Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

FAKULTÄT
FÜR MATHEMATIK, INFORMATIK
UND NATURWISSENSCHAFTEN

Masterthesis

HELMSimilarity — Substrukturfilter und Algorithmen zur Ähnlichkeitssuche in HELM-Notationen

Eva Bültel

Studiengang: Bioinformatik
Matrikelnummer: 6831811

Erstgutachter: Professor Dr. Matthias Rarey
Zweitgutachter: Professor Dr. Andrew Torda

Hamburg, den 11. April 2018

Zusammenfassung

Die computergestützte Verwaltung von molekularen Therapeutika hat sich traditionell auf kleine Moleküle und deren Repräsentation auf Atomebene beschränkt. Mittlerweile hat sich dies geändert. Es gelangen immer mehr Substanzen mit komplexem und oft undefiniertem chemischen Aufbau in die klinische Entwicklung. Diese steigende Tendenz hat die Verwaltung von chemischer Information stark erschwert. Die Hierarchical Editing Language for Macromolecules wurde entwickelt, um solche komplexen Moleküle kompakt und von Menschen lesbar zu repräsentieren. Molekülrepräsentationen ermöglichen das Arbeiten mit großen Mengen von Molekülen. Zudem werden sie verwendet, um molekulare Ähnlichkeit zu berechnen. Viele Versuche des Wirkstoffdesigns basieren auf dem Prinzip der molekularen Ähnlichkeit, welches besagt, dass strukturell ähnliche Verbindungen dazu tendieren, ähnliche Eigenschaften aufzuweisen.
In dieser Arbeit wurde ein Verfahren entwickelt, um in HELM-Notationen nach Ähnlichkeiten zu suchen. Außerdem können Notationen, die eine definierte Substruktur enthalten, herausgefiltert werden. Die Realisierung des Verfahrens fokussiert sich auf die Verarbeitung von HELM-Notationen zu molekularen Fingerprints. Verwendet wird ein pfadbasierter Hashed Fingerprint. Es wurden je zwei verschiedene Ansätze der Generierung von Pfaden und der Fingerprints realisiert und verglichen. Die Berechnung der Ähnlichkeit erfolgt über den Tanimoto-Koeffizienten, der von den Fingerprints zweier Notationen kalkuliert wird. Der Substrukturfilter überprüft, ob der Fingerprint einer Notation im Fingerprint einer anderen enthalten ist. Für eine einfache Bedienung wurde eine graphische Benutzeroberfläche implementiert, welche die Anwendung der Ähnlichkeitssuche und des Substrukturfilters in HELM-Notationen ermöglicht. Das Verfahren wurde mit Hilfe von verschiedenen Tests validiert und die Laufzeit analysiert.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ASCII American Standard Code for Information Interchange

CID PubChem Compound Identification

DNS Desoxyribonukleinsäure

HELM Hierarchical Editing Language for Macromolecules

ID Identifikation

RNS Ribonukleinsäure

SHA-2 Secure Hash Algorithm 2

SMILES Simplified Molecular Input Line Entry Specification

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

In der Chemieinformatik ist die Berechnung von Ähnlichkeit zwischen zwei Molekülen ein zentrales Anliegen. Die Anwendung umfasst verschiedene, meist medizinalchemische Felder, wie beispielsweise das virtuelle Screening, das auf chemischer Ähnlichkeit basiert [3]. Als virtuelles Screening wird die Suche nach neuen Wirkstoffen in großen Moleküldatenbanken mit computergestützten Methoden bezeichnet [16]. Es gibt unterschiedliche Möglichkeiten zur Molekülrepräsentation für die Datenbanken. Dazu gehört die Hierarchical Editing Language for Macromolecules (HELM), die sich bewährt hat, komplexe Moleküle zu beschreiben. Für die Darstellung, Umwandlung und Speicherung von HELM-Notationen existieren Toolkits, jedoch fehlt die Möglichkeit in den Notationen zu suchen. Das Suchen in HELM wird immer wichtiger, da die Moleküldatenbanken stetig wachsen [46]. In dieser Arbeit werden Algorithmen vorgestellt, die es erlauben, in HELM auf verschiedene Weisen zu suchen.

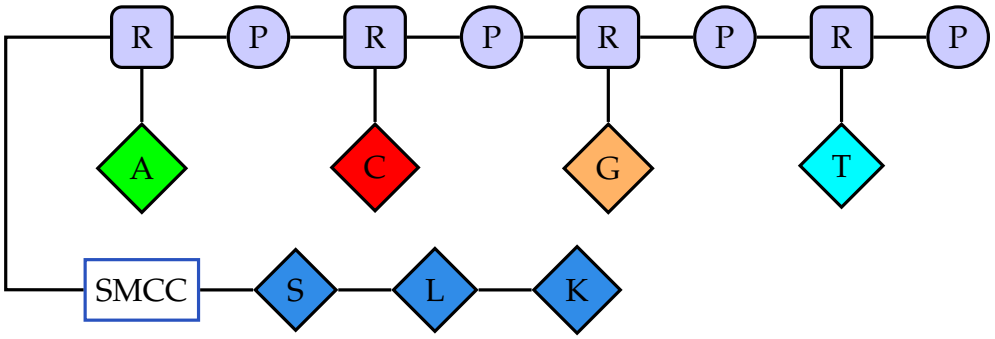
1.2 Grundlagen

Zum besseren Verständnis ist es notwendig, einen Überblick über HELM zu verschaffen und die Grundlagen zur molekularen Ähnlichkeit zu erläutern.

1.2.1 Hierarchical Editing Language for Macromolecules

Die hierarchische Strukturinformation von komplexen Biomolekülen in einer Notation prägnant zu repräsentieren ist eine Herausforderung. Eine solche Notation sollte simpel und dennoch ausreichend vielseitig sein, um alle Polymerstrukturen repräsentieren zu können. Noch wichtiger ist die vollständige Kompatibilität und Systematik, damit die Notation von Computerprogrammen generiert und in seine einzelnen Bestandteile zerlegt werden kann.

Die Pfizer Inc. entwickelte 2012 die Hierarchical Editing Language for Macromolecules (HELM) mit dem Ziel, die Repräsentation, Registrierung, Speicherung, Auswertung und Visualisierung einer stets wachsenden Menge von komplexen biomolekularen Strukturen zu ermöglichen [46]. HELM ist ähnlich zur Simplified Molecular Input Line Entry Specification (SMILES) und erfasst makromolekulare Informationen höherer Ordnung in einer kompakten und von Menschen lesbaren Weise. Sie behält dabei ein höchst strukturiertes und rechnerunterstütztes Format bei [20]. Eine Vielzahl von Unternehmen aus dem Bereich der Lebenswissenschaften integrieren die HELM-Notation in ihre Software. Sie benutzen die um HELM erbauten Programme (HELM Antibody Editor, HELM Editor, HELM Notation Toolkit, HELM Parser) [30] und entwickeln selber Programme, um die Möglichkeiten der Arbeit mit HELM zu erweitern. Die chemischen Datenbanken PubChem und ChEMBL enthalten bei vielen Molekülen die zugehörige HELM-Notation [29]. Die Strukturhierarchie der Notation besteht aus vier Ebenen (komplexes Polymer, simples Polymer, Monomer und Atom), wobei Komponenten auf einer höheren Ebene als Kombination von Komponenten aus einer niedrigeren Ebene definiert werden. Dieser Ansatz erlaubt es, ein Strukturformat für kleine Moleküle zur Repräsentation von Monomeren zu verwenden [46]. Wird ein Monomer durch SMILES repräsentiert, bezeichnet man das als in-line SMILES-Notation. In HELM werden nicht-natürliche Komponenten (wie z.B. nicht-natürliche Aminosäuren) und chemische Modifikationen sowie konventionelle natürliche Komponenten unterstützt [5]. Ein komplexes Polymer besteht aus einfachen Polymeren, die optional miteinander verbunden sein können. Ein einfaches Polymer besteht aus Monomeren vom selben Polymertyp (RNA, PEPTIDE oder CHEM). Monomere wiederrum bestehen aus Atomen. Um die Komplexität zu reduzieren, werden simple Polymere als lineare Ketten definiert. Dadurch muss die Ebene des einfachen Polymers keine Kettenverzweigungen oder Ringschlüsse behandeln. Die wichtigsten Eigenschaften der HELM-Notation werden anhand eines Beispiels dargestellt. Im Folgenden bezeichnet, wie auch in der Notation spezifiziert, der Polymertyp RNA alle Nukleotidpolymere inklusive Desoxyribonukleinsäure (DNS) und Ribonukleinsäure (RNS). Der Polymertyp PEPTIDE bezeichnet alle Polymere mit Peptidbindung zwischen Monomeren, die entweder natürliche oder synthetische Aminosäurereste sind. CHEM bezeichnet einen generischen Polymertyp für Monomere. Zu diesem Typ gehören beispielsweise chemische Linker [46]. In Abbildung 1.1 ist ein Beispiel für eine HELM-Notation dargestellt, anhand derer die einzelnen Komponenten im Folgenden verdeutlicht werden.



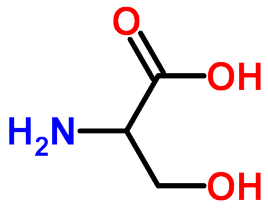
RNA1{R(A)P.R(C)P.R(G)P.R(T)}|CHEM1{[SMCC]}|PEPTIDE1{S.L.K}$
PEPTIDE1,CHEM1,1:R1-1:R2|RNA1,CHEM1,1:R1-1:R1$$$

Abbildung 1.1: Beispielmolekül bestehend aus drei verschiedenen Polymertypen. Dargestellt ist die HELM-Notation und darüber eine Graphrepräsentation des Moleküls.

Monomer

Ein Monomer besteht aus Atomen und Bindungen und kann durch ein bekanntes chemisches Strukturformat wie SMILES repräsentiert werden [46]. Es besitzt die folgenden Attribute: Struktur, ID, Bindungsstellen, Natürliches Analogon, Polymertyp, Monomertyp und Name [5]. Als Beispiel ist in Tabelle 1.1 das Monomer Serin vom Polymertyp PEPTIDE, welches auch im Beispielmolekül aus Abbildung 1.1 vorkommt, mit seinen Attributen definiert. Die Struktur des Monomers ist die Repräsentation der Atome und Bindungen. ID bezeichnet eine innerhalb eines Polymertyps eindeutige Identifikation (ID) wie zum Beispiel *S* für die Aminosäure Serin in einem Polymer des Typs PEPTIDE. Die Bindungsstellen sind die Punkte, an denen das Monomer eine Bindung zu einem anderen Monomer eingehen kann. Sie werden durch R-Gruppen mit eindeutigem Namen, wie etwa *R1*, spezifiziert. Natürliches Analogon ist ein optionales Attribut, das für nicht-natürliche Monomere zutrifft, welche in der Notationsschreibweise in eckige Klammern eingeschlossen werden und immer eine ID mit mehr als einem Zeichen besitzen. Als Beispiel lautet die ID des chemisch modifizierten N-Methyl-Serin *meS* und die ID des natürliche Analogons von *meS* lautet *S* (für Serin). Jedes Monomer muss zu einem der drei Polymertypen gehören. Der Monomertyp ist für RNA und PEPTIDE entweder Rückgrat oder Seitenkette, abhängig davon, ob das Monomer zum Grundgerüst des Moleküls gehört oder zu einer vom Rückgrat abzweigenden Seitenkette. Für CHEM-Monomere ist der Monomertyp „unbekannt“. Der Name ist eine vollständigere Bezeichnung für das Monomer, wie etwa Serin für die ID *S* [46].

Tabelle 1.1: Definition eines Serin-Monomers vom Typ PEPTIDE.

Struktur	
SMILES	OC[C@H](N[H:1])C([OH:2])=O
ID	S
Bindungsstellen	R1-H R2-OH
Natürliches Analogon	S
Polymertyp	PEPTIDE
Monomertyp	Rückgrat
Name	Serin

Simples Polymer

Die simple Polymer-Notation ist eine lineare Notation, die von links nach rechts gelesen wird. Der in Abbildung 1.1 dargestellte Polymerteil vom Typ PEPTIDE beinhaltet die drei Monomere *S*, *L* und *K*. Es besteht jeweils eine Bindung zwischen *S* und *L* sowie zwischen *L* und *K*. Abzweigende Monomere gibt es bei diesem Teil nicht. Die ID eines Seitenketten-Monomers wird in runde Klammern eingeschlossen und rechts des Rückgrat-Monomers geschrieben, von dem es abzweigt. Der RNA-Teil des Beispielmoleküls hat die Seitenketten-Monomere *A*, *C*, *G* und *T*, die jeweils von einem Rückgrat-Monomer *R* abzweigen, welches links von ihnen steht. Das Punktsymbol „.“ wird benutzt, um die simple Polymer-Notation in Gruppen einzuteilen. Die Notation *R(A)P.R(C)P.R(G)P.R(T)P* aus dem Beispielmolekül beschreibt neben den Bindungen zwischen den Monomeren auch, dass die Polymerkette in vier Gruppen aufgeteilt ist, die in diesem Fall jeweils ein Nukleotid repräsentieren. Der Polymertyp RNA behandelt DNS als modifizierte RNS. Zucker und Phosphate sind Rückgrat-Monomere und Basen sind Seitenketten-Monomere. Monomere vom Typ CHEM haben keine vordefinierten Bindungen zu anderen Monomeren. Daher besteht jedes CHEM-Polymer nur aus einem Monomer und ihre Verbindungen sind Bindungen zwischen simplen Polymeren und werden in der komplexen Polymer-Notation behandelt [46].

Komplexes Polymer

Im Gegensatz zur simplen Polymer-Notation, die einsträngige Polymere repräsentieren kann, dient die komplexe Polymer-Notation dazu, mehrere Polymerketten zu repräsentieren. Diese können optional miteinander verknüpft sein. Sie ist als Graphrepräsentation zu verstehen, bei der simple Polymerketten die Knoten darstellen und Bindungen zwischen den Polymerketten die Kanten [46]. Das Format der komplexen Polymer-Notation ist in Tabelle 1.2 in der linken Spalte dargestellt. Nur die erste Sektion (*ListOfSimplePolymers*) ist erforderlich, während das Sentinel-Symbol weitere Sektionen voneinander abgrenzt [5]. Die *ListOfSimplePolymers* enthält alle simplen Polymerketten mit einer eindeutigen ID im Format Polymertyp#. Dabei kann der Polymertyp entweder PEPTIDE, RNA oder CHEM sein und „#“ ist eine Zahl, die das Polymer eindeutig macht. Das Beispielmolekül besteht aus den Polymerketten RNA1, CHEM1 und PEPTIDE1. Die simplen Polymer-Notationen lauten *RNA1{R(A)P.R(C)P.R(G)P.R(T)}|CHEM1{[SMCC]}|PEPTIDE1{S.L.K}*. In geschweiften Klammern wird jeweils die zugehörige simple Polymer-Notation zur Polymer ID angegeben und die verschiedenen Polymere werden durch das Verkettungszeichen „|“ voneinander getrennt. In der *ListOfConnections* werden alle Bindungen zwischen Polymeren im Format

```
PID1,PID2,MonomerPos1:Bindungsstelle1-MonomerPos2:Bindungsstelle2
```

angegeben. PID1 und PID2 stehen für die Ursprungs- und Zielpolymer ID. MonomerPos1 und MonomerPos2 sind die Monomerpositionen im Ursprungs- und Zielpolymer und Bindungsstelle1 und Bindungstelle2 sind die Bindungsstellen der beteiligten Monomere. Ursprungs- und Zielpolymer ID können identisch sein z.B. bei zyklischen Verbindungen. Mehrere Bindungen werden durch das Verkettungszeichen voneinander getrennt. Die Bindung *PEPTIDE1,CHEM1,1:R1-1:R2* ist in Abbildung 1.1 angegeben. Daraus kann geschlossen werden, dass eine Bindung zwischen dem ersten Monomer von PEPTIDE1 (*S*) über die Bindungsstelle *R1* und dem ersten Monomer von CHEM1 (*[SMCC]*) über die Bindungsstelle *R2* besteht [46]. Polymere können in der *ListOfPolymerGroups* zu einer Gruppe zusammengefasst werden. Dabei wird der Gruppe eine ID, wie zum Beispiel *G1*, zugewiesen. Die Syntax einer Gruppe bestehend aus drei PEPTIDE-Polymeren würde

```
G1(PEPTIDE1+PEPTIDE2+PEPTIDE3)
```

lauten. Verschiedene Gruppen werden durch das Verkettungszeichung voneinander getrennt. Außerdem kann eine Gruppe Bestandteil einer anderen Gruppe sein.

Tabelle 1.2: Format der komplexen Polymer-Notation anhand des Beispiels aus Abb. 1.1.

ListOfSimplePolymers$	1. RNA1{R(A)P.R(C)P.R(G)P.R(T)} 2. CHEM1{[SMCC]} 3. PEPTIDE1{S.L.K.}
ListOfConnections$	1. PEPTIDE1,CHEM1,1:R1-1:R2 2. RNA1,CHEM1,1:R1-1:R1
ListOfPolymerGroups$	—
ExtendedAnnotations$	—

Die Sektion *ExtendedAnnotations* wird verwendet, um jegliche zusätzliche Annotation in validem JavaScript Object Notation (JSON)-Format hinzuzufügen [5]. Seit HELM 2.0 werden auch Funktionalitäten wie die Ambiguität unterstützt. Mehr Informationen über die HELM-Notation werden in der HELM-Spezifikation [5] und im Detail von Zhang *et al.* [46] erläutert.

1.2.2 Molekulare Ähnlichkeit

Das Konzept der molekularen Ähnlichkeit basiert auf dem Similar Property Principle [14], welches besagt, dass Verbindungen mit ähnlichen Strukturen dazu tendieren, ähnliche Bioaktivität aufzuweisen. Die ähnlichen Eigenschaften können durch ein Ähnlichkeitsmaß quantifiziert werden, welches eine Kombination aus einem molekularen Deskriptor und einem Ähnlichkeitskoeffizienten ist. Im Folgenden werden molekulare Deskriptoren und Ähnlichkeitskoeffizienten sowie zwei Arten des virtuellen Screenings (Substruktur– und Ähnlichkeitssuche) genauer erläutert.

Molekulare Deskriptoren

Deskriptoren repräsentieren Moleküle in reduzierter Form, wobei ein größerer Informationsverlust vermieden werden soll. Es gibt viele potentiellen Deskriptoren, weshalb es notwendig ist, eine passende Auswahl für die gewünschte Anwendung zu treffen. Die Ähnlichkeits– und Substruktursuche in großen chemischen Datenbanken benötigt Repräsentationen der Moleküle, die effektiv sind. Das heißt, sie müssen zwischen Molekülen, die sich unterscheiden, differenzieren können. Außerdem müssen sie effizient sein, also in der Praxis schnell zu kalkulieren. Es gibt einen generellen Konflikt zwischen diesen Anforderungen, dadurch, dass die effektivsten Methoden der Repräsentation dazu neigen, die am wenigsten effizienten zu sein und umgekehrt. Es muss also ein passender Kompromiss gefunden werden [44].

Eine einfache Möglichkeit, die Ähnlichkeit zwischen zwei Objekten zu beschreiben, ist die Bestimmung der Anzahl der gemeinsamen Merkmale. Bei der molekularen Ähnlichkeit gibt es viele Ansätze, dieses Konzept zu realisieren. Allen ist gemein, dass sie in einem linearen Deskriptor wie einem Bitstring oder Vektor für das Molekül resultieren. Jede Position in dem Bitstring repräsentiert ein einzelnes Merkmal. Der Bitstring für ein bestimmtes Molekül beinhaltet eine 1 an der Position, wenn das Merkmal vorhanden ist und ansonsten eine 0. Lineare Deskriptoren haben drei sehr nützliche Eigenschaften: Sie können einfach mit Computern gespeichert und verarbeitet werden, sie können einfach kombiniert werden, indem die entsprechenden Strings zu einem längeren String konkateniert werden und es kann ein Ähnlichkeitswert durch den Vergleich jeder Position zweier Strings berechnet werden. Häufig verwendete Funktionen für die Berechnung der Ähnlichkeit von zwei Bitstrings sind der Cosine-Koeffizient und der Tanimoto-Koeffizient. Letzterer wird in Abschnitt 1.2.2 genauer erläutert [23].

