass wenn es sich um einen Pfad handelt, es einen Start- und einen Endknoten gibt. An diesen Knoten existiert das Gleichgewicht von ein- und ausgehenden Kanten nicht. Der Startknoten hat eine eingehende Kante weniger als er ausgehende hat, während der Endknoten eine ausgehende Kante weniger als er eingehende Kanten hat. Die letzte Nebenbedingung (4.10) sichert ab, dass, falls ein Startknoten gewählt wird, auch ein Endknoten existieren muss. Ist die Lösung kein Pfad sondern ein Kreis, so existiert weder ein Start- noch ein Endknoten.

4.2.3 Ergebnisse des ILP

Für das oben modellierte ILP sind mehrere Lösungen zulässig. Ein Pfad oder ein Kreis der Länge L_{sMax} aus Formel (4.2) oder (4.3) sind die Lösung zum ersten Teilproblem. Zulässig sind aber auch mehrere disjunkte Kreise oder ein Pfad und mehrere disjunkte Kreise. Da Oligonukleotide gesucht werden, die aus relativ wenigen Basen bestehen, könnten auch nicht zusammenhängende Kreise eine Lösung sein. Haben die Kreise eine ausreichende Länge, kann die erzeugte Sequenz ein einzelnes Oligonukleotid darstellen. Alle anderen Kreise sind unbrauchbar. Um den letzteren Fall zu vermeiden, ist es denkbar die Zielfunktion abzuändern oder weitere Nebenbedingungen aufzustellen. Ein Ansatz ist es, durch weitere Nebenbedingungen, den Zusammenhang zwischen den einzelnen Kreisen und dem Pfad zu erzwingen. Diese Ansätze werden im Ausblick, Kapitel 7, näher erläutert.

Die Ausgabe des ILP bestimmt, welche Kanten im De Bruijn Graphen gewählt wurden. Um daraus Sequenzen zu erhalten, müssen noch einige Schritte abgearbeitet werden. Zunächst wird mit der ILP-Ausgabe ein Graph konstruiert. Anschließend wird in diesem Graphen der längste Pfad gesucht. Anhand einer Beispielinstanz sollen im Folgenden die einzelnen Schritte erläutert werden. Mit *Cplex 12.2* [IBM10] liefert das ILP für 3-gramme folgende Lösung:

> Objective: 32 CGstart=0, AAT=1, GTC=0, TTC=1, CGA=1, TTstart=0, GTA=0, TTA=0, CGC=0, TTende=0, TTG=0, CAG=1, CCende=0, AGstart=1, CAC=1, GTG=0, TGstart=0, CAA=1, TCG=0, TCende=0, AAG=1, AAA=0, AAC=0, TTT=1, TGT=0, TCA=1, CAT=0, ATT=0, AAstart=0, TCC=0, GAT=1, TCstart=0, GTende=0, TAT=0, CGT=1, CCstart=0, ACende=0, GAG=1, CTG=0, CAende=0, CTA=1, GAC=1, CTC=0, ATende=0, GAA=0, CGende=0, CGG=0, AGende=1, ACT=1, GCstart=0, CCA=0, CCC=0, GGstart=0, CCG=1, CAstart=0, TAA=1, ACC=1, TAC=1, ACG=0, AGT=0, AGG=1, GAstart=0, CCT=0, TAG=0, CTstart=0, AGA=1, ATG=1, GGA=1, GCT=0, GGC=1, ACstart=0, ATC=0, AGC=1, GGG=1, GAende=0, TAende=0, AAende=0, TGG=1, TCT=0, GTT=1, ACA=1, GGende=0, TGC=0, GCG=1, CTende=0, TGende=0, TGA=0, TAstart=0, ATstart=0, GGT=0, GCende=0, ATA=1, GCC=0, GCA=1, CTT=0, GTstart=0

Die Zielfunktion hat den Wert 32, es sind also 32 Kanten gewählt worden. Diese Anzahl entspricht der optimalen Anzahl aus Tabelle 4.1. Die gewählten Kanten bilden die Menge $Z = \{e \in E | x_e = 1\}$ der zulässigen Kanten. Aus diesem Ergebnis wird ein gerichteter Graph erstellt. Hierzu wird die Bibliothek *JGraphT* [Bar10] genutzt. Wenn das ILP die

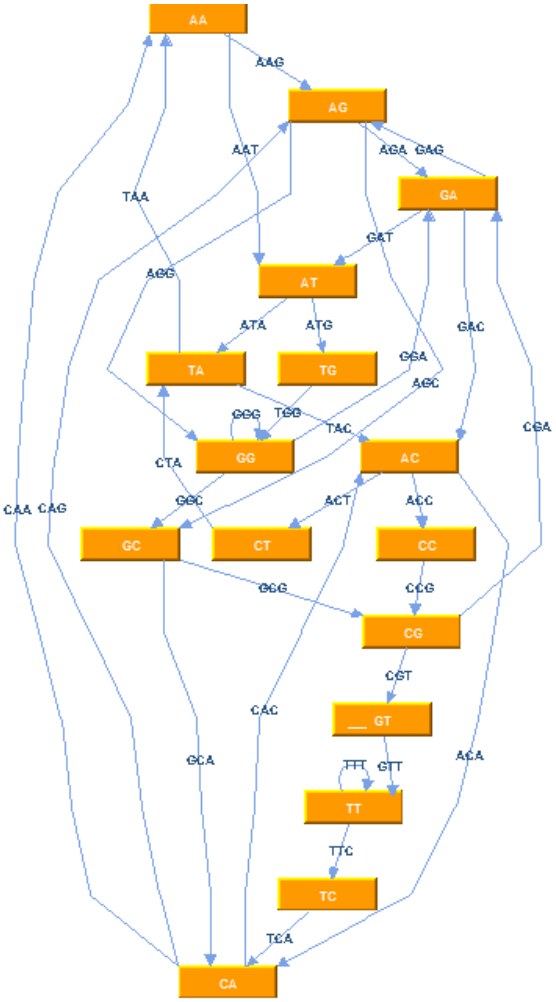


Abbildung 4.3: Graphische Darstellung der Lösung für 3-gramme

Variable x_e gleich eins gesetzt hat, dann muss die Kante e dem zunächst noch leeren Graphen hinzugefügt werden. Aus der Beschriftung der Kante kann sowohl Quell- als auch Zielknoten der Kante bestimmt werden. Da die Kanten mit q-grammen beschriftet sind, ist der Quellknoten einer Kante mit dem Präfix der Länge $q-1$ des jeweiligen q-gramms beschriftet. Der Zielknoten hingegen ist mit dem Suffix der Länge $q-1$ der Kante beschriftet. Falls einer oder beide Knoten im aktuellen Graphen noch nicht vorhanden sind, muss dieser oder diese zunächst hinzugefügt werden. Anschließend kann die verbindende Kante einfügt werden. Mit der Java-Bibliothek *JGraph* [JGr10] kann der vom ILP bestimmte Graph zusätzlich visualisiert werden. Dies ist jedoch nur für kleine Instanzen sinnvoll, wie zum Beispiel 2- und 3-gramme. Für größere Instanzen ist die Visualisierung des Graphen zu unübersichtlich. Die oben angegebene Lösung wird in Abbildung 4.3 visualisiert.

Das Ergebnis kann aus mehreren Kreisen und einem Pfad bestehen. Diese können auch verschiedene Teilgraphen bilden, das heißt sie müssen nicht zusammenhängend sein. Da die Teilgraphen Lösungen des ILP sind, existiert für jeden Teilgraphen ein Pfad, der jede Kante abläuft. Mit einer Tiefensuche kann für jeden Teilgraphen ein solcher Pfad gefunden werden. Jeder dieser Pfade ist wiederum eine q-eindeutige Sequenz. Eine weiter Möglichkeit ist, dass sich die verschiedenen Teilgraphen zu einem Pfad, der mehrere Kreise beinhaltet, verbinden lassen. Das Ergebnis des Beispiels besteht aus nur einem Pfad und

besitzt keinen Startknoten. Hier handelt es sich also um einen Kreis. Nach Formel 4.2 wird für dieses Beispiel die Länge der Sequenz wie folgt berechnet: $3 + (32 - 1) = 34$. Eine möglich Lösung nach Algorithmus 4.1 sieht wie folgt aus:

AAGACACTACCGAGGGCAGCGTTTCAATGGATAA

Hat das ILP einen Startknoten gewählt, so existiert in der Lösung ein Pfad. Dieser Pfad wird als erstes abgelaufen. Zunächst wird eine Kante gewählt, die den Startknoten als Quellknoten hat. Dann wird solange eine Nachfolgekante aus der Menge Z gewählt, bis keine gültige mehr gefunden werden kann. Eine Kante e' wird als Nachfolgekante einer Kante e bezeichnet, wenn der Quellknoten von e' auch der Zielknoten von e ist. Jedes Mal, wenn eine Kante gewählt wird, dann muss diese aus der Menge Z entfernt und dem Pfad P hinzugefügt werden. Nachdem ein Pfad gefunden wurde muss geprüft werden, ob weitere Zusammenhangskomponenten existieren. Die weiteren Komponenten können dann nur Kreise sein. Hierbei wird eine zufällige Kante gewählt und dann, wie bei der Pfadsuche, fortgefahren. Nach der Berechnung der Menge **P** muss geprüft werden, ob die einzelnen Pfade verknüpft werden können. Ist eine Subsequenz der Länge $q - 1$ in

Eingabe: $G = (V, Z)$
Ausgabe: $\boldsymbol{P}$ Menge von Pfaden $P_1, P_2, \ldots$

if existiert Startknoten **then**
  wähle $e \leftarrow$ eine vom Startknoten ausgehende Kante
  füge e zu P hinzu
  entferne e aus Z
  while e hat Nachfolgekante **do**
    wähle $e' \in Z$ mit e' Nachfolgekante von e
    setze $e \leftarrow e'$
    füge e' zu P hinzu
    entferne e' aus Z
  end while
  füge P zu $\boldsymbol{P}$
end if

while $Z \neq \emptyset$ **do**
  erzeuge neues leeres P
  $e \leftarrow$ zufällige Kante aus Z
  füge e zu P hinzu
  entferne e aus Z
  while e hat Nachfolgekante **do**
    wähle $e' \in Z$ mit e' Nachfolgekante von e
    setze $e \leftarrow e'$
    füge e' zu P hinzu
    entferne e' aus Z
  end while
  füge P zu $\boldsymbol{P}$
end while
return $\boldsymbol{P}$

Algorithmus 4.1: Pfadsuche im Graphen des ILP-Ergebnisses

Tabelle 4.3: Ergebnisse des ILP für Graphen ohne selbstkomplementäre Kanten

	Gesamt-kanten	selbstkompl.-Kanten	maximal wählbare Kanten	gewählte Kanten	Differenz
2-gramme	16	4	6	6	0
3-gramme	64	–	32	32	0
4-gramme	256	16	120	115	5
5-gramme	1024	–	512	512	0
6-gramme	4096	64	2016	1959	57
7-gramme	16384	–	8192	8192	0
8-gramme	65536	256	32640	32279	361
9-gramme	262144	–	131072	131072	0
10-gramme	1048576	1024	523776	–	–

zwei verschiedenen Pfaden identisch, können diese zu einem Pfad verknüpft werden. Im Graphen ist eine Subsequenz der Länge $q-1$ ein Knoten. Die zwei Pfade sind also im Graphen beide durch denselben Knoten gelaufen und werden deshalb zu einem längeren Pfad verknüpft.

Wie bereits in Tabelle 4.1 verdeutlicht, existiert eine maximale Anzahl an Kanten, die gewählt werden kann. In Tabelle 4.3 sind alle bisher mit *Cplex 12.2* gefundenen Lösungen aufgelistet. Weiterhin sind zu den q-grammen die Anzahl der selbstkomplementären Kanten, die maximalen Werte und die Differenz zwischen ihnen und den errechneten Werten aufgelistet. Die Ergebnisse sind auf einem Rechner mit Intel® Core™i5-430M Prozessor und 4 GB Speicher berechnet worden.

Bei geraden q-grammen, mit Ausnahme von 2-grammen, existiert eine Differenz zwischen maximalem und gefundenem Wert. Für ungerade q-gramme wurde stets der maximale Wert gefunden. Diese Differenz kann mittels des ungerichteten De Bruijn Graphen aus Kapitel 5 erklärt werden. Für 10-gramme war es *Cplex 12.2* nicht möglich eine Lösung zu errechnen.

4.3 Lösen der Teilprobleme des DSD-Problems auf Basis des De Bruijn Graphen

Wie schon aus Abschnitt 3.2 bekannt, existieren fünf verschiedenen Teilprobleme des DSD-Problems. Der Lösungsansatz des ILP für den De Bruijn Graphen aus dem vorherigen Abschnitt wird in diesem für drei der Teilproblem angepasst. Im Folgenden wird für das erste, zweite und fünfte Teilproblem die genaue Adaption des vorgestellten Ansatzes beschrieben. Das dritte und vierte Teilproblem sind eng miteinander verwandt und können deshalb mit ähnlichen Ansätzen gelöst werden. Dazu wird die Graphenstruktur des De Bruijn Graphen als Grundlage genutzt, während das vorgestellte ILP nicht nötig ist.

4.3.1 Erstes Teilproblem: Längster Pfad

Das erste Teilproblem wird direkt vom oben beschriebenen Ansatz gelöst. Die resultierende Sequenz kann in Oligonukleotide gewünschter Länge aufgeteilt werden. Beim Trennen der Sequenz werden an den Schnittstellen q-gramme gelöscht, die nun wieder verwendbar

sind. Zur Maximierung der Sequenzlänge werden diese wieder eingefügt. Die Anzahl der Zeichen, die gespart werden können, ist direkt abhängig von der Anzahl der Schnitte. An jeder Schnittstelle werden die letzten $q-1$ Zeichen des entstandenen Oligonukleotids kopiert und für das Nächste wiederverwendet. Sollen m Oligonukleotide aus der Sequenz entstehen, so stehen mit der vorgestellten Vorgehensweise m Zeichen mehr zur Verfügung.

$$m \cdot (q-1) \tag{4.11}$$

Am folgenden Beispiel wird die Vorgehensweise verdeutlicht.

Beispiel. Teilen der resultierenden Sequenz in Oligonukleotide

Sei $q = 5$ und s = TGACGATTAGC die resultierende Sequenz. Diese Sequenz soll in Oligonukleotide der Länge $l = 6$ geteilt werden. Das erste Oligonukleotid o_1 besteht aus den ersten sechs Zeichen. Das zweite Oligonukleotid o_2 würde dann aus den restlichen fünf Zeichen bestehen und nicht den Ansprüchen genügen.

1. Oligonukleotid o_1 = TGACGA
2. Oligonukleotid o_2 = TTAGC

Die 5-gramme, die beim Aufteilen beteiligt sind, gehen bei der beschriebenen naiven Methode verloren. Das erste Oligonukleotid o_1 besteht wie zuvor aus den ersten sechs Zeichen, also o_1 = TGACGA. In o_1 ist das 5-gramm ACGAT nicht vorhanden. Deshalb kann das zweite Oligonukleotid o_2 mit diesem 5-gramm anfangen und bis zum Ende von s alle Zeichen aufbrauchen. Dann ist o_2 = ACGATT. Wird dasselbe Prinzip noch einmal angewandt, so entsteht ein drittes Oligonukleotid o_3. Das 5-gramm GATTA wurde noch nicht verwendet und ist somit der Anfang für o_3. So entstehen aus der Sequenz s folgende Oligonukleotide:

1. Oligonukleotid o_1 = TGACGA
2. Oligonukleotid o_2 = ACGATT
3. Oligonukleotid o_3 = GATTAG
4. Oligonukleotid o_4 = TTAGC

4.3.2 Zweites Teilproblem: Längster Pfad bei gegebenem Kontext

Dieses Teilproblem beschäftigt sich mit dem Fall, dass bereits ein Kontext vorhanden ist. Das heißt, es sind schon Oligonukleotide vorgegeben. Die Menge der Oligonukleotide des Kontextes wird im Folgenden mit K bezeichnet und beinhaltet die vorgegebenen Oligonukleotide. Sind m Oligonukleotide vorgegeben, so ist $K = \{r_1, r_2, \ldots, r_m\}$. Dabei sind $r_1, r_2, \ldots, r_m$ die Sequenzen der vorgegebenen Oligonukleotide. Zunächst werden alle q-gramme dieser Oligonukleotide identifiziert. Die q-gramme bilden die Menge der benutzten Kanten Q. Die Anzahl der benutzten q-gramme kann nach oben abgeschätzt werden. Ist die Menge K q-eindeutig und die Länge jedes Oligonukleotids r_i gleich l_i, so existieren n verschiedene q-gramme $s_1, s_2, \ldots, s_n$ mit $n = (l_1-(q-1))+(l_2-(q-1))+\ldots+(l_m-(q-1))$. Die Menge Q ist dann wie folgt definiert: $Q = \{s_1, s_2, \ldots, s_n\}$.

Die Kanten, die mit den q-grammen aus Q und mit deren Komplementen beschriftet sind, werden aus dem De Bruijn Graphen gelöscht. Das Löschen dieser Kanten sorgt dafür, dass der erzeugte Pfad zusammen mit den schon vorgegebenen Oligonukleotiden

eine q-eindeutige Menge ist. Das formulierte ILP wird dazu durch folgendes Constraint ergänzt:

$$\forall e \in Q : x_e + x_{\gamma(e)} = 0 \tag{4.12}$$

4.3.3 Drittes Teilproblem: Modifikation der sticky ends

Zur Lösung dieses Problems ist das ILP nicht nötig. Abbildung 4.4 zeigt das Oligonukleotid BN-3 der Kachel B^3. Die unterstrichenen Subsequenzen sind die sticky ends, die modifiziert werden sollen. Das sticky end am Anfang eines Oligonukleotids wird im Folgenden erstes sticky end und das am Ende zweites sticky end genannt. In Abbildung 4.4 ist dann CCACA das erste und AGCCG das zweite sticky end des Oligonukleotids. Die nicht unterstrichene Sequenz ist der Teil des Oligonukleotids, der nicht verändert werden darf. Wie schon erwähnt, haben die sticky ends der hier behandelten DNA-Kacheln immer die Länge fünf. Die Modifikation der sticky ends muss mit 5-grammen modelliert werden, da die vorgestellten DNA-Kacheln mit 5-grammen erstellt wurden. Dabei besteht die Menge K aus allen Oligonukleotiden der zu modifizierenden Kachel und die Menge Q dementsprechend aus den vorkommenden 5-grammen. Es muss jedoch eine Korrektur der Menge Q erfolgen. In Abbildung 4.5 ist der Ausschnitt des De Bruijn Graphen zu sehen, der die Sequenz AGCCG des zweiten sticky ends erzeugt.

Der Pfad zur Erzeugung des sticky ends beginnt im Knoten CATT. Dieser Knoten besteht aus den letzten $q-1$ Zeichen der Sequenz, die nicht modifiziert werden soll. Die Kante, die mit dem sticky end beschrieben ist, bleibt in der Menge Q. Nach dem Entfernen des sticky ends sind alle anderen Kanten auf dem Pfad in Abbildung 4.5 wieder zulässig. Diese Kanten müssen, falls nicht durch andere Sequenzen verboten, aus der Menge Q entfernt werden. Analog dazu sind die Kanten beim ersten sticky end zu überprüfen und gegebenenfalls aus der Menge Q zu entfernen. Die Menge Q und die Komplemente ihrer Elemente werden aus der Menge E entfernt. Die restlichen Kanten sind genau die Menge Z. Für dieses Teilproblem existieren drei Fälle: Das erste sticky end soll modifiziert werden, das zweite sticky end soll modifiziert werden, oder beide sticky ends sollen modifiziert werden. Im Folgenden werden mit den Rekursion $FSE(Z, l, s)$ die möglichen Modifikationen des ersten sticky ends (**F**irst **S**ticky **E**nd) und mit Rekursion $SSE(Z, l, s)$ die des zweiten (**S**econd **S**ticky **E**nd) berechnet.