Die meistverwendeten linearen molekularen Deskriptoren sind rein strukturbasiert. Bei den Structural Keys beispielsweise wird eine Liste von kleinen Molekülfragmenten auf Bit-Positionen abgebildet. Beinhaltet das Molekül das Fragment, wird das zugehörige Bit auf 1 gesetzt, ansonsten auf 0. Ursprünglich wurden Structural Keys entworfen, um die Substruktursuche zu beschleunigen. Sie bringen eine relativ gute Leistung bei der Ähnlichkeitssuche. Der bedeutendste Nachteil ist, dass nur ein kleiner Anteil der funktionellen Gruppen direkt im Bitstring abgedeckt wird. Moleküle, die seltenere Fragmente beinhalten, können nicht richtig behandelt werden [23].

Dieses Problem kann angegangen werden, indem eine flexiblere Menge von Fragmenten genutzt wird, welche in einem Hashed Fingerprint resultiert. Das Auftreten von individuellen Fragmenten wird nicht geprüft, stattdessen werden alle Atompfade bis zu einer bestimmten Länge (in der Regel 5-7) aufgezählt. Da es zu viele verschiedene Pfade gibt, um für jeden ein Bit zu reservieren, wird eine Hashfunktion angewendet, die die Pfade auf eine bestimmte Menge an Bit-Positionen (meistens 0 bis 1.024) abbildet [23]. Dass mehrere Pfade auf dieselbe Bit-Position abgebildet werden können, ist nicht zu vermeiden. Hashed Fingerprints haben den Vorteil, dass sie nicht auf eine feste Fragmentmenge angewiesen sind und die Fingerprintlänge unabhängig von der Anzahl der Merkmale ist. Der Nachteil ist, dass dabei die 1-zu-1-Übereinstimmung zwischen Fragment und Bit-Position verloren geht und somit die Bit-Positionen auch nicht auf die Struktur rückabgebildet werden können [34].

Ähnlichkeitskoeffizienten

Die Idee der Berechnung eines numerischen Maßes von Ähnlichkeit (oder umgekehrt, Distanz) zwischen zwei Objekten wird in vielen unterschiedlichen Disziplinen verwendet. Daher wurde eine Menge wiederholter Aufwand betrieben und bekannte Ähnlichkeitskoeffizienten wurden mehrfach neu entwickelt. Das trug teilweise zu verschiedenen Namen für den gleichen Koeffizienten bei. Im Gegensatz zur Ähnlichkeit gibt es auch Koeffizienten, die die Distanz bzw. Unähnlichkeit zwischen Objekten erfassen [44].

Voraussetzung für die Anwendung eines Ähnlichkeitskoeffizienten ist ein Objekt, das mit Hilfe eines Vektors von Attributen beschrieben werden kann. Die Werte der Attribute können echte Zahlen über jeden Zahlenbereich oder auf dichotome Werte beschränkt sein, um das Vorhandensein oder Nichtvorhandensein einer bestimmten

Eigenschaft des Objekts anzuzeigen. Die Attribute können im Fall von molekularen Objekten eine Menge von berechneten physikochemischen Eigenschaften oder der Ja/Nein-Zustand von jedem der Bits in einem Fingerprint sein, welcher das Molekül repräsentiert. Meistens reicht der Wertebereich eines Ähnlichkeits- oder Distanzkoeffizienten von 0 bis 1. Für „absolute“ Vergleiche, beispielsweise zwischen zwei voneinander unabhängigen Molekülpaaren, eignet sich der Tanimoto-Koeffizient, welcher den maximalen Wert 1 für identische Objekte hat [44] und folgendermaßen berechnet wird:

$$S_{A,B} = \frac{\left[\sum_{j=1}^{n} X_{jA} X_{jB}\right]}{\left[\sum_{j=1}^{n} (X_{jA})^2 + \sum_{j=1}^{n} (X_{jB})^2 - \sum_{j=1}^{n} X_{jA} X_{jB}\right]} =: \frac{c}{(a+b-c)} \tag{1.1}$$

$S_{A,B}$ bezeichnet die Ähnlichkeit zwischen zwei Molekülen A und B und X_{jA} die j-te Eigenschaft des Moleküls A. a ist die Anzahl der Bits, die in Molekül A auf 1 gesetzt sind. b ist die Anzahl der 1-Bits in Molekül B, während c die Anzahl der gemeinsamen 1-Bits ist [3]. Auch bei dem Vergleich von einer Reihe von Ähnlichkeits– und Distanzkoeffizienten hinsichtlich ihrer Effizienz und Aussagekraft wird der Tanimoto-Koeffizient bevorzugt. Das basiert teilweise auf einer subjektiven Bewertung der aus der Ähnlichkeitssuche resultierenden Rangfolge und teilweise auf seiner Berechnung, die schneller ist als jene von Koeffizienten, die eine Wurzel beinhalten. Seitdem ist der Tanimoto-Koeffizient das Mittel der Wahl für molekulare Ähnlichkeitssuchen [43] [42].

Substruktursuche

Die häufigste Art des virtuellen Screenings in der modernen chemischen Forschung ist die Substruktursuche. Eine gemeinsame Substruktur von Molekülen ist die Menge von Atomen, die zwei Moleküle A und B gemeinsam haben [36]. Die Substruktursuche umfasst das Abrufen aller Moleküle einer Datenbank, die eine benutzerdefinierte Anfrage-Substruktur beinhalten, unabhängig von der Umgebung, in der die Anfrage-Substruktur vorkommt. Diese Suche hat jedoch einige Einschränkungen, die durch die Anforderung entstehen, dass eine Datenbankstruktur die ganze Anfrage-Substruktur enthalten muss. Das impliziert, dass der Nutzer, der die Datenbankabfrage stellt, schon ein relativ klares Bild vom Typ der Struktur haben muss, die abgerufen werden soll [44]. Graphentheoretisch ausgedrückt wird eine Reihe von topologischen Graphen auf eine Subgraph-Isomorphie zu einem bestimmten Anfragegraphen getestet [4]. Gegeben sind zwei Graphen G_1 und G_2 und

es gilt zu unterscheiden, ob Graph G_1 einen Teilgraph enthält, der isomorph zu G_2 ist [17]. Eine Subgraph-Isomorphie liegt vor, wenn alle Knoten (in diesem Beispiel die Atome eines Moleküls) des Graphen G_2 auf eine Teilmenge von Knoten des Graphen G_1 abgebildet werden können, sodass gleichzeitig die Kanten (in diesem Fall die Bindungen eines Moleküls) von G_2 auf eine Teilmenge der Kanten von G_1 abgebildet werden können. Wenn zwei Knoten in G_2 durch eine Kante verbunden sind, können sie genau dann auf zwei Knoten in G_1 abgebildet werden, wenn die beiden Knoten in G_1 auch durch eine Kante verbunden sind. Da eine echte Substruktursuche die Subgraph-Isomorphie voraussetzt, wird hier der Begriff des Substrukturfilters eingeführt. Dabei wird der Fingerprint eines Anfragemoleküls mit den Fingerprints der zu vergleichenden Moleküle verglichen. Kommt er als Teilmenge in einem oder mehreren der Fingerprints der Molekülmenge vor, werden diese als Ergebnis zurückgeliefert. Es wird hier von einem Substrukturfilter gesprochen, da die Fingerprints auf Deskriptoren basieren und somit keine echte Substruktursuche vorliegt.

Ähnlichkeitssuche

Die oben dargestellte Problematik der Substruktursuche, die voraussetzt, dass der Nutzer ein genaueres Bild von der abzurufenden Struktur haben muss, führte zur Entwicklung eines alternativen Zugriffsmechanismus, der Ähnlichkeitssuche. Eine Anfrage enthält in der Regel die Spezifizierung eines ganzen Moleküls, der Zielstruktur, statt einer Substruktur. Die Zielstruktur wird durch einen oder mehrere strukturelle Deskriptoren charakterisiert und diese Menge wird mit der korrespondierenden Menge von Deskriptoren jedes Moleküls in einer Datenbank verglichen. Diese Vergleiche ermöglichen die Berechnung eines Maßes an Ähnlichkeit zwischen der Zielstruktur und jeder der Datenbankstrukturen (wie etwa die Berechnung des Tanimoto-Koeffizienten). Die letzteren werden dann nach Ähnlichkeit zur Zielstruktur in absteigender Reihenfolge sortiert. Die Ausgabe der Suche ist dementsprechend eine Rangliste, bei der die als am ähnlichsten berechneten Strukturen am Anfang der Liste stehen. Diese werden am wahrscheinlichsten von Interesse für den Nutzer sein, wenn die Ähnlichkeit ausreichend ist [44]. Die Aussagekraft der Ähnlichkeitssuche hängt stark von der Fähigkeit des Deskriptors ab, so viele sinnvolle Informationen wie möglich zu gewinnen, um die Verbindung zu charakterisieren. Zudem hängt sie von der Fähigkeit des Koeffizienten ab, die Informationen des Deskriptors zu handhaben.

1.3 Problemstellung

Ein typischer Anwendungsfall für virtuelles Screening sieht folgendermaßen aus. Gegeben ist eine Datenbank von Molekülrepräsentationen, ein Anfragemolekül A sowie eine minimale Ähnlichkeit S_{min}. Die Anforderung besteht darin, alle Moleküle B in der Datenbank zu finden, deren Ähnlichkeit $S(A, B) \geq S_{min}$ ist [16]. Die gefundenen Moleküle werden nach Ähnlichkeit sortiert, in der Hoffnung, dass diejenigen mit der größten Ähnlichkeit zum Molekül A alternative Strukturen sind, die die essentiellen Eigenschaften von A beibehalten, während z.B. Patentierbarkeit und medizinalchemische Möglichkeiten gesteigert werden oder sogar optimierte physikochemische Profile erreicht werden [6]. Es ist gewünscht, dass das Screening durch eine Moleküldatenbank möglichst effizient abläuft, denn nicht nur die Datenbanken werden stets größer, sondern auch die Moleküle werden komplexer [46]. Als beliebte Molekülrepräsentation steht zum Beispiel SMILES [41] zur Verfügung, für die bereits Werkzeuge zur Berechnung von Ähnlichkeit existieren (Abschnitt 1.4). SMILES kann zwar beliebige Moleküle unabhängig von ihrer Größe und Komplexität darstellen, die resultierende Kette aus American Standard Code for Information Interchange (ASCII)-Zeichen wird jedoch extrem lang, dadurch, dass fast jedes einzelne Atom dargestellt wird. Daher wird SMILES hauptsächlich für die Repräsentation kleiner Moleküle verwendet. Hierarchical Editing Language for Macromolecules kann als neuer Standard für die Darstellung von Makromolekülen auch komplexe Moleküle kompakt und vor allem von Menschen lesbar repräsentieren. HELM ist eine relativ junge Sprache, daher bestehen im HELM-Projekt noch einige offene Herausforderungen. Dazu gehört unter anderem die HELM-Suche. Das Suchen in HELM-Notationen ist interessant, um gewünschte Informationen abzurufen, wie etwa gefundene Ähnlichkeiten. Für eine Datenbank von HELM-Notationen gibt es aktuell keine Möglichkeit, Vergleiche zu ziehen oder Übereinstimmungen zu finden [27]. Daher ist das Ziel dieser Arbeit, einen ersten Ansatz zu finden, wie in einer Datenbank von HELM-Notationen nach Ähnlichkeiten oder Teilmengen gesucht werden kann. Dazu soll es möglich sein, nach dem Schema des virtuellen Screenings vorzugehen. Gegeben ist eine HELM-Datenbank, eine Anfrage-HELM-Notation und eine minimale Ähnlichkeit. Gefunden werden sollen alle Notationen, dessen Ähnlichkeit mindestens der minimalen Ähnlichkeit entspricht oder alle Notationen, die die Anfrage-Notation als Teilmenge enthalten. Die Ergebnisse sollen nach absteigender Ähnlichkeit sortiert werden.

1.4 State of the Art

Der Begriff der Ähnlichkeit, nach dem Moleküle ihrer biologischen Wirkung oder physikochemischen Eigenschaften nach gruppiert werden können, hat ein breites Spektrum von Verwendungsmöglichkeiten in der Arzneimittelforschung gefunden. Virtuelles Screening, Abschätzungen und Vorhersagen von Eigenschaften oder Wirkungen sind Anwendungen, für die computergestützte Methoden unabdingbar geworden sind. Daher hat das Konzept molekularer Ähnlichkeit für die vorherrschenden Nutzer, Pharmaunternehmen, an praktischer Bedeutung drastisch zugenommen [6]. Obwohl einige häufig angewendete Methoden für molekulare Ähnlichkeitsberechnungen existieren, basieren die meisten auf praktischer Erfahrung. Inzwischen steht ein großer, virtueller „Methodenraum“ zur Verfügung, der darauf wartet erforscht zu werden. Dafür gibt es eine Fülle von Molekülrepräsentationen und eine große Anzahl an Ähnlichkeits– bzw. Distanzdefinitionen, um die Repräsentationen miteinander zu vergleichen. Das Wissen über die Auswirkungen, die die Wahl der Methoden auf molekulare Ähnlichkeitsberechnungen und –rankings hat, ist immer noch relativ knapp [3].

Ähnlichkeiten zwischen Molekülen können auf verschiedene Weisen berechnet werden. Dazu gehört die Ähnlichkeit basierend auf der Sequenz (Nukleotid– oder Peptidsequenz) eines Moleküls. Basic Local Alignment Search Tool (BLAST) ist eine Sammlung der weltweit am häufigsten verwendeten Programme zur Analyse biologischer Sequenzdaten [13]. Dieser Ansatz findet Regionen lokaler Ähnlichkeit zwischen Sequenzen. Der grundlegende Algorithmus kann auf verschiedene Weisen implementiert und in einer Vielzahl von Kontexten angewendet werden. Dazu gehört das Suchen in Nukleotid– und Proteindatenbanken, Mustersuche und Genidentifikationssuche [2]. Auf komplexe Makromoleküle, die aus unterschiedlichen Bausteinen bestehen (beispielsweise eine Nukleinsäure und ein Peptid verbunden über einen chemischen Linker) kann jedoch keins der BLAST-Programme angewendet werden. Dazu müsste der Nukleinsäure– und der Peptidteil jeweils einzeln in BLAST eingefügt werden.

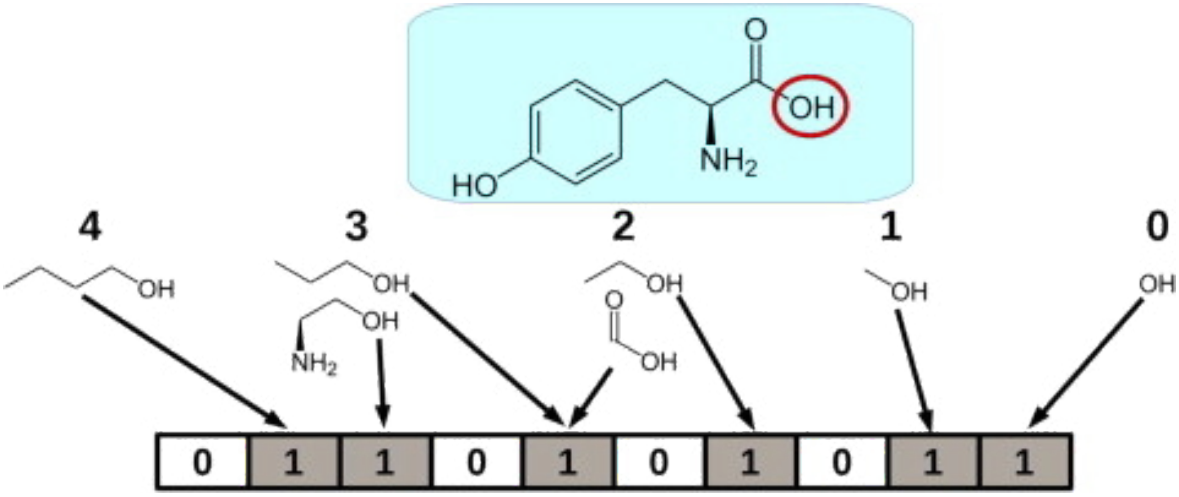


Abbildung 1.2: Daylight Fingerprint. Das Startatom ist in diesem Fall das rot umrandete Sauerstoffatom. Aufgelistet sind die vom Sauerstoff ausgehenden Pfade des Moleküls bis zu einer Länge von vier weiteren Atomen. Jeder Pfad gibt eine Bit-Position im Fingerprint an, welche auf 1 gesetzt wird, wenn der Pfad vorhanden ist [7].

Eine andere Weise, Ähnlichkeiten zwischen Molekülen zu finden, ist die chemische Ähnlichkeitssuche. Sie basiert auf der chemischen Struktur der Moleküle, also der Atomebene. Dabei ist die meistverwendete Methode die Deskriptor-basierte Ähnlichkeitssuche, die ein Molekül als eine Menge von Deskriptoren repräsentiert, sodass das Molekül als ein Punkt in einem multidimensionalen „Deskriptorraum“ gesehen werden kann. Bei den sogenannten molekularen Fingerprints wird nur das Vorhandensein oder Nichtvorhandensein eines Deskriptors festgestellt. Die Ähnlichkeit wird definiert durch die Anzahl an gemeinsamen Deskriptoren zweier Moleküle, normalisiert durch die Gesamtanzahl an Deskriptoren beider Moleküle [35]. Fingerprinting ist heutzutage zu einem unverzichtbarem Werkzeug geworden, um molekulare Ähnlichkeit zu bestimmen. Eine der bekanntesten Bibliotheken, die Computermethoden für verbreitete Problemstellungen der Bio– und Chemieinformatik zur Verfügung stellt, ist das Java Chemistry Development Kit (CDK). Die Funktionalität reicht von Input und Output von SMILES-Notationen über Visualisierung, Modellierung und Graphen (maximale, gemeinsame Substruktursuche und Ringsuche) bis zu Funktionen wie der Berechnung von Fingerprints [38]. Die Fingerprinter Klasse der CDK generiert Daylight-Fingerprints [12]. Sie werden erstellt, indem eine Breitensuche von jedem Atom des Moleküls durchgeführt wird, wobei Stringrepräsentationen von Pfaden der Länge von bis zu sechs Atomen entstehen (Abb. 1.2). Für jeden dieser Pfade wird ein Hash Code berechnet, mit dem eine pseudozufällige Zahl zwischen 0 und 1.023 erstellt wird. Diese Zahl gibt eine Position in einem

Fingerprint-Bitstring der Länge 1.024 an, welche dann auf 1 gesetzt wird. Basierend auf der Gesamtheit aller Pfade des Moleküls, erhält man einen molekularen Fingerprint in Form eines Bitstrings, der für Ähnlichkeits– und Substruktursuche verwendet werden kann [37]. Die Pfade können aus Molekülrepräsentationen wie SMILES erstellt werden. Auf der Basis von HELM-Notationen existieren bisher keine Methoden für Ähnlichkeits– oder Substruktursuchen. Eine Studentin der Cambridge Universität erstellte 2014 ein Proof of Concept, um zu demonstrieren, wie eine HELM-Suche implementiert werden könnte. Dabei konzentriert sich das Konzept darauf, dass Teile oder die gesamte HELM-Sequenz in SMILES umgewandelt wird und die Ähnlichkeit durch vorhandene Open Source Programme auf Atomebene berechnet wird [8].

1.5 Ziel der Arbeit

Die vorliegende Masterarbeit wurde bei der quattro research GmbH angefertigt, einem Unternehmen, das Softwarelösungen für Kunden aus der Life Science-, Pharma- und Chemieindustrie anbietet [1]. HELM wird von den Kunden des Unternehmens bereits zur Molekülrepräsentation verwendet, jedoch fehlt die Möglichkeit in den Notationen zu suchen.
Diese Arbeit beschreibt die Entwicklung von Algorithmen zur Ähnlichkeitssuche sowie die Implementierung eines Substrukturfilters in HELM-Notationen. Zunächst wird in Kapitel 2 die neue Methode aufgestellt und die Realisierung (mit unterschiedlichen Ansätzen) erläutert. Dazu gehört auch die Vorstellung einer graphischen Benutzeroberfläche. Im nächsten Schritt wird in Kapitel 3 das Verfahren validiert und die Laufzeit analysiert. In Kapitel 4 werden die Ergebnisse des Verfahrens dargelegt. Abschließend erfolgt eine Diskussion sowie ein Fazit mit weiterführender Arbeit. Als Programmiersprache wurde Java [25] verwendet mit Ausnahme von Teilen in SQL und kleineren Skripten, die in Python [33] geschrieben wurden.

2 Realisierung

2.1 Generierung des Datensets

Für die vorliegende Arbeit wurden Datensätze jeweils bestehend aus Peptiden, Oligonukleotiden und Chemical Compounds verwendet. Außerdem wurden Kombinationen aus den soeben beschriebenen Bestandteilen als Datensatz bestehend aus komplexen Molekülen erzeugt. Nachfolgend wird die Generierung der Daten beschrieben. Zunächst wurde ein Datensatz aus allen Molekülen, die in der PubChem Compound Datenbank mit Biologic Line Notation [22] registriert sind, erstellt. Dazu wurde eine Liste mit allen PubChem Compound Identification (CID)-Nummern heruntergeladen und eine Java-Funktion implementiert, die mit Hilfe eines RESTful Web Service Clients für jede CID-Nummer die Web-Ressource aufruft, in der die Molekülinformationen im JSON-Format [9] gespeichert sind. Über die Java JSON-Objekte konnte dann die HELM-Notation, falls vorhanden, entnommen werden. Diese wurden zusammen mit der CID-Nummer in einer Text-Datei gespeichert. Die resultierenden Notationen wurden mittels regulärer Ausdrücke gefiltert, um alle Notationen zu entfernen, die eine in-line SMILES-Notation beinhalten. Danach wurden Peptide und Oligonukleotide in separaten Text-Dateien gespeichert, um zunächst nur mit Peptiden als Datensatz zu beginnen und erst später auch Oligonukleotide zu verwenden. Die Peptid-Notationen sind einfacher, da sie abgesehen von möglichen Verbindungen nur lineare Ketten ohne Verzweigungen enthalten. Einen Sonderfall stellen die Oligonukleotide aus der PubChem dar. Die HELM-Notationen sind in der Datenbank fehlerhaft gespeichert.

```
Fehlerhaftes Format: RNA1{R(A).PR(C).PR(A)}$$$$
Korrektes Format:    RNA1{R(A)P.R(C)P.R(A)P}$$$$
```

Richtig wäre die Punktsetzung zwischen Phosphat und Zucker, um eine Einheit bestehend aus Zucker, Base und Phosphat kenntlich zu machen [46]. Aus diesem Grund wurde der Datensatz manuell verändert, indem in einem Python-Skript die Punkte entfernt und an die richtige Stelle gesetzt wurden. Insgesamt wurden aus der Datenbank 97.025 Peptide und 2.096 Nukleotide verwendet.

```
{
  "Record": {
  "RecordType": "CID",
  "RecordNumber": "129606597",
  "Section": [
    {
    "TOC Heading": "Biologic Description",
    "Description": "Information relevant to a biologic",
    "Section": [
      {
      "TOC Heading": "Biologic Line Notation",
      "Description": "A condensed line notation in one of a number of
         biologic formats",
      "Information": [
        {
        "ReferenceNumber": 1,
        "Name": "IUPAC Condensed",
        "StringValue": "cyclo[Ala-Gly-D-Arg-Gly-D-Asp-Ser-Pro]"
        },
        {
        "ReferenceNumber": 1,
        "Name": "PLN",
        "StringValue": "(cyclo)-AG{d}RG{d}DSP-(cyclo)"
        },
        {
        "ReferenceNumber": 1,
        "Name": "HELM",
        "StringValue": "PEPTIDE1{A.G.[dR].G.[dD].S.P}$PEPTIDE1,PEPTIDE1
           ,7:R2-1:R1$$$"
        },
        {
        "ReferenceNumber": 1,
        "Name": "IUPAC",
        "StringValue": "cyclo[L-alanyl-glycyl-D-arginyl-glycyl-D-alpha-
           aspartyl-L-seryl-L-prolyl]"
        }
       ]
      }
     ]
    }
  ],
  "Reference": [
    {
    "ReferenceNumber": 1,
    "SourceName": "PubChem",
    "SourceID": "PubChem",
    "Description": "Data deposited in or computed by PubChem",
    "URL": "https://pubchem.ncbi.nlm.nih.gov"
    }
   ]
  }
}
```


Abbildung 2.1: JSON-Format. Eintrag mit der CID-Nummer 129606597 in der PubChem Compound Datenbank im JSON-Format [21]. Unter Biologic Description - Biologic Line Notation ist der HELM-String zu finden.

In dem Datensatz waren nur wenige komplexe Moleküle enthalten, das heißt Kombinationen aus den drei Polymertypen. Daher wurden für Testzwecke Beispiele aus der HELM Spezifikation [5] verwendet und eigene Moleküle bestehend aus allen drei Polymertypen mit dem HELM Editor v1.4 [28] erstellt. Zusätzlich wurden eigene kleine Datensätze von fünf bis zwanzig HELM-Notationen erstellt, die für bestimmte Testzwecke angepasst wurden. Dazu gehören beispielsweise Notationen, die sich immer nur jeweils um ein weiteres modifiziertes Monomer unterscheiden.

2.2 Auswahl des Deskriptors

Als Deskriptor für die Ähnlichkeit zwischen HELM-Notationen wurde ein pfadbasierter Hashed Fingerprint ausgewählt. Dabei werden alle Monomerpfade der linearen simplen Polymer Sektion der HELM-Notation bis zu einer bestimmten Länge aufgezählt. Darüber hinaus können die Pfade auch über verschiedene simple Polymere gehen, wenn eine Bindung zwischen ihnen besteht. Jeder Pfad stellt ein Merkmal des Moleküls dar, über das das Molekül bzw. die HELM-Notation beschrieben werden kann. Die Pfadgenerierung wird unter Abschnitt 2.3.3 genauer beschrieben. Jeder Pfad wird auf eine Bit-Position im Fingerprint des Moleküls abgebildet. Die mögliche Anzahl der Pfade pro Notation ist lediglich durch die Größe des Moleküls begrenzt. Daher müssen mehrere Pfade auf das selbe Bit abgebildet werden und zu diesem Zweck wird eine Hashfunktion, die die Pfade auf eine bestimmte Menge an Bit-Positionen abbildet, angewendet. Der Hashed Fingerprint enthält eine 1 an der Bit-Position, falls ein bestimmter Pfad vorhanden ist, ansonsten eine 0.

2.3 Von der HELM-Notation zum Fingerprint

In Abbildung 2.2 ist der Workflow der gesamten Software schematisch dargestellt. Der Input besteht aus zwei Teilen; einer Textdatei mit HELM-Notationen sowie einer Anfrage-HELM-Notation und gegebenenfalls der gewünschten Mindestähnlichkeit, falls eine Ähnlichkeitssuche beabsichtigt ist. Aus den HELM-Notationen der Textdatei sowie der Anfrage-HELM-Notation werden zunächst Java Objekte generiert, welche in Graphen konvertiert werden. Aus dem Graph kann mit Hilfe der Pfade ein Fingerprint für jede Notation erstellt werden. Die Fingerprints können dann entweder in der Ähnlichkeitssuche oder im Substrukturfilter verwendet werden. Im Folgenden werden die einzelnen Schritte im Detail erklärt. Dafür wird der gesamte Ablauf exemplarisch anhand der HELM-Notation *RNA1{R(A)P.R(C)P}$$$$* verdeutlicht.

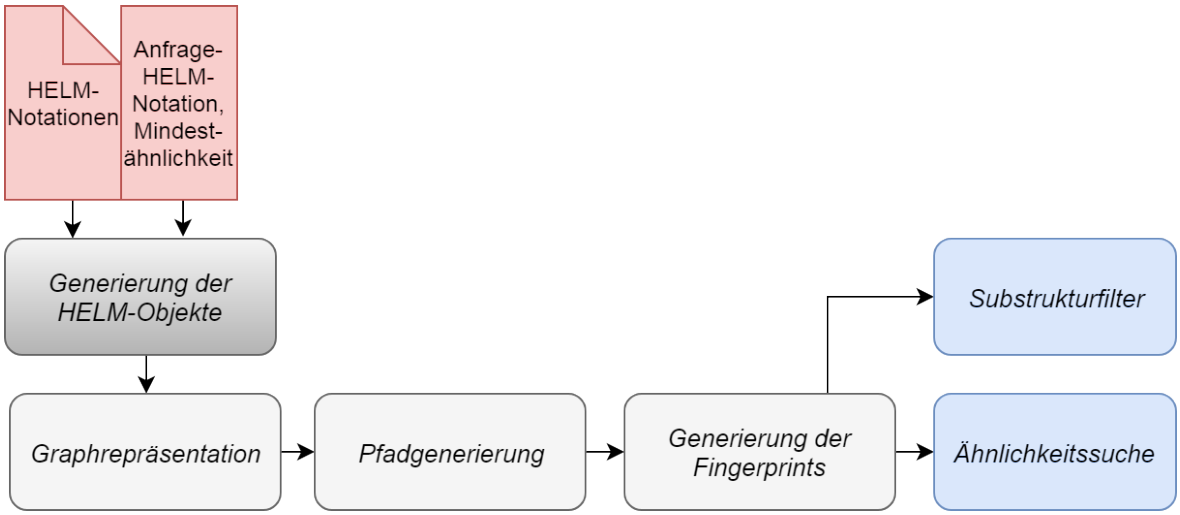


Abbildung 2.2: Workflow-Diagramm der Software. Rot hinterlegt ist der Input der Software, die Repräsentation und Encodierung der HELM-Objekte ist grau dargestellt. Die Generierung der HELM-Objekte verwendet die bereits existierende Softwarebibliothek HELMNotationParser [31] und ist deshalb dunkel hinterlegt. Blau dargestellt sind die beiden möglichen Anwendungen Ähnlichkeitssuche und Substrukturfilter.

2.3.1 Generierung der HELM-Objekte

Für die Generierung der HELM-Objekte wurde der HELMNotationParser verwendet, der eine Notation als ein Toolkit-internes Java Objekt speichert [31]. Dabei wird die Notation in ihre Bausteine zerlegt und der Zugriff auf diese erleichtert. Das ist notwendig, um die Graphrepräsentation umsetzen zu können, da jedes simple Polymer, jedes Monomer und jede Verbindung einer HELM-Notation prozessiert werden muss. Zusätzlich werden Informationen der Monomere benötigt, wie etwa die natürlichen Analoga von nicht-natürlichen Monomeren.

2.3.2 Graphrepräsentation

Die lineare HELM-Notation wird in einen ungerichteten Graphen konvertiert. Dabei stellen die Monomere die Knoten des Graphen dar und Bindungen zwischen den Monomeren die Kanten des Graphen. Die Graphrepräsentation ermöglicht das Traversieren durch das Molekül und erleichtert somit die Pfadgenerierung.

Algorithmus 1 zeigt die Generierung eines HELM-Graphen. Für ein komplexes Polymer wird der Graph in drei Schritten erstellt. Es wird zunächst für jedes simple Polymer der Polymertyp identifiziert und je nach Typ wird der Teil des Graphen

Algorithmus 1 Graphrepräsentation für eine HELM-Notation

```
1:  procedure BUILDMOLECULEGRAPH(POLYMERNOTATIONS, CONNECTIONS)
2:      for each polymer P ∈ PolymerNotations do:
3:          if P.polymerType is „RNA“ then
4:              BUILDRNAPART(P)
5:          else if P.polymerType is „PEPTIDE“ then
6:              BUILDPEPTIDEPART(P)
7:          else if P.polymerType is „CHEM“ then
8:              BUILDCHEMPART(P)
9:      ADDCONNECTIONS(MoleculeGraph, PolymerNotations, Connections)
10:     return moleculeGraph

11: procedure BUILDRNAPART(P)
12:     for each nucleotide n ∈ P do:
13:         for each monomer ∈ n do:
14:             moleculeGraph.add(monomer)
15:             monomerTypes.add(monomer.getType())
                                          ▷ monomerTypes: sugar, base, phosphate
16:     for vertices u, v ∈ moleculeGraph do:
17:         if CHECKBOND(monomerType(u), monomerType(v)) then
18:             CREATEBOND(u, v)

19: procedure BUILDPEPTIDEPART(P)
20:     for each monomer m ∈ P do:
21:         moleculeGraph.add(monomer)
22:     for vertex v ∈ moleculeGraph do:
23:         CREATEBOND(v, moleculeGraph[v+1])

24: procedure BUILDCHEMPART(P)
25:     moleculeGraph.add(P.getMonomer())

26: procedure ADDCONNECTIONS(MOLECULEGRAPH, CONNECTIONS)
27:     for each connection c ∈ Connections do:
28:         CREATEBOND(c.getFirstVertex(),c.getSecondVertex())
```

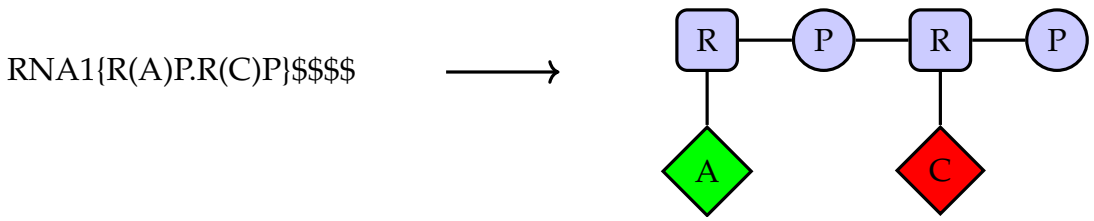


Abbildung 2.3: Graphrepräsentation aus der HELM-Notation (links) mit Monomeren als Knoten und Bindungen zwischen Monomeren als Kanten.

unterschiedlich aufgebaut. Bei einem simplen Polymer vom Typ RNA wird jedes Nukleotid in seine Monomere aufgeteilt, die entweder Zucker, Base und Phosphat sein können (Algorithmus 1 **buildRNAPart()**). Jedes Monomer wird zum *moleculeGraph* hinzugefügt und danach wird über die neu hinzugefügten Knoten im Graph iteriert. Für jeden Knoten wird überprüft, ob dieser mit dem nächsten oder mit dem übernächsten Knoten in der Liste verbunden ist. Letzteres ist nur der Fall, wenn es sich um einen Zucker und eine Base oder um einen Zucker und ein Phosphat handelt, sodass eine Bindung zwischen den Knoten hinzugefügt wird. Base und Phosphat können nicht miteinander verbunden sein, da die Base immer mit einem Zucker verbunden ist. Auf diese Weise wird das RNA-Polymer mit dem Rückgrat aus Zuckern und Phosphaten sowie den Seitenketten aus Basen richtig repräsentiert. Für ein simples Polymer des Typs PEPTIDE wird jedes Monomer zum *moleculeGraph* hinzugefügt. Es wird dann über die neu hinzugefügten Knoten im Graph iteriert und zwischen jedem Knoten und dem jeweils nächsten Knoten eine Bindung hinzugefügt. Damit ist die Erstellung des Peptid-Teils abgeschlossen, da es ausschließlich Rückgrat-Monomere gibt und mögliche Bindungen erst in der *listOfConnections* behandelt werden. Gehört ein simples Polymer zum Typ CHEM muss lediglich ein Monomer zum *moleculeGraph* hinzugefügt werden, da diese Polymerteile nur aus einem einzigen Monomer bestehen. Auch hier werden mögliche Bindungen erst in der *listOfConnections* behandelt. Nachdem die einzelnen Teile eines komplexen Polymers erstellt wurden, wird die *listOfConnections* behandelt. Für jede Verbindung werden die beteiligten Monomere und das zugehörige Polymer identifiziert und eine Bindung zwischen den Monomeren (bzw. ihren korrespondierenden Konten im *moleculeGraph*) hinzugefügt. In Abbildung 2.3 wird ersichtlich, warum es hilfreich ist, die HELM-Notation zunächst als Graph zu repräsentieren. Anhand der linearen Notation könnte man davon ausgehen, dass ein Pfad der Länge vier *RAPR* lautet. Der korrekte Pfad lautet jedoch *ARPR*, da die in runden Klammern stehende Base A ein vom Zucker R abzweigendes Monomer ist.

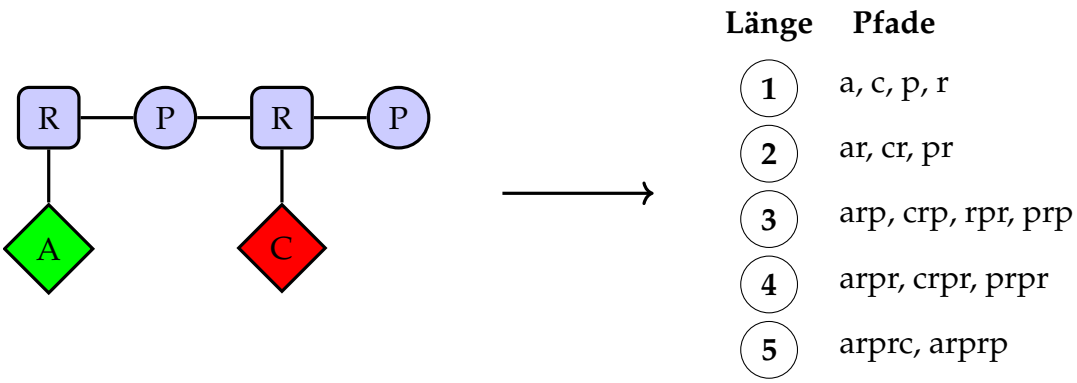


Abbildung 2.4: HELM-Pfade generiert nach Ansatz 1. Der Länge nach sortierte Pfade, die nach Traversierung des Graphen der HELM-Notation entstehen.

2.3.3 Pfadgenerierung

Ansatz 1

Für die Pfadgenerierung wird über die Menge der Knoten V im *moleculeGraph* iteriert. Für jeden Knoten v als Startknoten wird zunächst eine Liste *visited* über Werte des Typs *boolean* mit den Werten *false* gefüllt. Diese Liste repräsentiert die restlichen Knoten des Molekülgraphen und damit die weiteren Nachbarn von v. Sie wird benötigt, um zu verfolgen, welche Knoten ausgehend von v schon besucht wurden. Generiert wird eine Menge von Pfaden, das aus den IDs der Monomere besteht, die auch in der ursprünglichen HELM-Notation vorkommen. Dabei ist es unabhängig davon, ob es sich um ein natürliches oder ein nicht-natürliches Monomer handelt. In jeder Iteration wird ein aktueller Pfad gespeichert, zu dem die nächsten Knoten bzw. Monomer-IDs hinzugefügt werden können. Danach wird überprüft, ob der aktuelle Pfad valide ist. Ein valider Pfad existiert noch nicht in der Pfadmenge. Falls er schon als Palindrom existiert, wird nur das lexikographische Minimum der beiden Pfade gespeichert, während der andere Pfad verworfen wird. Ausgehend vom Startknoten v wird eine Tiefensuche (*depth-first search* (DFS)) durchgeführt. Dafür wird in der DFS jeder Nachbar von v besucht, bei dem *visited* auf *false* gesetzt ist. Die besuchten Knoten werden zum Pfad hinzugefügt, valide Pfade werden gespeichert und die DFS wird so lange rekursiv aufgerufen, bis die Rekursionstiefe größer als die maximale Pfadlänge ist. Für die maximale Pfadlänge wurde, wie beim Daylight Fingerprint und wie bei Hashed Fingerprints üblich, sechs gewählt [12] [23]. Die IDs der natürlichen Monomere vom Polymertyp RNA werden in Kleinbuchstaben im Pfad abgespeichert, um Überschneidungen mit dem Polymertyp PEPTIDE zu vermeiden. Beispielsweise kann die ID *R* für Ribose und auch für Arginin ste-

hen. Somit wird im Algorithmus die ID r für Ribose verwendet und R weiterhin für Arginin. Das Resultat des Algorithmus ist eine Menge von allen möglichen Pfaden der HELM-Notation. In Abbildung 2.4 sind alle Pfade zu sehen, die aus der Graphrepräsentation der Beispiel-HELM-Notation entstehen. Diese sind nach der Länge des Pfades sortiert. In diesem Beispiel gibt es keine Pfade länger als fünf, da die längste Monomerkette im Graph fünf Monomere hat.