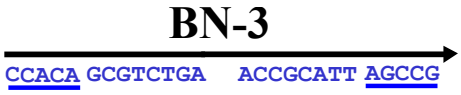


Abbildung 4.4: Oligonukleotid BN-3 der Kachel B^3

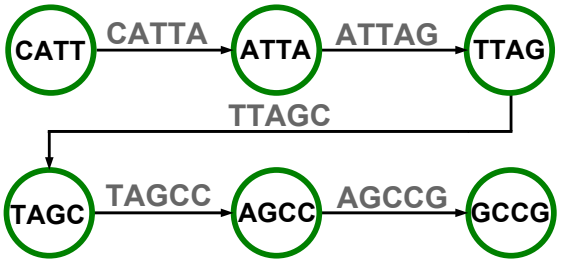


Abbildung 4.5: De Bruijn Graph zur Erzeugung von AGCCG

$FSE(Z, l, s)$:

Eingabe: Z Menge aller noch möglichen Kanten, l Länge der zu erzeugenden Pfade und s Startsequenz
if $l = 0$ **then**
füge s zu $\boldsymbol{S}$ hinzu
else
for all $k \in \{A, C, G, T\}$ **do**
$tmp \leftarrow k$ konkateniert mit den ersten $q-1$ Zeichen von s
if $tmp \in Z$ **then**
$FSE(Z \setminus \{tmp, \gamma(tmp)\}, l-1, ks)$
end if
end for
end if

Algorithmus 4.2: Rekursion zur Modifikation erstes sticky end

$SSE(Z, l, s)$:

Eingabe: Z Menge aller noch möglichen Kanten, l Länge der zu erzeugenden Pfade und s Startsequenz
if $l = 0$ **then**
füge s zu $\boldsymbol{S}$ hinzu
else
for all $k \in \{A, C, G, T\}$ **do**
$tmp \leftarrow$ letzte $q-1$ Zeichen von s konkateniert mit k
if $tmp \in Z$ **then**
$SSE(Z \setminus \{tmp, \gamma(tmp)\}, l-1, sk)$
end if
end for
end if

Algorithmus 4.3: Rekursion zur Modifikation zweites sticky end

Für den ersten Fall, muss zunächst die Menge Z wie oben beschrieben angepasst werden. Der Teil des Oligonukleotids ohne das zu bearbeitende sticky end ist die Ausgangssequenz und wird im Folgenden Startsequenz genannt. Die Startsequenz aus dem Oligonukleotid in Abbildung 4.4 ist s = GCGTCTGAACCGCATTAGCCG. Die Sequenz s muss unter Berücksichtigung der Sequenzdesign-Regeln um fünf Zeichen verlängert werden. Der rekursive Algorithmus 4.2 berechnet eine Menge von alternativen Oligonukleotiden für BN-3: $S = \{s_1, s_2, \ldots\}$. Als Eingabe wird die Startsequenz s, die Menge Z der zulässigen Kanten und die Länge l der sticky ends benötigt. Zunächst wird vorne an die Startsequenz s ein Zeichen aus Σ hinzufügt. Anschließend wird geprüft, ob die ersten q Zeichen der entstandenen Sequenz zulässig sind. Sind diese q Zeichen eine Sequenz aus der Menge Z, wird über den Rekursionsaufruf ein weiteres Zeichen anfügt. Wenn fünf Zeichen gefunden worden sind, wird die entstandene Sequenz in die Menge S aufgenommen.

Für die Modifikation des zweiten sticky ends wird analog zur Modifikation des ersten sticky ends verfahren. Mit dem rekursiven Algorithmus 4.3 werden der Startsequenz s hinten fünf Zeichen angehängt. In diesem Fall ist s = CCACAGCGTCTGAACCGCATT die Startsequenz.

Eingabe: Z Menge aller noch möglichen Kanten, l Länge der zu erzeugenden Pfade und s Startsequenz
Ausgabe: $\boldsymbol{T}$ Menge von Sequenzen $\{t_1, t_2, \ldots\}$
erezeuge neue leere Menge S
$FSE(Z, l, s)$
$T \leftarrow S$
$S \leftarrow \emptyset$
for all $t \in T$ **do**
$SSE(Z', l, s)$ (mit angepasster Menge Z')
end for
$T \leftarrow S$
return T

Algorithmus 4.4: Modifikation beider sticky ends

Für den Fall, dass beide sticky ends gleichzeitig modifiziert werden sollen, werden die vorgestellten Rekursionen kombiniert. Ist $S = \{s_1, s_2, \ldots, s_n\}$ die Menge der modifizierten Oligonukleotide berechnet für das erste sticky end, dann existiert für jedes $s \in S$ eine Menge R. Für jedes alternative Oligonukleotid aus S muss die Rekursion zur Modifikation des zweiten sticky ends aufgerufen werden. Die Menge $R_1 = \{r_1, r_2, \ldots\}$ für s_1 beinhaltet so alle Oligonukleotide, die durch Modifikation des zweiten sticky ends von s_1 entstehen. Zu jedem $s \in S$ muss die Menge Z' aktualisiert werden, da jedes der alternativen Oligonukleotide eine eigene Menge Q besitzt. In Algorithmus 4.4 sind die Rekursionen zur Modifikation der einzelnen sticky ends kombiniert. Die Menge T der alternativen Oligonukleotide, bei denen beide sticky ends modifiziert wurden, ist die Vereinigung der Mengen $R_1, R_2, \ldots, R_n$. Die Startsequenz ist der mittlere Teil des Oligonukleotids. In Abbildung 4.4 entspricht dies dem nicht unterstrichenen Teil von BN3.

4.3.4 Viertes Teilproblem: Erstellen eines Internals

Auch zur Lösung dieses Problems ist das ILP nicht notwendig. In Abbildung 4.6 ist der Kachel A^2 ein Internal hinzugefügt worden. Dazu wurde das Oligonukleotid NW in zwei Teile aufgetrennt. Das Internal ANW1 wird durch die Hybridisierungen mit dem Oligonukleotid AW-2 und C an die Kachel selbst gebunden. Das zweite Oligonukleotid NW2 hat die Aufgabe durch seine Hybridisierungen das Internal senkrecht zur Ebene der Kachel aufzustellen. Dazu muss das Oligonukleotid NW2 mit den Oligonukleotiden ANW1, C und AN hybridisieren, eine ausreichend lange Verbindung mit dem Oligonukleotid ANW1 haben und zwei Thyminbasen an einer bestimmten Stelle besitzen. An der Stelle der beiden Teile des Internals war vorher das äußere Oligonukleotid NW. Dieses wird zunächst an einer ausgewählten Position aufgetrennt. Aus dem ersten Teil entsteht das Oligonukleotid ANW1 und aus dem zweiten Teil das Oligonukleotid NW2.

Alle noch möglichen Sequenzen zur Erstellung eines Internals für eine bestimmte Kachel können mit dem rekursiven Algorithmus 4.3 berechnet werden. Vereinfacht ist das Internal ein längeres sticky end. Die Eingaben für die Rekursion sind eine Startsequenz s, die Länge l der zu erstellenden Sequenz und die Menge Z aller noch zulässigen Kanten. Für das Internal wird ein Startknoten statt einer Startsequenz übergeben. Der Startknoten ist, ähnlich wie bei der Erzeugung der sticky ends, der String aus den letzten $q-1$ Zeichen des ersten Teils. Die Länge der Internals ist fest vorgegeben und beträgt 28. Aus der Menge E aller Kanten des De Bruijn Graphen werden alle Kanten der Menge Q entfernt. Anschließend werden auch alle Komplemente der Kanten von Q entfernt. Die restlichen Kanten

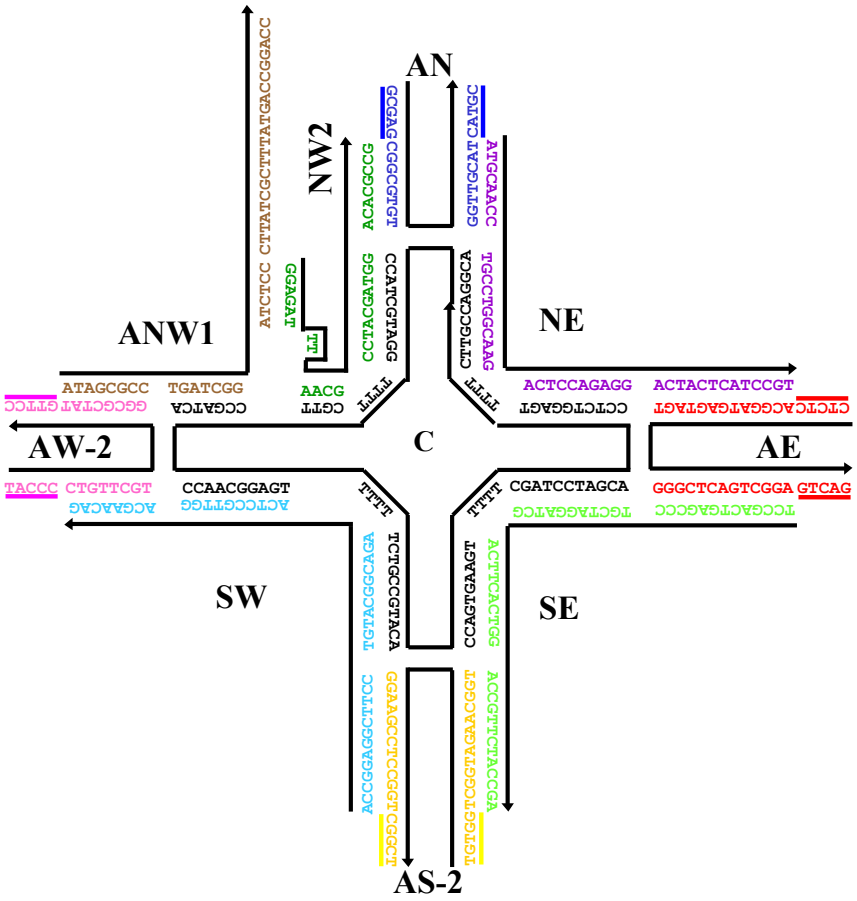


Abbildung 4.6: Kachel aA^2

sind genau die Menge Z. Da die behandelten DNA-Kacheln mit 5-grammen erstellt wurden, hat der Startknoten die Länge 4. An diese vier Zeichen werden weitere 28 hinzugefügt. Für dieses Teilproblem sind deshalb alle $s \in S$ des rekursiven Algorithmus 4.3 32 Zeichen lang. Diese müssen jeweils mit dem ersten Teil des alten Oligonukleotids verknüpft werden und ergeben den Arm des Internals. Bei relativ kleinem Kontext, beispielsweise nur einer DNA-Kachel, existieren möglicherweise sehr viele erlaubte Internals. Der zweite Teil des Oligonukleotids wird anschließend auch erweitert. An den Anfang des zweiten Teils wird das Komplement der ersten sechs Zeichen des Arms gesetzt. Dann werden die beiden Thyminbasen hinzugefügt und abschließend der zweite Teil des alten Oligonukleotids angehängt. Das Hinzufügen der zwei Thyminbasen wird nicht sequenzdesigntheoretisch überprüft. Deshalb könnten Komplemente oder Kopien von schon vorhandenen Subsequenzen entstehen.

4.3.5 Fünftes Teilproblem: Erstellen von DNA-Kacheln

Die hier beschriebenen Kacheln bestehen immer aus neun Oligonukleotiden, die mit Subsequenzen der Länge fünf modelliert werden. Jedes dieser Oligonukleotide hat eine bestimmte Länge, die aus strukturchemischen Gründen eingehalten werden muss. In Abbildung 4.7 ist der Aufbau der hier beschriebenen DNA-Kacheln schematisiert. Die C-Komponente (C steht für *center*) ist das Grundgerüst der Kachel. Die Oligonukleotide, die später für die sticky ends verantwortlich sind, werden N-Komponente (N steht für *north*), S-Komponente (S steht für *south*), W-Komponente (W steht für *west*) und E-Komponente (E steht für

east) genannt. Um diese vier Komponenten mit der C-Komponente zu verbinden, werden vier weitere Oligonukleotide benötigt. Die NE-Komponente (NE *northeast*) hybridisiert mit Teilen der C-Komponente, der N-Komponente und der E-Komponente. Die NW-, SE- und SW-Komponenten (NW *northwest*, SE *southeast*, SW *southwest*) verhalten sich analog.

Die Positionierung der vier Thyminbasen innerhalb der C-Komponente ist eine weitere Bedingung zur Erstellung einer DNA-Kachel. Diese vier Thyminbasen müssen an vier bestimmten Positionen der Komponente eingefügt werden und gewährleisten den planaren Aufbau der Kachel. Mittels der Lösung des ersten Teilproblems wird zunächst die C-Komponente der Kachel geformt. Dazu werden von der gefundenen Sequenz die ersten 84 Zeichen benötigt. Dann werden die vier Thyminbasen an den jeweiligen Positionen der Sequenz zugefügt. Die ersten vier Thyminbasen werden nach den ersten 15 Zeichen eingefügt. Die übrigen drei Positionen werden jeweils im Abstand von 21 Zeichen eingefügt. So erreicht die C-Komponente eine Länge von 100 Zeichen.

Die restliche Ergebnissequenz wird weiter in die vier Oligonukleotide N-, S-, W- und E-Komponente geteilt, mit den respektiven Längen von 26, 36, 26 und 36 Zeichen. Insgesamt werden zur Erstellung der Kachel 208 Zeichen benötigt. Die zu erstellende q-eindeutige Sequenz muss mindestens 208 Zeichen lang sein. Es werden vier Teilsequenzen für die C-Komponente gebraucht und weitere vier für die N-, S-, W- und E-Komponenten. Die Sequenz wird dazu an acht Stellen aufgetrennt, also $n = 8$. Bei $q = 5$ können mit der Formel (4.11) $m = n \cdot (q-1) = 8 \cdot (5-1) = 40$ Zeichen eingespart werden. Deshalb muss die q-eindeutige Sequenz zur Erstellung einer DNA-Kachel nur noch 168 Zeichen lang sein. Um eine q-eindeutige Sequenz der Länge l zu erzeugen, müssen nach Formel (3.1) $l-q+1$ Kanten im Graphen gewählt werden. Zur Erstellung einer DNA-Kachel wird demnach ein Pfad bestehend aus 164 Kanten benötigt. Aus Tabelle 4.3 geht hervor, dass erst bei 5-grammen eine ausreichende Anzahl an Kanten zur Erzeugung der Sequenz wählbar ist. Aus diesem Grund ist es nicht möglich eine DNA-Kachel aus einer q-eindeutigen Sequenz mit dem Grundgerüst aus Abbildung 4.7 und $q \leq 4$ zu generieren.

Anschließend müssen die restlichen vier Komponenten erstellt werden. Diese Komponenten müssen in Abhängigkeit zu den schon entstandenen Komponenten modelliert werden, da bestimmte Hybridisierungen gezielt erzeugt werden sollen. Seien die Zeichen der einzelnen Oligonukleotide von $5'$ nach $3'$ durchnummeriert und sei $subX(a,b)$ der Teilstring der X-Komponente von Position a bis b. Die Komponente NW besteht aus

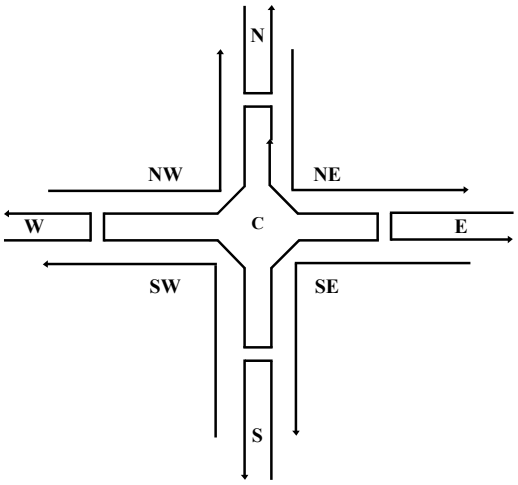


Abbildung 4.7: Allgemeiner Aufbau einer DNA-Kachel

dem Komplement des $subW(13, 20)$ konkateniert mit dem Komplement von $subC(19, 30)$, konkateniert mit dem Komplement von $subC(5, 14)$ und wiederum konkateniert mit dem Komplement von $subN(5, 12)$. Die NE-, SE- und SW-Komponente werden analog zur NW-Komponenten entworfen.

Die W-Komponente einer Kachel wird unabhängig von ihrer C-Komponente modelliert. Die NW-Komponente verbindet dann diese beiden Komponenten durch Komplementbildung der entsprechenden Stellen. Dabei entstehen q-gramme in der NW-Komponente, die nicht sequenzdesigntheoretisch überprüft wurden. Diese könnten eine in der Kachel schon vorhandene Subsequenz oder das Komplement einer schon vorhandenen Subsequenz bilden. Bei der Modellierung der SW-, NE- und SE-Komponenten kann es zu analogen Fällen kommen.

Kapitel 5

Ein ungerichteter De Bruijn Graph zur Lösung des DNA-Sequenzdesign-Problems

Der zweite Ansatz basiert auf einer Graphenstruktur, die eine hohe Ähnlichkeit mit dem De Bruijn Graphen aufweist. Der ungerichteter De Bruijn Graph genannt wird in Abschnitt 5.1 vorgestellt und anhand von Beispielen näher erläutert. In Abschnitt 5.2 wird die Reduktion des DSD-Problems auf das Eulerkreisproblem geschildert. Dazu ist der genaue Aufbau der einzelnen Knoten des ungerichteten De Bruijn Graphen wichtig. Abschließend legt Abschnitt 5.3 das Vorgehen zur Lösung der einzelnen Teilprobleme auf Basis des ungerichteten De Bruijn Graphen dar.

5.1 Ein ungerichteter De Bruijn Graph zur Erzeugung von DNA-Sequenzen: Aufbau und Eigenschaften

Der ungerichtete De Bruijn Graph ist eine Abwandlung des schon vorgestellten De Bruijn Graphen und wird im Folgenden auch kurz G_q-Graph genannt. Der Graph wird im Artikel [AFN06] definiert und analysiert. Die verschiedenen Knotenarten des ungerichteten De Bruijn Graphen wurden schon in dem Artikel untersucht. Zum besser Verständnis werden im Folgenden jedoch die Knotenarten im Detail beschrieben und mit Beispielen ihre Eigenschaften hervorgehoben.

Wie im De Bruijn Graphen sind die Knoten und Kanten des G_q-Graphen mit Wörtern beschriftet. Allerdings besteht dieser in etwa aus der Hälfte aller Knoten und Kanten des De Bruijn Graphen. Der Grund dafür, sind die doppelten Beschriftungen der Knoten und Kanten. Eine Beschriftungen bestehen für die Knoten aus einer Sequenz $r \in \Sigma^{q-1}$ und ihrem reversen Komplement $r^C \in \Sigma^{q-1}$ und für die Kanten aus einer Sequenz $s \in \Sigma^q$ und ihrem reversen Komplement $s^C \in \Sigma^q$. Zunächst folgt die formale Definition eines ungerichteten De Bruijn Graphen.