Ansatz 2

Algorithmus 2 Pfadgenerierung für eine HELM-Notation (Ansatz 2)

```
1:  procedure FINDPATHS(MOLECULEGRAPH G = (V, E))
2:      for each startvertex v ∈ V do:
3:          if v.isNonNatural() then
4:              currentNaturalPath ←currentNaturalPath + v.getNaturalAnalog()
5:          else
6:              currentNaturalPath ←currentNaturalPath + v.getMonomerID()
7:          currentOriginalPath ← v.getMonomerID()
8:          CHECKANDSTOREPATHS(currentOriginalPath, currentNaturalPath)
9:          for each u ∈ V do:
10:             visited[u] ← false                    ▷ potential neighbours of v
11:         DFS(G, visited, currentOriginalPath, currentNaturalPath, depth = 0)
12:     return originalPaths, naturalPaths
```

Der zweite Ansatz ist in Algorithmus 2 dargestellt. Hier werden zwei Mengen von Pfaden (originale Pfade und natürliche Pfade) generiert. Die IDs der Monomere, die auch in der ursprünglichen HELM-Notation vorkommen, sind wie in Ansatz 1 in den originalen Pfade enthalten unabhängig davon, ob es sich um ein natürliches oder ein nicht-natürliches Monomer handelt. Die natürlichen Pfade hingegen enthalten ausschließlich IDs natürlicher Monomere. Aus diesem Grund wird zu Beginn des Algorithmus überprüft, ob es sich bei dem Startknoten v um ein nicht-natürliches Monomer handelt. Ist das der Fall, wird zum aktuellen natürlichen Pfad (Algorithmus 2 *currentNaturalPath*) das natürliche Analogon des Monomers ermittelt und hinzugefügt. Dafür wird die bereits existierende Software HELMNotation-Toolkit [32] verwendet, die einen MonomerStore mit einer Reihe von Monomeren und ihren Attributen gespeichert hat (siehe Anhang E). Für die originalen Pfade wird das Monomer v in jedem Fall als ID zum aktuellen originalen Pfad (Algorithmus 2 *currentOriginalPath*) hinzugefügt. Die Überprüfung auf valide Pfade in **checkAndStorePath()** sowie die rekursive Tiefensuche **DFS()** (Algorithmus 2) läuft

wie in Ansatz 1 ab, allerdings mit beiden Pfaden, um beide vervollständigen zu können. Das Resultat von Ansatz 2 sind zwei Mengen von allen möglichen originalen und natürlichen Pfaden des Moleküls. Der aus den Pfaden berechnete Fingerprint beinhaltet dementsprechend beide Pfade: originale und natürliche Pfade.

2.3.4 Generierung der Fingerprints

Die Encodierung der Pfade erfolgt über einen Hashed Fingerprint. Der Fingerprint hat eine feste Länge N von entweder $N = 1.024$ Bits. Um von den Pfaden zu einem Integer zwischen 0 und $N - 1$ zu gelangen, wird zunächst auf jeden Pfad der HELM-Notation eine Hashfunktion angewendet.

Ansatz 1

Die erste verwendete Hashfunktion ist die von fast jeder Java-Klasse bereitgestellte Funktion **hashCode()**. Bei den Pfaden handelt es sich um Objekte der Klasse String. Der Hash Code eines String-Objekts s mit der Länge n berechnet sich wie folgt:

$$hashCode(s) = s[0] \cdot 31^{(n-1)} + s[1] \cdot 31^{(n-2)} + ... + s[n-1] \tag{2.1}$$

Dabei ist $s[i]$ die UTF-16-Code-Unit des i-ten Buchstaben des Strings [24]. Die Generierung von Bit-Positionen über Hash Codes ist in Algorithmus 3 dargestellt. Das Ergebnis der Hash Code-Berechnung ist ein 32-bit Integer [26], welcher als Startwert (Seed) für einen Zufallszahlengenerator der Klasse Random fungiert [40]. Mit Hilfe des Generators wird dann eine ganzzahlige Pseudo-Zufallszahl im Bereich 0 bis N als Länge des Fingerprints generiert. Die Pseudozufallszahl ist die Bit-Position im Fingerprint, die für den Pfad steht und auf 1 gesetzt werden soll.

Algorithmus 3 Generierung von Bit-Positionen über Hash Codes (Ansatz 1)

```
1: procedure HASHCODEFINGERPRINT(PATHS, N)
2:     for each path p ∈ Paths do:
3:         hashCode ← hashCode(p)
4:         bitPosition ← Random(hashCode) between [0...N − 1]
5:         set bitPosition in fingerprint
6:     return fingerprint
```

Ansatz 2

Im zweiten Ansatz wurde eine andere Hashfunktion verwendet. Ausgewählt wurde die kryptologische Hashfunktion Secure Hash Algorithm 2 (SHA-2), die es in den vier Varianten SHA-224, SHA-256, SHA-384, SHA-512 gibt. Dabei stehen die Ziffern für die Anzahl der Bits des Hashes, den sie produzieren [19]. Da der weiter unten beschriebene Faltalgorithmus auf den Hash angewendet wird, wird die Funktion mit der kleinsten Anzahl an Bits genutzt, die von der verwendeten Java Klasse unterstützt wird: SHA-256. In Algorithmus 4 ist die Generierung von Bit-Positionen über SHA-2 gezeigt. Für jeden Pfad der HELM-Notation wird der Hash aus SHA-2 ermittelt. Der Hash wird in ein Bitset konvertiert, das aus 256 Bits besteht. Um

Algorithmus 4 Generierung von Bit-Positionen über SHA-2 (Ansatz 2)]

```
1:  procedure SHA2FINGERPRINT(PATHS, N)
2:      for each path p ∈ Paths do:
3:          hash ← SHA2(p)
4:          convert hash to bitSet
5:          bitPosition ← FOLD(bitSet, N)
6:          set bitPosition in fingerprint
7:      return fingerprint

8:  procedure FOLD(BITSET, N)
9:      while length of bitSet > log2(N) do:
10:         bitSet[0...(length/2) − 1] XOR bitSet[(length/2)...(length − 1)]
11:     convert bitSet to decimal
12:     return bitSet
```

daraus eine Bit-Position zwischen 0 und N zu erhalten und so wenig Informationen wie möglich aus dem Hash zu verlieren, wird eine Methode angewendet, die das Bitset so lange faltet bis es aus $log_2(N)$ vielen Bits besteht, um $N-1$ viele Zahlen darstellen zu können. Bei der Faltung des Bitsets wird dieses in einer Schleife jeweils in der Hälfte aufgetrennt und auf die beiden Hälften das bitweise exklusive Oder angewendet. Das Ergebnisbit ist 1, wenn zwei Bits unterschiedlich sind, und 0, wenn sie gleich sind. Das resultierende Bitset wird in eine Dezimalzahl umgewandelt und dient wie in Ansatz 1 als Bit-Position, die für den Pfad auf 1 gesetzt wird. Abb. 2.5 zeigt schematisch die Encodierung von Pfaden einer HELM-Notation in Fingerprints. Dargestellt sind zwei HELM-Notationen, die sich nur in einem Monomer unterscheiden (*C* und *G*). Der Pfeil zeigt jeweils auf einen Ausschnitt des zugehörigen Fingerprints der Notation. Über bzw. unter dem Fingerprint sind die Pfade zur Bit-Position angegeben. Daran ist beispielsweise erkennbar, dass in der

oberen HELM-Notation der Pfad *CR* vorhanden ist und das Bit im Fingerprint an dieser Position auf 1 gesetzt ist. In der unteren Notation ist der Pfad nicht vorhanden, dementsprechend ist das Bit an der Position auf 0 gesetzt.

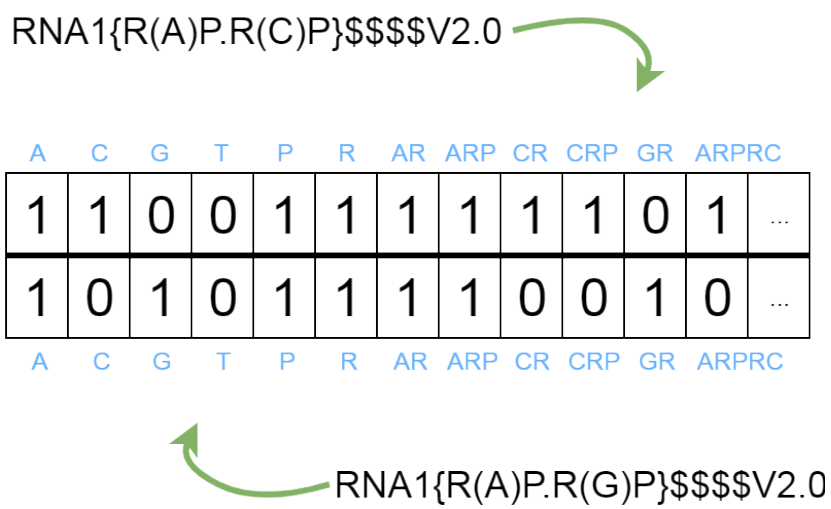


Abbildung 2.5: Schematische Darstellung der Encodierung von HELM-Pfaden in Fingerprints. Zu sehen sind zwei HELM-Notationen, die sich in einem Monomer unterscheiden (*C* und *G*). Ein Ausschnitt der korrespondierenden Fingerprints zeigt, welche Pfade vorhanden sind und welche Bits somit auf 1 gesetzt wurden. Der zugehörige Pfad zur Bit-Position ist jeweils über bzw. unter dem Fingerprint angegeben.

2.4 Berechnung der Ähnlichkeit

Um einen Ähnlichkeitswert zwischen zwei HELM-Notationen zu erhalten, wird der Tanimoto-Koeffizient der beiden Fingerprints berechnet. Dazu wird die Anzahl an Bits der Fingerprints, die jeweils auf 1 gesetzt sind sowie die Anzahl der gemeinsamen 1-Bits gezählt. Damit kann der Tanimoto-Koeffizient nach Gleichung 1.1 berechnet werden. Die Berechnung soll beispielhaft an den beiden HELM-Notationen aus Abbildung 2.5 erläutert werden. Die Anzahl der 1-Bits des ersten Fingerprintausschnitts beträgt neun, die des zweiten sieben. Es gibt fünf gemeinsame 1-Bits. Aus der Berechnung $\frac{5}{9+7-5} = 0.\overline{45}$ ergibt sich ein Tanimoto-Wert von $0.\overline{45}$.

2.5 Berechnung von Teilmengen

Algorithmus 5 Substrukturfilter

```
1: procedure CHECKFORSUBSET(FINGERPRINT1, FINGERPRINT2)
2:     if fingerprint1 ∧ fingerprint2 == fingerprint1 then
3:         fingerprint1 is a subset of fingerprint2
4:         return true
5:     return false
```

Neben der Ähnlichkeitssuche für HELM-Notationen wurde auch ein Substrukturfilter implementiert. Dieser ist in Algorithmus 5 dargestellt und basiert auf den generierten Fingerprints für jede HELM-Notation. Ob ein Fingerprint eine Teilmenge des anderen ist, wird über die bitweise Operation des logischen Und herausgefunden. Die Operation wird auf die beiden Fingerprints gleicher Länge angewendet und damit eine logische Und-Verknüpfung (logische Konjunktion) auf jedem Paar korrespondierender Bits durchgeführt. Das Bit im Ergebnisfingerprint ist 1, wenn beide Bits 1 sind, ansonsten 0 [18]. Ist der Ergebnisfingerprint gleich dem ersten Fingerprint (siehe Algorithmus 5 *fingerprint*1), so ist *fingerprint*1 vollständig im anderen Ausgangsfingerprint (Algorithmus 5 *fingerprint*2) enthalten und somit eine Teilmenge von diesem.

2.6 Graphische Benutzeroberfläche

Für die Benutzung der implementierten Algorithmen wurde eine graphische Benutzeroberfläche entwickelt. Diese dient der erleichterten Bedienung für den Nutzer und einem strukturierteren Output. Dafür wurde die plattformübergreifende Anwendung JavaFX von Java 8 [25] verwendet. Die Nutzeroberfläche ist in Abbildung 2.6 als Screenshot zu sehen und in drei Bereiche aufgeteilt: *Input*, *Options* und *Results*. Im *Input*-Bereich muss der Nutzer eine Text-Datei hochladen, in der eine HELM-Notation in jeder Zeile gespeichert ist. Wahlweise kann vor der Notation eine ID stehen, die eindeutig und mit einem Tab von der Notation getrennt sein muss. Wenn keine ID angegeben wird, wird sie automatisch für jede HELM-Notation erzeugt. Bei der Verarbeitung der Text-Datei wird eine SQLite Datenbank erstellt, in der die ID und die zugehörige Notation gespeichert werden. Im *Options*-Bereich muss zwischen der Ähnlichkeitssuche und dem Substrukturfilter gewählt werden (Abb. 2.6). Bei beiden Varianten wird auf die zuvor erstelle SQLite Datenbank zugegriffen und für jede eingetragene HELM-Notation der Fingerprint wie in Abschnitt 2.3.4 Ansatz 2 erstellt. Die Ähnlichkeitssuche teilt sich in weitere Optionen. Der Benutzer kann entweder einen gewünschten Ähnlichkeitswert in % angeben und sich damit alle HELM-Notationen anzeigen lassen, die zur Anfrage-HELM-Notation mindestens so ähnlich sind wie der angegebene Mindestwert. Die andere Möglichkeit ist die Anzeige von den zehn ähnlichsten HELM-Notationen, die zur Anfrage-HELM-Notation gefunden werden konnten. Zusätzlich gibt es hier ein Kontrollkästchen, mit dem angegeben werden kann, ob nicht-natürliche Monomere in ihre natürlichen Analoga umgewandelt werden und für die Ähnlichkeitsberechnung mit einbezogen werden sollen. Bei dem Substrukturfilter gibt es keine weiteren Optionen. Der Filter findet alle HELM-Notationen, in denen der Fingerprint der Anfrage-HELM-Notation als Teilmenge vorkommt. Bevor eine jeweilige Funktion gestartet werden kann, muss der Benutzer seine Anfrage-HELM-Notation in das vorgegebene Textfeld eingeben. Im *Results*-Bereich werden die Ergebnisse aufgelistet (Abb. 2.6). Die Ergebnisnotationen werden in absteigender Reihenfolge der Ähnlichkeit mit ihrer ID, der HELM-Notation und der Ähnlichkeit angezeigt. Die Ergebnisse können als Textdatei exportiert werden. Für die Berechnung der Ähnlichkeit und des Substrukturfilters wird die Pfadgenerierung aus Abschnitt 2.3.3 Ansatz 1 und Ansatz 2 genutzt, je nachdem ob das Kontrollkästchen für die natürlichen Analoga selektiert wurde oder nicht.

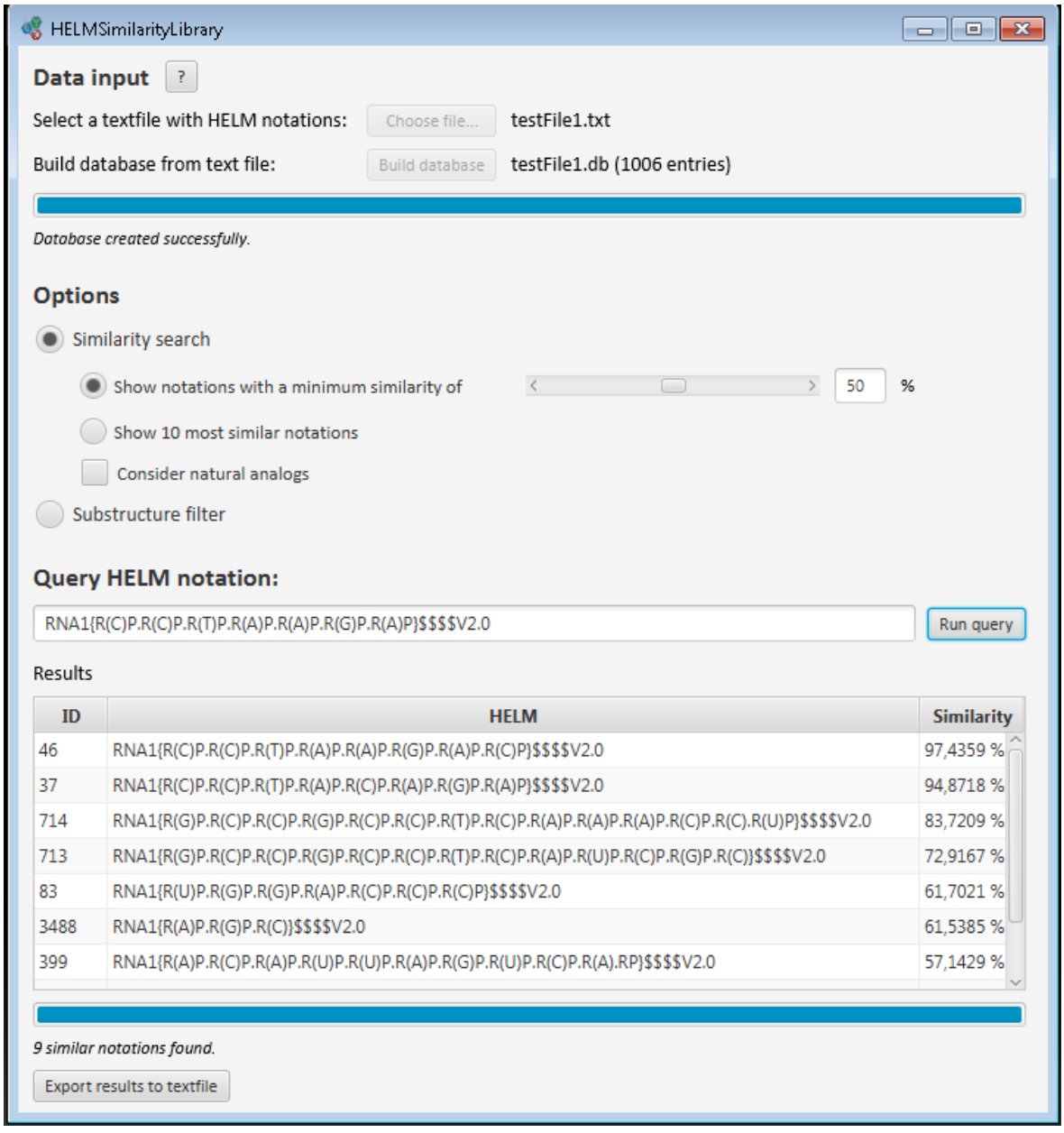


Abbildung 2.6: Screenshot der graphischen Benutzeroberfläche der Softwarebibliothek HELMSimilarityLibrary. Beispielhaft wurde eine Anfrage-HELM-Notation verwendet, die aus einem RNA-Polymer besteht. Diese wurde mit einem Datenset von 1.006 HELM-Notationen [22] verglichen.