5.1 Definition. Ungerichteter De Bruijn Graph G_q

Sei $\Sigma = \{A, C, G, T\}$ und q ein Parameter. Dann ist der ungerichtete De Bruijn Graph zu (Σ, q) durch die Menge aller Knoten V und die Menge aller Kanten E definiert.
Ist q **gerade** dann existieren $4^{q-1}/2$ Knoten und $4^q/2 + 4^{q/2}/2$ Kanten im Graphen. Für **ungerade** q enthält der Graph $4^{q-1}/2 + 4^{(q-1)/2}/2$ Knoten und $4^q/2$ Kanten.
Die Knoten sind wie folgt definiert:

$$V = \{\{r; r^C\} : r \in \Sigma^{q-1}\}.$$

Die Kanten lassen sich wie folgt beschreiben:

$$\begin{aligned} E = &\{(\{\sigma_1 s; s^C \sigma_1^C\}, \{s\sigma_2; \sigma_2^C s^C\}) : \sigma_1, \sigma_2 \in \Sigma, s \in \Sigma^{q-2}, \\ &\{\sigma_1 s; s^C \sigma_1^C\}, \{s\sigma_2; \sigma_2^C s^C\} \in V\}. \end{aligned}$$

Jede Kante $e \in E$ mit Quellknoten $\{\sigma_1 s; s^C \sigma_1^C\} \in V$ und Zielknoten $\{s\sigma_2; \sigma_2^C s^C\} \in V$ trägt die Beschriftung:

$$\{\sigma_1 s \sigma_2; \sigma_2^C s^C \sigma_1^C\} : \sigma_1, \sigma_2 \in \Sigma, s \in \Sigma^{q-2}$$

Die Besonderheit an dieser Graphenstruktur ist, dass eine Sequenz r und ihr reverses Komplement r^C in direktem Zusammenhang stehen, da beide Sequenzen Teil der Beschriftung einer Kante sind. Jede Kante und jeder Knoten ist also mit einer zweielementigen Menge von Strings beschriftet. Das Durchlaufen des Graphen ist dann in zwei Richtungen möglich, da die Kanten und Knoten mit zwei Strings beschriftet sind. Die Knoten $v_1 = \{\sigma_1 s; s^C \sigma_1^C : \sigma_1 \in \Sigma, s \in \Sigma^{q-2}\}$ und $v_2 = \{s\sigma_2; \sigma_2^C s^C : \sigma_2 \in \Sigma, s \in \Sigma^{q-2}\}$ sind durch die Kante $\{\sigma_1 s \sigma_2; \sigma_2^C s \sigma_1^C : \sigma_1, \sigma_2 \in \Sigma, s \in \Sigma^{q-2}\}$ verbunden. Führt ein Pfad von Knoten v_1 zu Knoten v_2, so muss der String $\sigma_1 s$ um das Zeichen σ_2 verlängert werden. Soll jedoch der Pfad von Knoten v_2 zu Knoten v_1 abgelaufen werden, so muss der String $\sigma_2^C s^C$ um das Zeichen σ_1^C verlängert werden. In Abbildung 5.1 ist ein Ausschnitt eines ungerichteten De Bruijn Graphen zu sehen. Dieser Ausschnitt soll im Folgenden beispielhaft durchlaufen werden.

Der Startknoten des Graphen in Abbildung 5.1 sei $v = \{\mathbf{AA;TT}\}$ und die Startsequenz, wie im Beispiel des De Bruijn Graphen in Abschnitt 4.1, sei $s = \mathbf{AAA}$. Damit ist entschieden, welche der beiden Sequenzen des Knotens v weiterverfolgt werden soll. Anschließend kann der Knoten v nur durch den String **AAC** der Kante {AAC;GTT} verlassen werden. In diesem Fall ist die Kante eine ausgehende Kante. Dabei wird der String s durch ein C erweitert und die Sequenz s ist dann **AAAC**. Der String **GTT** der Kante {AAC;GTT} würde bedeuten, dass der Knoten v betreten wird. Dann wäre diese Kante eine eingehende Kante. Einen Knoten zu durchlaufen bedarf also einer Einschränkung, die im Folgenden als *Erlaubte Kantenpaarung* bezeichnet und im Unterabschnitt 5.1.1 erklärt wird. In Tabelle 5.1 ist ein beispielhafter Durchlauf in beide Richtungen des Graphenausschnitts aufgelistet.

Eine weitere Besonderheit der Graphenstruktur ist, dass bei geradem und ungeradem q unterschiedliche Knotenarten vorhanden sind. Bei ungeradem q stellen sich drei verschiedene Knotenarten heraus, während sich bei geradem q vier verschiedene Knotenarten

herausstellen. Die Knotenarten bei ungeradem q werden in Unterabschnitt 5.1.2 vorgestellt und analysiert. Unterabschnitt 5.1.3 erläutert analog dazu die Knotenarten für gerade q.

5.1.1 Erlaubte Kantenpaare

Um einen Knoten im G_q-Graphen zu durchlaufen, muss eine Kante in den Knoten führen und eine aus dem Knoten herausführen. Die Richtung der einzelnen Kanten des Graphen wird durch die Wahl der Beschriftung der Kante selbst bestimmt. Ein Knoten $v = \{s; s^C\}$ wird mit einer Kante $e_1 = \{\sigma_1 s; s^C \sigma_1^C\}$ erreicht, wenn der String $r_1 = \sigma_1 s$ gelesen wird, und verlassen, wenn der String $r_2 = s^C \sigma_1^C$ gelesen wird. Ist der String r_1 gewählt worden, so kann zum Verlassen des Knotens nur eine Kanten mit den folgenden Strings gewählt werden: $sA,\ sC,\ sG$ oder sT. Für jede Kante eines Knotens gibt es eine Menge von erlaubten Kanten. Zur Kante e_1 erlaubte Kanten sind alle Kanten der Form $e = \{sa; a^C s^C : a \in \Sigma\}$. Die Kante e_1 und jede dieser Kanten bilden ein erlaubtes Kantenpaar. In Abbildung 5.2 ist ein Knoten des ungerichteten De Bruijn Graphen zu sehen. Zusätzlich sind die Richtungen der Strings durch Pfeile gekennzeichnet, um die erlaubten Kantenpaare zu verdeutlichen.

Wird der Knoten $v = \{ACT; AGT\}$ durch die Kante $e = \{CACT; AGTG\}$ mit dem String $r_1 = CATC$ erreicht, kann der Knoten nur durch eine der folgenden Kanten verlassen werden. Dabei wird der String r erzeugt:

- $\{ACTA; TAGT\}$, mit dem String $r_2 = ACTA$ und $r = r_1 r_2 = CACTA$
- $\{ACTC; GAGT\}$, mit dem String $r_2 = ACTC$ und $r = r_1 r_2 = CACTC$
- $\{ACTG; CAGT\}$, mit dem String $r_2 = ACTG$ und $r = r_1 r_2 = CACTG$
- $\{ACTT; AAGT\}$, mit dem String $r_2 = ACTT$ und $r = r_1 r_2 = CACTT$

Wird der Knoten v hingegen durch die Kante e mit dem String $r_3 = AGTG$ verlassen, so muss der Knoten durch eine der folgenden Kanten erreicht worden sein:

- $\{ACTT; AAGT\}$, mit dem String $r_4 = AAGT$ und $r = r_3 r_4 = AAGTG$
- $\{ACTG; CAGT\}$, mit dem String $r_4 = CAGT$ und $r = r_3 r_4 = CAGTG$
- $\{ACTC; GAGT\}$, mit dem String $r_4 = GAGT$ und $r = r_3 r_4 = GAGTG$
- $\{ACTA; TAGT\}$, mit dem String $r_4 = TAGT$ und $r = r_3 r_4 = TAGTG$

Für jede Kante aus Abbildung 5.2 existieren vier verschiedene erlaubte Kantenpaare. Im optimalen Fall wird dieser Knoten deshalb genau vier Mal durchlaufen.

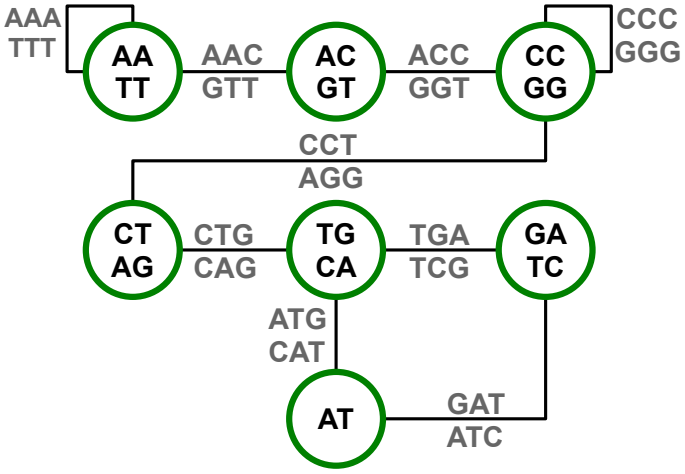


Abbildung 5.1: Ausschnitt des G_q-Graphen mit $q = 3$

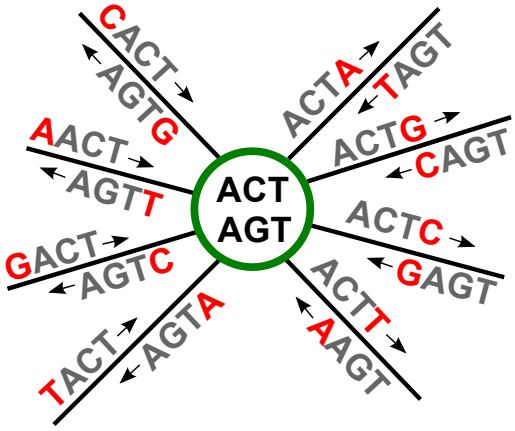


Abbildung 5.2: Erlaubte Kantenpaarungen

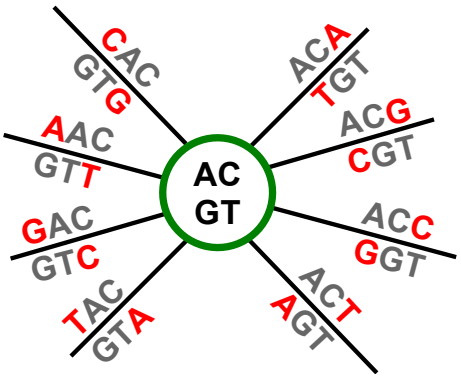
5.1.2 Ungerades q

Für ungerade q existieren im G_q-Graphen drei Knotenarten: *Knoten mit Vielfachen eines Zeichens*, *Knoten mit selbstkomplementärer Beschriftung* und *Normale Knoten*. Diese werden im Folgenden genauer erörtert und sind in Abbildung 5.3 zu sehen.

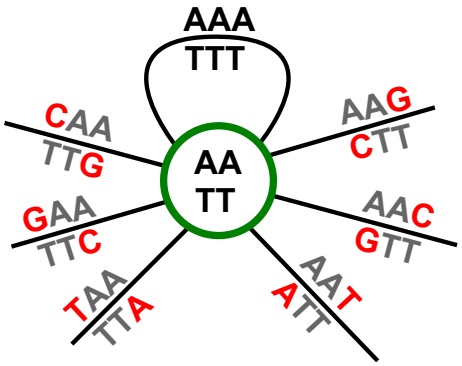
Knoten mit Vielfachen eines Zeichens Dieser Knoten besitzt eine Schleife, die selbst ein Vielfaches des Zeichens ist. Von dieser Knotenart existieren in jedem G_q-Graphen genau zwei Knoten: $\{AAA \dots A; T \dots TTT\}$ und $\{CCC \dots C; G \dots GGG\}$. In Abbildung 5.3(b) ist ein solcher Knoten aus dem G_q-Graphen, mit $q = 3$, abgebildet. Die rot gefärbten Zeichen kennzeichnen die hinzugefügten Zeichen, um die nächsten Knoten zu erreichen. Ist für Knoten $\{AA; TT\}$ der String AA gewählt, dann entsteht durch das Anhängen eines Zeichens jeweils eine Kante. Wird in diesem Fall das Zeichen A angehängt, so entsteht die Schleife $\{\mathbf{AAA}; TTT\}$. Das Anhängen der Zeichen C, G und T erzeugt die einfachen Kanten $\{\mathbf{AAC}; GTT\}$, $\{\mathbf{AAG}; CTT\}$ und $\{\mathbf{AAT}; ATT\}$. Ist für den Knoten der String TT gewählt und wird dann ein weiteres T gelesen, entsteht die Schleife $\{AAA; \mathbf{TTT}\}$ ein zweites Mal. Für die weiteren Zeichen entstehen die Kanten $\{TAA; \mathbf{TTA}\}$, $\{GAA; \mathbf{TTC}\}$ und $\{CAA; \mathbf{TTG}\}$. Da die beiden entstandenen Schleifen identisch sind, wird dem Knoten nur eine Schleife hinzugefügt. Die sechs entstandenen Kanten sind alle unterschiedlich

Tabelle 5.1: Exemplarischer Durchlauf des G_3-Graphen aus Abbildung 5.1

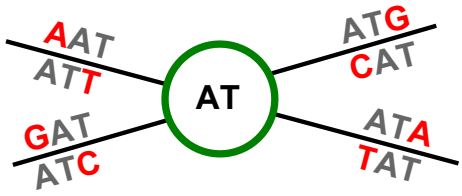
Schritt	Kante	Sequenz (1. Richtung)	Schritt	Kante	Sequenz (2. Richtung)
1	AAA, TTT	AAA	9	AAA, TTT	CATCAGGGTTT
2	AAC, GTT	AAAC	8	AAC, GTT	CATCAGGGTT
3	ACC, GGT	AAACC	7	ACC, GGT	CATCAGGGT
4	CCC, GGG	AAACCC	6	CCC, GGG	CATCAGGG
5	CCT, AGG	AAACCCT	5	CCT, AGG	CATCAGG
6	CTG, CAG	AAACCCTG	4	CTG, CAG	CATCAG
7	TGA, TCA	AAACCCTGA	3	TGA, TCA	CATCA
8	GAT, ATC	AAACCCTGAT	2	GAT, ATC	CATC
9	ATG, CAT	AAACCCTGATG	1	ATG, CAT	CAT



(a) Normaler Knoten



(b) Knoten mit Vielfachen eines Zeichens



(c) Knoten mit selbstkomplementärer Beschriftung

Abbildung 5.3: Für ungerade q: Die drei Knotenarten des G_3-Graphen

voneinander. Dieser Knoten hat Grad 8, da eine Schleife den Grad des Knotens um zwei erhöht und weitere sechs unterschiedliche Kanten existieren.

Knoten mit selbstkomplementärer Beschriftung Der Knoten mit selbstkomplementärer Beschriftung ist mit nur einem String der Länge $q-1$ beschriftet. Da es sich um einen selbstkomplementären String handelt, ist das reverse Komplement gleich dem String selbst. In jedem G_q-Graphen mit ungeradem q existieren genau $4^{q-1/2}$ dieser Knoten, da ebenso viele selbstkomplementäre Strings existieren. In Abbildung 5.3(c) ist exemplarisch der Knoten $\{AT\}$ mit all seinen Kanten abgebildet. Auch hier sind die gelesenen Zeichen rot gefärbt. Der String AT kann durch die vier Zeichen A, C, G und T verlängert werden. Dabei entstehen die Kanten $\{ATA;TAT\}$, $\{ATC;GAT\}$, $\{ATG;CAT\}$ und $\{ATT;AAT\}$. Das Anhängen eines Zeichens σ an einen String s ($s\sigma$) ist gleichzeitig auch das Anhängen des reversen Komplements s^C an das Komplement von σ ($\overline{\sigma}s^C$). Wenn s, wie in diesem Fall, selbstkomplementär ist, dann sind s und s^C identisch und das Anhängen eines Zeichens an s ist gleichzeitig das Anhängen von s an das Komplement des Zeichens. Deshalb existieren in diesem Fall nur vier Kanten und diese Knotenart hat immer Knotengrad 4.

Normale Knoten Ein normaler Knoten zeichnet sich dadurch aus, dass keine Schleife vorhanden ist und das Lesen jedes Zeichens eine neue Kante beschreibt. In Abbildung 5.3(a) ist exemplarisch der normale Knoten $\{AC;GT\}$ mit allen inzidenten Kanten abge-

bildet. Wird der String AC durch die Zeichen A, C, G und T verlängert, entstehen folgende Kanten:

- $\{\mathbf{ACA}; TGT\}$
- $\{\mathbf{ACC}; GGT\}$
- $\{\mathbf{ACG}; CGT\}$
- $\{\mathbf{ACT}; AGT\}$

Wird hingegen der String GT gelesen und dann durch die vier Zeichen verlängert, entstehen die folgenden Kanten:

- $\{TAC; \mathbf{GTA}\}$
- $\{GAC; \mathbf{GTC}\}$
- $\{CAC; \mathbf{GTG}\}$
- $\{AAC; \mathbf{GTT}\}$

Da keine Kantenbeschriftung doppelt vorkommt, hat diese Knotenart immer den Knotengrad 8. In einem G_q-Graphen existieren genau $4^{q-1}/2$ Knoten. Davon besitzen zwei Knoten eine Schleife und $4^{q-1/2}$ sind mit einem selbstkomplementären String beschriftet. Dann existieren in einem G_q-Graphen mit ungeradem q genau $4^{q-1}/2 - 4^{q-1/2} - 2$ normale Knoten.

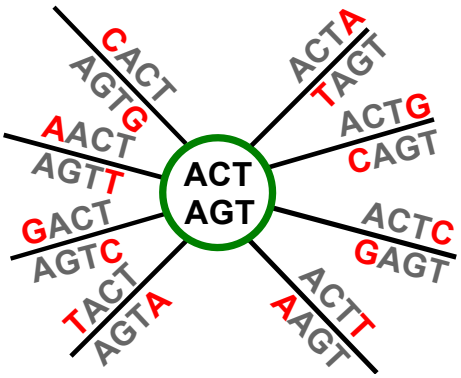
5.1.3 Gerades q

Für gerade q existieren im G_q-Graphen vier Knotenarten: *Normale Knoten*, *Knoten mit Vielfachen eines Zeichens*, *Knoten inzident zu zwei selbstkomplementären Kanten* und *Knoten inzident zu einer selbstkomplementären Kante*. Statt der Knoten, die mit einer selbstkomplementären Sequenz beschrieben sind, existieren für gerade q Kanten, die mit einer selbstkomplementären Sequenz beschriftet sind. Die selbstkomplementären Kanten sind für das weitere Vorgehen wichtig und stellen eine Herausforderung für das Erzeugen einer q-eindeutigen Sequenz dar. Die vier Knotenarten sind in Abbildung 4.1 dargestellt und im Folgenden näher beschrieben.

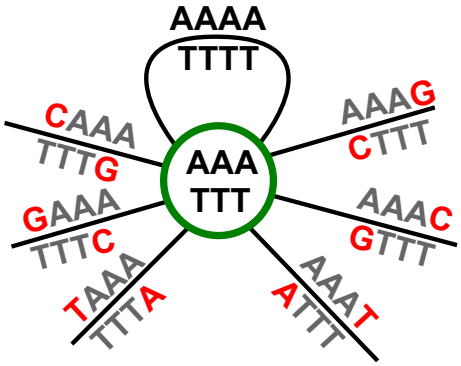
Knoten mit Vielfachen eines Zeichens Der Aufbau des Knotens mit Vielfachen eines Zeichens für gerade q ist analog zum Knoten für ungerade q. Auch hier existieren im G_q-Graphen genau zwei dieser Knoten. Diese Art von Knoten hat Grad 8 und ist in Abbildung 5.4(b) dargestellt.

Knoten inzident zu zwei selbstkomplementären Kanten Für gerade q, existieren zwei Knoten, deren Beschriftung wie folgt aussieht: $\{\sigma\overline{\sigma}\sigma\overline{\sigma}\ldots\sigma; \overline{\sigma}\sigma\overline{\sigma}\sigma\ldots\overline{\sigma} : \sigma \in \Sigma\}$. In Abbildung 5.4(c) ist der Knoten $\{ATA; TAT\}$ für $\sigma = \text{A}$ und $q = 4$ abgebildet. Wird der String ATA gelesen und anschließend eines der Zeichen A, C, G und T, so entstehen folgende Kanten:

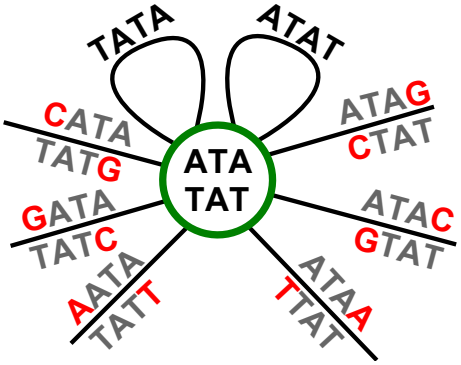
- $\{\mathbf{ATAA}; TTAT\}$
- $\{\mathbf{ATAC}; GTAT\}$
- $\{\mathbf{ATAG}; CTAT\}$
- $\{\mathbf{ATAT}; ATAT\}$



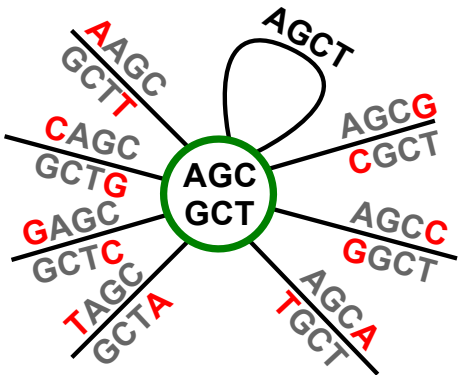
(a) Normaler Knoten



(b) Knoten mit Vielfachen eines Zeichens



(c) Knoten inzident zu zwei selbstkomplementären Kanten



(d) Knoten inzident zu einer selbstkomplementären Kante

Abbildung 5.4: Für gerade q: Die vier Knotenarten des G_4-Graphen

Die letzte Kante ist in zweierlei Hinsicht interessant. Zunächst ist die Sequenz selbstkomplementär, da String und reverses Komplement dieselbe Sequenz beschreiben. Des Weiteren sind die ersten $q-1$ Zeichen (ATA) genau das reverse Komplement der letzten $q-1$ Zeichen (TAT). Da diese beiden Strings beide genau den Knoten $\{ATA; TAT\}$ beschreiben, ist die selbstkomplementäre Kante $\{ATAT\}$ eine Schleife. Analog dazu, entstehen durch Anfügen der Zeichen an TAT folgende Kanten:

- $\{TATA; \mathbf{TATA}\}$
- $\{GATA; \mathbf{TATC}\}$
- $\{CATA; \mathbf{TATG}\}$
- $\{AATA; \mathbf{TATT}\}$

Hier ist die erste Kante $\{TATA\}$ eine selbstkomplementäre Schleife. Die beiden selbstkomplementären Schleifen entstehen durch das abwechselnde Auftreten zweier komplementärer Zeichen. Der Knoten inzident zu zwei selbstkomplementären Kanten hat Knotengrad 10, da die zwei Schleifen den Grad um vier erhöhen und sechs weitere Kanten existieren.