3 Evaluation

3.1 Validierung

Die Korrektheit der einzelnen Funktionen des entwickelten Programms wurde mit Hilfe von JUnit 5 für Java 8 überprüft [15]. Dafür wurde vor allem darauf geachtet, dass die Generierung des Molekülgraphen, die Pfadgenerierung sowie die Pfadspeicherung korrekt abläuft. Die Validierung wurde in den Testfunktionen mit mehreren Beispiel-HELM-Notationen durchgeführt.

3.1.1 Test: Graphrepräsentation

Beim Test der Graphrepräsentation wird überprüft, ob die implementierte Funktion **buildMoleculeGraph()**, die einen Graph aus einer HELM-Notation erstellt, alle Monomere zum Graph hinzugefügt und alle Bindungen korrekt anlegt. Dazu werden zwei Graphen erstellt. Einer davon wird direkt aus einer HELM-Notation in **buildMoleculeGraph()** erstellt. Der andere Graph wird manuell erstellt. Bei der manuellen Generierung werden neue Knoten für die Monomere erzeugt und die benachbarten Knoten miteinander verknüpft. Die beiden Graphen werden dann auf Gleichheit überprüft. Dafür bietet JUnit mit der Klasse Assert viele Funktionen, die die Überprüfung von Objekten auf Gleichheit erleichtert. Das Objekt, also der Graph selber, muss dafür eine Methode **equals()** implementieren, die alle Attribute der beiden Graphen vergleicht, die gleich sein müssen. Das Ergebnis wird dann an die JUnit-Funktion zurückgegeben. Der Test ist erfolgreich, wenn die Graphen gleich sind.

3.1.2 Test: Pfadgenerierung

Die Pfadgenerierung wird getestet, indem zunächst ein Molekülgraph aus einer HELM-Notation erstellt wird. Die implementierte Funktion **findPaths()**, die getestet werden soll, wird anschließend für den Molekülgraph aufgerufen und damit

zwei Mengen von Pfaden erstellt (originale und natürliche Pfade). Zusätzlich werden alle möglichen (originale und natürliche) Pfade manuell generiert. Dabei werden Duplikate verworfen und bei Palindromen nur das lexikographische Minimum berücksichtigt, wie es auch im eigentlichen Algorithmus der Fall ist. Über die Methode **equals()** werden die beiden Mengen von originalen und natürlichen Pfaden jeweils miteinander verglichen. Bei Gleichheit gibt die JUnit-Klasse Assert keinen Fehler. Der Test würde Abweichung der Pfade fehlschlagen.

3.1.3 Test: Pfadspeicherung

Der Test zur Pfadspeicherung bezieht sich auf die implementierte Funktion **checkAndStorePath()**, in der nur valide Pfade gespeichert werden sollen. Um die Korrektheit der Funktion zu überprüfen, wird ein neuer Pfad zu einer Menge von Pfaden hinzugefügt, welcher ein lexikographisch kleineres Palindrom von einem schon in der Menge existierenden Pfad ist. Der Pfad in der Menge soll gelöscht und stattdessen der neue Pfad hinzugefügt werden. Ist das der Fall, ist der Test erfolgreich. Zusätzlich werden auch noch Negativ-Tests durchgeführt. Diese sollen fehlschlagen, da eine neu erstellte Menge von Pfaden, das einen nicht korrekten Pfad enthält, mit einer vom Programm erstellten korrekten Menge von Pfaden verglichen wird. Die Tests wurden auch mit Pfaden durchgeführt, die in natürliche Analoga umgewandelt werden sollen.

3.2 Laufzeitanalyse

Die Laufzeitanalyse wurde auf einem Rechner mit Intel(R)Core(TM)i5-4690 CPU 3.50 GHz und 16GB RAM durchgeführt.

Die hier angegebenen Laufzeiten sind asymptotische Worst-Case Laufzeiten. In der Funktion **buildMoleculeGraph()** erfolgt eine Iteration über die Liste der Polymere einer HELM-Notation. In jeder Iteration wiederrum wird zwei Mal über die Liste der Monomere iteriert, um die Knoten und die Kanten des Graphen aufzubauen. Danach wird über die Liste der Connections iteriert. Daraus ergibt sich eine Laufzeit von $O(P \cdot 2\overline{M} + C) = O(P\overline{M} + C/2)$. Dabei ist P die Anzahl an Polymeren einer HELM-Notation, $\overline{M}$ die durchschnittliche Anzahl an Monomeren eines Polymers, da jedes Polymer unterschiedlich viele Monomere haben kann, und C die Anzahl von Connections einer HELM-Notation. In der Funktion **findPaths()** wird

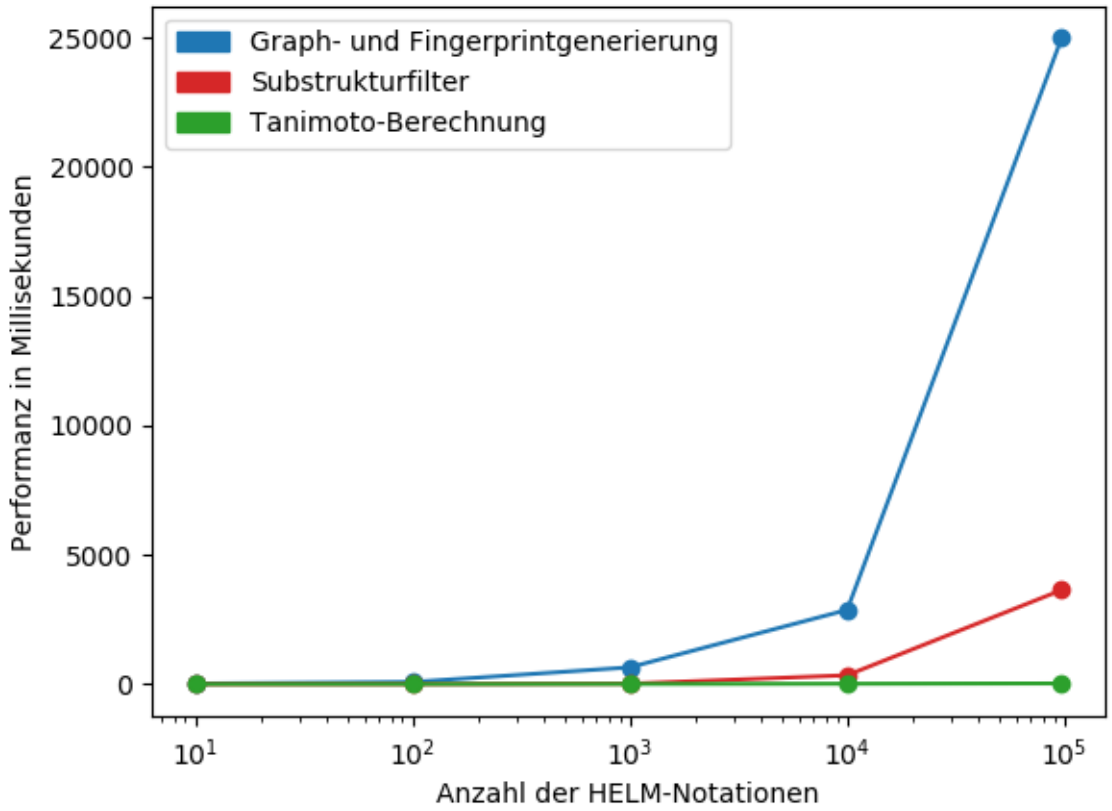


Abbildung 3.1: Laufzeit der Software. Die Laufzeit in Millisekunden ist gegen die Anzahl der HELM-Notationen aufgetragen. Die blaue Kurve gibt die Laufzeit der Graph- und Fingerprintgenerierung an, die rote Kurve die des Substrukturfilters und die grüne Kurve die der Tanimoto-Berechnung.

eine Tiefensuche von jedem Knoten aus durchgeführt. Es wird also einmal über alle Knoten und Kanten des Molekülgraphen iteriert. Dies führt zu einer Laufzeit von $O(P \cdot \overline{M} + P \cdot \overline{M} + C) = O(P\overline{M} + C/2)$. Die Funktion **generateFingerprint()** berechnet für jeden Pfad den SHA-2 und eine Bit-Position. Daraus ergibt sich eine Laufzeit von $O(p \cdot n)$ wobei p die Anzahl von Pfaden ist, über die iteriert wird und n die Länge eines Pfads. Geht man davon aus, dass die Funktionen **calculateSimilarity()** und **isSubset()** jeweils auf einen Fingerprint angewendet werden, der mit einer Menge von anderen Fingerprints verglichen wird, haben beide Funktionen eine Laufzeit von $O(F)$, wobei F die Anzahl an Fingerprints ist. Die Laufzeit ist in Abbildung 3.1 dargestellt. Bei einem Datenset von zehn HELM-Notationen dauert die Generierung der Graphen und der Fingerprints 24 Millisekunden (blaue Kurve). Für ein Datenset von 10.000 dauert die Generierung 2.873 Millisekunden und bei ungefähr

100.000 HELM-Notationen 24.986 Millisekunden. Die Generierung stellt damit den zeitaufwändigsten Teil der Software dar. Die Tanimoto-Berechnung (grüne Kurve) braucht auch bei dem größten getesteten Datenset von rund 100.000 Notationen nur 29 Millisekunden. Auffällig ist die Dauer des Substrukturfilters, die bei einer Datensetgröße von 10.000 Notationen 345 Millisekunden und bei 100.000 Notationen 3.642 Millisekunden dauert. Die Laufzeit des Substrukturfilters ist demnach in der Praxis langsamer als in der Komplexitätsanalyse angenommen. Der Grund dafür liegt bei Implementierungsdetails der Java Klasse BitSet. Ein Vorteil des Verfahrens ist, dass die Fingerprints immer verwendet werden können, sobald sie einmal berechnet und gespeichert wurden. Die eigentliche Ähnlichkeitsberechnung und Anwendung des Substrukturfilters hat dann eine Laufzeit von $O(F)$ und ist linear in der Anzahl der zu vergleichenden Fingerprints F.

4 Ergebnisse

4.1 Pfadgenerierung

Ansatz 1

Beim ersten Ansatz der Pfadgenerierung werden die Pfade aus der HELM-Notation aus den Monomeren erstellt, wie sie ursprünglich in der Notation vorkommen. Die Pfade bestehen somit aus natürlichen und nicht-natürlichen Monomeren. Bei der Berechnung der Ähnlichkeit eines ersten kleinen Testsets bestehend aus zehn Peptiden mit der Länge neun (bezogen auf die Anzahl der Monomere) wurde ein Ausgangspeptid verwendet, welches nur aus natürlichen Monomeren besteht (Abb. 4.1 Peptid mit der ID „0“). Bei jedem weiteren Peptid wurde jeweils ein Monomer durch ein nicht-natürliches Monomer, dessen natürliches Analogon dem Ausgangsmonomer entspricht, ausgetauscht. Das zehnte Peptid mit der ID „9“ besteht damit ausschließlich aus nicht-natürlichen Monomeren. Berechnet wurden die Tanimoto-Werte jeweils zwischen dem ersten Peptid (Abb. 4.1 Notation „0“) und einem der anderen Peptide. Als Ergebnis zeigt sich, wie in Abbildung 4.2 Ansatz 1 zu sehen, mit größerer Anzahl an modifizierten Monomeren ein fallender Tanimoto-Wert. Der Tanimoto zwischen der HELM-Notation „0“ und der gleichen HELM-Notation, also mit keinen modifizierten Monomeren, beträgt 1.0. Das Ergebnis entspricht den Erwartungen, da dabei identische Moleküle miteinander verglichen wurden. Schon bei einem einzigen ausgetauschten Monomer beträgt der Tanimoto nur noch 0.74 und bei zwei ausgetauschten Monomeren 0.53 bis eine Ähnlichkeit bei neun ausgetauschten Monomeren kaum noch vorhanden ist (Tanimoto von 0.00). Diese Ergebnisse führten zu einem anderen Ansatz der Pfadgenerierung.

Ansatz 2

Da es sich bei den modifizierten Monomeren jeweils um ein nicht-natürliches Analogon des Ausgangsmonomer handelt, soll diese Tatsache berücksichtigt werden. Wie in Abschnitt 2.3.3 Ansatz 2 beschrieben, wurden bei der Pfadgenerierung auch

```
0  PEPTIDE{C.R.H.Y.I.N.L.I.T}$$$$
1  PEPTIDE{[dC].R.H.Y.I.N.L.I.T}$$$$
2  PEPTIDE1{[dC].[dR].H.Y.I.N.L.I.T}$$$$
3  PEPTIDE1{[dC].[dR].[meH].Y.I.N.L.I.T}$$$$
4  PEPTIDE1{[dC].[dR].[meH].[meY].I.N.L.I.T}$$$$
5  PEPTIDE1{[dC].[dR].[meH].[meY].[meI].N.L.I.T}$$$$
6  PEPTIDE1{[dC].[dR].[meH].[meY].[meI].[dN].L.I.T}$$$$
7  PEPTIDE1{[dC].[dR].[meH].[meY].[meI].[dN].[Nle].I.T}$$$$
8  PEPTIDE1{[dC].[dR].[meH].[meY].[meI].[dN].[Nle].[meI].T}$$$$
9  PEPTIDE1{[dC].[dR].[meH].[meY].[meI].[dN].[Nle].[meI].[meT]}$$$$
```

Abbildung 4.1: Individuell erstelltes Datenset bestehend aus zehn HELM-Notationen, die sich in modifzierten Monomeren unterscheiden. Links davon eine ID, die gleich der Anzahl an modifizierten Monomeren ist.

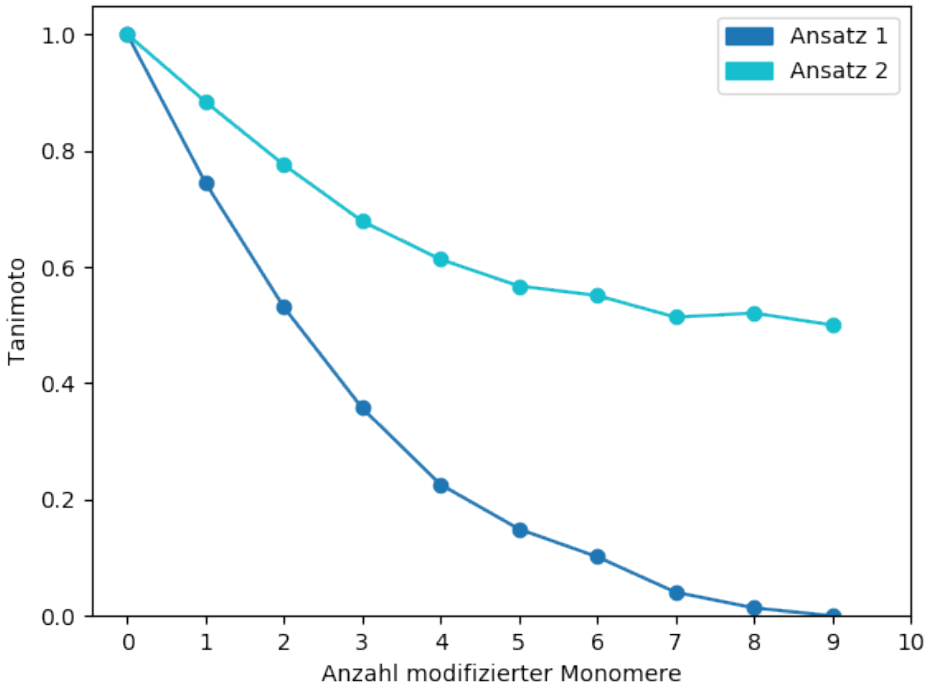


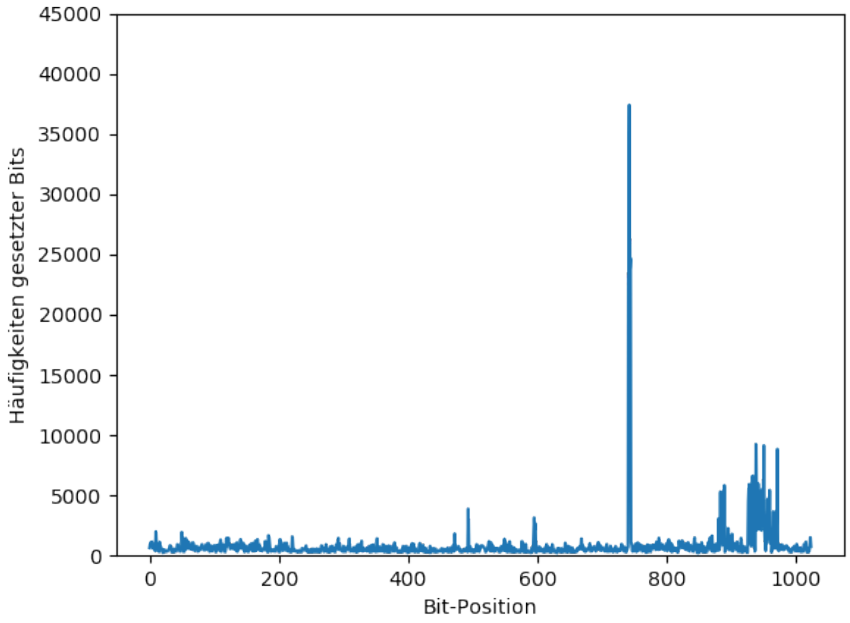
Abbildung 4.2: Auftragung des Tanimoto-Werts gegen die Anzahl an modifizierten Monomeren. Zugrundeliegend ist der Datensatz aus Abb. 4.1. Dabei entspricht die Anzahl der modizifierten Monomere der ID der Notation, die mit Notation „0“ verglichen wurde, also zwischen denen der Tanimoto berechnet wurde. Ansatz 1 ist dunkelblau dargestellt und Ansatz 2 in hellblau.

alle Pfade erstellt, bei denen jedes nicht-natürliche Monomere in sein natürliches Analogon umgewandelt wurde. Die zusätzlichen Pfade wurden in den Fingerprint der HELM-Notation mit einberechnet. Die Ergebnisse des neuen Ansatzes sind in Abbildung 4.2 Ansatz 2 zu sehen. Im Gegensatz zu Ansatz 1 fällt die Kurve mit steigender Anzahl an modifizierten Monomeren deutlich weniger. Der Tanimoto beträgt bei einem ausgetauschten Monomer 0.85 und bei zwei ausgetauschten Monomeren 0.88. Bei allen neun ausgetauschten Monomeren liegt der Tanimoto bei 0.76. Die Ähnlichkeit fällt also bei diesem Ansatz und diesem Datenset nie unter 50 %. Den hier dargestellten Ergebnissen liegt der Fingerprint aus Abschnitt 4.2 Ansatz 2 zu Grunde.

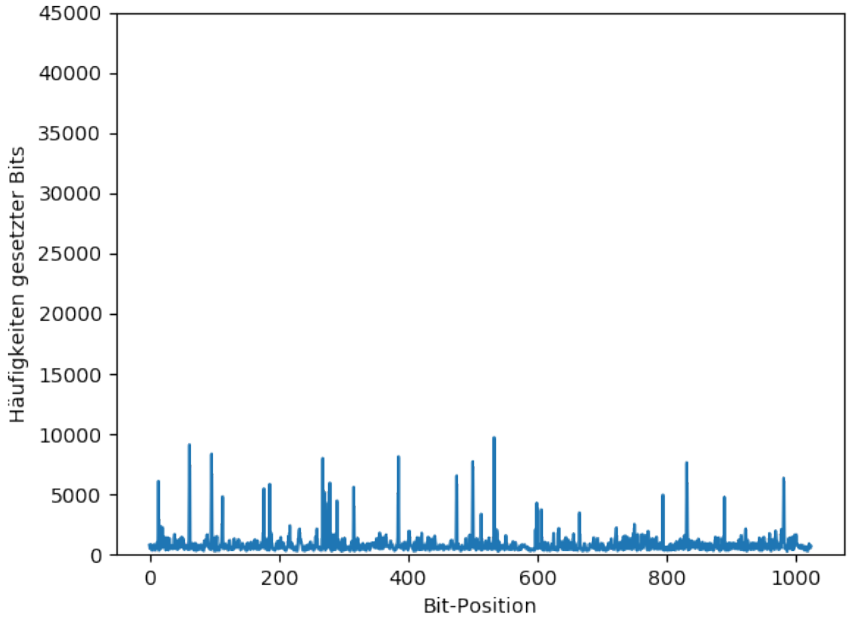
4.2 Generierung der Fingerprints

Ansatz 1

Bei dem ersten Ansatz der Fingerprintgenerierung wird jeder berechnete Pfad der HELM-Notation in einen Hash Code umgewandelt und für diesen eine Pseudozufallszahl zwischen 0 und $N - 1$ generiert, wobei N die Länge des Fingerprints ist. Da die Anzahl der Bit-Positionen begrenzt ist, die Anzahl der möglichen Pfade jedoch nicht, werden mehrere Pfade auf die gleiche Bit-Position abgebildet. Bei der Anwendung dieser Fingerprints wurde ein Testset aus 10.000 zufällig ausgewählten Peptiden aus der PubChem und eine Fingerprintlänge von 1.024 verwendet. Es fiel auf, dass verschiedene Pfade besonders häufig auf die gleiche Bit-Position abgebildet werden. Abbildung 4.3 zeigt die Bit-Positionen zwischen 0 und 1.023 auf der x-Achse und auf der y-Achse die Häufigkeiten der gesetzten Bits an einer bestimmten Position für zwei verschiedene Ansätze der Fingerprintgenerierung. Der erste Ansatz ist in Abbildung 4.3a zeigt. Es gibt einen besonders auffälligen Anstieg zwischen den Bit-Positionen 741 bis 744, bei denen die Häufigkeit dieser gesetzten Bits bis über 37.000 reicht. In einem weiteren Bereich bei den Bit-Positionen 939 bis 972 liegen erhöhte Häufigkeiten von bis über 9.000 der gesetzten Bits vor. In Tabelle 4.1 ist eine Auswahl von Pfaden aus dem selben Testset von 10.000 Peptiden und ihre Häufigkeit, ihr Hash Code und ihre pseudozufällige Bit-Position zu sehen. Betrachtet man die Pfade der Länge eins, so ist erkennbar, dass ihre Hash Codes sehr nah beieinander liegen. Dieses Ergebnis entspricht auf Grund der Berechnung nach Gleichung 2.1 den Erwartungen, da sie auf der UTF-16-Codierung basiert.



(a) Ansatz 1 (Hash Code)



(b) Ansatz 2 (SHA-2)

Abbildung 4.3: Auftragung der Häufigkeiten gesetzter Bits gegen die Bit-Position. Gemeint ist damit, wie häufig die Pfade des zugrunde liegenden Datensets aus 1000 HELM-Notationen aus der PubChem [22] vom Polymertyp PEPTIDE auf eine bestimmte Bit-Position zwischen 0 und 1.023 abgebildet werden.

Tabelle 4.1: Häufigkeit des Vorkommens der Pfade im Datenset, der zugehörige Hash Code nach Gleichung 2.1 und die daraus resultierende Bit-Position nach Ansatz 1 der Fingerprintgenerierung. Auswahl von 14 Pfaden, die im Datenset von 1000 HELM-Notationen vom Polymertyp PEPTIDE aus der PubChem [22] vorkommen.

Pfad	**Pfad-Häufigkeit**	**Hash Code**	**Bit-Position**
A	7850	65	742
C	3624	66	742
D	4753	67	742
E	4375	68	742
F	7102	69	742
G	9080	70	742
H	3780	71	741
I	5181	72	741
K	7081	73	741
L	8469	74	741
M	2705	75	741
N	3912	76	741
AA	1373	2080	933
AAA	564	64545	50

Auffällig ist, dass diese ähnlichen Hash Codes wie zum Beispiel die der Pfade *A*, *C*, *D*, *E* und *F* auf die selbe Bit-Position 742 abgebildet werden. Außerdem kommen Pfade der Länge eins deutlich häufiger vor als Pfade der Länge zwei oder drei wie *AA* und *AAA*. Um auszuschließen, dass die verwendete Random Klasse eine ungleichmäßige Verteilung auf die möglichen Bit-Positionen bewirkt, wurde ein Datenset von Pfaden generiert, das alle möglichen Kombinationen von aus ASCII-Zeichen bestehenden Pfaden ohne Palindrome der Längen eins bis sechs enthält. Für jeden ASCII-Pfad wurde ein Hash Code und die daraus resultierende pseudozufällige Bit-Position berechnet. Wie auch für die Pfade der HELM-Notationen wurden die Häufigkeiten der gesetzten Bits bestimmt. Das Ergebnis zeigt, dass alle möglichen Bit-Positionen gleichmäßig häufig genutzt werden (Abb. 4.4). Die von der Random Klasse erstellte Verteilung auf die Bit-Position ist demnach zwar gleichmäßig, das besonders häufige Setzen von bestimmten Bit-Positionen korreliert somit mit der Pfad-Häufigkeit und den Hash Codes. Der Grund wird in Kapitel 5 genauer erläutert.

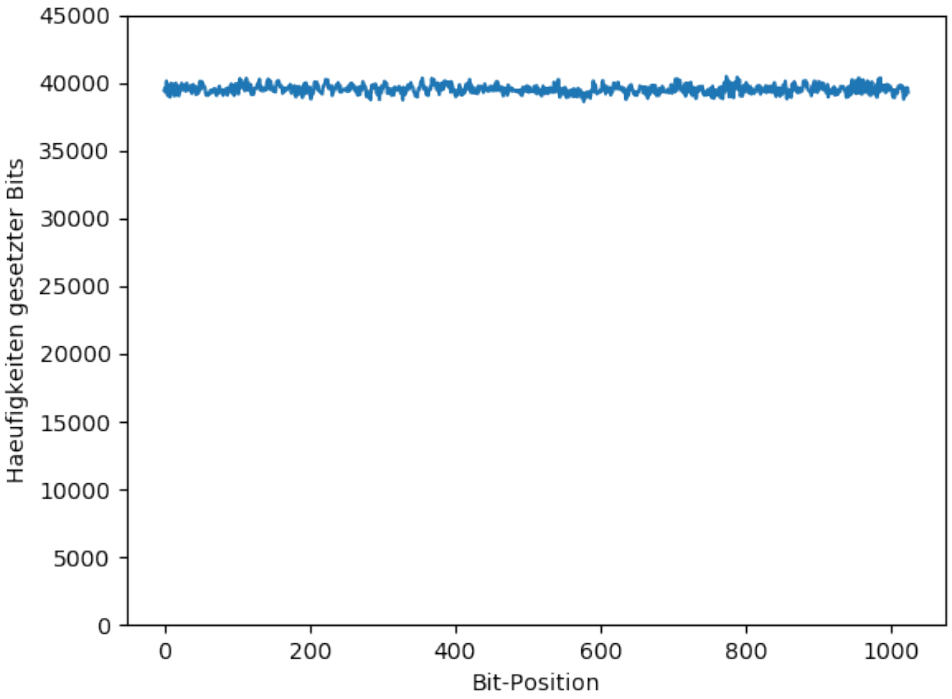


Abbildung 4.4: Auftragung der Häufigkeiten gesetzter Bits gegen die Bit-Position. Gemeint ist damit, wie häufig die Pfade des zugrunde liegenden Datensets bestehend aus allen möglichen ASCII-Pfaden ohne Palindrome der Längen 1 bis 6 auf eine bestimmte Bit-Position zwischen 0 und 1.023 abgebildet werden.

Ansatz 2

Ein zweiter Ansatz, der für die Fingerprintgenerierung genutzt wurde, ist die Berechnung der Bit-Positionen über eine kryptologische Hashfunktion anstelle von Hash Codes und Pseudozufallszahlen. Die Fingerprintberechnung wurde auf das gleiche Testset von 10.000 Peptiden wie in Ansatz 1 angewendet. Die Auswertung, wie häufig eine jeweilige Bit-Position im zweiten Ansatz auf 1 gesetzt wird, ist in Abbildung 4.3b zu erkennen. Es zeigt sich, dass es nur wenige kleine Anstiege gibt, die maximal knapp über einer Häufigkeit von 10.000 liegen. Vergleicht man die Ergebnisse mit denen aus Abbildung 4.3a, so entsteht bei diesem Ansatz der Fingerprintgenerierung eine deutlich gleichmäßigere Verteilung auf die Positionen, zumal die höchste Häufigkeit der gesetzten Bits in Ansatz 1 fast das Vierfache des maximalen Häufigkeitswerts in Ansatz 2 beträgt.

Tabelle 4.2: Resultierende Bit-Position für jeden Pfad nach Ansatz 2 der Fingerprintgenerierung. Gleiche Auswahl von 14 Pfaden wie in Tabelle 4.1, die im Datenset von 1000 HELM-Notationen vom Polymertyp PEPTIDE aus der PubChem [22] vorkommen.

Pfad	Bit-Position
A	96
C	186
D	316
E	177
F	268
G	533
H	599
I	279
K	500
L	62
M	277
N	113
AA	633
AAA	699

Der Grund dafür ist in Tabelle 4.2 zu erkennen. Darin sind die gleichen Pfade aufgelistet wie in Tabelle 4.1, aber mit den zugehörigen Bit-Positionen, die aus dem SHA-2 hervorgehen. Zwischen den Bit-Positionen der einzelnen Pfade gleicher Länge ist kein Zusammenhang zu erkennen, wie es im Gegensatz dazu bei Ansatz 1 der Fall ist.

4.3 Berechnung der Ähnlichkeit

Die Berechnung der Ähnlichkeit erfolgt mit Hilfe des Tanimoto-Koeffizienten. Dafür wird der Tanimoto-Wert zwischen zwei Fingerprints von HELM-Notationen berechnet. Tabelle 4.3 zeigt, wie eine beispielhafte HELM-Notation als Anfrage mit sechs anderen Notationen verglichen wird. Die zu vergleichenden Notationen sind in absteigender Reihenfolge der Ähnlichkeit sortiert und die Teile, die der Anfrage-Notation entsprechen, grün gekennzeichnet. Die erste Vergleichs-Notation ist zu 97 % ähnlich zur Anfrage-Notation. Der Tanimoto-Wert liegt also bei 0.97. Der einzige Unterschied besteht in den letzten drei Monomeren, die zusätzlich am Ende der Vergleichs-Notation vorkommen. Bei der zweiten Vergleichs-Notation sind die gleichen drei Monomere allerdings in der Mitte eingeschoben. Das macht einen Unterschied beim Tanimoto-Wert von 0.2 aus.

Tabelle 4.3: Beispiel für die berechnete Ähnlichkeit zwischen einer Anfrage-HELM-Notation (erste Zeile) und sechs anderen Notationen in den Zeilen darunter. Die Ähnlichkeit ist gerundet in % angegeben; 97 % Ähnlichkeit entspricht einem Tanimoto-Wert von 0.97. Die übereinstimmenden Teile mit der Anfrage-Notation sind grün gekennzeichnet.

RNA1{R(C)P.R(C)P.R(T)P.R(A)P.R(A)P.R(G)P.R(A)P}$$$$	Ähnlichkeit
RNA1{R(C)P.R(C)P.R(T)P.R(A)P.R(A)P.R(G)P.R(A)P.R(C)P}$$$$	97 %
RNA1{R(C)P.R(C)P.R(T)P.R(A)P.R(C)P.R(A)P.R(G)P.R(A)P}$$$$	95 %
RNA1{R([5meC])P.R(G)P.R(G)P.R(A)P.R(C)P.R(C)P.R(C)P}$$$$	68 %
RNA1{R(U)P.R(G)P.R(G)P.R(A)P.R(C)P.R(C)P.R(C)P}$$$$	62 %
RNA1{R(A)P.R(G)P.R(C)}$$$$	62 %
RNA1{[dR](C)P.[dR](C)P.[dR](T)P.[dR](A)P.[dR](A)P.[dR](G)P}$$$$	40 %

Das Ergebnis entspricht den Erwartungen, da andere Pfade entstehen, je nachdem an welcher Stelle der Unterschied ist. Auch die weiteren Ähnlichkeitswerte entsprechen den Erwartungen. Sie werden niedriger mit zunehmend unterschiedlichen Monomeren und Monomeranordnungen.

4.4 Berechnung von Teilmengen

Für die Berechnung von Teilmengen wurden die Fingerprints von Abschnitt 2.3.4 Ansatz 2 verwendet. Ist eine Anfrage-HELM-Notation gegeben, so kann man durch Anwendung des Substrukturfilters überprüfen, ob diese eine Teilmenge von einer oder mehreren anderen Notationen ist. In Tabelle 4.4 ist ein Beispiel für den Substrukturfilter gezeigt. *PEPTIDE1{Y.G.G.F.M.R.F}$$$$* ist die Anfrage-Notation. In diesem Fall ist sie eine Teilmenge von den drei darunter angegeben Notationen. Die übereinstimmenden Teile sind grün gekennzeichnet. Zudem ist die Ähnlichkeit der Notationen angegeben. Die erste Vergleichs-Notation stimmt vollständig mit der Anfrage-Notation überein und hat daher eine Ähnlichkeit von 100 %. Bei der zweiten Notationen ist die Anfrage-Notation in der Mitte als Teilmenge vorhanden und in der letzten Notation am Ende. Die Vergleichs-Notationen sind auch hier nach Ähnlichkeit sortiert.

Tabelle 4.4: Beispiel für die Verwendung des Substrukturfilters. Die Anfrage-HELM-Notation (erste Zeile) ist eine Teilmenge von den drei darunter angegebenen HELM-Notationen. Zusätzlich ist die Ähnlichkeit gerundet in % angegeben; 100 % Ähnlichkeit entspricht einem Tanimoto-Wert von 1.0. Die übereinstimmenden Teile mit der Anfrage-Notation sind grün gekennzeichnet.

PEPTIDE1{Y.G.G.F.M.R.F}$$$$	Ähnlichkeit
PEPTIDE1{Y.G.G.F.M.R.F}$$$$	100 %
PEPTIDE1{M.L.Y.G.G.F.M.R.F.I}$$$$	61 %
PEPTIDE1{E.I.G.V.Y.G.L.P.Y.G.G.F.M.R.F}$$$$	38 %

5 Diskussion

Ziel dieser Arbeit war es, eine HELM-Suche zu implementieren. Dazu gehörte die Entwicklung von Algorithmen, um in HELM-Notationen, wie beim virtuellen Screening, Ähnlichkeiten zu berechnen und sie zu sortieren. Außerdem sollten Substrukturen in HELM-Notationen herausgefiltert werden.

Zunächst soll die Ähnlichkeitssuche betrachtet werden. Wie in Abschnitt 2.3.1 beschrieben, werden HELM Objekte mit Hilfe des HELMNotationToolkit erstellt. Danach erfolgt die Erstellung eines Graphen zur Repräsentation der HELM-Notation. Da der Graph ungerichtet ist, gibt es keine Kennzeichung der Richtung des Moleküls bezogen auf das 3'- und 5'-Ende von RNA beziehungsweise den N– und C-Terminus von Peptiden. Zudem wird beim Hinzufügen von Kanten des Graphen nicht zwischen den unterschiedlichen Bindungsstellen (wie R1 oder R2) von Monomeren unterschieden.

Der Algorithmus wird mit zwei verschiedenen Ansätzen der Generierung der Pfade fortgesetzt. Im ersten Ansatz der Pfadgenerierung werden nur Pfade basierend auf den original in der HELM-Notation vorkommenenden Monomeren erstellt. Dabei stellte sich heraus, dass die Ähnlichkeit zwischen zwei Notationen stark abnimmt, wenn sie sich zunehmend in modifizierten Monomeren unterscheiden (Abb. 4.2 Ansatz 1). Der Grund dafür liegt bei dem Vergleich der Pfade, denn der Algorithmus berücksichtigt nicht, dass zwei Pfade der Länge eins, wie etwa *C* und *[dC]*, das gleiche natürliche Analogon haben. Die Hashfunktion berechnet unterschiedliche Hashwerte für die beiden Pfade, was wiederrum zu unterschiedlichen Bit-Positionen im Fingerprint führt. Vergleicht man nur diese Pfade, haben sie einen Tanimoto-Wert von 0. Das ist für eine Ähnlichkeitssuche in HELM-Notationen nicht von Vorteil, wenn man von verschiedenen Anwendungsfällen ausgeht. Möglicherweise ist es gewünscht, dass Notationen gefunden werden, dessen Monomere nur Modifizierungen zur Anfrage-HELM-Notation aufweisen. Diese unterscheiden sich zwar von der Anfrage-Notation, jedoch ähneln sie sich insofern, dass sie das glei-

che natürliche Analogon teilen. Im zweiten Ansatz der Pfadgenerierung wird die Berücksichtigung der natürlichen Analoga umgesetzt. Betrachtet man die Beispielpfade von oben, so haben diese nun einen Tanimoto-Koeffzient von 0.5. Trotzdem besteht für mögliche Anwendungen, in denen die exakten Monomere ohne Abweichungen in Form von Modifizierungen gewünscht sind, weiterhin die Möglichkeit, die natürlichen Analoga von der Generierung des Fingerprints auszuschließen. Ein Problem, das bei dem zweiten Ansatz auftreten kann, ist die fehlende Angabe des natürlichen Analogons eines nicht-natürlichen Monomers. In einem solchen Fall wird wieder auf das originale Monomer zurückgegriffen und dieses in den natürlichen Pfad eingefügt. So kann jedoch die Pfadgenerierung nach Ansatz 2 nicht gewährleistet werden. An dieser Stelle ist anzumerken, dass die HELM-Grammatik nicht eindeutig definiert ist [27]. Unter anderem ist die ID für ein Monomer nicht eindeutig, sie muss lediglich innerhalb eines simplen Polymers eindeutig sein [46]. Zudem können Verwender von HELM auch eigene Monomere für ihre Monomerdatenbank definieren, bei denen das natürliche Analogon nicht angegeben wird oder undefiniert sein kann. In Abschnitt 2.3.3 wird beschrieben, dass die IDs natürlicher Monomere vom Polymertyp RNA durch Kleinbuchstaben ersetzt werden. Es kann zu weiteren Überschneidungen von identischen IDs in verschiedenen Polymertypen kommen.