Knoten inzident zu einer selbstkomplementären Kanten Eine selbstkomplementäre Sequenz der Länge q besteht aus den ersten $q-1$ Zeichen s und den letzten $q-1$ Zeichen s^C. Da Knoten des G_q-Graphen genau mit s und s^C der Länge $q-1$ beschriftet sind, ist eine selbstkomplementäre Sequenz im Graph immer eine Schleife. In Abbildung 5.4(d) ist ein solcher Knoten $v = \{AGC; GCT\}$ abgebildet. Wird nach dem String AGC ein Zeichen aus Σ gelesen, entstehen die folgenden Kanten:

- $\{\mathbf{AGCA}; TGCT\}$
- $\{\mathbf{AGCC}; GGCT\}$
- $\{\mathbf{AGCG}; CGCT\}$
- $\{\mathbf{AGCT}; AGCT\}$

Hier ist die letzte Kante $\{AGCT\}$ selbstkomplementär und eine Schleife. Analog dazu kann der String GCT durch die vier Zeichen erweitert werden. Dann entstehen folgende Kanten:

- $\{TAGC; \mathbf{GCTA}\}$
- $\{GAGC; \mathbf{GCTC}\}$
- $\{CAGC; \mathbf{GCTG}\}$
- $\{AAGC; \mathbf{GCTT}\}$

In diesem Fall entsteht keine selbstkomplementäre Sequenz und alle Kanten führen zu anderen Knoten. Jeder dieser Knoten ist inzident zu genau einer selbstkomplementären Sequenz. Bei geradem q existieren genau $4^{q/2}$ selbstkomplementäre Sequenzen. Vier dieser Sequenzen sind von den Kanten repräsentiert, die im vorherigen Paragraphen vorgestellt wurden. Von dieser Knotenart existieren also $4^{q/2} - 4$. Der Knoten inzident zu einer selbstkomplementären Kante hat Grad 9, da eine Schleife und sieben weitere Kanten existieren.

Normale Knoten Wie bei ungeradem q existieren auch für gerades q normale Knoten mit Knotengrad 8. Jede Kante dieser Knoten führt zu einem anderen Knoten. In Abbildung 5.4(a) ist einer dieser Knoten mit allen zugehörigen Kanten abgebildet. Die Anzahl der normalen Knoten in einem G_q-Graphen mit geradem q ist $4^{q-1} - (4^{q/2} - 4) - 2$.

5.2 Reduktion des DSD-Problems auf das Eulerkreisproblem

Der eben beschriebene G_q-Graph kann wie der De Bruijn Graph zum Entwurf von Oligonukleotiden genutzt werden. Die zweite Sequenzdesign-Regel wird, im Gegensatz zum De Bruijn Graphen, schon durch die Struktur des G_q-Graphen erfüllt. Wenn jede Kante genau einmal gelesen wird, dann kommt die Sequenz der Länge q nur einmal in der entworfenen Sequenz vor. Die gelesene Kante ist aber auch mit dem Komplement beschriftet, welches somit nicht mehr gelesen werden kann. Das erste Teilproblem sucht nach der längsten Sequenz, die keine der Sequenzdesign-Regeln verletzt. Ist der G_q-Graph eulersch, dann ist die längste Sequenz, also der längste Pfad im Graphen, ein Eulerkreis oder -pfad. Um im G_q-Graphen einen Eulerkreis zu finden, muss der Graph den Satz von Euler aus Abschnitt 2.4 erfüllen. In einem Graphen existiert genau dann ein Eulerkreis, wenn alle Knoten geraden Grad besitzen. Es existiert ein Eulerpfad, wenn bis auf zwei Knoten alle anderen geraden Grad besitzen. In Abschnitt 5.1 wurden die einzelnen Knotenarten des G_q-Graphen analysiert. Ist q ungerade, so haben alle Knotenarten des Graphen geraden Grad und es ist mit dem Linearzeitalgorithmus und aus [Bra94] möglich einen Eulerkreis zu finden. Dieser muss durch leichte Modifikationen für die erlaubten Kantenpaare an das hier beschriebene Problem angepasst werden. Im Falle des G_q-Graphen mit geradem q existiert eine Knotenart, die keinen geraden Grad hat. Die Knoten inzident zu einer selbstkomplementären Kante sind problematisch, da diese einen ungeraden Grad aufweisen. Diese Knoten werden im Weiteren *Problemknoten* genannt. Im Folgenden wird zunächst die Behandlung der Problemknoten vorgestellt. Weiter wird der Linearzeitalgorithmus aus [Bra94] angepasst und näher beschrieben. Abschließend werden Ergebnisse präsentiert.

Das erste Teilproblem sucht die längste Sequenz, in diesem Fall einen Pfad, der möglichst viele Kanten des G_q-Graphen genau einmal wählt. Da ein Eulerkreis aufgrund der Problemknoten nicht möglich ist, muss der G_q-Graph bei geradem q angepasst werden. Ziel ist es hierbei, die minimale Anzahl an Kanten zu löschen, so dass ein Eulerkreis möglich wird. Der grundlegende Ansatz ist, das Löschen von Kanten inzident zu Problemknoten. In Unterabschnitt 5.2.1 wird die Behandlung der Problemknoten diskutiert, wenn selbstkomplementäre Kante erlaubt sind. Unterabschnitt 5.2.2 hingegen erläutert die Vorgehensweise für die Problemknoten, wenn keine selbstkomplementäre Kanten erlaubt sind.

5.2.1 Behandlung von Problemknoten unter Verwendung von selbstkomplementären Kanten

Ein Problemknoten besitzt nach dem Löschen einer Kante einen geraden Grad. Ist die gelöschte Kante jedoch nur zu einem Problemknoten inzident, so entsteht ein neuer Problemknoten. Statt eine beliebige Kante zu löschen, soll eine Kante inzident zu zwei Problemknoten gelöscht werden. Dazu muss es eine gerade Anzahl an Problemknoten in jedem G_q-Graphen mit geradem q geben. Dass die Anzahl der Problemknoten gerade ist, hat sich schon über die Struktur des G_q-Graphen gezeigt. In Abschnitt 5.1 wurde die Anzahl der Problemknoten $p = 4^{q/2} - 4$ festgestellt.

Um die längste Sequenz zu erhalten, wird zunächst die minimale Anzahl an Kanten gelöscht. Das Löschen der Kanten soll anschließend einen Eulerkreis ermöglichen. Gibt es zwischen zwei Problemknoten immer eine Kante, kann diese ohne weitere Anpassungen gelöscht werden, sofern die restlichen Kanten erlaubte Kantenpaarungen ermöglichen. Bei p Problemknoten werden dann höchstens $p/2$ Kanten gelöscht. Zunächst wird im Folgenden der Aufbau eines einzelnen Problemknotens beschrieben.

5.2 Beobachtung. Aufbau eines Problemknotens

Sei s ein selbstkomplementärer String der Länge $q - 2$, dann ist die selbstkomplementäre Kante $\sigma s\overline{\sigma}$ inzident zu dem Knoten $\{\sigma s, s\overline{\sigma}\}$. Dieser Knoten ist ein Problemknoten.
Die Knotenart aus Abbildung 5.4(c) ist eine Ausnahme, da zwei inzidente Schleifen existieren und der Knotengrad gerade ist.

Die zwei Problemknoten $v_1 = \{AGC; GCT\}$ und $v_2 = \{GCA; TGC\}$ aus Abbildung 5.5 sind durch die Kante $e = \{AGCA; TGCT\}$ verbunden. Diese kann, wie oben schon erwähnt, gelöscht werden, sodass $deg(v_1)$ und $deg(v_2)$ anschließend gerade sind. Wird der String AGC von Knoten v_1 um das Zeichen T erweitert, entsteht die Schleife $\{AGCT\}$. Das Lesen der Schleife bedeutet, dass der String AGC durch den String GCT erweitert wird. Der Knoten v_1 wird somit nicht verlassen, sondern es wird der zweite String des Knotens gewählt.Wird die Kante e zwischen v_1 und v_2 gelöscht, sind $deg(v_1)$ und $deg(v_2)$ gerade und die restlichen Kanten der jeweiligen Knoten können durch erlaubte Kantenpaare beschrieben werden. Für Knoten v_1 könnten die Kanten wie folgt gepaart werden:

- Kante $\{AAGC; GCTT\}$ mit Kante $\{AGCC; GGCT\}$
- Kante $\{CAGC; GCTG\}$ mit Kante $\{AGCG; CGCT\}$

Dann bleiben noch zwei ungepaarte Kanten und eine Schleife übrig. Wird der String GAGC der Kante $\{GAGC; GCTC\}$ genutzt, um den Knoten v_1 zu erreichen, kann nur die Schleife $\{AGCT\}$ gelesen werden. Die letzte Kante $\{TAGC; GCTA\}$ kann erst nach dem Lesen

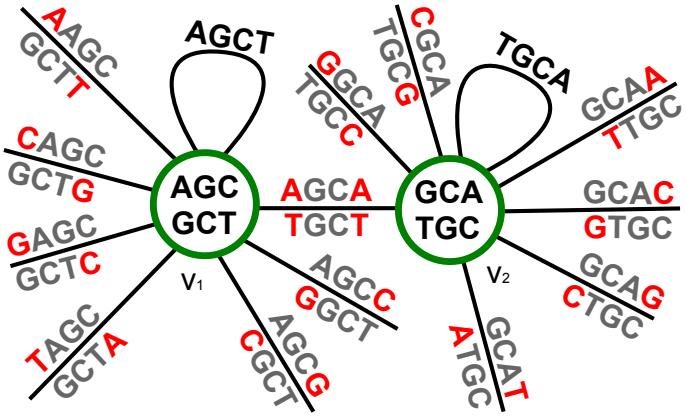


Abbildung 5.5: Zwei Problemknoten verbunden durch eine Kante

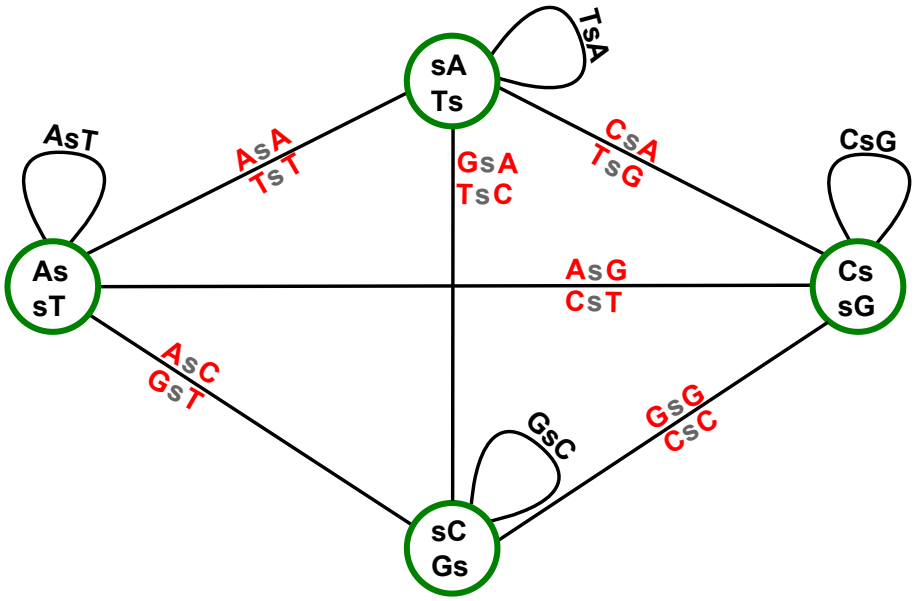


Abbildung 5.6: Verallgemeinerte Problemknotengruppe

der Schleife gewählt werden. So können alle noch übrigen Kanten des Knoten v_1 genutzt werden. Der Knoten v_2 kann analog dazu behandelt werden.

Ausgehend von dem String AGC von Knoten v_1 können weitere zwei Problemknoten durch folgende Kanten erreicht werden:

- $\{AGCC; GGCT\}$, die zum Knoten $v_3 = \{GCC; GGC\}$ führt.
- $\{AGCG; CGCT\}$, die zum Knoten $v_4 = \{GCG; CGC\}$ führt.

Der Problemknoten v_1 ist durch jeweils eine Kante mit drei weiteren Problemknoten verbunden. In Abbildung 5.6 ist eine Gruppe von Problemknoten und ihre Verbindungen dargestellt. Das Alphabet besteht aus vier Zeichen, deshalb kann jeder Knoten um vier Zeichen erweitert werden. Handelt es sich um einen Problemknoten $\{\sigma s, s\overline{\sigma}\}$, so ist s der selbstkomplementär Kern des Knotens. Wird der String σs um ein Zeichen des Alphabets verlängert, dann wird s komplett an den nächsten Knoten weitergereicht. Deswegen sind alle Knoten, die von solch einem String erreicht werden, auch Problemknoten. Diese Gruppe wird im Weiteren *Problemknotengruppe* oder auch kurz *PKG* genannt.

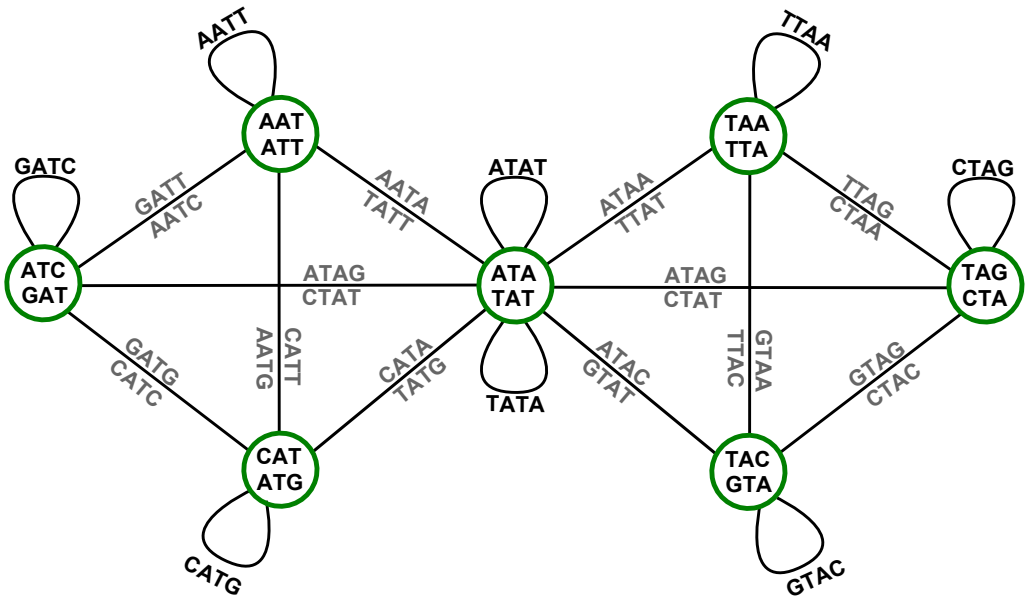


Abbildung 5.7: Spezielle Problemknotengruppe

In einer PKG existieren sechs Kanten zwischen den vier Problemknoten. Abbildung 5.6 zeigt eine verallgemeinerte PKG mit diesen sechs Kanten. Von diesen Kanten müssen nur zwei gelöscht werden, damit die Problemknoten geraden Grad erhalten. Die Knotenpaare der zu löschenden Kanten müssen disjunkt sein.

Wie in Abschnitt 5.1 existieren in jedem G_q-Graphen genau zwei Knoten inzident zu zwei selbstkomplementären Kanten. Diese Knoten bilden eine *spezielle Problemknotengruppe* (abgekürzt *sPKG*), die für den Knoten $\{ATA, TAT\}$ in Abbildung 5.7 zu sehen ist. An einer sPKG sind sieben Knoten beteiligt, von denen nur sechs Problemknoten sind. In einer sPKG müssen vier Kanten gelöscht werden, damit die sechs Problemknoten geraden Grad erhalten. Der Knoten inzident zu zwei selbstkomplementären Kanten hat geraden Grad. Damit der Grad dieses Knotens gerade bleibt, müssen zwei Kanten des Knotens gelöscht werden. In Abbildung 5.7 können beispielsweise die Kanten $\{AATA; TATT\}$ und $\{ATAA; TTAT\}$ gelöscht werden. Anschließend müssen zwei weitere Kanten gelöscht werden. Die Kanten müssen jeweils zwei der restlichen vier Problemknoten verbinden. In diesem Beispiel müssen die Kanten $\{GATG; CATC\}$ und $\{GTAG; CTAC\}$ gelöscht werden.

Es existieren $4^{q/2}$ selbstkomplementäre Kanten. Acht dieser Kanten sind jeweils an einer sPKG, wie in Abbildung 5.7 zu sehen ist, beteiligt. Es existieren in jedem G_q-Graphen genau zwei dieser Gruppen und somit sind 16 selbstkomplementäre Kanten abgedeckt. Für jede dieser sPKG müssen vier Kanten gelöscht werden, also werden für die beiden Gruppen insgesamt acht Kanten gelöscht. Die restlichen $4^{q/2} - 16$ selbstkomplementären Kanten sind jeweils an einen Problemknoten beteiligt. Es gibt also $4^{q/2} - 16$ weitere Problemknoten. Diese Problemknoten sind jeweils zu viert in einer PKG zusammengefasst. Demnach existieren genau $(4^{q/2} - 16)/4 = 4^{q/2-1} - 4$ PKG, für die jeweils zwei Kanten

gelöscht werden müssen. Insgesamt müssen dann genau d Kanten gelöscht werden, um einen Eulerkreis in einem G_q-Graphen mit geradem q zu ermöglichen:

$$\begin{aligned} d &= 2 \cdot 4 + (4^{q/2-1} - 4) \cdot 2 \\ &= 8 + 2 \cdot 4^{q/2-1} - 8 \\ &= \frac{2 \cdot 4^{q/2}}{4} \\ &= \frac{4^{q/2}}{2} \end{aligned}$$

Zwischen den Problemknoten $p_1 = \{AAT; ATT\}$ und $p_2 = \{TAA; TTA\}$ aus Abbildung 5.7 existiert zusätzlich die Kante $e = \{TAAT; ATTA\}$. Das Löschen dieser Kante ist jedoch nicht möglich, da die restlichen Kanten mit den Schleifen anschließend keine erlaubten Kantenpaarungen mehr ermöglichen.