Nach der Pfadgenerierung wurden zwei Ansätze für die Generierung von Fingerprints untersucht. Sie unterscheiden sich in der verwendeten Hashfunktion. Der erste Ansatz verwendet den Java Hash Code. Dieser eignet sich nicht für die Fingerprintgenerierung. Wie in den Ergebnissen unter Abschnitt 4.2 Ansatz 1 geschildert, bildet die Hashfunktion zu viele unterschiedliche Pfade auf die gleiche Bit-Position ab. Der Grund dafür liegt in der Berechnung des Hash Codes für String-Objekte nach Gleichung 2.1. Da die Berechnung auf den UTF-16-Codes der Zeichen basiert [24], erhalten Pfade der gleichen Länge ähnliche Hash Codes, die sich nur minimal unterscheiden (siehe Tabelle 4.1). Die Random Klasse wiederrum muss für viele unterschiedliche Hash Codes pseudozufällige Zahlen eines bestimmten Intervalls generieren. Somit werden mehrere ähnliche Hash Codes auf die gleiche Bit-Position abgebildet. Das Problem wird dadurch verstärkt, dass in jeder HELM-Notation am häufigsten Pfade mit der Länge eins vorkommen, von denen sowieso schon viele die gleiche Bit-Position erhalten. Bemerkbar wird das, indem Kollisionen auftreten, weil zu viele unterschiedliche Pfade auf die gleiche Bit-Position abgebildet werden. Im Fingerprint einer einzigen HELM-Notation wird zwar nicht erkennbar,

dass bestimmte Bit-Positionen mehrfach auf 1 gesetzt werden (da das Bit auf 1 gesetzt bleibt, sobald es einmal gesetzt wurde). Jedoch handelt es sich stets um wenige gleiche Bit-Positionen, wie etwa 742, die extrem häufig gesetzt werden, sodass zwei HELM-Notationen einen höheren Ähnlichkeitswert erhalten können, als sie eigentlich erhalten dürften. Es handelt sich dann um eine Pseudoähnlichkeit. Kollisionen in Hashfunktion sind eine bekannte Problematik, weshalb es eine Reihe von Literatur über kollisionsfreie Hashfunktionen (Injektivität) gibt [11] [10]. Eine Funktion ist injektiv, wenn es zu jedem Element aus der Zielmenge höchstens ein Element aus der Ausgangsmenge gibt [45]. Das setzt allerdings voraus, dass die Zielmenge nicht kleiner sein darf, als die Ausgangsmenge. Diese Voraussetzung ist im vorliegenden Algorithmus jedoch nicht gegeben . Der zweite Ansatz der Fingerprintgenerierung schwächt das Problem insofern ab, als dass die Hashwerte von ähnlichen, kurzen Pfaden keinen Zusammenhang haben und sich somit nicht ähneln, da eine kryptographische Hashfunktion verwendet wird. Die Funktion SHA-2 gehört zum neuen Standard mächtiger Hashfunktionen [39] und entfernt einen Teil der auftretenden Pseudoähnlichkeiten.

Im Laufe der Arbeit wurde eine kurze Umfrage mit einem Kunden von quattro research, der HELM verwendet, durchgeführt. Dabei ging es um die Ähnlichkeit von HELM-Notationen. Es sollte vor allem Aufschluss darüber geben, ob die von dem entwickelten Verfahren berechneten Ähnlichkeiten den Erwartungen entsprechen und für die Praxis geeignet sind. Die Umfrage ist nicht zur Veröffentlichung bestimmt, es ist jedoch zu sagen, dass die Ergebnisse größtenteils den Erwartungen entsprechen. Aufgefallen ist allerdings, dass das entwickelte Verfahren bei der Berechnung der Ähnlichkeit von RNA limitiert ist. Das liegt vor allem an dem Grundgerüst aus Zuckern und Phosphaten, das zu repetitiven Pfaden und zu einer Art Hauptpfaden, die häufig vorkommen, führt. Mit repetitiven Pfaden ist gemeint, dass die Pfade des Grundgerüsts immer aus R und P zusammengesetzt sind, vorausgesetzt es handelt sich um natürliche Monomere. Da die Pfade nur bis zu einer Länge von sechs erstellt werden, würden bei einer HELM-Notation mit mehr als sechs Grundgerüst-Monomeren bestehend aus R und P die weiteren Pfade nicht bei der Ähnlichkeit einberechnet werden, da sich die Pfade wiederholen. Es besteht die Möglichkeit, die Pfadlänge zu vergrößern, um mehr Monomere einer Notation abzudecken, das führt jedoch zu mehr Kombinationsmöglichkeiten von Pfaden und somit zu mehr Bit-Positionen, die von den zusätzlichen Pfaden eingenommen werden. Somit bilden mehr Pfade auf die gleiche Bit-Position ab und es kann

Tabelle 5.1: Beispiel für die Berechnung von HELM-Ähnlichkeit in RNA. Die erste Vergleichs-Notation hat verglichen mit der oben stehenden Anfrage-Notation zwei ausgetauschte Basen und ist mit 58 % ähnlicher als die zweite Vergleichs-Notation mit 31 %. Diese hat keine Unterschiede in den Basen, sondern zwei ausgetauschte Zucker-Monomere. Die ausgetauschten Monomere sind rot gekennzeichnet. Das Beispiel wurde aus einer Kundenumfrage entnommen und verändert.

RNA1{R(G)P.R(A)P.R(T)P.R(T)P.R(T)P}$$$$	Ähnlichkeit
RNA1{R(C)P.R(G)P.R(T)P.R(T)P.R(T)P}$$$$	58 %
RNA1{[LR](G)P.[dR](A)P.R(T)P.R(T)P.R(T)P}$$$$	49 %

zu Pseudoähnlichkeiten kommen. Daraus resultierend könnte die Fingerprintlänge vergrößert werden, um mehr Bit-Positionen zu schaffen. Das Problem dabei ist allerdings, dass die Fingerprints kurzer HELM-Notationen wenige Bits auf 1 gesetzt haben. Da der Tanimoto-Koeffizient nicht die Abwesenheit von Eigenschaften einkalkuliert, erhalten kleinere Moleküle geringe Ähnlichkeitswerte [44]. Aus diesem Grund wurde die übliche Pfadlänge von sechs und eine Fingerprintlänge von 1.024 beibehalten [12] [23]. Die Problematik der Hauptpfade soll anhand des Beispiels in Tabelle 5.1, das dem Ergebnis der Kundenumfrage ähnelt, erläutert werden. Gegeben ist *RNA1{R(G)P.R(A)P.R(T)P.R(T)P.R(T)P}$$$$* als Anfrage-HELM-Notation. In der ersten Vergleichs-Notation sind die ersten beiden Basen-Monomere durch andere Basen ausgetauscht. Die zweite Vergleichs-Notation unterscheidet sich in den ersten beiden Zuckern, welche modifiziert sind. Der Austausch der Basen macht einen geringeren Unterschied in der Ähnlichkeit als die Modifizierung des Zuckers, obwohl die befragten Kunden das gegenteilige Ergebnis erwartet hätten. Die Modifizierung des Zuckers bewirkt eine Änderung der Pfade des Grundgerüstes. Deshalb entstehen mehr unterschiedliche Pfade und somit Bit-Positionen als bei der ersten Vergleichs-Notation, bei der Seitenketten-Monomere verändert wurden.

Bei dem Substrukturfilter können ähnliche Probleme wie bei der Ähnlichkeitssuche auftreten. Durch die limitierte Anzahl von Bit-Positionen und nicht-limitierte Anzahl von Pfaden, werden unterschiedliche Pfade auf die selbe Bit-Position abgebildet. Vergleicht man die Fingerprints zweier HELM-Notationen miteinander, ist nicht gewährleistet, dass die Anfrage-HELM-Notation tatsächlich exakt in einer Vergleichs-Notation enthalten ist. Im Gegensatz zur Ähnlichkeitssuche sind Pseudoähnlicheiten bei dem Substrukturfilter jedoch in der Praxis bei keinem Test vorgekommen.

6 Fazit und weiterführende Arbeit

In dieser Arbeit wurde ein erster Ansatz für die Ähnlichkeitssuche und Substrukturfilter in HELM-Notationen vorgestellt. Das gesteckte Ziel wurde erreicht, indem die implementierten Algorithmen eine Möglichkeit bieten, um komplexe Moleküle, die mit HELM repräsentiert werden, zu vergleichen. Insgesamt bestand die größte Schwierigkeit darin, einen Rahmen für die Ähnlichkeit von HELM-Notationen zu finden. Die Definition der Ähnlichkeit benötigt einerseits weitere wissenschaftliche Betrachtung und ist außerdem von der Anwendung abhängig.
Mit verschiedenen Tests wurde das Verfahren validiert und die Laufzeit analysiert. Vorteilhaft ist, dass die Verarbeitung der HELM-Notationen zu Fingerprints nur einmal erfolgen muss. Die gespeicherten Fingerprints können dann für die Ähnlichkeitsberechnung mit dem Tanimoto-Koeffizienten und den Substrukturfilter genutzt werden.
Die Methode unterstützt im Gegensatz zur BLAST-Suche auch Makromoleküle, welche aus verschiedenen Polymertypen zusammengesetzt sind. Vergleicht man die HELM-Suche mit der chemischen Ähnlichkeitssuche auf der Basis von Atomen, so ist zu sagen, dass die HELM-Suche den Ähnlichkeitsbereich zwischen 0 % und 100 % besser ausnutzt. Möchte man beispielsweise Moleküle vergleichen, die sich lediglich in Modifikationen einzelner Bausteine (wie Aminosäuren oder Nukleotidteile) unterscheiden oder die eine andere Anordnung der Bausteine im Molekül aufweisen, so fällt die berechnete Ähnlichkeit bei der chemischen Ähnlichkeitssuche höher aus als bei der HELM-Suche. In Abbildung 6.1 sind die Tanimoto-Werte von Molekülen, repräsentiert durch SMILES, gegen die Tanimoto-Werte von den gleichen Molekülen, repräsentiert durch HELM, aufgetragen. Dabei wurde ein Datenset von 50 Nukleotidketten verwendet, bei der alle Moleküle untereinander verglichen wurde. Die Tanimoto-Koeffizienten basierend auf SMILES liegen im Bereich 0.7 bis 1.0, während die Werte basierend auf HELM zwischen 0.2 und 1.0 liegen. Daran ist zu erkennen, dass sich alle Moleküle auf der Atomebene sehr ähnlich sind, während die HELM-Ähnlichkeit stärker differenziert.

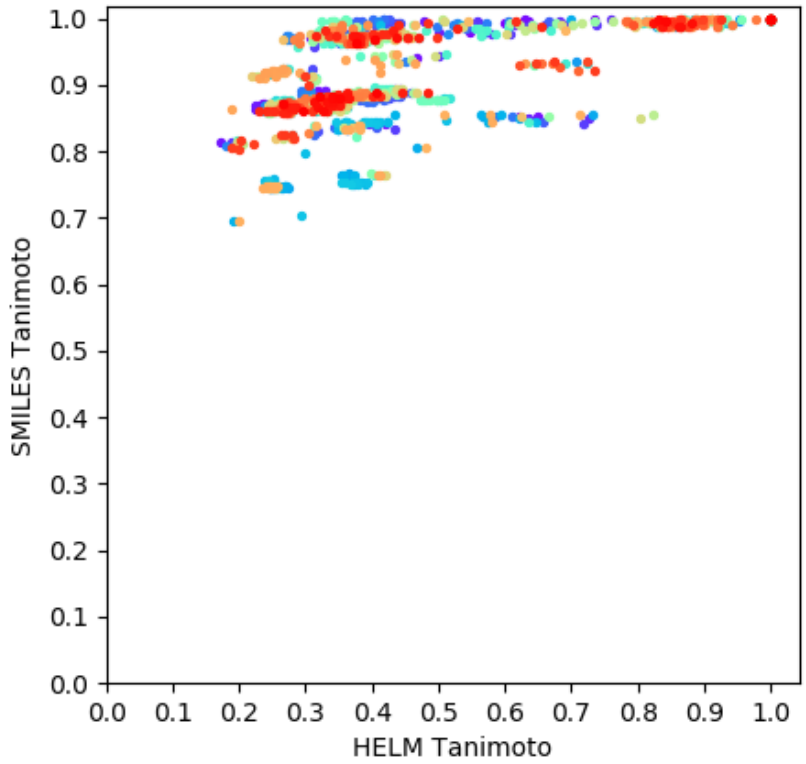


Abbildung 6.1: Auftragung der Tanimoto-Werte - SMILES gegen HELM. Es ist zu sehen, dass bei SMILES der Tanimoto-Wert im Bereich 0.7 bis 1.0 liegt, während der Tanimoto-Wert bei HELM zwischen 0.2 und 1.0. Zugrunde liegt ein Datenset aus 50 Nukleotidketten, bei dem jedes Molekül mit jedem verglichen wurde.

An einigen Stellen kann die Software noch verbessert und erweitert werden. Ein Aspekt zur Verbesserung der Laufzeit ist die Änderung der Datenstruktur für den Fingerprint. Das aktuell verwendete Java BitSet hat eine schwache Laufzeit und könnte durch eine leistungsstärkere Datenstruktur ersetzt werden. Der Substrukturfilter könnte durch eine echte Substruktursuche ersetzt werden. Damit wäre das exakte Suchen der gesamten Anfrage-HELM-Notation in den Vergleichs-Notationen gewährleistet. Als Erweiterung kann die Unterstützung der in Kapitel 1 erwähnten in-line SMILES-Notation innerhalb einer HELM-Notationen entwickelt werden. Außerdem kann der ungerichtete Graph durch einen gerichteten Graphen ersetzt werden, um die Molekülrichtung anzuzeigen. Diese genannten Erweiterungen sind voraussichtlich unproblematisch umsetzbar. Eine größere Herausforderung stellt die Unterstützung von Ambiguität in HELM dar. Seit HELM 2.0 können auch Makromoleküle, welche undefinierte Teile beinhalten, dargestellt werden. Die Ambi-

guität kann auf allen vier Strukturhierarchieebenen auftreten. Undefinierte Teile von einer Notation in die Berechnung von Ähnlichkeit mit einzubeziehen ist jedoch problematisch und bedarf einer gründlichen Analyse. Die für die Software entwickelte graphische Benutzeroberfläche bietet grundlegende Funktionen, um die Bedienung einer Ähnlichkeitssuche und eines Substrukturfilters zu erleichtern. Weiterführend könnte die Benutzeroberfläche um eine Hervorherbung der ähnlichen oder übereinstimmenden Teile in den Vergleichs-Notationen erweitert werden. Damit werden diese Teile sofort für den Nutzer ersichtlich. Die implementierte Software geht im nächsten Schritt in eine Testphase des Kunden.
Zusammenfassend ist zu sagen, dass die vorgestellten Algorithmen eine geeignete Methode darstellen, um in HELM-Notationen zu suchen. Verglichen mit anderen Methoden der Bestimmung von Ähnlichkeit, erlaubt die HELM-Suche die Verarbeitung von komplexen Molekülen, die aus verschiedenen Polymertypen bestehen können. Zudem erfolgt die Ähnlichkeitsberechnung auf einer höheren Strukturhierarchieebene als die der Atome. Daher wird ein aussagekräftigeres Ergebnis der Ähnlichkeit für Moleküle ermöglicht, die beispielsweise dasselbe Grundgerüst von Atomen aufweisen, sich aber in chemischen Modifikationen unterscheiden.

A Software User Guide

Das Programm wird über die Datei **HELMSimilarityLibrary.jar** aufgerufen. Es öffnet sich die graphische Benutzeroberfläche der HELMSimilarityLibrary.

A.1 Input

Erforderlich ist eine Textdatei (Dateieendung: .txt) mit HELM-Notationen. Die Datei darf zeilenweise entweder ausschließlich HELM-Notationen oder eine pro Notation eindeutige ID mit der zugehörigen Notation beinhalten, die durch einen Tab von der ID getrennt sein muss.

```
Beispiel Format 1:
PEPTIDE1{[dY].G.G.F.L.[dR]}$$$$
PEPTIDE1{G.R.G.[dD].[dS].[dP].[dK]}$$$$
PEPTIDE1{G.R.G.[dD].[dS].P.[dK]}$$$$
PEPTIDE1{[dY].A.[dF].[dD].V.[dV].G.[am]}$$$$
PEPTIDE1{[dY].G.G.F.L.R}$$$$

Beispiel Format 2:
129606642    PEPTIDE1{[dY].G.G.F.L.[dR]}$$$$
129606680    PEPTIDE1{G.R.G.[dD].[dS].[dP].[dK]}$$$$
129606681    PEPTIDE1{G.R.G.[dD].[dS].P.[dK]}$$$$
129606684    PEPTIDE1{[dY].A.[dF].[dD].V.[dV].G.[am]}$$$$
129606641    PEPTIDE1{[dY].G.G.F.L.R}$$$$
```

Bei der Wahl von Format 1, also der HELM-Notationen ohne ID, wird für jede Notation eine eindeutige ID automatisch erstellt, um eine exakte Zuordnung der Notationen zu gewährleisten. Über die Schaltfläche **Build database** (siehe Abb. 2.6) wird die Textdatei verarbeitet und die weiteren Funktionen werden zur Benutzung freigegeben. Ein Fortschrittsbalken weist darauf hin, wenn der Vorgang beendet wurde.

A.2 Optionen

Es ist zwischen den beiden Optionen **Similarity search** für Ähnlichkeitssuche und **Substructure filter** für Substrukturfilter zu wählen. Bei der Ähnlichkeitssuche kann eine Minimalähnlichkeit zwischen 0 % und 100 % angegeben werden. Damit werden alle Notationen zurückgegeben, die mindestens eine Ähnlichkeit entsprechend der Minimalähnlichkeit zur Anfrage-HELM-Notation aufweisen. Die andere Möglichkeit ist die Anzeige der zehn ähnlichsten Notationen, die gefunden wurden, unabhängig von einem Schwellenwert für die Ähnlichkeit. Für beide Möglichkeiten der Ähnlichkeitssuche gibt es optional ein Kontrollkästchen, mit dem angegeben werden kann, ob die Umwandlung von nicht-natürlichen Monomeren in ihre natürlichen Analoga bei der Berechnung der Ähnlichkeit berücksichtigt werden soll. Der Substrukturfilter gibt alle HELM-Notationen zurück, die die Anfrage-HELM-Notation als Teilmenge enthalten. Dabei ist zu beachten, dass hier keine echte Substruktursuche vorliegt, sondern lediglich diejenigen Notationen herausgefiltert werden, deren Fingerprints den Fingerprint der Anfrage-Notation vollständig enthalten. Außerdem erforderlich ist die Eingabe der Anfrage-HELM-Notation in das dafür vorgesehene Textfeld, bevor eine Programmfunktion mit der Schaltfläche **Run query** ausgeführt werden kann.

A.3 Ergebnisse

Der Ergebnisteil besteht hauptsächlich aus einer Tabelle mit den Spalten ID, HELM und Ähnlichkeit in %. Ein Fortschrittsbalken gibt den Status des Programmladevorgangs an. In der Tabelle werden die gefundenen HELM-Notationen aufgelistet und nach Ähnlichkeit in absteigender Reihenfolge sortiert. Die Reihenfolge kann manuell durch Klicken auf den jeweiligen Spaltennamen geändert werden. Für die Ergebnisse steht das Kontextmenü „Kopieren“ zur Verfügung, mit dem jeweils eine Zelle der Tabelle kopiert werden kann. Es besteht außerdem die Möglichkeit über die Schaltfläche **Export results to textfile**, alle Ergebnisse in eine Textdatei zu exportieren. Die Ausgabedatei erhält den gleichen Namen wie die Input-Datei mit dem Suffix „_results.txt“.

A.4 Problembehandlung

Tabelle A.1: Mögliche Probleme bei der Verwendung der Software.

Problem	**Lösung**
Ladebalken der Input-Sektion eingefroren	Überprüfen Sie das Format der Eingabedatei und stellen Sie sicher, dass die HELM-Notationen keine Syntax-Fehler aufweisen. Starten sie das Programm erneut mit der korrekten Eingabedatei.
Falsche Textdatei geladen	Wählen sie die korrekte Textdatei über die Auswahl der Schaltfläche „Choose file“.
Datenbank der falschen Textdatei erstellt	Starten sie das Programm erneut mit der korrekten Eingabedatei. Löschen sie gegebenenfalls die fälschlicherweise erstellte SQLite Datenbank (Dateiendung .db).
Ladebalken der Ergebnis-Sektion eingefroren	Überprüfen sie die Validität der HELM-Notationen ihrer Eingabedatei und starten sie dann das Programm erneut.
Fehlermeldung: Invalid HELM-Notation	Überprüfen sie die Eingabe ihrer Anfrage-HELM-Notation (*Query HELM notation*). Überprüfen sie die Validität der HELM-Notationen ihrer Eingabedatei und starten sie dann das Programm erneut.