Wenn nur ein Eulerpfad gesucht wird, dann können zwei Kanten weniger gelöscht werden. Diese zwei Kanten müssen jedoch inzident zu einem Knoten sein, der wiederum inzident zu zwei selbstkomplementären Kanten ist. So existieren im Graphen genau zwei Problemknoten, die jeweils Start- und Endknoten des Eulerpfades sind. Die minimale Anzahl der zu löschenden Kanten, um einen Eulerpfad zu ermöglichen, ist dann:

$$d = \frac{4^{q/2}}{2} - 2$$

Sind selbstkomplementäre Sequenzen erlaubt, dann ist n eine obere Schranke für die Anzahl der maximal wählbaren Kanten in einem G_q-Graphen mit geradem q:

$$n = \frac{4^q}{2} + \frac{4^{q/2}}{2}.$$

Wird nun die Anzahl der zu löschenden Kanten abgezogen, dann ist n_{max} die obere Schranke für die Anzahl maximal wählbarer Kanten in einem G_q-Graphen mit geradem q:

$$\begin{aligned} n_{max} &= n - d \\ &= (\frac{4^q}{2} + \frac{4^{q/2}}{2}) - (\frac{4^{q/2}}{2} - 2) \\ &= \frac{4^q}{2} + \frac{4^{q/2}}{2} - \frac{4^{q/2}}{2} + 2 \\ &= \frac{4^q}{2} + 2. \end{aligned}$$

Die hier beschriebene Methode kann nur dann angewandt werden, wenn selbstkomplementäre Kanten erlaubt sind. Dies ist allerdings durch die dritte Sequenzdesign-Regel untersagt. Ist die Länge der zu erzeugenden Sequenz wichtiger als die Einhaltung aller Sequenzdesign-Regeln, so kann mit dieser Vorgehensweise die längste Sequenz erzeugt werden. Diese ist dann q-eindeutig unter Berücksichtigung der ersten beiden Regeln. Zur Erstellung der DNA-Kacheln ist die dritte Regeln jedoch sehr wichtig. Deshalb wird im nächsten Unterabschnitt eine Methode vorgestellt, die keine selbstkomplementären Kanten verwendet.

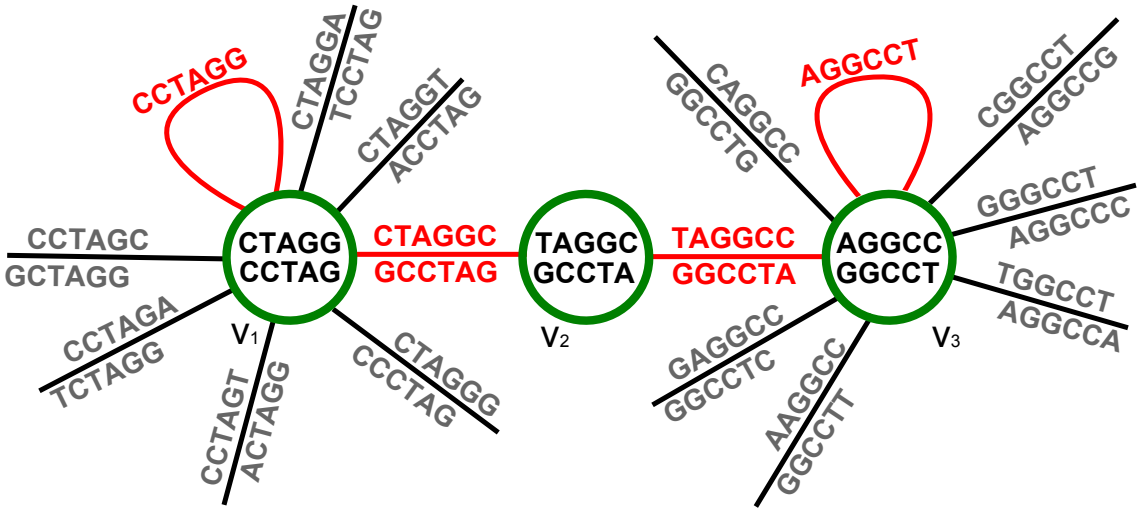


Abbildung 5.8: Weg zwischen zwei Problemknoten

5.2.2 Behandlung von Problemknoten ohne Verwendung von selbstkomplementären Kanten

In der Arbeit von *James W. Anderson et al.* [AFN06] wird eine Vorgehensweise vorgestellt, die es ermöglicht einen Eulerkreis im G_q-Graphen mit geradem q zu finden. Einige Kanten werden doppelt gewählt, um so Kreise zu erzeugen. Die erzeugten Kreise enthalten jeweils zwei Problemknoten und können zunächst aus dem G_q-Graphen entfernt werden. Dann kann in dem restlichen Graphen ein Eulerkreis gefunden werden. Dieser wird anschließend mit den vorher gefundenen Kreisen vereint. Diese Vorgehensweise kann für das DSD-Problem genutzt werden, indem die doppelt gewählten Kanten gelöscht werden. Ein selbstkomplementärer String s der Länge q kann wie folgt beschrieben werden:

$$\sigma_1\sigma_2\ldots\sigma_{q/2-1}\sigma_{q/2}\ldots\sigma_q$$

Dabei sei der erste Teilstring $s_1 = \sigma_1\sigma_2\ldots\sigma_{q/2-1}$ das reverse Komplement des zweiten $s_2 = s_1^C = \sigma_{q/2}\ldots\sigma_q$. Der String s kann dann wie folgt permutiert werden:

$$\begin{aligned}
s &= \sigma_1\sigma_2\ldots\sigma_{q/2-1}\sigma_{q/2}\ldots\sigma_q \\
r_1 &= \sigma_2\ldots\sigma_{q/2-1}\sigma_{q/2}\ldots\sigma_q\sigma_1 \\
r_2 &= \sigma_3\ldots\sigma_{q/2-1}\sigma_{q/2}\ldots\sigma_q\sigma_1\sigma_2 \\
\vdots &= \ldots \\
r_{q/2-1} &= \sigma_{q/2-1}\sigma_{q/2}\ldots\sigma_q\sigma_1\ldots\sigma_{q/2-2} \\
r_{q/2} &= \sigma_{q/2}\ldots\sigma_q\sigma_1\ldots\sigma_{q/2-1}
\end{aligned}$$

Der String $r_{q/2}$ ist selbstkomplementär. Im G_q-Graphen kann jeder Problemknoten über $q/2-1$ Kanten einen weiteren Problemknoten erreichen. Abbildung 5.8 zeigt einen Ausschnitt aus einem G_6 Graphen. Der Knoten $v_1 = \{CTAGG; CCTAG\}$ ist inzident zur

selbstkomplementären Sequenz $s = CCTAGG$. Wird s wie oben beschrieben permutiert, ergeben sich folgende Sequenzen:

$$\begin{aligned} s &= CCTAGG \\ r_1 &= CTAGGC \\ r_2 &= TAGGCC \\ r_3 &= AGGCCT \end{aligned}$$

In Abbildung 5.8 wird im Knoten v_1 mit dem String **CCTAG** begonnen. Dann wird s gewählt und im Knoten v_1 ist deshalb der String **CTAGG** gelesen worden. Als nächstes werden die Kanten mit den Sequenzen r_1 und r_2 gewählt. Als letztes ist dann der String **AGGCC** im Problemknoten $v_3 = \{AGGCC; GGCCT\}$ gelesen worden. Dieser String ermöglicht das Lesen der selbstkomplementären Sequenz r_3, die wieder im Knoten v_3 endet. Ausgehend vom String **GGCCT** wird mit den Sequenzen r_2^C und r_1^C über den Knoten v_2 wieder der Knoten v_1 erreicht. So entsteht der folgende Kreis:
$k = (CCTAGG, CTAGGC, TAGGCC, AGGCCT, GGCCTA, GCCTAG)$.
Dieser ist in Abbildung 5.8 rot gekennzeichnet. Der Kreis k kann anschließend gelöscht werden und die Problemknoten v_1 und v_3 haben geraden Grad. Der Knoten v_2 wurde mit zwei Pfaden durchlaufen und behält deshalb auch geraden Grad. Die restlichen Kanten der beiden Problemknoten bilden anschließend erlaubte Kantenpaare. Eine mögliche erlaubte Kantenpaarung für Knoten v_1 ist:

- $\{CCTAGC; GCTAGG\}$ mit $\{CTAGGA; TCCTAG\}$
- $\{CCTAGA; TCTAGG\}$ mit $\{CTAGGT; ACCTAG\}$
- $\{CCTAGT; ACTAGG\}$ mit $\{CTAGGG; CCCTAG\}$

Für Knoten v_3 ist beispielsweise folgende Kantenpaarung möglich:

- $\{CAGGCC; GGCCTG\}$ mit $\{AGGCCG; CGGCCT\}$
- $\{GAGGCC; GGCCTC\}$ mit $\{AGGCCC; GGGCCT\}$
- $\{AAGGCC; GGCCTT\}$ mit $\{AGGCCA; TGGCCT\}$

Sind alle so entstandenen Kreise gelöscht, existiert ein Eulerkreis im G_q-Graphen. Die Anzahl der Kanten, die zusätzlich gelöscht werden müssen, wird wie folgt bestimmt. Die Pfade zwischen zwei Problemknoten haben die Länge $q/2 - 1$. Es existieren $4^{q/2}$ selbstkomplementäre Kanten. Vier von diesen Kanten sind inzident zu zwei Knoten, die keine Problemknoten sind. Bleiben also $4^{q/2} - 4$ selbstkomplementäre Kanten, die jeweils inzident zu einem Problemknoten sind. Zwischen jeweils zwei Problemknoten soll ein Pfad gefunden werden, der nach der vorgestellten Vorgehensweise zusammen mit den selbstkomplementären Kanten einen Kreis bildet. Also existieren $(4^{q/2} - 4)/2$ Pfade der Länge $q/2 - 1$. Die Anzahl der zu löschenden Kanten d ist:

$$d = \frac{4^{q/2} - 4}{2} \cdot (q/2 - 1)$$

Um die Anzahl der wählbaren Kanten zu maximieren, kann wie in Unterabschnitt 5.2.1 ein Eulerpfad gesucht werden. Deshalb dürfen zwei Problemknoten mit ungeradem Grad im Graphen existieren. Diese sind der Start- und Endknoten des Eulerpfades. Die Anzahl der Problemknoten ist $4^{q/2} - 4 - 2$ und die Anzahl d ist dann:

$$d = \frac{4^{q/2} - 6}{2} \cdot (q/2 - 1)$$

Sind keine selbstkomplementären Sequenzen erlaubt, dann kann die obere Schranke für die Anzahl der maximal wählbaren Kanten n_{max} in einem G_q-Graphen bei geradem q wie folgt berechnet werden:

$$\begin{aligned} n_{max} &= \frac{4^q}{2} - \frac{4^{q/2}}{2} - d \\ &= \frac{4^q}{2} - \frac{4^{q/2}}{2} - \frac{4^{q/2}-6}{2} \cdot (q/2-1) \\ &= \frac{4^q - 4^{q/2}}{2} - \frac{(4^{q/2}-6)(q-2)}{4}. \end{aligned}$$

5.2.3 Ein Linearzeitalgorithmus zur Berechnung längster Pfade

Der Algorithmus 5.1 berechnet einen Eulerkreis für einen G_q-Graphen mit ungeradem q. Für Graphen mit geradem q berechnet der Algorithmus 5.1 nach Behandlung der Problemknoten einen Eulerpfad. Nach dem Satz von Euler aus Abschnitt 2.4 existiert ein Eulerkreis in einem Graphen, wenn alle Knoten geraden Grad aufweisen. Es existiert ein Eulerpfad, wenn genau zwei Knoten ungeraden Grad und alle anderen geraden Grad aufweisen. Für gerade q muss der Graph zuerst angepasst werden.

Nach der Behandlung der Problemknoten entsteht der Graph $G_q = (V, E')$. Die Menge V enthält genau zwei Knoten, die ungeraden Grad besitzen. Einer dieser Knoten wird als Startknoten markiert, während der zweite Knoten der Endknoten des Eulerpfades ist. Die Menge E' ist die angepasste Menge der Kanten in einem G_q-Graphen mit geradem q. Für ungerade q weisen alle Knoten des Graphen geraden Grad auf.

Zu jeder Kante e existiert im Graphen eine Kante e'. Diese beiden Kanten sind ein erlaubtes Kantenpaar (e, e'). Ein erlaubtes Kantenpaar ist wie in Unterabschnitt 5.1.1 beschrieben aufgebaut. Zunächst prüft der Algorithmus, ob ein Startknoten gegeben ist. Ist dies der Fall, so wird ein String des Knoten $v_{start} = \{s; s^C\}$ gewählt. Als Nächstes wird eine Kante inzident zu v_{start} betrachtet, die den gewählten String um ein Zeichen verlängert. Diese Kante ist die Startkante des Eulerpfades. Existiert kein Startknoten, dann kann eine zufällige Kante als Startkante gewählt werden. An jedem erreichten Knoten wird dann überprüft, ob die aktuelle Kante e eine Kante e' an diesem Knoten besitzt. Existiert ein erlaubtes Kantenpaar (e, e'), wird die Kante e aus der Menge E gelöscht, dem Pfad P hinzugefügt und die Kante e' ist dann die aktuelle Kante e. Wenn keine erlaubten Kantenpaare für die aktuelle Kante existieren, dann wird der Pfad der Menge $\boldsymbol{P}$ hinzugefügt. Ist die Menge E leer, sind alle Kanten abgelaufen worden. Sind in E noch Kanten vorhanden, müssen diese Kreise ergeben. Aus der übrigen Menge E wird eine zufällige Kante gewählt und wie schon beschrieben fortgefahren. Jede Kante wird genau einmal betrachtet und dann aus der Menge E gelöscht. Also ist der Zeitaufwand $\mathcal{O}(|E|)$. Die einzelnen Pfade der gefundene Menge $\boldsymbol{P}$ müssen anschließend zu einem Pfad vereinigt werden. In dieser Mengen existieren nur Kreise und höchstens ein Pfad. Zunächst werden alle Sequenzen der Länge $q-1$ überprüft. Im Graphen sind das alle Knoten. Zwei Pfade der Menge $\boldsymbol{P}$ können vereinigt werden, wenn ein Knoten in beiden Pfaden vorkommt. Insgesamt beträgt der Zeitaufwand bei geschickter Implementierung $\mathcal{O}(|E| + |V|)$. Der resultierende Pfad $P = (e_1, e_2, \ldots)$ kann in zwei Richtungen gelesen werden, da die Kanten mit zwei Strings beschriftet sind. Das Lesen in beide Richtungen erzeugt eine q-eindeutige Sequenz. Diese beiden Sequenzen sind jeweils das reverse Komplement der anderen. Ein Beispiel dazu ist bereits in Abbildung 5.1 und Tabelle 5.1 gegeben.

5.2.4 Ergebnisse des Linearzeitalgorithmus

In Tabelle 5.2 sind die Ergebnisse des Linearzeitalgorithmus und des ILP für gerade q aufgelistet. Es wurden nur q-gramme bis zu einer Länge von neun Basen betrachtet. Es geht hervor, dass für ungerade q das Ergebnisse beider Ansätze mit den theoretisch errechneten Werten übereinstimmt. Für gerade q hingegen erzielt das ILP bessere Lösungen.

Die Abweichung der maximal wählbaren Kanten liegt offenbar an der Berechnung der zu löschenden Kanten d. In einem G_q-Graphen mit geradem q wurde zwischen zwei Problemknoten immer ein Pfad der Länge $q/2 - 1$ gelöscht. Dieser Pfad wurde durch das Permutieren der Basen eines selbstkomplementären Strings erzeugt und in Unterabschnitt 5.2.2 dargelegt. Doch es existieren auch Pfade, die kürzer sind und gelöscht werden können. Ein Beispiel dafür ist in Abbildung 5.9 zu sehen. Der selbstkomplementäre String s = ATTAAT ist inzident zu dem Knoten $v_1 = \{TTAAT; ATTAA\}$. Der String s würde nach der Vorgehensweise aus Unterabschnitt 5.2.2 zum String s' = AATATT permutiert werden. Der Problemknoten v_1 könnte deshalb durch einen Pfad mit dem Problemknoten $v_1' = \{AATAT; ATATT\}$ verbunden werden. Die zwei selbstkomplementären Strings und der erzeugte Pfad ergeben dann einen Kreis, der aus dem Graphen gelöscht wird.

Es existiert auch eine andere Lösung. Die Kante $e = \{TTAATT; AATTAA\}$ aus Abbildung 5.9 verbindet den Problemknoten v_1 mit dem Knoten $v_2 = \{TAATT; AATTA\}$. Diese Kante ergibt mit den zwei selbstkomplementären Strings s und r = TAATTA einen

Eingabe: $G_q = (V, E)$
Ausgabe: $\boldsymbol{P}$ Menge von Pfaden $P_1, P_2, \ldots$

if existiert Startknoten **then**
  wähle $e \leftarrow$ eine vom Startknoten ausgehende Kante
  füge e zu P hinzu
  entferne e aus E
  while e und $e' \in E$ ist erlaubtes Kantenpaar **do**
    füge e' zu P hinzu
    entferne e' aus E
    setze $e \leftarrow e'$
  end while
  füge P zu $\boldsymbol{P}$
end if

while $E \neq \emptyset$ **do**
  erzeuge neues leeres P
  $e \leftarrow$ zufällige Kante aus E
  while e und $e' \in E$ ist erlaubtes Kantenpaar **do**
    füge e zu P hinzu
    entferne e aus E
    setze $e \leftarrow e'$
  end while
  füge P zu $\boldsymbol{P}$
end while
return $\boldsymbol{P}$

Algorithmus 5.1: Eulerkreis im G_q-Graphen

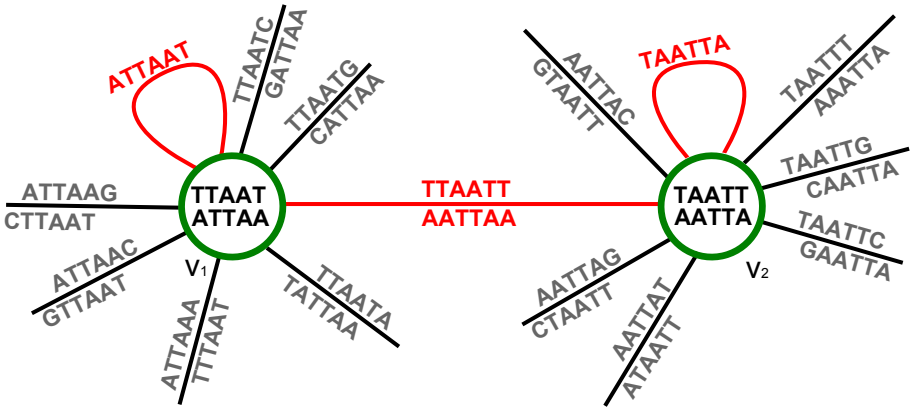


Abbildung 5.9: Löschen einer Kanten zwischen zwei Problemknoten

Kreis, der gelöscht werden kann. Der erzeugte Kreis ist $k = (s, e, r, e)$. Die restlichen Kanten der Problemknoten v_1 und v_2 ergeben erlaubte Kantenpaare.

Nun sind die Knoten $v_1' = \{AATAT; ATATT\}$ und $v_2' = \{TAATA; TATAA\}$ übrig. Diese Problemknoten müssen miteinander verbunden werden, da v_1 und v_2 schon miteinander verbunden sind. In Abbildung 5.10 ist ein Pfad zwischen den beiden Knoten angegeben, der mit den Schleifen s' = AATATT und r' = TTATAA einen Kreis ergibt. Dieser kann anschließend wieder gelöscht und die restlichen Kanten der Problemknoten können zu erlaubten Kantenpaaren kombiniert werden. Der Pfad zwischen den zwei Problemknoten v_1' und v_2' ist in diesem Beispiel drei Kanten lang. Da ein Eulerpfad gesucht wird, müssen nicht beide Pfade gelöscht werden. In der optimalen Lösung wird der Pfad minimaler Länge gelöscht, der in diesem Beispiel eine Kante lang ist. Der Pfad der Länge drei aus Abbildung 5.10 wird nicht gelöscht und die beiden Problemknoten v_1' und v_2' sind Start- und Endknoten des gesuchten Eulerpfades. Für die Anzahl der zu löschenden Kanten d bedeutet dies, dass nicht alle Pfade zwischen zwei Problemknoten $q/2 - 1$ lang sind.

Tabelle 5.2 gibt für $q = 6$ eine Lösung bestehend aus 1958 Kanten an. Wenn zwischen den zwei Knoten v_1 und v_2 nur eine Kante statt $q/2 - 1 = 6/2 - 1 = 2$ Kanten gelöscht werden, dann können 1959 Kanten gewählt werden. Diese Lösung ist optimal, da das ILP dieselbe Anzahl an maximal wählbaren Kanten errechnet. Die Anzahl der zu löschenden Kanten d ist für den Fall $q = 4$ optimal, da jeder Pfad zwischen Problemknoten genau eine Kante lang ist: $q/2-1 = 4/2-1 = 1$. Für gerade $q \geq 6$ ist die Anzahl d aus Unterabschnitt 5.2.2 dann eine obere Schranke.

Tabelle 5.2: Vergleich zwischen ILP und Linearzeitalgorithmus

q-gramme	Gesamtkanten	Algorithmus mit selbstkomp. Kanten	ILP mit selbstkomp. Kanten	Algorithmus ohne selbstkomp. Kanten	ILP ohne selbstkomp. Kanten
2	16	10	10	6	6
4	256	130	130	115	115
6	4096	2050	2050	1958	1959
8	65536	32770	32770	32265	32279

5.3 Lösen der Teilprobleme des DSD-Problems auf Basis des ungerichteten De Bruijn Graphen

Der Linearzeitalgorithmus aus Abschnitt 5.2 kann wie das ILP aus Abschnitt 4.2 an drei der Teilprobleme des DSD-Problems angepasst werden.

Für das erste Teilproblem ist die Adaption des Algorithmus analog zu der Adaption des ILP. Zunächst wird mittels des Algorithmus 5.1 die längste q-eindeutige Sequenz erzeugt. Wie schon in Unterabschnitt 5.2.4 diskutiert existiert für gerade q nicht eine Abweichung des Ergebnisses vom Optimum. Diese Abweichung ist jedoch akzeptabel. Das Teilen der erzeugten q-eindeutigen Sequenz in die verschiedenen Oligonukleotide erfolgt nach der Methode aus Abschnitt 4.3.

Das Erstellen der sticky ends und des Internals wird wie schon in Abschnitt 4.3 beschrieben gelöst. Dazu ist keine Graphenstruktur nötig, da die Subsequenzen durch die Mengen Z und Q verwaltet werden. Dieser Sachverhalt wird ausgenutzt, um korrekte Lösungen für die Modifikation der sticky ends zu berechnen.

Im Folgenden wird die Adaption des ungerichteten De Bruijn Graphen an das zweite und fünfte Teilproblem des DSD-Problems erläutert.

5.3.1 Zweites Teilproblem: Längster Pfad bei gegebenem Kontext

Ist ein Kontext gegeben, dann müssen die Sequenzen der einzelnen Oligonukleotide aus dem Graphen gelöscht werden. Nach dem Löschen der Oligonukleotide aus dem vorgegebenen Kontext ist der Graph möglicherweise nicht mehr eulersch. Das Löschen der einzelnen Sequenzen führt dazu, dass im Graphen G_q Knoten mit ungeradem Grad entstehen. Sei zum Beispiel der Kontext $K = \{GTCCAGT\}$ und $q = 6$, dann existieren zwei 6-gramme, die nicht mehr verwendet werden dürfen. Die Menge der nicht erlaubten Kanten ist dann $Q = \{GTCCAG, TCCAGT\}$. Abbildung 5.11 zeigt einen Ausschnitt dieses Graphen G_6. Wird die Menge Q entfernt, dann beschreibt der gestrichelte Pfad die Sequenz aus K. Durch das Wegfallen der Kanten $\{GTCCAG; CTGGAC\}$ und $\{TCCAGT; ACTGGA\}$ haben die Knoten v_1 und v_2 ungeraden Grad. Der Algorithmus 5.1 benötigt als Eingabe einen Graphen, der höchstens zwei Knoten ungeraden Grades beinhaltet. Besteht der Kontext aus mehreren Oligonukleotiden, dann muss der Graph G_q nach dem Löschen der Oligonukleotide des Kontextes angepasst werden. Das Einfügen einer Kante zwischen zwei

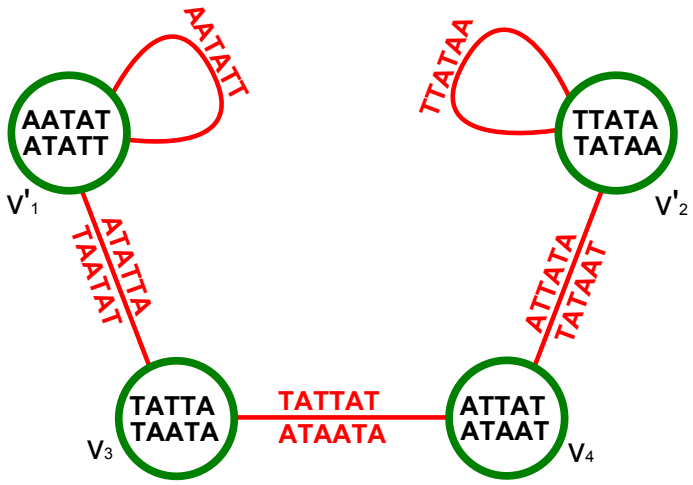


Abbildung 5.10: Pfad zwischen zwei Problemknoten

Knoten ungeraden Grades erlaubt es wieder einen Eulerkreis zu finden. In Abbildung 5.11 ist die rot gestrichelte Kante eine *Pseudokante*. Diese weist keine Beschriftung auf, sondern verbindet nur zwei Knoten ungeraden Grades. Die Pseudokanten in einem Graphen sind Stellvertreter für gelöschte Sequenzen.

Die Sequenzen des Kontextes müssen nicht den Sequenzdesign-Regeln entsprechen. Deshalb können schon bei einem Kontext aus nur einem Oligonukleotid auch mehr als zwei Knoten mit ungeradem Grad entstehen. Wird beispielsweise die Sequenz $s = \text{CACTAGTT}$ aus dem Graphen G_3 gelöscht, dann entstehen mehr als zwei Knoten mit ungeradem Grad. In Abbildung 5.12 ist der Graphenausschnitt der Sequenz s mit den dazugehörigen Pseudokanten skizziert.

In einem Graphen existiert nach dem Satz 5.3 immer eine gerade Anzahl an Knoten mit ungeradem Grad.

5.3 Satz. *Für jeden Graphen existiert eine gerade Anzahl Knoten mit ungeradem Grad.*

Beweis. Die Summe der Grade aller Knoten V in einem Graphen ist gerade, da jede Kante zu genau zwei Knoten inzident ist. Zunächst werden die Knoten aus V in Knoten mit geradem Grad V_g und Knoten mit ungeradem Grad V_u aufgeteilt. Dann läßt sich die Summe der Grade wie folgt formulieren:

$$\sum_{v \in V} deg(v) = \sum_{v \in V_g} deg(v) + \sum_{v \in V_u} deg(v)$$

Die Summe über die Knotengrade aus V_g ist gerade. Daraus folgt, dass die Summe der Knotengrade aus V_u gerade sein muss, damit die Summe der Knotengrade über V gerade sein kann. Somit muss die Anzahl der Knoten aus V_u gerade sein. □

Nach dem Löschen der Sequenzen aus dem Kontext entsteht also immer eine gerade Anzahl an Knoten mit ungeradem Grad. Deshalb ist es immer möglich eine Pseudokante zwischen zwei Knoten mit ungeradem Grad einzufügen. Für das Beispiel aus Abbildung 5.12 sind die Knoten v_1, v_2, v_4 und v_5 durch zwei Pseudokanten verbunden. Da ein Eulerpfad zur Lösung dieses Problems genügt, kann eine Pseudokante vernachlässigt werden und die

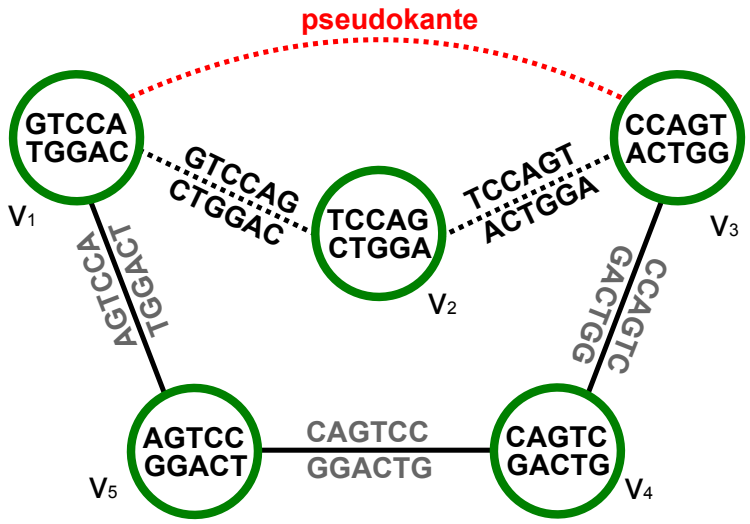


Abbildung 5.11: Entstehung einer Pseudokante durch das Löschen der Sequenz $s = \text{GTCCAGT}$

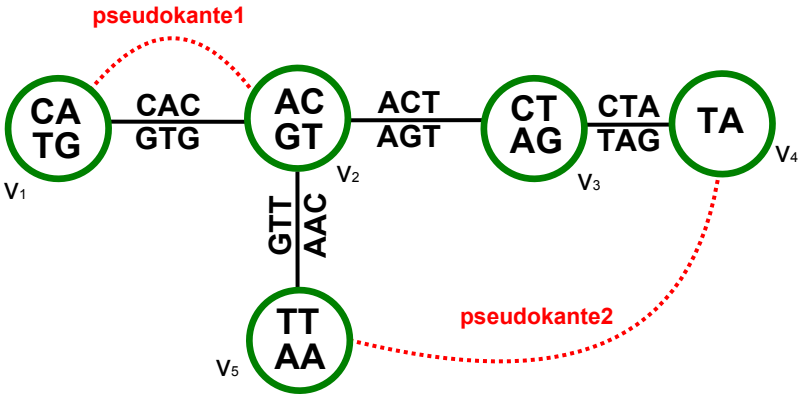


Abbildung 5.12: Entstehung zweier Pseudokanten durch das Löschen der Sequenz s = CAC-TAGTT

betroffenen Knoten sind dann Start- und Endknoten. Nach dem Einfügen der Pseudokanten existieren im Graphen zwei Knoten mit ungeradem Grad. Alle anderen Knoten weisen geraden Grad auf.

Mittels der Pseudokanten kann der Algorithmus 5.1 auf dem angepassten Graphen arbeiten. Die Pseudokante e_p ersetzt den Pfad $P = (e_1, e_2, \ldots, e_n)$, der durch die Sequenz $p \in K$ beschrieben wird. Eine Pseudokante e_p und eine Kante e' sind ein erlaubtes Kantenpaar an dem Quellknoten der Pseudokante, wenn die Kante e_1 und e' ein erlaubtes Kantenpaar sind. Für den Zielknoten sind e_p und e' ein erlaubtes Kantenpaar, wenn e_n und e' ein erlaubtes Kantenpaar sind. Der Algorithmus wählt zunächst alle Kanten an einem Knoten, die keine Pseudokanten sind. Gibt es keine andere Möglichkeit, so wird die Pseudokante gewählt. Diese fügt keine Zeichen an die bis dahin entstandene Sequenz, sondern beendet die Erzeugung der Sequenz. Der Knoten am anderen Ende der Pseudokante wird der Startknoten des nächsten Pfades. Der angegebene Algorithmus findet eine Menge $\boldsymbol{P} = \{P_1, P_2, \ldots\}$, die anders als für das erste Teilproblem mehrere Pfade enthält. In dieser Menge wird anschließend nach Kreisen gesucht. Dann wird geprüft, ob eine Subsequenz der Länge $q-1$, also ein Knoten, gleichzeitig in einem Pfad und einem Kreis der Menge $\boldsymbol{P}$ existiert. Ist dies der Fall, so kann der Kreis in den Pfad integriert werden und der Pfad wird um die Länge des Kreises verlängert. Aus der Menge $\boldsymbol{P}$ kann abschließend die längste Sequenz ausgewählt werden. Die restlichen Pfade können nach Bedarf entweder verworfen oder auch für andere Zwecke verwendet werden. Sind diese Pfade ausreichend lang, dann können weitere Oligonukleotide gewonnen werden ohne die Sequenzdesign-Regeln zu verletzen.

5.3.2 Fünftes Teilproblem: Erstellen von DNA-Kacheln

Das Erstellen einer DNA-Kachel nach dem Vorbild der Kacheln A^2 und B^3 muss mit mindestens 5-grammen modelliert werden. Die C-Komponente dieser DNA-Kacheln besteht aus vier Subsequenzen von vier Thyminbasen und vier Sequenzen der Länge 21. Die vier Subsequenzen werden ohne Überprüfung der Regeln in die erzeugte Sequenz eingefügt. Dabei könnten 5-gramme doppelt entstehen und reverse Komplemente vorhanden sein. Wird der String t = TTTT in den String s = ATTAAAAA an dritter Position eingefügt, dann entsteht der String s_t = AT TTTT TAAAAA. Der String s_t besteht dann aus folgenden 5-grammen:

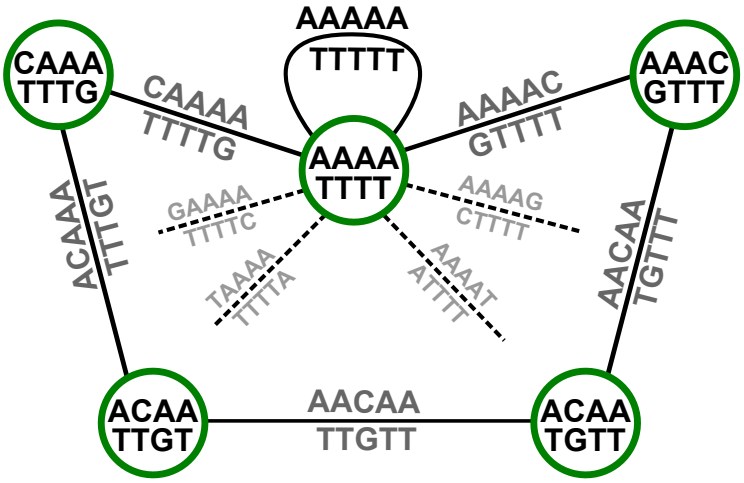


Abbildung 5.13: T-Kreise im G_q Graphen

- ATTTT
- TTTTT
- TTTTT
  ⋮
- AAAAA

In der C-Komponente würde in diesem Fall das 5-gramm TTTTT doppelt und das Komplement AAAAA einmal enthalten sein. Die Knoten der Graphenstruktur G_5 sind genau 4-gramme. Die doppelten 5-gramme beeinträchtigen den Assemblierungsprozess aber nicht, solange kein reverses Komplement vorkommt. Der String t wird im Graphen als Knoten $v_T = \{TTTT; AAAA\}$ dargestellt.

Der oben beschriebene Fall kann im Graphen G_q vermieden werden, wenn es nicht möglich ist den Knoten v_T zur Erzeugung der Sequenz zu nutzen. Zunächst wird dieser Knoten aus dem Graphen entfernt, um bei dem Entwurf der Sequenz genau alle Sequenzen zu vermeiden, die den String t enthalten. Dadurch werden gleichzeitig auch alle Strings verboten, die mit t^C = AAAA beschriftet sind.

Damit aber ein Eulerkreis möglich bleibt, müssen die adjazenten Knoten von v_T bearbeitet werden. Dazu werden alle kürzesten Kreise gesucht, die den Knoten v_T beinhalten. Es existieren sieben Kanten am Knoten v_T. Eine der Kanten ist die Schleife $\{AAAAA; TTTTT\}$, die ohne weiteren Aufwand gelöscht werden kann. Die restlichen sechs Kanten führen zu drei Kreisen. In Abbildung 5.13 ist der Knoten v_T mit einem der Kreise skizziert. Nach dem Entfernen aller Kreise und der Schleife ist der Graph G_q wieder eulersch. Im modifizierten Graphen kann nach Algorithmus 5.1 ein Eulerkreis gesucht werden und anschließend eine q-eindeutige Sequenz erzeugt werden. Diese Sequenz enthält weder den String t noch sein Komplement AAAA. Wird beim Generieren der DNA-Kachel die Subsequenz t an einer beliebigen Position eingefügt, so können folgende Fälle auftreten:

1. String t wird in die Teilsequenz TTT eingefügt.
2. String t wird zwischen TT eingefügt.
3. String t wird zwischen σ T oder Tσ eingefügt, mit $\sigma \neq$ T.
4. String t wird zwischen $\sigma_1\sigma_2$ eingefügt, mit $\sigma_1, \sigma_2 \neq$T.

Im ersten Fall entsteht drei Mal das 5-gramm TTTTT. Werden also zufällig alle vier Subsequenzen t zwischen drei Thyminbasen platziert, so existiert das 5-gramm TTTTT

12 Mal. Für den zweiten Fall würden an der Stelle zwei 5-gramme entstehen und in der gesamten C-Komponente höchstens acht. Der dritte Fall würde jeweils ein solches 5-gramm erzeugen und in der gesamten Komponente wären es vier. Im letzten Fall kommt das 5-gramm kein Mal in der Sequenz vor.

In den ersten drei Fällen ist die C-Komponente nicht mehr 5-eindeutig. Dennoch ist die Modellierung solcher Komponenten erlaubt. In allen vier Fällen ist relevant, dass das reverse Komplement von TTTTT nicht vorkommt. Dieses könnte den Assemblierungsprozess erheblich beeinträchtigen, da es in den ersten drei Fällen mindestens eine nicht erwünschte Hybridisierung eingehen könnte.

Auch bei dieser Vorgehensweise wird nicht vermieden, dass bei der Modellierung der NW-, SW-, NE- und SE-Komponenten schon vorhandene q-gramme oder Komplemente von schon vorhandenen Subsequenzen entstehen.

Kapitel 6

ONlibrary - Oligonukleotid Bibliothek für die DNA-Nanotechnologie

Das im Zusammenhang mit der vorliegenden Arbeit entstandene Tool **"ONlibrary - Oligonukleotid Bibliothek für die DNA-Nanotechnologie"**, kurz ONlibrary genannt, ist die Umsetzung der vorgestellten Lösungsansätze aus Kapitel 4 und 5. Dieses Kapitel beschreibt die Funktionalitäten von ONlibrary. Das beschriebene Tool befindet sich auf dem beiliegenden Datenträger.

Jedes der fünf Teilprobleme des DSD-Problems wird durch einen geeigneten Lösungsansatz bearbeitet. Zunächst wird in Abschnitt 6.1 das Hauptfenster des Tools vorgestellt. Mit diesem ist es möglich die ersten vier Teilprobleme zu bearbeiten. Die längste q-eindeutige Sequenz kann ohne oder auch mit vorgegebenem Kontext berechnet werden. Für die sticky ends eines Oligonukleotids aus einer Struktur können alle möglichen Alternativen berechnet werden. Weiter können alle möglichen Internals an einer beliebigen Stelle eines Oligonukleotids unter Berücksichtigung der Strukturvorgabe erstellt werden.

Dann wird in Abschnitt 6.2 die benutzerdefinierte Erstellung von Strukturen, unter anderem auch kompletter Kacheln, erläutert. Hier ist es möglich eine komplette Kachel aus neun Oligonukleotiden zu erstellen, die die Form der Kacheln A^2 und B^3 hat. Dabei wird zufällig eine q-eindeutige Sequenzen erzeugt, die dann in die Oligonukleotide der einzelnen Komponenten aufgeteilt wird.

Abschließend werden in Abschnitt 6.3 zwei Analysemöglichkeiten näher beschrieben. Diese sollen der Unterstützung beim Erstellen von benutzerdefinierten Strukturen, aber auch zur Untersuchung bisher schon bekannter Strukturen, dienen.

6.1 Hauptfenster

In Abbildung 6.1 ist das Hauptfenster von ONlibrary zu sehen. Dieses besteht aus zwei Hauptkomponenten. Auf der rechten Seite befindet sich die Strukturvorgabe und auf der Linken alle Optionen zur Berechnung der alternativen Oligonukleotide. Die Liste auf der rechten Seite beinhaltet also alle Oligonukleotide der Struktur, also den Kontext. In der Liste auf der linken Seite des Hauptfensters werden nach den Berechnungen alle gefunden Alternativen zu einem vorgegebenen Oligonukleotid aus dem Kontext aufgelistet. Der Benutzer kann über den Slider „Länge der Subsequenzen“ die Länge der q-gramme einstellen. Wird dann die Checkbox „Struktur auffüllen“ aktiviert und der „berechnen“-Button

betätigt, so wird eine längste q-eindeutige Sequenz ausgegeben. Diese wird dann mit ihren biochemischen Eigenschaften in der linken Liste angezeigt. Die Sequenz wird nach der in Abschnitt 4.3 beschriebenen Methode erzeugt und ist die Lösung zum ersten Teilproblem des DSD-Problems.