B Software Developer Guide

B.1 Wichtige Klassen und Funktionen

Klasse	**Funktionen**
HELM2Object	*makeHELM2NotationObject()*
MoleculeGraph	*buildMoleculeGraph()*
	buildRNAPart()
	buildPeptidePart()
	buildChemPart()
	addConnections()
PathGenerator	*findPaths()*
	depthFirstSearch()
	checkAndStorePath()
	getNaturalPeptide()
	getNaturalRNA()
	getPaths()
	getNaturalPaths()
Fingerprinter	*getHashedFingerprint()*
	getBitposFromHashcode()
	foldBitSet()
	getDecimalFromBitSet()
Tanimoto	*calculateSimilarity()*
Subset	*isSubset()*

B.2 APIs

Um eine Ähnlichkeitssuche oder Substrukturfilter mit HELM-Notationen durchzuführen, müssen folgende Funktionen aufgerufen werden.

`Graph buildMoleculeGraph(List<PolymerNotation> P, List<ConnectionNotation> C);`
Die Funktion der Klasse MoleculeGraph baut einen Graphen mit einer Liste von Knoten (Monomere) und Kanten (Bindungen) zwischen den Knoten auf. Als Übergabeparameter muss die Liste der Polymere und die Liste der Verbindungen einer HELM-Notation übergeben werden.

`void findPaths(Graph moleculeGraph);`
Diese Funktion der Klasse PathGenerator generiert alle Pfade des Molekülgraphen ausgehend von jedem Monomer bis zu einer Maximallänge von 6. Die Pfade werden durch eine Tiefensuche ermittelt.

`BitSet getHashedFingerprint(Set<String> totalPaths);`
Die Funktion der Klasse Fingerprinter generiert einen Hashed Fingerprint aus einer Menge von Pfaden. Dabei wird die kryptographische Hashfunktion SHA-2 verwendet.

`double calculateSimilarity(BitSet fingerprint1, BitSet fingerprint2);`
Die Funktion der Klasse Tanimoto berechnet den Tanimoto-Wert von zwei Fingerprints, die als Parameter übergeben werden.

`boolean isSubset(BitSet fingerprint1, BitSet fingerprint2);`
Die Funktion der Klasse Subset überprüft, ob *fingerprint1* eine Teilmenge von *fingerprint2* ist und gibt entsprechend *true* oder *false* zurück.

B.3 Softwarearchitektur

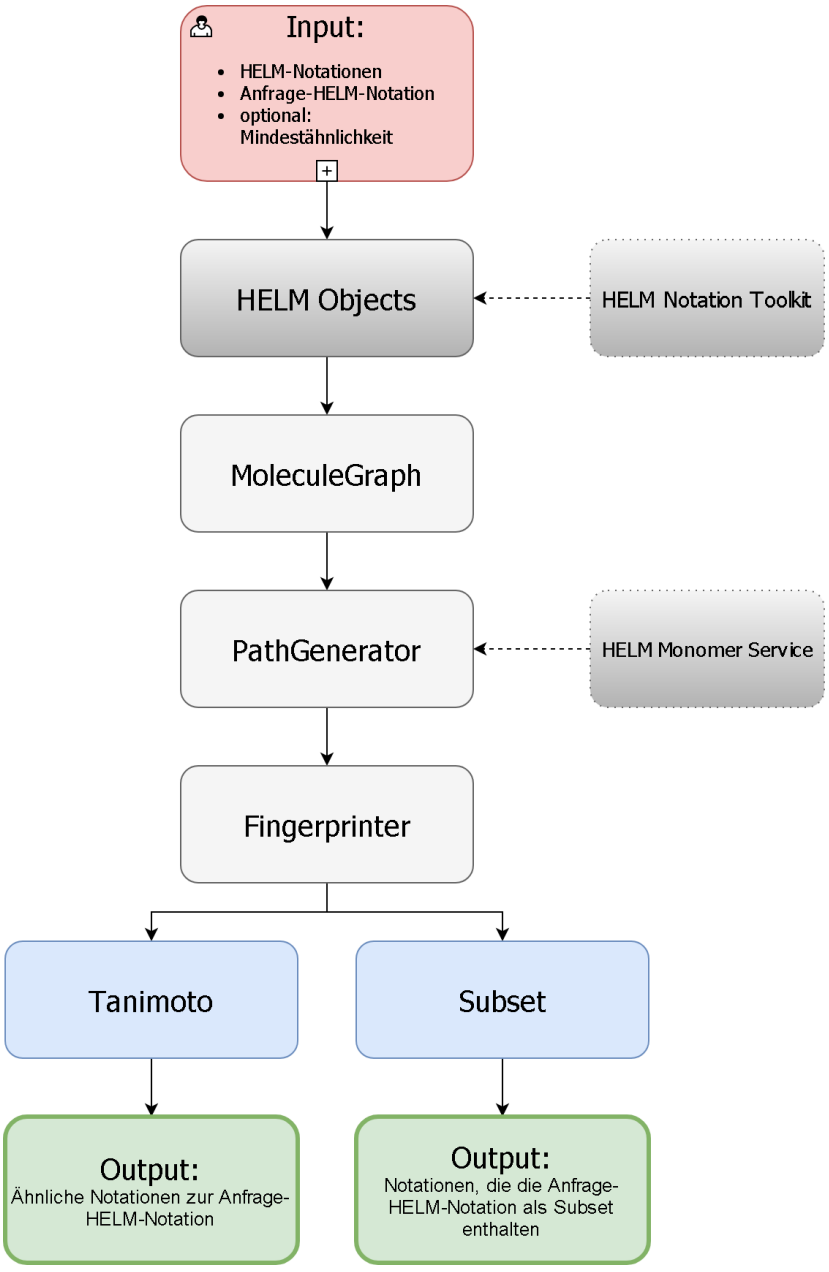


Abbildung B.1: Schematische Darstellung der Softwarearchitektur. Der Input besteht aus einer Anfrage-HELM-Notation und einer Textdatei mit Vergleichs-HELM-Notationen. Optional ist je nach Anwendungsart die Angabe einer Mindestähnlichkeit. Die Pfeile zeigen den Ablauf des Programms an. Grau dargestellt ist die Verarbeitung der HELM-Notationen zu den Fingerprints. Dabei sind die Teile dunkel hinterlegt, die eine bereits existierende Software (gestrichelte Umrandung) verwenden. Der Output hängt davon ab, ob eine Ähnlichkeitssuche (Tanimoto, blau dargestellt) oder der Substrukturfilter (Subset, blau dargestellt) angewendet wurde.

C Genutzte Datensätze

Die für die vorliegende Arbeit genutzten Datensätze stammen aus der Datenbank PubChem [22] oder wurden individuell mit Hilfe des MonomerStores des HELM-NotationToolkits [32] erstellt. Die Datensätze befinden sich auf dem beigefügten Datenträger.

D Implementation

Die gesamte Implementation der Software befindet sich auf dem beigefügten Datenträger.

E Monomer Datenbank des HELMNotationToolkits

Dargestellt ist ein Auszug aus der Monomer Datenbank (MonomerStore), die im Extensible Markup Language (XML)-Format vorliegt. Die Datenbank gehört zum HELMNotationToolkit [32]. Als Beispiele für die Monomere werden Serin und Ribose gezeigt.

```
<?xml version="1.0" encoding="UTF-8"?>
<MonomerDB xmlns="lmr"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<PolymerList>
  <Polymer polymerType="PEPTIDE">
    <Monomer>
      <MonomerID>S</MonomerID>
      <MonomerSmiles>OC[C@H](N[H:1])C([OH:2])=O</MonomerSmiles>
      <MonomerMolFile>H4sIAAAAAAAAAKWSuw7CMAxF93yFJVixHDtp7ZkiJh
        5iYGdkYWDg+0kDKmmpBBQrUqVj9yS5igPYnK638wWA1BspU1RuoCvnAB
        SgTv1ivcrM4MhElOZgwUhioeWMUUKeJExdgiX0FeNrYBGUqFZYdr9Y6i
        j5X49sVSgs2ykWRs8+jN7If7IIhkrl31wCeuW6O0uclotH4WCjlsPsa0
        u+0TMNVbEpFs7J8YAmFAaRPmj9TtOgjNKYqS9pQtX7rOTn3acbgMN6n0
        /Wbpq/2r6jtrPaNs7NU7k7Ruw0UjwDAAA=</MonomerMolFile>
      <MonomerType>Backbone</MonomerType>
      <PolymerType>PEPTIDE</PolymerType>
      <NaturalAnalog>S</NaturalAnalog>
      <MonomerName>Serine</MonomerName>
      <Attachments>
        <Attachment>
          <AttachmentID>R2-OH</AttachmentID>
          <AttachmentLabel>R2</AttachmentLabel>
          <CapGroupName>OH</CapGroupName>
```

```
        <CapGroupSmiles>O[*:2]</CapGroupSmiles>
      </Attachment>
      <Attachment>
        <AttachmentID>R1-H</AttachmentID>
        <AttachmentLabel>R1</AttachmentLabel>
        <CapGroupName>H</CapGroupName>
        <CapGroupSmiles>[*:1][H]</CapGroupSmiles>
      </Attachment>
    </Attachments>
  </Monomer>
</Polymer>
<Polymer polymerType="RNA">
  <Monomer>
    <MonomerID>tR</MonomerID>
    <MonomerSmiles>[H:1]O[C@H]1CO[C@@H]([OH:3])
      [C@@H]1O[H:2]</MonomerSmiles>
    <MonomerMolFile>H4sIAAAAAAAAAJ2TMU8EIRCFe37FJF4rmZmFgam9i9
      WpucLe0sbCwt8vA67Hhk1OmUyy8Pbx7VsCDuD89vn1/gGAkRgVNWY+wm
      85B4TW0Pe1VBVeGRGdzcgHoWQj9pyxjtCXtwjPsEXst2srUkjaeFkSd5
      SHv1PYU5K8UnSfwrezqHBsFEybP7pS6BaFCiWElcI0R7Esy08CDnFydy
      1LlJWSaI5iu9so7BUxdJTL3T+yiEQ7T/foF8zLHIW9ZKRGIYlTWaieBR
      rVZVSLMYxqMcZdNVeVejXseosqo7cYU5WkV8tER0KqN3WrngEujy81iU
      WpT7tc9h3bejbH6eno3KGU+waaCI8kEgQAAA==</MonomerMolFile>
    <MonomerType>Backbone</MonomerType>
    <PolymerType>RNA</PolymerType>
    <NaturalAnalog>R</NaturalAnalog>
    <MonomerName>Ribose</MonomerName>
    <Attachments>
      <Attachment>
        <AttachmentID>R3-OH</AttachmentID>
        <AttachmentLabel>R3</AttachmentLabel>
        <CapGroupName>OH</CapGroupName>
        <CapGroupSmiles>O[*:3]</CapGroupSmiles>
      </Attachment>
      <Attachment>
        <AttachmentID>R1-H</AttachmentID>
        <AttachmentLabel>R1</AttachmentLabel>
        <CapGroupName>H</CapGroupName>
```

```
          <CapGroupSmiles>[*:1][H]</CapGroupSmiles>
        </Attachment>
        <Attachment>
          <AttachmentID>R2-H</AttachmentID>
          <AttachmentLabel>R2</AttachmentLabel>
          <CapGroupName>H</CapGroupName>
          <CapGroupSmiles>[*:2][H]</CapGroupSmiles>
        </Attachment>
      </Attachments>
    </Monomer>
    </Polymer>
</PolymerList>
</MonomerDB>
```

Literaturverzeichnis

[1] *Quattro research GmbH*. http://www.quattro-research.com/, Abruf: 16.03.2018

[2] ALTSCHUL, S. F. ; GISH, W. ; MILLER, W. ; MYERS, E. W. ; LIPMAN, D. J.: Basic Local Alignment Search Tool. In: *Journal of Molecular Biology* 215 (1990), S. 403–410

[3] BAJUSZ, D. ; RÁCZ, A. ; HÉBERGER, K. : Why is Tanimoto index an appropriate choice for fingerprint-based similarity calculations? In: *Journal of Cheminformatics* 7 (2015)

[4] BARNARD, J. M.: Substructure Searching Methods: Old and New. In: *Journal of Chemical Information and Computer Sciences* 33 (1993), S. 532–538

[5] BELLAMY, C. ; BHAMIDIPATI, R. ; KLOSTERMANN, S. ; KNISPEL, R. ; NOLTE, M. ; PAPP, A. ; ZHANG, T. ; WEISSER, M. : *HELM Specification*. 2017

[6] BENDER, A. ; GLEN, R. C.: Molecular similarity: a key technique in molecular informatics. In: *Organic & Biomolecular Chemistry* 22 (2004), S. 3204–3218

[7] CERETO-MASSAGUE, A. ; OJEDA, M. J. ; VALLS, C. ; MULERO, M. ; GARCIA-VALLVE, S. ; PUJADAS, G. : Molecular fingerprint similarity search in virtual screening. In: *Journal of Molecular Biology* 71 (2015), S. 58–63

[8] CHLEBIKOVA, A. : *HELM Search Proof of Concept*. https://pistoiaalliance.atlassian.net/wiki/spaces/PUB/pages/13795373/Search+Proof+of+Concept, Abruf: 23.03.2018

[9] CROCKFORD, D. : *JavaScript Object Notation*. https://tools.ietf.org/html/rfc4627, Abruf: 09.04.2018

[10] DAMGARD, I. : Collision Free Hash Functions and Public Key Signature Schemes. In: *Advances in Cryptology EUROCRYPT 87* (1987), S. 203–216

[11] DAMGARD, I. : A Design Principle for Hash Functions. In: *Advances in Cryptology CRYPTO 89 Proceedings* (1989), S. 416–427

[12] DAYLIGHT: *Daylight Theory Manual*. http://www.daylight.com/dayhtml/doc/theory/theory.finger.html#RTFToC80, Abruf: 25.03.2018

[13] HANCOCK, J. M. ; ZVELEBIL, M. J.: *Dictionary of Bioinformatics and Computational Biology*. Wiley, 2004

[14] JOHNSON, M. A. ; MAGGIORA, G. M.: *Concepts and Applications of Molecular Similarity*. Wiley, 1990

[15] JUNIT: *JUnit 5*. https://junit.org/junit5/, Abruf: 15.03.2018

[16] KRISTENSEN, T. G. ; NIELSEN, J. ; PEDERSEN, C. N. S.: Methods For Similarity-Based Virtual Screening. In: *Computational and Structural Biotechnology Journal* 5 (2013)

[17] KRUMKE, S. O. ; NOLTEMEIER, H. : *Graphentheoretische Konzepte und Algorithmen*. Teubner, 2005

[18] LOUIS, D. ; MÜLLER, P. : *Java - Der umfassende Programmierkurs*. O'Reilly, 2014

[19] MCEVOY, R. P. ; CROWE, F. M. ; MURPHY, C. C. ; MARNANE, W. P.: Optimisation of the SHA-2 Family of Hash Functions on FPGAs. In: *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures* (2006), S. 6

[20] MILTON, J. ; ZHANG, T. ; BELLAMY, C. ; SWAYZE, E. ; HART, C. ; WEISSER, M. ; HECHT, S. ; ROTSTEIN, S. : HELM Software for Biopolymers. In: *Journal Of Chemical Information and Modeling* 57 (2017), S. 1233–1239

[21] NCBIPUBCHEM: *Beispiel des Compound Views*. https://pubchem.ncbi.nlm.nih.gov/rest/pug_view/data/compound/129606597/JSON?heading=Biologic+Line+Notation, Abruf: 28.11.2017

[22] NCBIPUBCHEM: *Compounds with Biologic Line Notation*. https://www.ncbi.nlm.nih.gov/pccompound?DbFrom=pchierarchy&Cmd=Link&Db=pccompound&LinkName=pchierarchy_pccompound&IdsFromResult=1856910, Abruf: 18.09.2017

[23] OPREA, T. I.: *Chemoinformatics in Drug Discovery*. Wiley-VCH, 2015

[24] ORACLE: *Java 8 Documentation: Class String*. https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#hashCode--, Abruf: 14.03.2018

[25] ORACLE: *Java SE Development Kit 8*. http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html, Abruf: 13.03.2018

[26] ORACLE: *The Java Language Specification, Java SE 8 Edition*. 2015

[27] PISTOIAALLIANCE: *HELM Challenges*. https://pistoiaalliance.atlassian.net/wiki/spaces/PUB/pages/20512783/HELM+Challenges, Abruf: 23.03.2018

[28] PISTOIAALLIANCE: *HELM Editor v1.4*. https://github.com/PistoiaHELM/HELMEditor/releases, Abruf: 12.03.2018

[29] PISTOIAALLIANCE: *HELM in Online Databases*. https://pistoiaalliance.atlassian.net/wiki/spaces/PUB/pages/14286930/HELM+in+Online+Databases, Abruf: 08.03.2018

[30] PISTOIAALLIANCE: *HELM Project*. https://github.com/PistoiaHELM/, Abruf: 09.04.2018

[31] PISTOIAALLIANCE: *HELMNotationParser*. https://github.com/PistoiaHELM/HELMNotationParser, Abruf: 13.03.2018

[32] PISTOIAALLIANCE: *HELMNotationToolkit*. https://github.com/PistoiaHELM/HELMNotationToolkit, Abruf: 28.03.2018

[33] PYTHON: *Python Reference Version 3.0*. https://www.python.org/download/releases/3.0/, Abruf: 13.03.2018

[34] RAREY, M. : *Vorlesung: Grundlagen der Chemieinformatik*. 2015. – WS2015/2016

[35] SHERIDAN, R. P. ; KEARSLEY, S. K.: Why do we need so many chemical similarity search methods? In: *Journal of Molecular Biology* 7 (2002)

[36] SHERIDAN, R. P. ; MILLER, M. D.: A Method for Visualizing Recurrent Topological Substructures in Sets of Active Molecules. In: *Journal of Chemical Information and Computer Sciences* 38 (1998), S. 915–924

[37] STEINBECK, C. ; HAN, Y. ; KUHN, S. ; HORLACHER, O. ; LUTTMANN, E. ; WILLIGHAGEN, E. : The Chemistry Development Kit (CDK): An Open-Source Java Library for Chemo- and Bioinformatics. In: *Journal of Chemical Information and Computer Sciences* 43 (2003), S. 493–500

[38] STEINBECK, C. ; HOPPE, C. ; KUHN, S. ; FLORIS, M. ; GUHA, R. ; WILLIGHAGEN, E. L.: Recent Developments of the Chemistry Development Kit (CDK) - An Open-Source Java Library for Chemo- and Bioinformatics. In: *Current Pharmaceutical Design* 12 (2006), S. 2111–2120

[39] SUN, W. ; GUO, H. ; HE, H. ; DAI, Z. : Design and optimized implementation of the SHA-2(256, 384, 512) hash algorithms. In: *2007 7th International Conference on ASIC* (2007), S. 858–861

[40] ULLENBOOM, C. : *Java ist auch eine Insel*. Rheinwerk Verlag, 2009

[41] WEININGER, D. : Smiles, a chemical language and information system. 1. Introduction to methodology and encoding rules. In: *Journal of Chemical Information and Computer Sciences* 28 (1988), S. 31–36

[42] WILLETT, P. : Similarity-based virtual screening using 2D fingerprints. In: *Drug Discovery Today* 11 (2006), S. 1046–1053

[43] WILLETT, P. ; WINTERMAN, V. : A Comparison of Some Measures for the Determination of Inter-Molecular Structural Similarity Measures of Inter-Molecular Structural Similarity. In: *Molecular Informatics* 5 (1986), S. 18–25

[44] WILLETT, P. : Chemical Similarity Searching. In: *Journal Of Chemical Information and Computer Sciences* 38 (1998), S. 983–996

[45] WITT, K.-U. : *Mathematische Grundlagen für die Informatik: Mengen, Logik, Rekursion*. Springer, 2013

[46] ZHANG, T. ; LI, H. ; XI, H. ; STANTON, R. V. ; ROTSTEIN, S. : HELM: A Hierarchical Notation Language for Complex Biomolecule Structure Representation. In: *Journal Of Chemical Information and Modeling* 52 (2012), S. 2796–2806

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.
Ich bin mit einer Einstellung in den Bestand der Bibliothek des Fachbereiches Informatik einverstanden.

Hamburg, den _____ Unterschrift: _____