Es besteht die Möglichkeit die Kacheln A^2, B^3 oder eine im „fasta“-Format abgespeicherte Struktur als Vorgabe zu wählen. Vor dieser Wahl kann der Benutzer die Molarität einstellen. Die rechte Liste füllt sich mit den einzelnen Oligonukleotiden der gewählten Strukturvorgabe. Zu jedem Oligonukleotid werden der prozentuale GC-Gehalt, die Gibbs-Energie und die Schmelztemperatur in Abhängigkeit zur eingegebenen Molarität errechnet und angezeigt. Der Kontext kann anschließend um beliebig viele Strukturen erweitert werden. Die Liste der Oligonukleotide wird mit denen der hinzugefügten Struktur aufgefüllt. Die so erzeugten Strukturen können jederzeit über „Struktur speichern“ im „fasta“-Format gespeichert werden. Der Button „Struktur leeren“ erlaubt es dem Benutzer den Kontext zu leeren. Wird nun die Checkbox „Struktur auffüllen“ aktiviert und der „berechnen“-Button betätigt, so wird die Lösung des zweiten Teilproblems ausgegeben. Unter Berücksichtigung des gewählten Kontext wird nach der Vorgehensweise aus Abschnitt 4.3 die längste q-eindeutige Sequenz erzeugt und wie schon zuvor in der linken Liste angezeigt.

Weiter bietet sich die Möglichkeit ein Oligonukleotid aus dem Kontext zu wählen. Auf dem ausgewählten Oligonukleotid können zwei Aktionen ausgeführt werden. Ist das gewählte Oligonukleotid verantwortlich für sticky ends, so können diese nach der Vorgehensweise aus Abschnitt 4.3 modifiziert werden. Dazu kann der Benutzer über die Checkboxen „$5'$“ und „$3'$“ wählen, ob nur ein Ende oder beide Enden des Oligonukleotids verändert werden sollen. Über den Slider wird wieder die Länge der q-gramme und gleichzeitig der sticky ends bestimmt. Weiter hat der Benutzer mit der Checkbox „korrektes Design“ die Möglichkeit einzustellen, ob das Anfügen der noch möglichen sticky ends den Sequenzdesign-Regeln entsprechen muss. Ist diese Checkbox aktiviert, werden die alternativen Oligonukleotide wie in Abschnitt 5.3 beschrieben berechnet. Sollte diese Option eine zu geringe Anzahl an alternativen Oligonukleotiden erzeugen, so kann sie deaktiviert werden. Dann werden alle noch möglichen q-gramme jeweils an die gewünschte Stelle gesetzt, ohne zu prüfen, ob dabei die Regeln verletzt werden. Nach Betätigen des „berechnen“-Buttons füllt sich die linke Liste mit alternativen Oligonukleotiden. Bei den entstandenen Oligonukleotiden sind die modifizierten Stellen schwarz eingefärbt, während der bleibende Teil grün eingefärbt ist. Im Fall der modifizierten sticky ends sind diese in jedem der alternativen Oligonukleotide schwarz, während der Mittlere Teil grün dargestellt ist. Die Spalte „Name“ bleibt leer. Es ist dem Benutzer überlassen, wie die berechneten Oligonukleotide benannt werden sollen. Ist ein alternatives Oligonukleotid interessant, kann im Kontext das alte Oligonukleotid durch die Alternative ausgetauscht werden. Dazu muss dieses zunächst benannt und dann markiert werden. Mit dem Button „ersetzen“ wird das markierte Oligonukleotid aus dem Kontext durch das markierte alternative Oligonukleotid ersetzt. Der so neu entstandene Kontext kann anschließend gespeichert werden.

Eine weitere Möglichkeit das gewählte Oligonukleotid zu modifizieren, ist das Einfügen eines Internals. Die Methode zur Erzeugung eines Internals wurde bereits in Abschnitt 4.3 vorgestellt. Soll aus dem ausgewählten Oligonukleotid eine Internal erzeugt werden, so muss zunächst die Checkbox „Internal“ aktiviert werden. Der Benutzer entscheidet anschließend, an welcher Position das Internal senkrecht zur Kachel entstehen soll. Die Position muss in das Textfenster eingetragen werden. In der Liste werden dann alle möglichen Internals aufgelistet. Internals bestehen wie in Abschnitt 4.3 beschrieben aus zwei Oligonukleotiden. Jedes in der Liste auftauchende Oligonukleotid ist mit einer Zahl benannt. Für jedes Paar von Oligonukleotiden, die ein Internal bilden, sind die Zahlen gleich. Im ersten

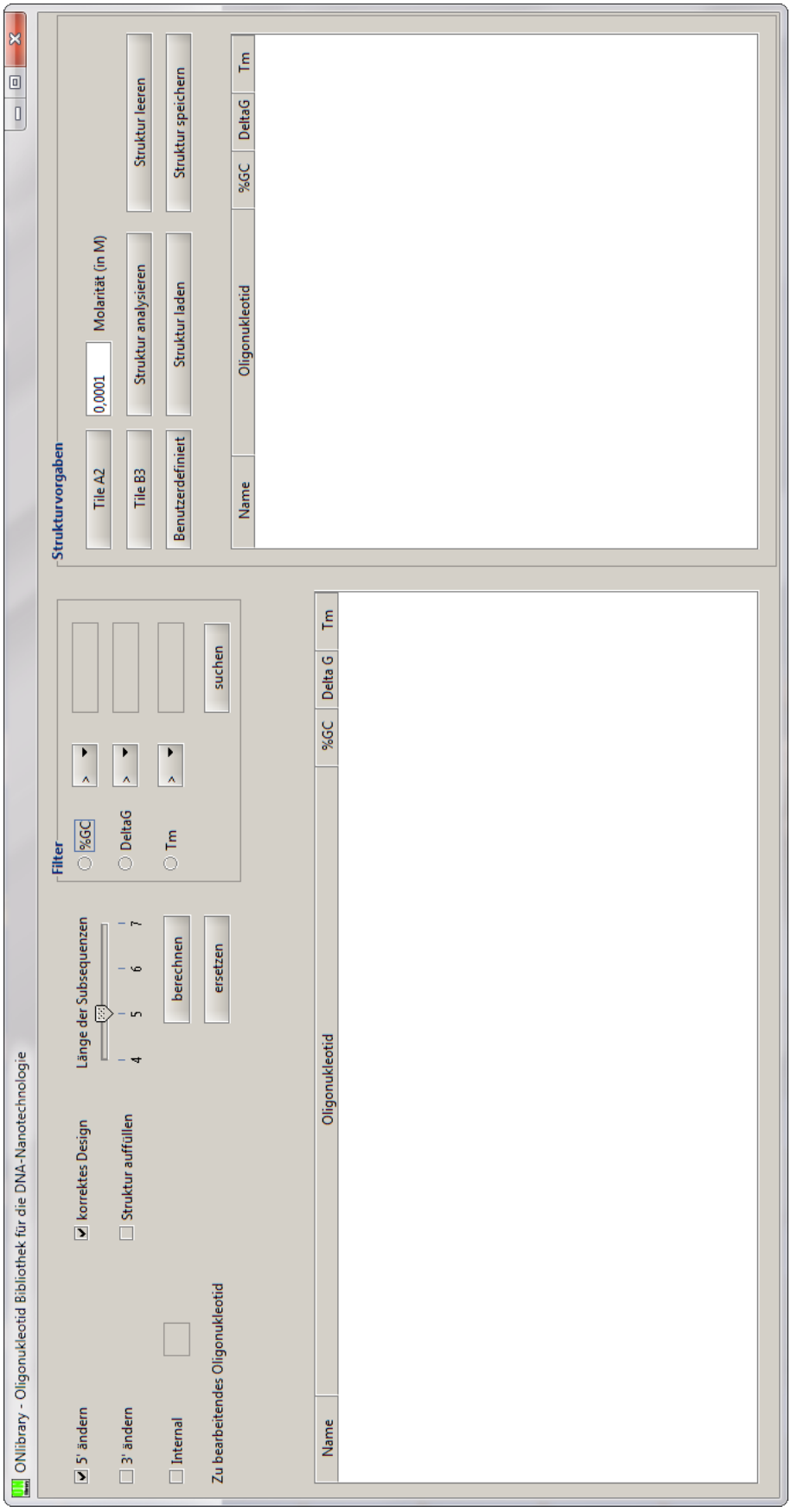


Abbildung 6.1: Hauptfenster des ONlibrary

Oligonukleotid eines Paares sind die Zeichen bis zur gewählten Position grün gefärbt, da diese unverändert aus dem gewählten Oligonukleotid übernommen wurden. Dann folgen 28 Zeichen, die schwarz gefärbt sind und den Arm repräsentieren. Beim zweiten Oligonukleotid des Paares sind die ersten sechs Zeichen schwarz gefärbt. Diese Zeichen sind das reverse Komplement zu den ersten sechs Zeichen des erzeugten Arms. Anschließend sind zwei Tyhminbasen rot eingefärbt. Diese sind notwendig, um den Arm senkrecht zur Kachel aufzustellen. Die restlichen Zeichen des zweiten Oligonukleotids sind wieder grün eingefärbt, da es sich um die Zeichen aus dem ursprünglichen Oligonukleotid handelt. Auch bei der Erzeugung von Internals gibt es die Möglichkeit, wie bei der Modifikation der sticky ends, die Checkbox „korrektes Design“ zu aktivieren. Ist diese deaktiviert, wird der Arm unter Berücksichtigung der Struktur erstellt und an die gewünschte Position gestellt ohne zu prüfen, ob die dabei entstandenen q-gramme den Regeln entsprechen. Ist die Checkbox hingegen aktiviert, wird wie in Abschnitt 5.3 verfahren. Auch die erzeugten Internals können der ausgewählten Struktur durch den Button „ersetzen“ hinzugefügt werden. Dazu wird zunächst das zu ersetzende Oligonukleotid im Kontext markiert. Anschließend genügt es eines der Beiden Oligonukleotide zu markieren, um beide dem Kontext hinzuzufügen.

In einigen Fällen kann die erzeugte Anzahl an alternativen Oligonukleotid sehr hoch sein. Um einen besseren Überblick zu bekommen, ist dem Benutzer die Möglichkeit gegeben, anhand der biochemischen Eigenschaften zu filtern. Mit den Radiobuttons im Filterbereich kann nach einer bestimmten Eigenschaft gefiltert werden. Nach Auswahl einer Eigenschaft kann der Benutzer den gewünschten Wert eingeben und sich alle Oligonukleotide anzeigen lassen, die diesem entsprechen. In Abbildung 6.2 ist das Ergebnis einer Suche zu sehen. Im erscheinenden Fenster kann der Benutzer zunächst die gefilterten Oligonukleotide näher betrachten. Anschließend kann er entscheiden, ob die gefilterten Oligonukleotide seinen Wünschen entsprechen. Ist das der Fall, so kann die Auswahl übernommen werden. Anderenfalls kann durch das Schließen des Fensters die Auswahl verworfen werden. Nach einer übernommenen Auswahl kann wiederum gefiltert werden.

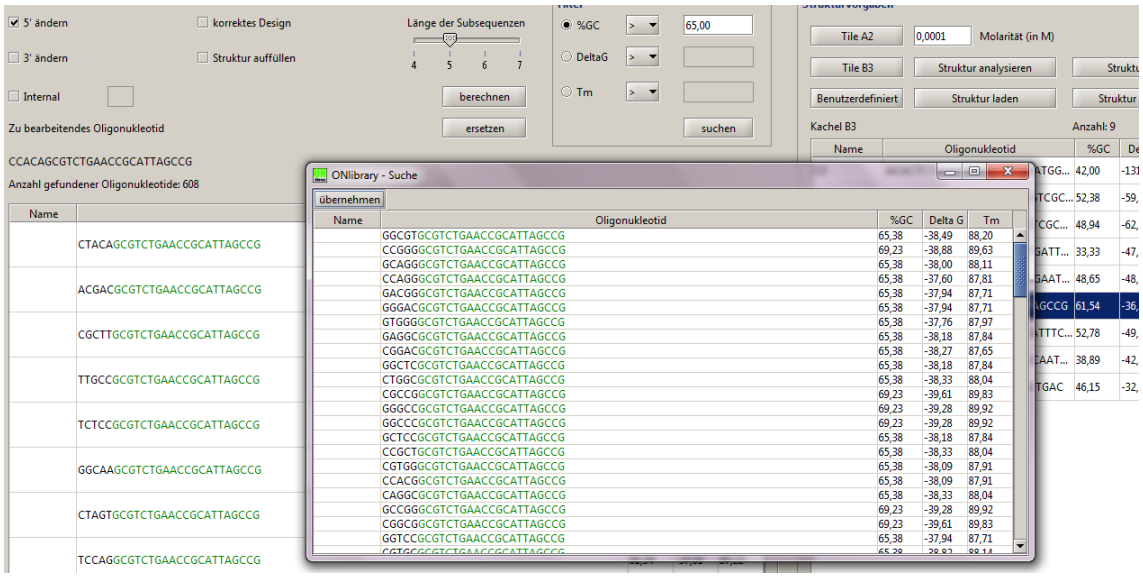


Abbildung 6.2: Suche in den erzeugten Oligonukleotiden

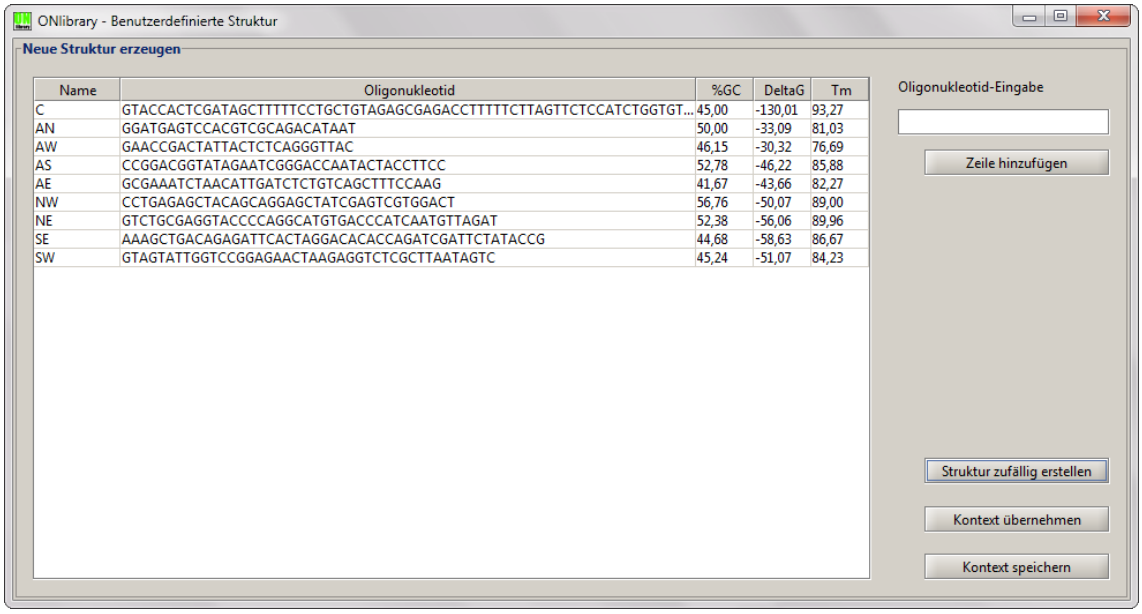


Abbildung 6.3: Erzeugung benutzerdefinierter Strukturen

6.2 Benutzerdefinierte Strukturvorgaben

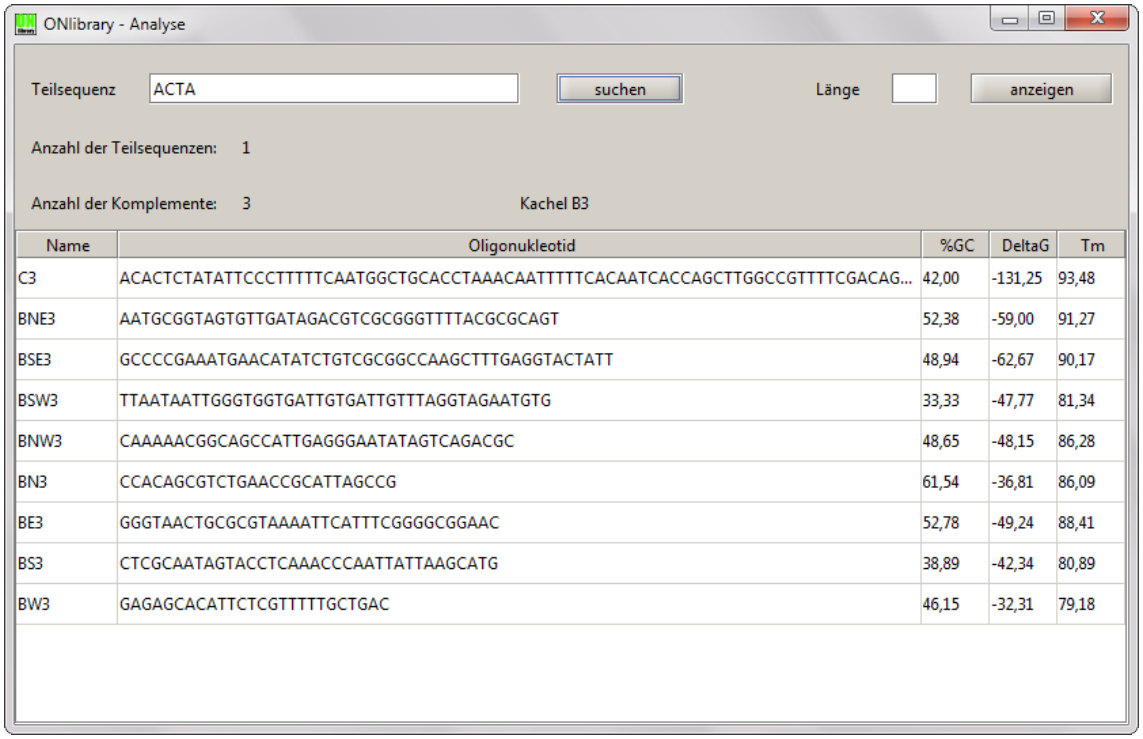
Sind die Kacheln A^2 oder B^3 nicht die gewünschten Kontexte, so kann der Benutzer einen eigenen Kontext definieren. Dazu ruft er aus dem Hauptfenster mit dem Button „Benutzerdefiniert“ das Fenster aus Abbildung 6.3 auf. Hier existieren zwei Möglichkeiten für den Benutzer einen Kontext zu definieren.

Zunächst können in das Eingabefenster auf der rechten Seite einzelne Oligonukleotide eingetragen werden und mit dem „Zeile hinzufügen“-Button der Liste auf der linken Seite hinzugefügt werden. So entsteht ein benutzerdefinierter Kontext, der nicht auf die Einhaltung der Sequenzdesign-Regeln geprüft wird. Dieser Kontext kann dann gespeichert werden und in das Hauptfenster übernommen werden. Wenn im Hauptfenster schon ein Kontext vorgegeben ist, dann wird der benutzerdefinierte Kontext hinzugefügt.

Die andere Möglichkeit einen Kontext zu definieren ist den Button „Struktur zufällig erstellen“ zu benutzen. Dieser füllt die Liste mit neun Oligonukleotiden, die nach der Methode aus Abschnitt 5.3 erstellt werden. Dabei entsteht eine Kachel, die aus 5-eindeutigen Sequenzen besteht. Diese Kachel hat dieselbe Form wie die Kacheln A^2 und B^3. Der Benutzer kann hier nicht eine neue Struktur erstellen, sondern nur eine Kachel der bekannten Form mit zufällig erstellen Oligonukleotiden. Diese kann gespeichert und dann in das Hauptfenster übernommen werden, um weiter damit zu arbeiten.

6.3 Analyse der Strukturen

Das Tool bietet noch zwei zusätzliche Analysefunktionen für Kontexte die im „fasta“-Format vorliegen. Beliebige Kontexte können unter diesem Format in ONlibrary geladen und analysiert werden. Zunächst muss der gewünschte Kontext in das Tool geladen werden. Dies geschieht über den „lade“-Button im Hauptfenster aus Abbildung 6.1. Anschließend gelangt der Benutzer über den Button „Struktur analysieren“ in das Fenster aus Abbildung 6.4.



Name	Oligonukleotid	%GC	DeltaG	Tm
C3	ACACTCTATATTCCCTTTTTCAATGGCTGCACCTAAACAATTTTTCACAATCACCAGCTTGGCCGTTTTCGACAG...	42,00	-131,25	93,48
BNE3	AATGCGGTAGTGTTGATAGACGTCGCGGGTTTTACGCGCAGT	52,38	-59,00	91,27
BSE3	GCCCCGAAATGAACATATCTGTCGCGGCCAAGCTTTGAGGTACTATT	48,94	-62,67	90,17
BSW3	TTAATAATTGGGTGGTGATTGTGATTGTTTAGGTAGAATGTG	33,33	-47,77	81,34
BNW3	CAAAAACGGCAGCCATTGAGGGAATATAGTCAGACGC	48,65	-48,15	86,28
BN3	CCACAGCGTCTGAACCGCATTAGCCG	61,54	-36,81	86,09
BE3	GGGTAACTGCGCGTAAAATTCATTTCGGGGCGGAAC	52,78	-49,24	88,41
BS3	CTCGCAATAGTACCTCAAACCCAATTATTAAGCATG	38,89	-42,34	80,89
BW3	GAGAGCACATTCTCGTTTTTGCTGAC	46,15	-32,31	79,18

Abbildung 6.4: Analyse hinsichtlich bestimmeter Subsequenzen

In diesem Fenster ist der Kontext wieder aufgelistet, um die Übersicht zu behalten. Die erste Analysefunktion beschränkt sich auf eine bestimmte Teilsequenz beliebiger Länge. Der Benutzer kann eine Sequenz in das Textfenster „Teilsequenz“ eintragen und nach dieser suchen. Die Analysefunktion gibt aus, wie oft die angegebene Sequenz im gesamten Kontext vorkommt. Außerdem wird auch die Anzahl der Komplemente der angegebenen Sequenz angezeigt. Beide Zahlen werden in zwei Zeilen unter dem Textfeld der Teilsequenz ausgegeben. In Abbildung 6.4 wurde nach der Teilsequenz ACTA in der Kachel B^3 gesucht, die genau einmal existiert. Das Komplement hingegen ist dreimal in der Kachel zu finden.

Die zweite Analysefunktion befasst sich mit der Suche aller Teilsequenzen einer bestimmten Länge. In Abbildung 6.5 wurde die Länge 4 eingegeben. Der Button „anzeigen“ öffnet ein weiteres Fenster, das dem Benutzer eine Liste bestehend aus vier Spalten anzeigt. In dieser Liste sieht der Benutzer alle Teilsequenzen der Länge vier. In der ersten Spalte steht die Teilsequenz selbst. In dieser Spalte ist es möglich die Teilsequenzen alphabetisch zu sortieren. In der zweiten Spalte ist die Häufigkeit der Subsequenz in der gesamten Struktur angegeben. Die dritte Spalte zeigt an, wie häufig das Komplement der Subsequenz in der Struktur vorkommt. Zur besseren Übersicht werden aber nicht alle Subsequenzen der gewählten Länge angezeigt. Es werden nur Subsequenzen aufgelistet, die mindestens zweimal vorkommen und deren Komplement mindestens einmal vorkommt. Nur diese Sequenzen verletzten zwei der in Abschnitt 3.1 angegebenen Sequenzdesign-Regeln. Eine Subsequenz, die öfter vorkommt, aber dessen Komplement gar nicht vorkommt, kann im Assemblierungsprozess nicht unkontrolliert hybridisieren. Eine Teilsequenz, die gemeinsam mit ihrem Komplement genau einmal auftauchen, ist erwünscht, obwohl dies die Sequenzdesign-Regeln verletzt. Die Selbstorganisation einer Struktur geschieht über eben diese gezielten Hybridisierungen. Die vierte Spalte besteht aus zwei Typen von Einträgen.

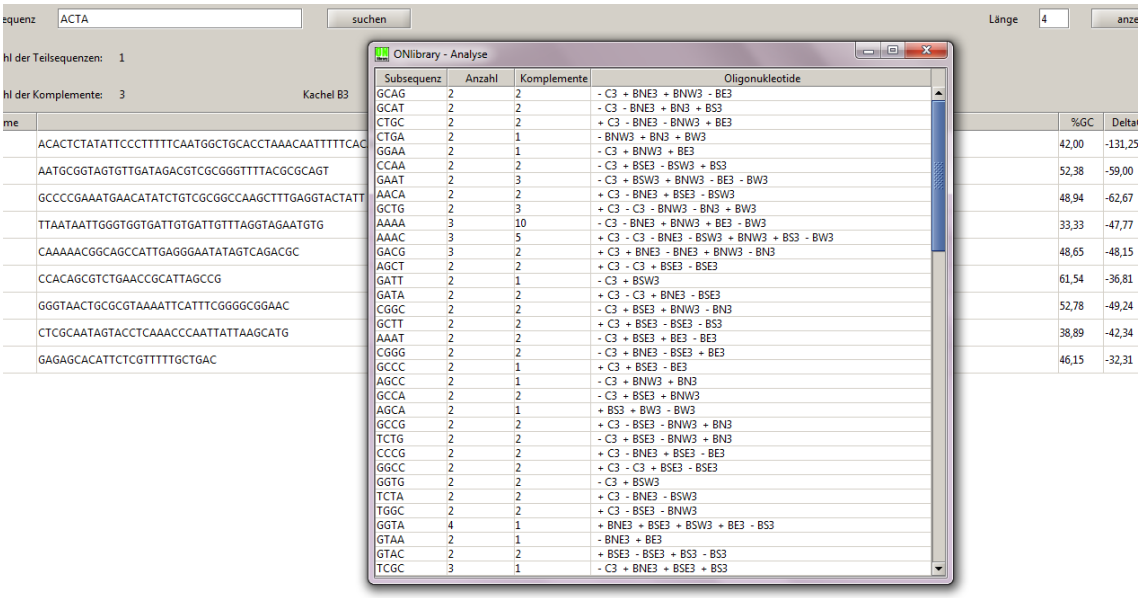


Abbildung 6.5: Analyse hinsichtlich aller Subsequenzen einer vorgegebenen Länge

Hier werden die Namen der Oligonukleotide aus dem Kontext aufgelistet, in denen entweder die Subsequenz oder ihr Komplement enthalten ist. Wenn die Subsequenz selbst in dem Oligonukleotid enthalten ist, dann steht vor dem Namen das Zeichen „+“. Ist hingegen das Komplement enthalten, so ist das Zeichen „-“ vor dem Namen des Oligonukleotids zu sehen. Existiert eine Teilsequenz zum Beispiel vier Mal in der Struktur und in der letzten Spalte sind nur drei Oligonukleotide mit einem „+“ gekennzeichnet, so muss in einem dieser Oligonukleotide die Teilsequenz doppelt vorkommen.

Kapitel 7

Zusammenfassung und Ausblick

In der Zusammenfassung wird ein Überblick der erreichten Ziele dieser Arbeit geboten. Der Ausblick hingegen weist auf weitere Problemstellungen und mögliche Erweiterungen des Tools "ONlibrary - Oligonukleotid Bibliothek für die DNA-Nanotechnologie" hin.

7.1 Zusammenfassung

Ziel der vorliegenden Arbeit war es, das in Kapitel 3 vorgestellte DSD-Problem zu lösen. Zusätzlich sollte ein Tool zum Entwurf von Oligonukleotid-Bibliotheken für die DNA-Nanotechnologie implementiert werden. Dazu wurden in Kapitel 2 zunächst biochemische und graphentheoretische Grundlagen vermittelt. In Kapitel 3 wurde das DSD-Problem definiert und in fünf Teilprobleme unterteilt. Auf Basis von zwei Graphenstrukturen wurden in Kapitel 4 und 5 zwei Ansätze vorgestellt, die zur Lösung des DNA-Sequenzdesign-Problems genutzt wurden. In Kapitel 6 wurden abschließend die einzelnen Funktionen des entstandenen Tools ONlibrary beschrieben.

Der erste Ansatz auf Basis des De Bruijn Graphen liefert, durch das in Kapitel 4 vorgestellte ILP, Ergebnisse, mit denen drei der fünf Teilprobleme gelöst werden konnten. Für ungerade q-gramme wird die beschriebene obere Schranke immer erreicht. Bei geradem q wurden Unterschiede zwischen der gewählten Anzahl der Kanten und der oberen Schranke für die Anzahl maximal wählbarer Kanten gefunden. Das Erzeugen der längsten q-eindeutigen Sequenz, das Erzeugen der längsten q-eindeutigen Sequenz unter Berücksichtigung eines Kontextes und das Erstellen einer DNA-Kachel sind mittels der Ergebnisse des ILP möglich. Die Modifikation der sticky ends und das Generieren eines Internals an einer vorgegebenen Position hingegen sind schon alleine durch die Graphenstruktur selbst möglich. Dazu wird zunächst der Startknoten festgelegt. Dieser ist mit den letzten $q-1$ Zeichen der bleibenden Sequenz beschriftet. Für die sticky ends werden von diesem Startknoten aus alle noch möglichen Pfade der Länge fünf gesucht. Dabei muss in einem De Bruijn Graphen immer berücksichtigt werden, dass das Wählen einer Subsequenz den Ausschluß der komplementären Subsequenz impliziert. Die Berechnung der möglichen Internals an einer vorgegebenen Position erfolgt mit demselben Vorgehen und wählt alle möglichen Pfade der Länge 28.

Die Graphenstruktur des ungerichteten De Bruijn Graphen birgt den Vorteil die Subsequenzen mit ihren reversen Komplementen kombinieren zu können. Dieses Tupel von Subsequenzen bildet die Beschriftung einer Kante. Wird also eine Kante im Graphen gewählt, so sind beide Sequenzen des Tupels nicht mehr wählbar. Zunächst konnte gezeigt werden, dass es für ungerade q möglich ist einen Eulerkreis in einem ungerichteten De Bruijn Graphen zu finden. Da ein Eulerkreis alle Kanten eines Graphen beinhaltet, ist

dieser auch der längste Pfad in einem Graphen. Der längste Pfad erzeugt auch die längste Sequenz und so konnte das DSD-Problem für ungerade q direkt auf das Eulerkreisproblem reduziert werden. Auch für diesen Ansatz wurde für ungerade q-gramme immer die obere Schranke erreicht. Für gerade q mussten Vorverarbeitungsschritte durchgeführt werden, um einen Eulerpfad zu ermöglichen.

Weiterhin konnten durch die Kombination der Subsequenzen mit ihren reversen Komplementen Erkenntnisse über die Lösungen des ILP gewonnen werden. Dazu wurden in Abschnitt 5.2 zwei Fälle betrachtet. Sind selbstkomplementäre q-gramme erlaubt, konnte die dafür errechnete obere Schranke immer erreicht werden. Die Anzahl der gewählten Kanten in diesem Fall entsprach immer dem vom ILP berechneten Optimum. Wurden selbstkomplementäre q-gramme nicht zugelassen, konnte für die Anzahl maximal wählbarer Kanten eine obere Schranke gefunden werden. Diese konnte mit einem Verfahren aus dem Artikel [AFN06] gefunden werden. Für den Fall $q = 6$ konnte außerdem gezeigt werden, dass durch geschickte Vorverarbeitung das Optimum doch erreicht werden konnte. Für die Anwendung sind die Fälle der q-eindeutigen Strukturen mit $5 \leq q \leq 7$ besonders interessant. Die Lösungen für diese Fälle sind auch mit dem Ansatz auf Basis des ungerichteten De Bruijn Graphen optimal.

Die Implementierung der einzelnen Lösungen im Tool ONlibrary sind eine Kombination beider Ansätze. Für die ersten beiden Teilprobleme des DSD-Problems ist der Ansatz basierend auf dem De Bruijn Graphen verwendet worden. Die restlichen drei Teilprobleme wurden unter Verwendung des ungerichteten De Bruijn Graphen gelöst.

7.2 Ausblick

Für das erste Teilproblem wird die längste q-eindeutige Sequenz erzeugt. Beide Ansätze finden genau eine Sequenz. Eine weitere Aufgabenstellung ist das Finden aller möglichen q-eindeutigen Sequenzen der errechneten Länge. Für den ersten Ansatz werden dann alle möglichen Traversierungen des Graphen aus der ILP-Lösung und für den zweiten Ansatz alle möglichen Eulerkreise oder -pfade in einem G_q-Graphen gesucht.

Das in Abschnitt 4.2 vorgestellte ILP kann bei gegebenem Kontext durch das Hinzufügen weiterer Nebenbedingungen verbessert werden. Die Nebenbedingungen sollen vermeiden, dass ein Pfad und mehrere disjunkte Kreise entstehen. Dazu würden sich Subtour-Ungleichungen anbieten, die mindestens zwei Kanten zwischen jeweils zwei disjunkten Knotenmengen fordern. Bei dem De Bruijn Graphen handelt es sich um einen gerichteten Graphen. Deshalb muss eine Kanten von der ersten Knotenmenge in die Zweite führen und die zweite Kante entgegengesetzt gerichtet sein.

Die vorliegenden Arbeit stellt ein Verfahren vor, das sticky ends einer einzelnen DNA-Kachel modifiziert. In der DNA-Nanotechnologie werden aber DNA-Nanoarrays erstellt, die aus mehreren DNA-Kacheln bestehen. Die sticky ends sind die Bindestellen zwischen den einzelnen DNA-Kacheln der Nanoarrays. Aus diesem Grund müssen die sticky ends der Kacheln aufeinander abgestimmt sein, damit diese auch gezielt hybridisieren. Das Verändern eines sticky ends führt deshalb zur Modifikation des sticky ends der zweiten Kachel. Die vorgestellten Verfahren sind der erste Schritt zur Modifikation der sticky ends in den Kacheln eines DNA-Nanoarrays. Des Weiteren könnten in zukünftigen Anwendungen die Längen der sticky ends variieren. Dazu müsste das vorgestellt Verfahren angepasst werden und für jedes sticky end die gewünschte Länge angegeben werden.

Das Erstellen stabiler DNA-Kacheln ist eine komplexe Aufgabe. Das Lösen des DSD-Problems ist ein erster Schritt. Die erzeugte q-eindeutige Sequenz kann nach der beschriebenen Methode in die verschiedenen Oligonukleotide einer DNA-Kachel aufgeteilt werden.

Weitere Aspekte, die die Selbstorganisation der verschiedenen Oligonukleotide fördern, sind die thermodynamischen Eigenschaften. Diese Eigenschaften sollten schon bei der Erzeugung der q-eindeutigen Sequenz berücksichtigt werden. Die Schmelztemperatur ist beispielsweise ein wichtiger Parameter für den Assemblierungsprozess. Eine weiteres Vorgehen zur Steigerung der Stabilität von DNA-Kacheln könnte das Minimieren von Subsequenzen der Länge $q - 1$ bei der Modellierung mit q-grammen sein.

Ein weiteres Feature des Tools ONlibrary könnte eine graphische Darstellung der Analysefunktionen sein. Diese könnten weitere Hinweise auf die Stabilität der DNA-Kacheln geben. Position und Häufigkeit bestimmter Subsequenzen und deren Komplemente könnten weitere Indikatoren für stabile DNA-Strukturen sein. Ein weiterer und noch viel aufwendigerer Schritt ist das Einrichten eines graphischen Editors zur Erstellung verschiedener DNA-Strukturen. Dabei soll dem Benutzer die Möglichkeit gegeben werden das Grundgerüst der Nanostruktur über ein Zeichentool zu erstellen. Anschließend sollen die Sequenzen der einzelnen Oligonukleotide nach den Sequenzdesign-Regeln erzeugt werden.

Abbildungsverzeichnis

Literaturverzeichnis

[AFN06] Anderson, J.W., K.R. Fox und G.A. Niblo: *A fast algorithm for the construction of universal footprinting templates in DNA*. Journal of mathematical biology, 52(3):307–342, 2006.

[Ami08] Amiji, M.M.: *The Handbook of Nanomedicine. By Kewal K. Jain.* ChemMedChem, 3(12):1977, 2008.

[Bar10] Barak Naveh and Contributors: *JGraphT*. http://www.jgrapht.org/, August 2010.

[BJ07] Baumgartner, W. und B. Jäckli: *Nanotechnologie in der Medizin*. Nano, Seiten 149–164, 2007.

[Bra94] Brandstädt, A.: *Graphen und Algorithmen*, Band 89. Teubner, 1994.

[FBR+00] Feldkamp, U., W. Banzhaf, H. Rauhe et al.: *A DNA sequence compiler*. In: *Poster presented at the 6th International Meeting on DNA Based Computers, Leiden, June*, 2000.

[Fel09] Feldkamp, U.: *Search Strategies for DNA Sequence Design Software*. 2009.

[FN06] Feldkamp, U. und C.M. Niemeyer: *Rationaler Entwurf von DNA-Nanoarchitekturen*. Angewandte Chemie, 118(12):1888–1910, 2006.

[FRB03] Feldkamp, U., H. Rauhe und W. Banzhaf: *Software tools for DNA sequence design*. Genetic Programming and Evolvable Machines, 4(2):153–171, 2003.

[FS06] Fuchs, G. und H.G. Schlegel: *Allgemeine Mikrobiologie*. Georg Thieme Verlag, 2006.

[IBM10] IBM: *IBM ILOG CPLEX Optimizer*. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/, July 2010.

[JGr10] JGraph Ltd: *JGraph*. http://www.jgraph.com/, August 2010.

[Kni06] Knippers, R.: *Molekulare Genetik*. Georg Thieme Verlag, 2006.

[LK08] Latscha, H.P. und U. Kazmaier: *Chemie für Biologen*. Springer, 2008.

[PCF+04] Paschen, H., C. Coenen, T. Fleischer, D. Oertel und C. Revermann: *Nanotechnologie: Forschung, Entwicklung, Anwendung*. Springer, 2004.

[PRS98] Păun, G., G. Rozenberg und A. Salomaa: *DNA computing: new computing paradigms*. Springer Verlag, 1998.

[Sch00] Schrijver, A.: *Theory of linear and integer programming.* John Wiley & Sons, 2000.

[Sei08] Seiffert, D.I.J.: *Ein Sequenzdesign-Algorithmus für verzweigte DNA-Strukturen.* Doktorarbeit, 2008.

[Shi09] Shi, D.: *Nanoscience in biomedicine.* Springer Verlag, 2009.

[SJAS96] SantaLucia Jr, J., H.T. Allawi und P.A. Seneviratne: *Improved Nearest-Neighbor Parameters for Predicting DNA Duplex Stability.* Biochemistry, 35(11):3555–3562, 1996.

[SJH04] SantaLucia Jr, J. und D. Hicks: *The thermodynamics of DNA structural motifs.* Annual review of biophysics and biomolecular structure, 33:415, 2004.

[SK83] Seeman, N.C. und N.R. Kallenbach: *Design of immobile nucleic acid junctions.* Biophysical journal, 44(2):201–209, 1983.

[SMF+08] Saccà, B., R. Meyer, U. Feldkamp, H. Schroeder und C.M. Niemeyer: *Hochdurchsatz-Analyse der Selbstorganisation von DNA-Nanostrukturen in Echtzeit mittels FRET-Spektroskopie.* Angewandte Chemie, 120(11):2165–2168, 2008.

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 10. März 2011

Marianna D'Addario

Einverständniserklärung des Urhebers

Hiermit erkläre ich mich einverstanden, dass meine Diplomarbeit nach §6(1) des URG der Öffentlichkeit durch die Übernahme in die Bereichsbibliothek zugänglich gemacht wird.
Damit können Leser der Bibliothek die Arbeit einsehen und zu persönlichen wissenschaftlichen Zwecken Kopien aus dieser Arbeit anfertigen. Weitere Urheberrechte werden nicht berührt.

Dortmund, den 10. März 2011

Marianna D'Addario